12/5/2018

مقدمة للبرمجة في لغة جافا

تصميم المبرمج الممتاز

Excellent programmer
HTTPS://WWW.FACEBOOK.COM/SAMEH5122002/

الفهرس

الباب الأول

البرمجة و البرمجيات

الفصل الأول

عناصر عامة عن المعلوماتية و البرمجة

في هذا الفصل الوجيز نلقي نظرة عامة على ميدان المعلوماتية لنعرّف القارئ على أهم مقوماته و أبرز مفاهيمه. فندرس على التوالي: أهم تطبيقات المعلوماتية، ثم أهم فروعها، ثم أهم أطوارها التاريخية. قبل ذلك، نقدم بعض الألفاظ و نعينها و نميز بينها.

1.1 تعيينات أساسية

إن كلمة المعلوماتية صادرة عن كلمة المعلومات. فنفتتح هذا الجزء بطرح السؤال: ما هي المعلومات. نهتم خاصة بالسؤال التالي: ما هو الفرق بين المعلومات، البيانات، المعطيات، و المعرفة. فنقترح التعيينات التالية:

اللفظة	التعيين	الترجمة
المعطيات، البيانات	رموز مجردة قد نحولها أو نعالجها لكن لا نعرف ضرورة ماذا تمثل	Data
المعلومات	بيانات أو معطيات نستطيع تأويلها و نفهم أبعادها	Information
المعرفة	كفاءات نكتسبها بمقتضى تطلعنا على المعلومات	Knowledge

على باب المثال، نعتبر ثلاثة أشخاص يفتحون صندوق إقتراع في ختام إنتخاب تشارك فيها مترشحان إثنان. نعتبر أن الشخص الأول مسؤول عن فرز بطاقات التصويت بين ثلاثة أقسام: قسم المترشح الأول؛ قسم المترشح الثاني؛ و قسم البطاقات الملغات. نعتبر أن هذا الشخص ليس طرفا في هذه الإنتخابات، و قد لا يتقن حتى قراءة الأسماء على البطاقات، لكنه يفرق بين الثلاثة أقسام. نعتبر أن الشخص الثاني ممثل أحد المترشحين، و أنه يراقب عملية الفرز ليتأكد أن بطاقات مترشحه يقع فرزها لصالحه. أما الشخص الثالث، فهو محلل سياسي يراقب نتائج الإنتخابات ليحلل حالة الرأي العام في البلاد و إنعكاسات التطورات السياسية الأخيرة على نتيجة الإنتخابات. فنقترح ما يلي: بالرغم من أن هؤلاء الثلاثة أشخاص ينظرون إلى نفس البطاقات، فيتلقون علامات مختلفة: يرى الشخص الأول هذه البطاقات كمصدر بيانات (معطيات)، و يراها الشخص الثاني كمصدر معلومات، و يراها الثالث كمصدر معرفة.

بناء على هذه التعيينات، نقدم بعض الألفاظ المركبة أو المشتقة، مثل المعلوماتية و معالجة (أو تحويل) المعلومات و معالجة (أو تحويل) البيانات و علم الحاسوب و علم الحساب الآلي و نظرية الحساب و هندسة المعلومات و أنظمة المعلومات و الأنظمة التصرفية للمعلومات و تقنيات المعلومات. فنقترح التعيينات التالية:

اللفظة	التعيين	الترجمة

Informatics	ميدان علمي و هندسي يهتم بعمليات معالجة و تحويل و تحليل و إستعمال المعلومات؛ كما يهتم بتصميم و إنجاز و تحليل العتاد و البرمجيات التي تمكننا من هذه العمليات.	المعلوماتية
Information processing	ميدان علمي و هندسي يهتم أساسا بتطبيقات المعلوماتية في معالجة المعلومات.	معالجة المعلومات، تحويل المعلومات
Data processing	ميدان علمي و هندسي يهتم أساسا بتطبيقات المعلوماتية في معالجة البيانات، خاصة بيانات التصرف في المؤسسات.	معالجة البيانات، تحويل البيانات
Computer science	ميدان علمي و هندسي تهتم أساسا بتصميم و إنجاز و تحليل و إستخدام وسائل آلية لمعالجة و تحويل و تحليل و إستعمال المعلومات.	علم الحاسوب
Computing Science	تشير هذه اللفضة لنفس المعنى كلفضة علم الحاسوب، مع إبراز النواحي الهندسية عوضا عن النواحي العلمية.	علم الحساب الآلي
Theory of Computing	ميدان علمي يهتم بنظرية الحساب الآلي و التلقائيات و الخوارزميات.	نظرية الحساب
Information Engineering	ميدان هندسي يهتم بتمثيل و خزن و صيانة و إستعمال المعلومات.	هندسة المعلومات
Information Systems	ميدان هندسي و علمي يهتم بإستعمال أنظمة آلية لخزن و إستعمال المعلومات.	أنظمة المعلومات
Management Information Systems	تطبيق أنظمة المعلومات في ميدان التصرف في المؤسسات.	الأنظمة التصرفية للمعلومات
Information Technology	لفضة عامة تشير للتقنيات المتعلقة بالمعلوماتية و تطبيقاتها.	تقنيات المعلومات
Data Engineering	ميدان هندسي يهتم بتمثيل و خزن و صيانة و إستعمال المعلومات.	هندسة البيانات

2.1 تطبيقات المعلوماتية

إنتشرت تطبيقات المعلوماتية لجميع الميادين الحياتية في المجتمعات العصرية، فأصبح العتاد و البرمجيات يلعبون دورا جوهريا في جميع القطاعات. نقسم تطبيقات المعلوماتية إلى ستة أقسام، حسب الترتيب التالي:

- تطبيقات التصرف. بالرغم من أن الإعلامية نشأت أساسا كوسيلة حساب آلي، فوجدت أوسع تطبيقاتها، منذ أطوارها الأولى، في ميدان التصرف في المؤسسات. يشمل هذا القسم تطبيقات مثل التصرف في العمليات التجارية، و التصرف في الموارد البشرية، و التصرف في العمليات الحسابية و الجبائية، و التصرف في خزن البضائع، و التصرف في قائمات المزودين و قائمات الحرفاء، و جدولة العمليات، و عمليات الحجز (كحجز البقاع في طائرة و حجز سيارات الكراء و حجز البقاع في تظاهرات ثقافية أو ترفيهية)، و التجارة الإلكترونية، إلخ.

- التطبيقات المالية. تكاملا مع تطبيقات التصرف، و تزامنا مع إنتشارها في ميدان التصرف، وقع إستعمال وسائل المعلوماتية في ميدان التطبيقات المالية. إذ أن جل عمليات التصرف غالبا ما تكون لها نواحي مالية لا تتجزء من النواحي التصرفية. فنجد ضمن هذا القسم تطبيقات مثل: تسجيل العمليات البنكية (تحويل من حساب إلى آخر)، و توفير العمليات البنكية على العنكبوتية، و تسيير موزعات النقود، و الدفع عن طريق بطاقات الإعارة و التخطيط المالي و تصرف في الحسابات البنكية، إلخ.

- التطبيقات الصناعية. غالبا ما يتطلب تسيير العمليات الصناعية طاقة في تحليل الأوضاع و تقييم الإختيارات و أخذ القرارات قد تفوق الطاقة البشرية العادية. بالتالي، فإنتشر إستعمال الوسائل الإعلامية في التطبيقات الصناعية ليأخذ مقام المسير البشري، خاصة في تطبيقات تتطلب إعتمادية رفيعة. تم إستعمال الحاسوب في الميدان الصناعي في عدد من التطبيقات، بما فيها: تسيير آلات الصناعة، مراقبة و تسيير تفاعلات (كالتفاعلات الكيميائية و التفاعلات النووية و غيرها)، جدولة العمليات الصناعية، أنظمة تأييد التصميم الصناعي، أنظمة تأييد الإنجاز الصناعي، تسيير آلات المراقبة و التحكم، إلخ.

- التطبيقات الإدارية. إنتشرت الوسائل المعلوماتية لتشمل قطاع الإدارة، فنجد في هذا القطاع تطبيقات مثل خزن دفاتر الحالة المدنية و التصرف فيها، خزن البيانات المتعلقة بالملكية و التصرف فيها، التصرف في المحاكم و الأحكام العدلية، خزن البيانات المتعلقة بالأبحاث في الجرائم و التصرف فيها، تقديم الخدمات الإدارية للمواطنين، التصرف في ميزانية الهياكل الإدارية في مختلف مستوياتها (الوزارات و المؤسسات و الولايات و المعتمديات و البلديات، إلخ)، التصرف في الموارد البشرية للإدارة، نمذجة المظاهر الإقتصادية و الإجتماعية، أنظمة تأييد القرارات، أنظمة الترجمة الآلية، إلخ.

- التطبيقات العلمية. كما ذكرنا أعلاه، فإن أول دافع لعلوم الحساب الآلي و المعلوماتية كان في ميدان التطبيقات العلمية. يشمل هذا الميدان تطبيقات مثل نمذجة المظاهر الطبيعية و التجارب الإفتراضية و نماذج التكهنات و حل المشاكل العددية و الأنظمة الخبيرة الطبية و أنظمة تسيير الآلات الطبية، إلخ.

- التطبيقات الفنية. إنتشر الحاسوب في العديد من القطاعات الفنية، فوقع إستعماله في خلق الأدوات الموسيقية و في تأليف الألحان الموسيقية و في إنتاج الأشرطة السينيمائية و في إنجاز الصور المتحركة، كما وقع إستعماله في قطاع الألعاب، الذي ما يزال يتسع و ينزاد جودة.

- التطبيقات التربوية. منذ أوائل تاريخ المعلوماتية، إهتم المدرسون بإستعمال الحاسوب لتوفير أنظمة تسمح للتلميذ أن يتعلم موضوعا ما لا من مدرس بشري، بل من مدرس آلي يكون على ذمة المتعلم بصفة مستمرة. بالإضافة لأنظمة التعليم الآلي، تشمل تطبيقات الحاسوب في قطاع التعليم أنظمة تؤيد التعليم العادي (كوابسيتي و واب بورد) و أنظمة تؤيد التعليم عن بعد و الجامعات الإفتراضية، و أنظمة تكشف التلاميذ الذين ينقلون تمارينهم عن آخرين (مثل نظام تورنت إين).

إن كل هذه التطبيقات ترتكز على توفير العتاد (في شكل آلات الحساب الآلي و آلات خزن البيانات و آلات حويل البيانات على متن شبكة الإنترنت)؛ لكنها ترتكز بالأخص على توفير برمجيات ضخمة و معقدة تسير هذا الآلات و تنسق بينها؛ نهتم في هذا الفصل بتصميم و إنجاز هذه البرمجيات.

3.1 فروع المعلوماتية

بينما إستعرضنا في الجزء السابق تطبيقات المعلوماتية، ففي هذا الجزء نهتم بإستعراض أهم فروعها. يمثل كل فرع مجموعة من المبادئ و المناهج و أساليب التفكير و الإستنتاج. نقدم فيما يلي عشرة فروع، ثم نناقش العلاقات بين الفروع و التطبيقات.

- التصميم المنطقي و معمارية الحاسوب. يهتم هذا الفرع بأهم المواضيع المتعلقة بتصميم و صنع و صيانة المدارات الإلكترونية التي تمثل مكونات الحاسوب، و بتركيبها لصنع الحاسوب.

- الخوارزميات و هياكل البيانات. إن الحاسوب في حد ذاته مجرد مدارات إلكترونية لا حياة لها. ما يجعله مفيدا هو البرامج التي ننفذها بواسطته. تتكون البرامج أساسا من عنصرين إثنين: هياكل البيانات، التي تمثل البيانات التي نريد تحويلها أو تحليلها أو معالجتها؛ و الخوارزميات، التي تضبط أساليب التحويل و التحليل و المعالجة. غالبا ما نهتم بهذين الموضوعين مع بعضهما بعضا، لأن كلاهما يؤثر على الآخر.

- هندسة البرمجيات. يهتم هذا الفرع بتصميم و تطوير و صيانة البرمجيات، و هي أنظمة برمجية ذات حجم كبير. و الجدير بالذكر أن هندسة البرمجيات، بما فيها التصميم و الصيانة و التطوير، إلخ، تختلف عن تصميم البرامج و الخوارزميات و هياكل البيانات لا من حيث الحجم فقط، بل أيضا من حيث النوعية. إذ أن التصرف في أنظمة برمجية كبيرة (يتراوح حجمها بين مئات الآلاف سطر و ملايين السطور) يثير مشاكل تنظيمية و تصرفية لا تبرز في نطاق البرامج الصغيرة الحجم.

- الذكاء الإصطناعي. بالرغم من أن المعلوماتية تطورت في مراحلها الأولى حول التطبيقات الحسابية أساسا (التي تعالج الأعداد)، فشهد فرع الذكاء الإصطناعي (الذي يعالج الرموز) إهتماما خاصا من طرف الباحثين و الممارسين. نخص بالذكر ميادين مثل الترجمة الآلية و معالجة اللغات الطبيعية و التعليم الآلي و الأنظمة الخبيرة و تمثيل المعرفة و غير ذلك من الوظائف التي ننسبها عادة لمقدرة الذكاء.

- قواعد البيانات. جل تطبيقات الحاسوب، خاصة منها التطبيقات التصرفية، ترتكز على قاعدة بيانات. تقوم قاعدة البيانات بخزن البيانات المعنية و صيانتها و وضعها تحت تصرف المستخدم. يشمل هذا الفرع الهام مسائل مثل تمثيل البيانات؛ و تنظيم هياكل الخزن؛ و تصميم خوارزميات التفتيش و تحليلها؛ و التصرف في القواعد المركزية و القواعد الموزعة؛ و الحرص على صيانة البيانات من الكشف و التغيير و الإباحة.

- أنظمة الإستخدام. كما ذكرنا سابقا، فإن الحاسوب مجرة مدارات إلترونية لا تتعاما إلا بالبيانات الثنائية (الصفر و الواحد) و لا تقوم إلا بعمليات بدائية. ما يجعل الحاسوب مرنا و قابلا للإستعمال هو البرمجيات التي تتنفذ عليه. يمكن أن نتصور أن هذه البرمجيات منظمة في شكل طبقات متتالية، أولها (أي الطبقة التي تتعامل مع الحاسوب مباشرة) هي نظام الإستخدام. تتمثل وظيفة هذا النظام في التصرف في موارد الحاسوب و تقديم الخدمات للمستعمل و حماية الحاسوب من المستخدمين و حماية المستخدمين من بعضهم و حماية المستخدمين من نفسهم.

- لغات البرمجة. تمثل لغات البرمجة أهم وسيلة نبلغ بها تعليماتنا للحاسوب لكي يقدم لنا الخدمات المطلوبة؛ بالتالي، فإن لغات البرمجة إكتست مقاما هاما في نطاق تطور المعلوماتية. يهتم هذا الفرع بتصميم و إنجاز و تحليل لغات البرمجة، كما يهتم بإنجاز الأنظمة الآلية التي تسمح لنا أن نستعمل هذه اللغات، مثل المصرفين و المؤولين و محيطات البرمجة، و محيطات الإختبار، إلخ.

- التحليل العددي. كان أول دافع لصنع آلات الحساب، تاريخيا، هو التطبيقات العددية. فنشأ فرع في الرياضيات

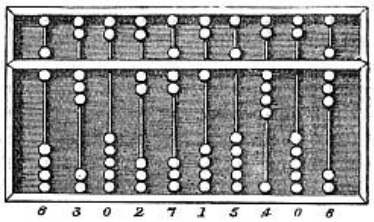
نلقي عليه إسم التحليل العددي أو الأساليب العددية أو الرياضيات التطبيقية. يهتم هذا الفرع بحل مشاكل عددية؛ و تصميم و تحليل خوارزميات تنطبق على بيانات عددية؛ و تصميم و تحليل آلات الحساب المختصة في العمليات العددية.

- واجهات التعامل بين الحاسوب و المستخدم. يمكن الإتصال بالحاسوب عن طريق عدد طبير من الوسائل. يبلغ المستخدم بيانات للحاسوب عن طريق لوحة المفاتيح، أو عن طريق الشاشات الحسّاسة، أو عن طريق الكرة المتجولة، أو عن طريق وحدات ذاكرة خارجية، أو عن طريق آلات الحس، إلخ. من ناحية أخرى، يبلغ الحاسوب نتائج للمستخدم عن طريق الشاشة، أو عن طريق آلة الطبع، أو عن طريق آلات الصوت، أو عن طريق وحدات ذاكرة خارجية، أو عن طريق آلات التحكم، إلخ. يهتم هذا الفرع بتصميم و تحليل وسائل التعامل بين الحاسوب و المستخدم، بما في ذلك الوسائل العتادية (التي ذكرناها أعلاه)؛ كما يشمل هذا الفرع أيضا الوسائل البرمجية، كتقديم البيانات على الشاشة و تصميم لوحات البيانات الواردة' إلخ.

يبين الجدول التالي العلاقات بين تطبيقات المعلوماتية و فروعها، فنضع علامة في خلية من الجدول كلما يتدخل الفرع المذكور أفقيا في تقديم التطبيق المذكور عموديا.

	الميادين	واجهات التعامل بين الحاسوب و المستخدم	التحليل العددي	لغات البرمجة	أنظمة الإستخدام	قواعد البيانات	الذكاء الإصطناعي	هندسة البرمجيات	الخوارزميات و هياكل البيانات	التصميم المنطقي و معمارية الحاسوب
التطبيقات										
في التصرف		√				√	√	√		
في المالية		√				√	√	√	√	
في الصناعة		√	√			√	√	√	√	√
في الإدارة		√				√	√	√		
في العلم			√	√	√			√	√	√
في الفنون		√	√				√		√	
في التعليم		√				√	√			

4.1 تاريخ المعلوماتية



يعتبر المؤرخون أن آلة المعداد، المتكونة من كويرات مصفوفة على مساطر متتالية، هي أول آلة حساب، فبالتالي يمكن إعتبارها كأول حاسوب إستعملها الناس. لا يعرف المؤرخون مصدر هذه الآلة، مع أنهم يعرفون أن إستعمالها كان جاريا في الصين و العراق منذ ثلاثة ألف سنة تقريبا، كما أن العديد من التجار ما زالوا يستعملونه في عدة مناطق في آسيا للقيام بعمليات حسابية.

1.4.1 الخوارزميات

بعد إنتشار الإسلام في الشرق الأوسط و شمال إفريقيا و غرب آسيا و جنوب أوروبا، إزدهرت الحياة العلمية في شتى الميادين العلمية كالطب و علم الفلك و الفيزياء و الرياضيات و غيرها. في العديد من هذه الميادين، ينطلق النشاط العلمي بترجمة المصادر اليونانية و تأويلها و تحليلها و تطويرها. في ميدان الرياضيات خاصة، قام عدد من العلماء بتحليل عمليات عددية عرفها اليونانيون و تعميمها لتشمل معادلات أصعب و عمليات أرقى.

ولد العالم الفارسي أبو جعفر محمد إبن موسى الخوارزمي سنة 783 ميلادي في مدينة خيفا بقطاع الخوارزم و توفي في بغداد سنة 850. نشر هذا العالم سنة 825 كتابا عنوانه الكتاب المختصر في حساب الجبر و المقابلة. يشمل هذا الكتاب ستة فصول، خصص كل فصل منهم لحل نوع خاص من المعادلات؛ و كان يقدم أساليب دقيقة لحل كل معادلة بواسطة خطوات متتالية مضبوطة. عندما قام الغربيون بترجمة الكتب العربية للغة اللاتينية، ألقوا إسم الخوارزم لكل أسلوب دقيق يُستعمل لحل مشكلة بواسطة خطوات مضبوطة. تطورت هذه اللفضة في اللغات الغربية لتدل على وصف مجرد لبرامج في المعلوماتية.

الجدير بالذكر أن كلمة الجبر، التي تدل على فرع من فروع الرياضيات يهتم بالهياكل الشكلية و بحل المعادلات، يرجع الفضل فيها لأبي جعفر الخوارزمي، و لا سيما لعنوان كتابه المذكور. كما نذكر أيضا أن إستعمال حرف الإكس (x) لتعيين مجهول المعادلة في جل اللغات الغربية، يرجع الفضل فيه أيضا لنفس العالم، الذي كان يمثل المجهول بحرف الش (الشئ المجهول).

2.4.1 مساطر الحساب

في القرن السابع عشر، قدم العالم السكوتلاندي جوهن نابيير جهازا يتمثل في لوحات عمودية رسم عليها أعدادا حسب الجدول المرسوم في الرسم المرافق. نتمكن بفضل هذا الجهاز من القيام بأي عملية ضرب بين عدد من 2 إلى 9 و عدد عشوائي بترصيف اللوحات العمودية التابعة للعدد المضروب فيه و قراءة النتائج الجزئية في السطر المناسب للضارب. مثلا، إذا أردنا أن نضرب 7 في 937 فنصفف اللوحات العمودية التابعة للمضروب فيه (937) و نجمع النتائج الموجودة في السطر التابع للضارب (7)، حيث نجد على التوالي:

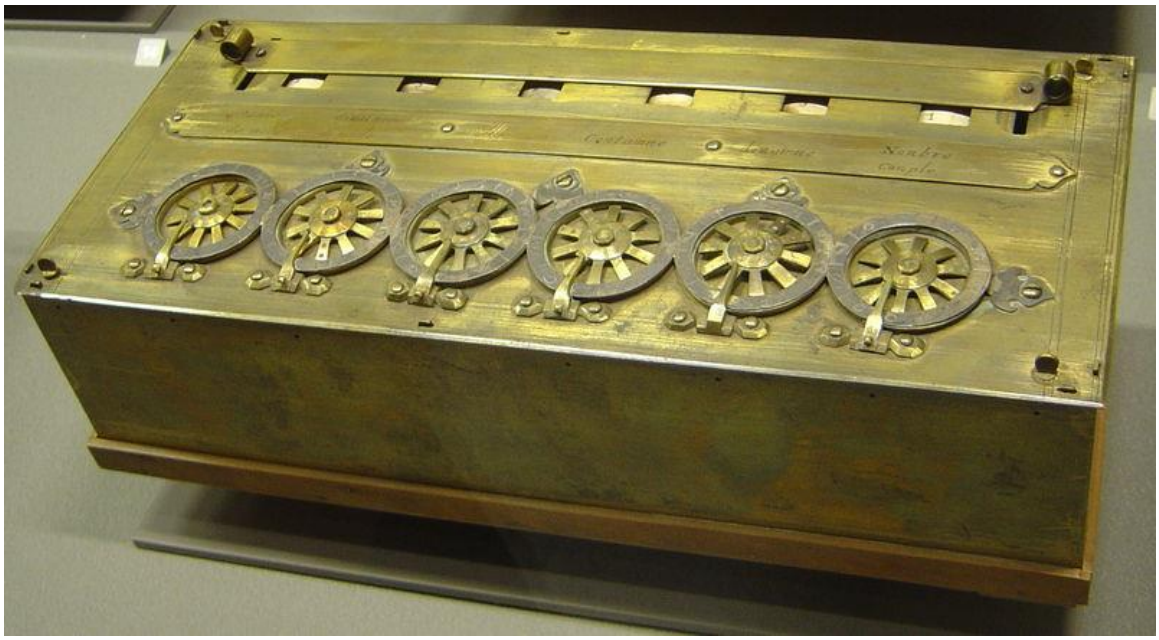
6
2+3
4+1

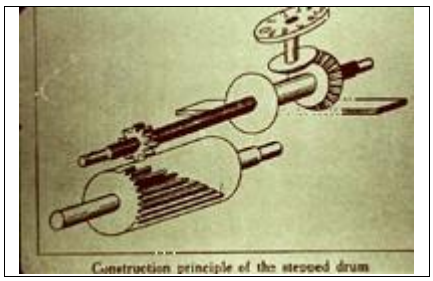
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

إذا قمنا بعمليات الجمع و رصفنا هذه الأعداد من اليسار إلى اليمين، لوجدنا:
6559،
و هو نتيجة عملية الضرب: 7 في 937. بالإضافة لهذه المائدة، قدم العالم نابيير جداول تمثل قيمات وظيفة اللوغاريثما. بناء على هذه المائدات، صنع العالم الأنجليزي ويليام أوتراد آلات تقوم بعمليات عددية مستعملين وظيفة اللوغاريثما. تطورت فكرة ويليام أورتاد فأصبحت مسطرة الحساب، التي واصل الممارسون إستعمالها حتى منتصف القرن العشرين؛ لكن في نفس الوقت، تطورت فكرة الحساب الآلي، كما نستعرض فيما يلي.

3.4.1 آلات الحساب

كان العالم الفرنسي بلاز باسكال (مولود في 1623 ميلادي و متوفي في 1662 ميلادي) أول من إخترع آلة تقوم بعمليات حسابية عن طريق أسلوب آلي. كانت آلة باسكال، التي أطلق عليها إسم باسكالين، تقوم بعمليات الجمع و الطرح بواسطة عدد من الدواليب (ستة أو ثمانية)، تحرك بعضها بعضا، حيث أن كل دولاب يتكون من عشرة أسنان. تمثل هذه الدواليب الأعداد الطبيعية في تمثيلها العشري، و تتمثل عمليات الجمع و الطرح في دوران هذه الدواليب حيث أن كل دولاب يتجاوز التسعة (في الجمع) أو الصفر (في الطرح) يحرك الدولاب الموالي (كما نفعل يدويا في العمليات العددية). صنع باسكال خمسين نسخة من آلته؛ تمثل الورة أسفله نسخة من هذه الآلة ذات ستة دواليب، صنعها سنة 1652، و هي معروضة حاليا في متحف التقنيات و المهن بباريس، فرنسا. بالرغم من أنها لم تعرف نجاحا تجاريا كبيرا، فإن آلة باسكال تسببت في مظاهرات من طرف المحتسبين، الذين إعتبروا أنها سعوضهم فتقضي على قطاعهم المهني.





بعد وفاة باسكال بحوالي عشر سنوات، قام العالم الألماني غوتفريد ويلهالم فون لايبنيتز (عاش من سنة 1646 إلى سنة 1716) بتحسين آلة باسكال، حيث قدم آلة حساب تتمكن من القيام بعمليات الجمع و الطرح، مثل آلة باسكال، كما تقوم أيضا بعمليات الضرب و القسمة. و كانت من أهم مكونات هذه الآلة دولابا ذات أسنان متزايدة الطول، مصفوفة على إسطوانة طويلة، كما يدل الرسم المرافق. فيقوم هذا

الدولاب بتدوير دواليب مختلفة في أطوار مختلفة. تم إنجاز هذه الآلة سنة 1694، و تم توزيعها على مستوى واسع نسبيا.

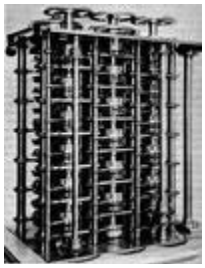
عندما قرر الأستاذ السويسري نيكلاوس ويرث أن يخترع لغة برمجة بسيطة يستعملها قصد تدريس البرمجة في المعهد التقني الإتحادي بزوريخ، ألقى عليها إسم باسكال، تكريما للعالم الفرنسي بلاز باسكال.

4.4.1 البرامج المخزونة

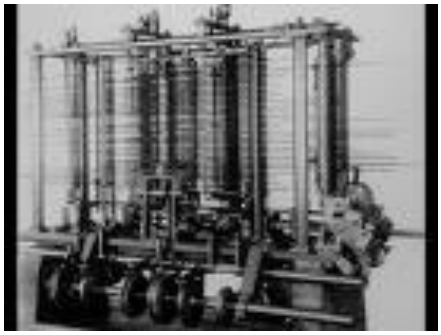
من خصائص الآلتين التين إستعرضناهما سابقا (آلتي باسكال و لايبنيتز) أنهما لا تقومان إلا بعمليات معينة وقع ضبطها عند صنع الآلة. في سنة 1801، قدم مهندس فرنسي جوسف ماري جاكارد آلة نسيج تتمكن من نسيج أشكال مختلفة حسب نماذج مسجلة في لوحات حديدية؛ حيث أن المستخدم يختار لوحة ضمن مجموعة من اللوحات، فتنسج الآلة النموذج المطابق للوحة المختارة. نعتبر أن أهم فرق بين آلة حساب (التي تقوم بعمليات حسابية تحت مراقبة المستخدم) و حاسوب (الذي يقوم بتنفيذ برنامج عشوائي الطول و التعقيد) هو أن الحاسوب يتمكن من تنفيذ برامج عشوائية وقع تصميمها بصفة مستقلة من صنع الحاسوب. بالرغم من أن منسج جاكارد لا يُعتبر حاسوبا، فنعتبر أن لوحات جاكارد تمثل أول مثال لبرنامج بالمعنى العصري للمفهوم. نالت آلة جاكارد نجاحا نسبيا عبر أوروبا، مع أنها تسببت في مظاهرات من طرف النساجين، خاصة في بريطانيا، الذين إعتبروا أنها ستعوضهم في عملهم.

5.4.1 أول حاسوب و أول مبرمج

بينما آلات بسكال و لايبنيتز تقوم بعمليات حسابية لكنها لا تقبل برامج، و أن منسج جاكارد يقبل برامج مخزونة لكنه لا يقوم بعمليات حساب (بل ينفذ تعليمات نسيج)، فإن الآلة الموالية، التي ندرسها فيما يلي، تقوم بعمليات حسابية و تخضع لبرامج مخزونة. في أواخر القرن التاسع عشر، كان الممارسون في الهندسة و المحاسبة و غيرها من الميادين المبنية على عمليات حسابية، يقومون بجل العمليات الحسابية بواسطة جداول وقع تأليفها سابقا. كانت هذه الجداول صعبة و بطيئة الإستعمال، و كانت غالبا ملئانة بالأخطاء.



قام العالم البريطاني شارل بابّج (الذي عاش من سنة 1792 إلى سنة 1871) بتصميم آلة حاسوب، تقوم بعمليات حسابية متتالية حسب تعليمات معينة، و بادر بإنجازها إبتداء من سنة 1822، بفضل تمويل و تشجيع الحكومة البريطانية؛ ألقى بابج على هذه الآلة إسم محرك الطرح. تتكون هذه الآلة من دواليب مصفوفة تدور بعضها بعضا، و كان بابّج ينوي إستعمالها قصد تقييم وظائف عددية مبنية على عمليات الجمع و الطرح و الضرب. بعد إحدى عشر سنة، عدل بابّج عن صنع هذه الآلة قبل أن يتم إنجازها تماما، و وجه إهتمامه لآلة أخرى.



إنتداء من سنة 1833، شرع شارل بابّج في تصميم آلة حاسوب ثانية، ألقى عليها إسم محرك التحليل، مستعملا الدروس التي تلقاها في إنجاز محرك الطرح. تتكون هذه الآلة من خمسين ألف قطعة، و هي أرقى بكثير من حيث تصميمها و خدماتها و إمكانياتها.
الجدير بالذكر أن من خلال هذه الآلة، سطر العالم شارل بابّج أهم أسس معمارية الحاسوب العصري، بما فيها الأربعة مقومات الأساسية: وحدة المعالجة، وحدة الخزن (الذاكرة)، وحدة القراءة (لجلب البيانات الواردة) و وحدة الكتابة (لنشر البيانات الصادرة). كانت وحدة المعالجة تقوم بالعمليات العددية العادية و كانت الذاكرة تستعيب ما لا يقل عن ألف عدد ذات خمسين رقم. هذا و كانت الآلة تقبل البيانات الواردة في شكل لوحات مثقوبة و تنشر البيانات الصادرة عن طريق آلة طبع. كان شارل بابّج ينوي بناء هذه الآلة لكي تسير بواسطة البخار، و كان

يتوقع أن يكون حجمها يساوي تقريبا حجم محرك القطار. بالرغم من أن هذه الآلة لم يتم إنجازها، فنعتبرها أول حاسوب، لأنها تمثل أهم الأفكار و المفاهيم التي يتسم بها الحاسوب العصري.

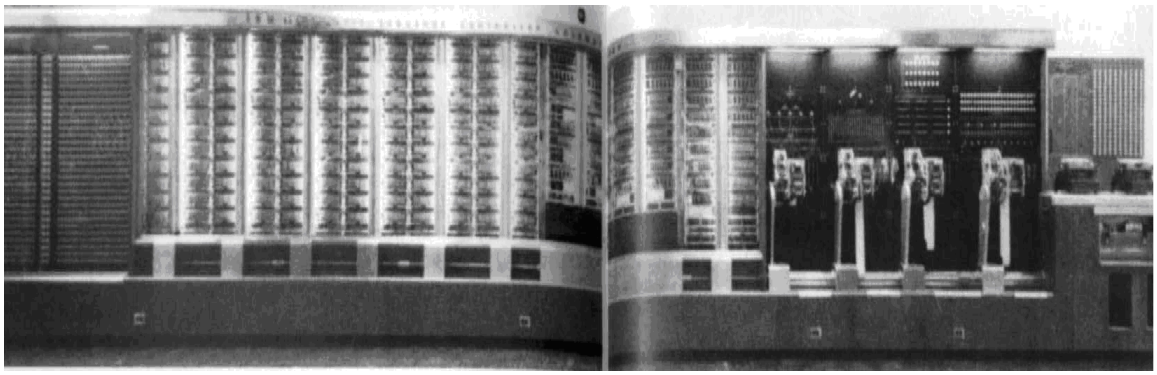
بينما كان في مخاض تصميم و إنجاز محرك التحليل، إلتقى شارل بابّج بالعالمة البريطانية أوغسطا آدا بايرون (التي عاشت من 1815 إلى 1852)، التي تأثرت كثيرا بمشروعه، و أدركت أبعاده و آفاقه. كما أنها تكهنت بأن محرك التحليل، عندما يتم إنجازه، سيكون قادرا على تأليف الألحان الموسيقية، و معالجة الصور و التكهن بالمظاهر الطبيعية و تطبيقات علمية و هندسية. كما نعلم اليوم، فإن تكهنات أوغسطا آدا بايرون كلها تحققت. من أهم مساهمات السيدة آدا في تاريخ المعلوماتية هي أنها عرضت على بابّج برنامجا يمكن تنفيذه على محرك التحليل قصد إستخراج قيمات وظائف هامة معروفة بإسم أعداد برنويي (يقال أن هذه العالمة كانت الشخص الوحيد، بالإضافة لبابّج نفسه، الذي يفهم سير و تسيير محرك التحليل). بالرغم من أن محرك التحليل لم يتم إنجازه، يعتبر المؤرخون أن البرنامج الذي كتبته أوغسطا آدا بايرون لتقييم أعداد برنويي هو أول برنامج كُتب في التاريخ، فقيل عن أوغسطا آدا بايرون أنها أول مبرمج في التاريخ. في أواخر السبعينات (من القرن العشؤين) نظمت وزارة الدفاع الأمريكية مناظرة دولية لبعث لغة برمجة جديدة، ففاز بالمناظرة فريق فرنسي ألقى على لغته إسم آدا، تكريما لأوغسطا آدا بايرون.

6.4.1 أول حاسوب ثنائي

بينما شارل بابّج قام بتصميم حاسوب عصري ذات معمارية متطورة، فهو لم يقم بإنجازه، نظرا لأن متطلبات هذا الحاسوب تتجاوز إمكانيات الآلات البخارية التي إستعملها لهذا الغرض. فجاء العالم الألماني كُنراد زوس في العشرية الرابعة من القرن العشرين و أنجز حاسوبا حسب معمارية بابّج، مستعملا في ذلك تمثيلا تنائيا للبيانات بواسطة محوّلات كهربائية كانت مُستعمَلة آنذاك في صنع أجهزة الربط الهاتفي. بالإضافة لهذا الحاسوب، قدم كنراد زوس لغة برمجة عصرية ألقى عليها إسم بلان كالكول (أي: مخطط الحساب)، و كتب فيها عددا من التطبيقات، تشمل عمليات عددية و عمليات ترتيب جداول و ألعاب الشطرنج، إلخ.

لم يكن كنراد زوس راضيا عن آلته، نظرا لأنها بطيئة جدا، فإقترح للحكومة الألمانية بناء نموذجا آخر من هذا الحاسوب يستعمل أنابيب الفراغ عوضا عن المحولات الكهربائية. عكسا للمحولات الكهربائية، التي تتطلب حركات بدنية، فإن أنابيب الفراغ مبنية أساسا على عمليات إلكترونية، و بالتالي فهي أسرع بكثير. كانت ألمانيا آنذاك في مخاض الحرب العالمية الثانية، و كانت واثقة من النصر، فلم تر داعيا لتمويل هذا المشروع.

بينما تعرض تطور المعلوماتية لهذه الحواجز في صفوف المحور، فكانت البحوث تتطور بسرعة فائقة في صفوف الحلفاء. نذكر بالخصوص أعمال العالم البريطاني آلان تورنغ الذي أشرف على إنجاز عدد من الآلات تختص في الكشف عن أساليب التلغيز التي إستعملها الألمانيون في إتصالاتهم العسكرية؛ كما قدم آلان تورنغ الاسس النظرية لميدان المعلوماتية و البرمجة، بما فيها مفهوم آلة تورنغ المعروف. كما نذكر أيضا صنع حاسوب في الولايات المتحدة مماثل لحاسوب كنراد زوس، من طرف جامعة هارفرد و شركة أي بي أم. تمّ صنع هذا الحاسوب سنة 1945، و أُلقي عليه إسم مارك الأول.

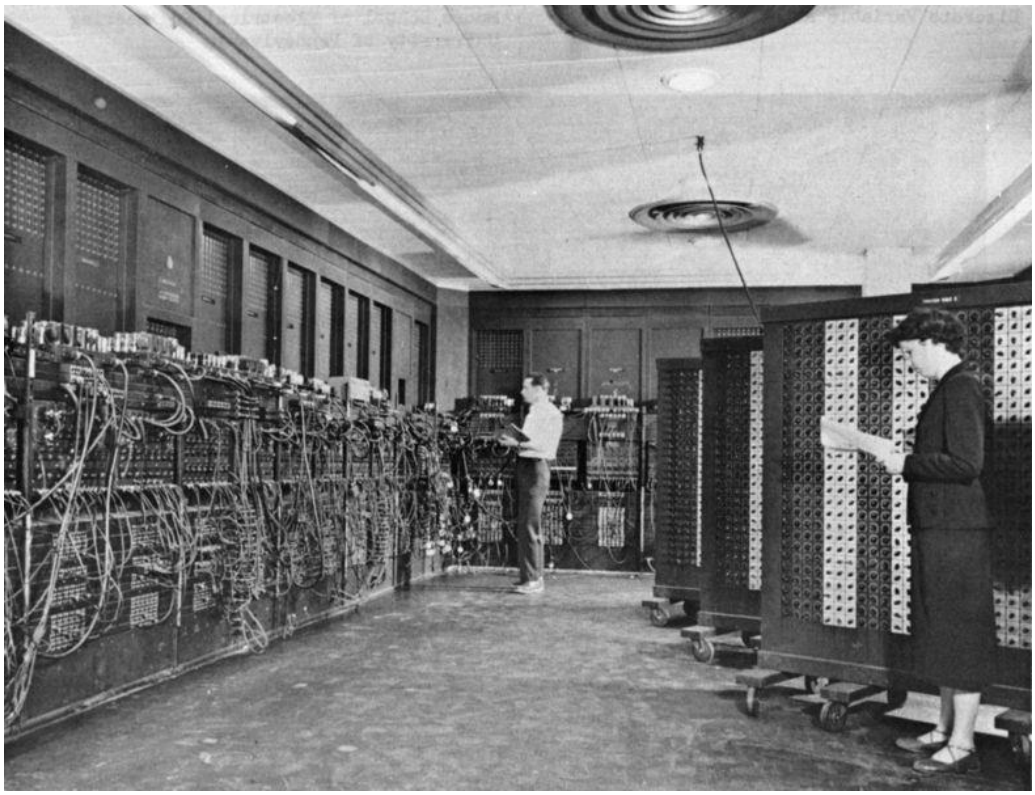


7.4.1 أول حاسوب إلكتروني

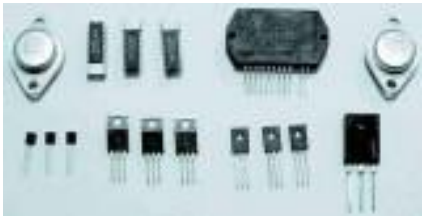
قام الباحثان جوهن آتاناسوف و كلفورد باري بإنجاز أول حاسوب إلكتروني في جامعة آيُوَا ستات بالولايات المتحدة من سنة 1937 إلى سنة 1942، و كان تصميمهم يجسم أهم الأفكار المعاصرة في صنع الحواسيب، بما فيها إستعمال أنابيب الفراغ و التمثيل الثنائي للبيانات و التصميم المنطقي للمدارات الإلكترونية. من سنة 1943 إلى سنة 1946 قام باحثان آخران من جامعة بنسيلفانيا بالملايات المتحدة، براسبر آكارت و جوهن موشلي، بتصميم حاسوب آخر يستعمل نفس الوسائل التقنية، فأطلقوا عليه إسم الإينياك. كان هذا الحاسوب يتكون من 18000 أنبوب فراغ و 70000 أنبوب مقاومة، فكان يزن ثلاثين طنا و يستهلك طاقة كهربائية تتجاوز المائة و ستين ألف واطّ. كانت هذه الآلة تقوم بعمليات عددية بسرعة تفوق سرعة مارك الأول بألف مرة، لكنها كانت تتعطل كثيرا بسبب تعطل أنابيب الفراغ العديدة. هذا، و كانت أيضا تتلقى البيانات الواردة عن طريق عملية بطيئة جدا تتمثل في تغيير المدارات الإلكترونية للحاسوب، مما يحول دون إستغلال السرعة النسبية المرتفعة لوحدة الحساب و يجعل عملية البرمجة معرضة للأخطاء.

تفاديا لهذا العيب الكبير في تصميم حاسوب آكارت و موشلي، تقدم العالم الأمريكي جون فون نويمان بمعمارية تنص على أن يقع تأليف و طبع البرنامج خارج الحاسوب، مع إدخاله عند التنفيذ و خزنه في ذاكرة الحاسوب قصد تنفيذه. تنص هذه المعمارية على عدد من الأفكار الجذرية، منها أن الحاسوب يتكون من وحدة معالجة و وحدة ذاكرة و وحدات توريد و إصدار البيانات و هيكلية تحتية تتحمل تبادل البيانات و البرامج بين هذه الوحدات. أُطلق على هذه المعمارية أسم معمارية فون نويمان، و مازالت سائرة المفعول إلى يومنا هذا.

نقول عن الحواسيب المذكورة في هذا الجزء، و التي تتسم بأنها مصنوعة من أنابيب الفراغ، أنها حواسيب الجيل الأول. أشهر حاسوب يمثل هذا الجيل هو حاسوب الإينياك، الذي نمثله في الصورة الموالية.



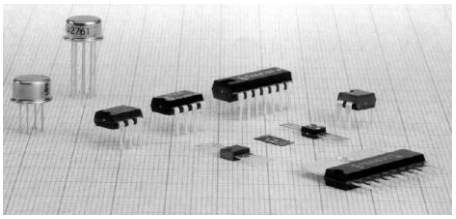
8.4.1 حواسيب الجيل الثاني: من 1956 إلى 1963



في أواخر الأربعينات، قام باحثون من مخابر شركة الهاتف بالّ (في ولاية نيو جيرسي) بإختراع مكونة إلكترونية ألقوا عليها إسم ترانزيزتور، يمكن إستعمالها لمراقبة عبور (أو توقيف) علامة كهربائية بين نقطتين؛ نلقي عليها إسم عدسة العبور، أو العبورية. سرعان ما تفطن الباحثون لتطبيقات العبورية في صنع الحواسيب، فتم إنجاز أول حاسوب مبني بالعبوريات سنة 1958، فعوضت العبورية أنبوب الفراغ بصفة نهائية في صنع الآلات الإلكترونية. قامت شركة آي بي آم الأمريكية بصنع هذا الحاسوب، تحت إسم آي بي آم 7090، و إحتلت بفضل هذا الحاسوب مكانة في طليعة القطاع، إحتفظت بها مدة طويلة. ساهم إختراع العبورية في تطور تصميم الحواسيب، و ساهم أيضا و خصوصا في دفع الهندسة الإلكترونية بصفة عامة، بما فيها جهازات الراديو و التلفزة و التسجيل و غيرها.

شهدت فترة أواخر الخمسينات تطورات جذرية في تصميم اللغات البرمجية، فصدرت على التوالي لغات هامة جدا مثل لغة فورطران (للتطبيقات العلمية / الهندسية) و كوبول (للتطبيقات التصرفية) و ليسب (لتطبيقات الذكاء الإصطناعي) و آلغول (الموجه لتصميم الخوارزميات). كما شهدت هذه الفترة تطورات هامة في إنجاز مصرفات و مؤولات لهذه اللغات البرمجية.

9.4.1 حواسيب الجيل الثالث: الستينات



بينما تتسم حواسيب الجيل الثاني بإستعمال العبورية كعنصر أساسي، فإن الجيل الثالث يتسم بإستعمال المدار المدمج. تتمثل المدارات المدمجة في عناصر إلكترونية صغيرة تقوم بعمليات مختصة (كعمليات عددية أو عمليات مراقبة أو خزن بيانات). يعوض كل مدار مدمج آلاف العبوريات، من حيث الوضيفة التي يقوم بها، بينما يكون حجمه أصغر بكثير من العدد المطابق للعبوريات، و يكون إستغلال الطاقة الكهربائية أقل بكثير من العدد المطابق للعبوريات. بينما أتاخ إختراع العبورية بعث هندسة الإلكترونية، فإن إختراع المدار المدمج أتاح بعث هندسة المعالج الصغير. فإنتشر إستعمال المعالج الصغير ليشمل تطبيقات مختلفة كالمراقبة الآلية للمراكب (قطارات، حافلات، طائرات، مثلا)، و تسيير الآلات المنزلية و مراقبة الآليات و تسيير المفاعلات الكيميائية، إلى غير ذلك.

كان أول حاسوب يمثل هذا الجيل هو حاسوب أي بي أم 360, من صنع شركة أي بي أم. من أهم الأفكار الثورية التي جاء بها هذا الحاسوب هي أنه حاسوب ذات هدف عامّ. أي أنه لم يقع تصميمه لحل مشكلة خاصة أو لتطبيق خاص، كما جرت العادة حتى ذلك الوقت، لكنه صُنع لمجموعة واسعة من المستخدمين، تختلف متطلباتهم و شروطهم. مما يجعل هذا الحاسوب يكتسي أهمية تاريخية كبيرة هو أيضا أنه كان أول فرصة للتعرض لما نسميه اليوم أزمة البرمجيات. فبعثت شركة أي بي أم مشروع صنع عتاد الحاسوب في نفس الوقت الذي بعثت فيه مشروع صنع نظام الإستغلال المرافق للعتاد. فبينما تم إنجاز العتاد في وقته و في نطاق ميزانيته المالية، فإن نظام الإستغلال تجاوز الميزانية المالية المخصصة له، كما تأخر كثيرا في الإنجاز، فأخر بالتالي تسليم الآلات لأصحابها، و تسبب في خسائر كبيرة لشركة أي بي أم. بالرغم من هده الصعوبات الأولية، عرف حاسوب ال أي بي أم 360 نجاحا تجاريا كبيرا، خيث كان يباع منه قرابة الألف نسخة شهريا، فتلاه حاسوب آخر من نفس المعمارية لكنه أرقى، أُطلق عليه إسم أي بي أم 370. تمثل الصورة التالية حاسوب أي بي أم 360.

كان حاسوب أي بي أم 360 يلعب دورا مركزيا في المؤسسات التي تستعمله، سواء كانت مؤسسات تجارية أو صناعية أو تربوية/ جامعية/ بحثية. نظرا لتكاليفه المرتفعة جدا، كان هذا الحاسوب يتحمل جميع الوظائف التي يمكن إسنادها له، كالتطبيقات التصرفية و الهندسية و التجارية و غيرها. فأصبح الناس يشسرون لهذا الحاسوب (و لحواسيب مماثلة) بلفضة الحاسوب المركزي، معتبرين أن كل مؤسسة لا يمكن أن تملك أكثر من واحد.

بعد إصدار حاسوب الأي بي أم 360 بسنة، أي سنة 1965، صنعت شركة داك (شركة العتاد الرقمي) حاسوبا صغيرا، إسمه بي دي بي 8، أقل مقدرة من حاسوب أل360، لكنه أقل ثمنا بكثير. فعرف هذا الحاسوب نجاحا تجاريا كبيرا بفضل بساطته، ثمنه الباهض، م عدد من الأفكار الجديدية المتعلقة بالعمارية و أنظمة الإسغلال.

أسفر نجاح البي دي بي 8 على إهتمام متزايد بالحواسيب الصغيرة، و بالتالي بالحواسيب المتنقلة، أي الحواسيب التي توضع على متن مراكب لتراقب سيرها أو تؤيد تسييرها. ففي سنة 1968 صنعت شركة رايثيون حاسوبا مختصا، حسب تصميم من المعهد التقني لماساشوستس، يتكلف بالقيادة الرئيسة، البحارة و نظام التحكم، التابعة للمراكب الفضائية من سسلة آبولو. وُضع هذا الحاسوب على متن آبولو السابع سنة 1968 لتقوده في دورانه حول الأرض، ثم وُضع على متن آبولو الحادي عشر في رحلته إلى القمر. يختلف هذا الحاسوب المختص عن الحواسيب الأخرى في نفس البرنامج، كالحاسوب الذي يسير صاروخ ساترن الخامس عند إنطلاق المهمة و الحاسوب الذي يتحكم في هبوط المركب القمري على القمر.

شهدت أواخر الستينات عددا من التطورات الهامة في ميدان الحاسوب نذكر منها بالخصوص إختراع لغة باسكال (التي كانت لها أثر عميق على تطور ميدان لغات البرمجة) و تصميم نظام الإستغلال يونكس (الذي مازال يُستعمل في العديد من الأنظمة العصرية). كما شهدت أيضا تصميم و إنجاز أول شبكة حواسيب مرتبطة عبر مسافات طويلة

(شبكة أربانات، التي تطورت أفكارها لتصبح الهيكلية التحتية للعنكبوتية العالمية المعروفة حاليا)؛ و شهدت أيضا بعث الوسائل العصرية للمعاملة بين الحاسوب و المستخدم، بما فيها إدخال البيانات / التوجيهات عن طريق الفأرة و نشر البيانات / النتائج عن طريق النوافذ على الشاشة (مما لعب دورا عظيما في تسخير الحاسوب ليصبح في متناول عامية المستخدمين).

10.4.1 حواسيب الجيل الرابع: السبعينات

تتسم فترة السبعينات من القرن العشرين بعدد من الإختراات و التطورات الجذرية في ميدان المعلوماتية، نبوبها إلى خمسة أبواب.

الباب الأول: المكونات الإلكترونية. بينما كانت حواسيب الجيل الثالث مبنية على أساس المدارات المدمجة، فإن أهم مكون أساسي لحواسيب الجيل الرابع هو المدارات الدقيقة، التي تضع في مساحات صغيرة من عاج السيليكون عشرات آلاف البوابات المنطقية، فتكون مقدرتها في سرعة العمليات و في خزن البيانات تفوق بكثير حواسيب كبيرة سابقة، كانت تفوقها ثمنا و حجما و إستغلالا. دفع هذا الإختراع تصميم الحواسيب في إتجاهين إثنين: الإتجاه الأول يخص الحواسيب الدقيقة، التي أصبحت ممكنة نظرا للحجم الصغير و الثمن الباهض و الإستغلال الضئيل؛ و الإتجاه الثاني هو الحواسيب العضيمة، التي أصبحت ممكنة نظرا لإمكانية المعالجة السريعة و الخزن الواسع النطاق. نظر في هذين الإتجاهين فيما يلي.

الباب الثاني: الحاسوب الدقيق. في سنة 1974، قام مهندسان إسمهما ويليام غاتس و بول آلان بإنجاز مصرف لحاسوب صغير، بُني على أساس مدار دقيق من شركة إنتال. في نفس السنة، قام مهندسان إسمهما ستيفن جوبس و ستيفن فوسنياك بصنع حاسوب صغير أطلقوا عليه إسم آبل الأول؛ يتبعه بعد سنتين حاسوب آخر سنة 1976 إسمه آبل الثاني. نظرا لثمنه الباهض و لوجود برمجيات أساسية له، نال الآبل الثاني (الصورة المجاورة) نجاحا تجاريا كبيرا، خاصة لدى المؤسسات التربوية و الجامعية. يحتوي هذا الحاسوب على أهم العناصر التي يتسم بها الحاسوب الشخصي العصري، بما فيها لوحة المفاتيح و الشاشة و وحدات القرص، إلخ. كما نعلم اليوم، كان مجهود ويليام غاتس و بول آلان أول خطوة في تأسيس شركة مايكروسوفت، و كان مجهود ستيفن جوبس و ستيفن فوسنياك أول خطوة في تأسيس شركة آبل. تأسست هذه الشركتان سنة 1976.

الباب الثالث: الحاسوب العظيم. بينما إستغل المهندسون المشار إليهم أعلاه السعر المنخفض و الحجم الصغير و الإستهلاك الضئيل للمدارات الدقيقة لبناء حواسيب شخصية، إستغل المهندس سيمور كراي مقدرة هذه المدارات في المعالجة السريعة و خزن البيانات الضخمة لصنع حاسوب عضيم، سنة 1976، سماه كراي الأول. تتكون هذه الآلة من 1662 لوحة إلكترونية و تقوم بما لا يقل عن 800 مائة مليون عملية عددية في الثانية. بُني هذا الحاسوب في شكل نصف دائرة (كما نرى في الصورة المجاورة) لكي تكون المكونات التي تتبادل الكثير من المعلومات بجانب بعضها بعضا. توضع اللوحات الإلكترونية في الأبراج العمودية، بينما تختص الوحدات السفلى في تبريد الآلة بضخ سائل الفريون بين اللوحات الإلكترونية بإستمرار. كانت هذه الآلة تزن 5.5 طنا، بما فيها سائل الفريون المُستعمل في التبريد، و كانت تستهلك 115 ألف واط من الطاقة الكهربائية، مع أن ذاكرتها تستعيب مليون كلمة ذات 64 ثنائي. كانت تختص هذه الآلة، طبعا، في البرامج العددية، و كانت مزودة بنظام إستغلال مماثل ليونكس و بمصرف للغة فرطران. كان مخبر لوس آلاموس التابع لوزارة الطاقة الأمريكية أول من تسلم نسخة من هذه الآلة، إذ أن هذا المخبر يحتاج لمقدرة عملية كبيرة في تطبيقات مثل النمذجة و التكهن و تحليل المفاعل النووي، إلخ.

الباب الر ابع: لغات البرمجة. واصل الباحث كان طومسن أشغاله على نظام الإستخدام يونكس، فعندما أراد تحويل هذا النظام إلى حواسيب أخرى دون الحاسوب الذي وُضع عليه في البداية، وجد أن كل لغات البرمجة الموجودة أنذاك غير لائقة لهذا الغرض. فإخترغ لغة برمجة جديدة ألقى عليها إسم سي و إستعملها ليكتب 90 بالمائة من نظام الإستغلال. أصبحت هذه اللغة سائدة في أوساط البحث و التعليم و الممارسة، فتولدت عنها لغات أخرى مثل سي++ و جافا و غيرها.

الباب الخامس: الشبكات. في نطاق شبكات الحواسيب، قام باحثون في مخبر زيروكس بارك معمارية للشبكات المحلية عُرفت بإسم إيثرنت (شبكة الأثير). فتبنت المجموعة هذه المعمارية كمعيار في هذا الميدان.

11.4.1 حواسيب الجيل الخامس: الثمانينات

إن المعلوماتية كما نمارسها اليوم ناتجة عن ثلاثة تطورات طرأت بصفة متزامنة (و مستقلة، نوع ما) في الثمانينات، و هي: تطور الحاسوب الشخصي، و برمجة العنكبوتية و شبكة الحواسيب. تظافرت هذه التطورات لتكون بنية أساسية للعنكبوتية العالمية كما نعرفها اليوم، و للنشاطات المختلفة التي تدور عليها. في نفس الفترة، نذكر تطوران إثنان كانت لهما أهمية تاريخية كبيرة: الأول هو مجهود حواسيب الجيل الخامس، و البرمجة المنطقية المصاحبة له؛ و الثاني هو مجهود الإستعمالية في البرمجيات، و البرمجة الموجهة للكائنات المصاحبة له.

الباب الأول: الحاسوب الشخصي. في آوخر السبعينات، كانت شركة أي بي أم من أكبر الشركات التي تصنع الحواسيب في العالم، لكنها كانت غائبة عن سوق الحواسيب الدقيقية (الحواسيب الشخصية)، إذ أنها كانت تختص أساسا في صنع و صيانة حواسيب كبيرة و ثمينة. فقررت هذه الشركة أن تدخل هذه السوق الجديدة، و إتخذت لهذا الغرض تدابير تبعد كل البعد عن عاداتها السابقة في التصميم و التسويق. من أهم القرارات التي إتخذها فريق التصميم في صنح هذا الحاسوب، نذكر مثلا: 1) صنع هذا الحاسوب، لا من مكونات تصنعها أي بي أم، بل من مكونات موجودة في السوق. 2) إستعمال نظام إستغلال خارجي، دون الأنظمة الموجودة لدى الشركة آنذاك، فوقع إختيارهم على نظام أم أس دوس، من صنع شركة صغيرة إسمها مايكروسوفت. 3) تبني معمارية مفتوحة، تسمح لشركات أخرى أن يصنعوا و يصدروا آلات تتماشى مع هذا الحاسوب، فتكمل خدماته. 4) لم تتكلف شركة أي بي أم بتسويق هذا الحاسوب، فتحالفت مع شركات تجارية عامة لتتكلف بتسويقه (إذ أن هيكل التسويق لهذه الشركة كان عندئذ مختص في تسويق آلات كبيرة و ثمينة لشركات كبيرة، لا لتسويق آلات صغيرة و رخيصة لعامة المستعملين). صدر هذا الحاسوب في شهر أوت 1981، تحت إسم أي بي أم بي سي (الذي يشير للفضة الحاسوب الشخصي)، فأسفر إصداره عن إنطلاق قطاع واسع من النشاطات سواء في العتاد أو في البرمجيات. نظرا للمعمارية المفتوحة لهذا الحاسوب، تمكنت العديد من الشركات أن تختص في صنع قطع ملحقة لهذا الحاسوب، مثل آلات الطبع و وحدات الأقراص و الشاشات و لوحات المفاتيح و غيرها. من ناحية أخرى، نظرا أيضا للمعمارية المفتوحة، إنطلقت عدة شركات في صنع حواسيب مماثلة لهذا الحاسوب، نذكر منها شركة كمباك التي أصدرت أول حاسوب مطابق للاي بي أم بي سي سنة 1982. بفضل مكانة شركة أي بي أم آنذاك، يمكن القول أن إصدار هذا الحاسوب يمثل في الواقع إصدار معيار أكثر مما هو إصدار آلة خاصة. تمثل أهمية البي سي في أنه وضع أدوات المعلوماتية في متناول مجموعة كبيرة من الأفراد و المؤسسات الصغيرة، ففتح المجال للعديد من الإمكانيات.

الباب الثاني: شبكة الحواسيب. شهدت الثمانيات تطورات هامة في ميدان شبكات الحواسيب، فتطورت فكرة شبكة الحواسيب من الطور التجريبي (خلال شبكة آربانت) إلى الطور التطبيقي (خلال شبكة أن أس أف نات). بينما كانت شبكة آربانت مجرد مشروع بحث، فإن أن أس أف نات أصبحت مؤسسة تستخدم شبكة حواسيب لتمكن عددا متزايدا من المؤسسات التربوية و غيرها من الإتصال ببعضها بعضا. في نفس الفترة، نشأت مراسم جديدة للإتصالات بين الحواسيب و شبكات الحواسيب، تحت إسم تي سي بي / أي بي (مراسم مراقبة التنقل / مراسم الإنترنت). فأصبحت هذه معيارا للإتصال بين الشبكات الموجودة، حتى لو كانت صادرة عن مؤسسات مختلفة. ساهمت هذه المراسم الموحدة في إنتشار شبكات الحواسيب، فأصبحت تشمل عددا متزايدا من البلدان في العالم. إن العنكبوتية العالمية كما نعرفها و نمارسها اليوم ناتجة عن توفير شبكة الحواسيب، لكنها لم تكن ممكنة لولا مفهوما آخر، و هو مفهوم شبكة النصوص. ففي أوائل عهد شبكات الحواسيب، كان المستخدمون يستعملون الإنترنت للمراسلة الإلكترونية و لتبادل البيانات، و للمعالجة الموزعة، و للعمل على حاسوب عن بعد، إلى غير ذلك.

الباب الثالث: شبكة النصوص. عندما نكتب نصا لنعبر عن فكرة ما، نظطر لتنظيم هذا النص بصفة خطية، بحيث نكتب فصلا بعد فصل، و جزءا بعد جزء، و فقرة بعد فقرة، و كلمة بعد كلمة، إلخ. لكن العديد من الأفكار ليست خطية، بل تتمثل في علاقات متداخلة و معقدة بين العديد من الأطراف. لهذا، إهتم العديد من الباحثين بمفهوم شبكة النصوص، الذي ينص على تمثيل الأفكار بشكل يعكس هيكل الفكرة، عوضا عن الشكل الخطي المسترسل للنصوص. وجدت هذه الفكرة أول تطبيق لها سنة 1980 في المركز ألأوروبي للطاقة النووية بجينيف، سويسرا، حيث ألف الباحث البريطاني تيم بارنارس لي نظاما، سماه إنكواير (عن كلمة أنجليزية تعني: إستخبر)، يمكنه من الإطلاع على عدد من الوثائق الموجودة على حواسيب مختلفة بمجرد إتباع علاقات تربط بين هذه الوثائق. في سنة 1989، إهتدى الباحث تيم بارنارس لي للإمكانيات الشاسعة التس قد تنجر عن تصريف مفاهيم شبكة النصوص مع شبكة الحواسيب

فقدم نظاما ألقى عليه إسم العنكبوتية العالمية، برّره بأنه يمكن باحثين موزعين في جميع أنحاء العالم من أن يساهموا في مشروع بحث مشترك، بواسطة موارد مشتركة (قد تكون بيانات أو برامج أو نتائج، إلخ).

الباب الرابع: البرمجة الموجهة للكائنات و الإستعمالية. بعد أن أيقن الممارسون و الباحثون من عمق و حدة إزمة البرمجيات خلال الستينات، و بعد أن حاولوا العديد من الحلول، بدون جدوى، خلال السبعينات، أيقنوا خلال الثمانيات أن الحل الوحيد لأزمة البرمجيات تتمثل في إتقان إستعمالية البرمجيات. عوضا عن إنجاز كل منتوج برمجي من جديد، تنص إستعمالية البرمجيات على صنع و خزن و تنظيم منتوجات برمجية مهيئة للإستعمال المكرر، بحيث أن إنجاز برمجات كبيرة يتم بتركيب و تنسيق منتوجات موجودة. يخول هذا المنهج إنتاج أنظمة برمجية بأقل تكاليف (حيث أننا نستعمل منتوجات موجودة عوضا عن إعادة تأليف كل نظام جديد) و بأحسن جودة (إذ أن المنتوجات المستعملة يقع التأكد من جودتها خلال إستعمال واسع النطاق) و في أقصر وقت (إذ أن إنجاز نظام جديد يتمثل أساسا في تركيب و تنسيق مكونات موجودة). فجائت البرمجة الموجهة للكائنات لتقدم وسائل لغوية تأيد إستعمالية البرمجيات، فتطور إنتشارها و الإهتمام فيها خلال الثمانيات، مواكبا إنتشار الإستعمالية و الإهتمام بها.

الباب الخامس: البرمجة المنطقية و الذكاء الإصطناعي. خلال الثمانيات قامت وزارة الصناعة و التقنيات في اليابان بمجهود واسع النطاق يرمي لبعث ما سمّوه حواسيب الجيل الخامس. ترمي هذه البادرة لإدماج أرقى الأفكار في العتاد و البرجيات لإنجاز الجيل الجديد من الحواسيب. في نطاق العتاد، إهتم الباحثون بإنجاز معماريات تنسق بين الآلاف من وحدات المعالجة؛ كما إهتموا بصنح حواسيب تختص، لا في العمليات العددية (كما جرت العادة إلى ذلك الوقت) لكن تختص في عمليات الإستنتاج المنطقي. أما في نطاق البرمجيات، فإهتم الباحثون بتطبيقات تهم الذكاء الإصطناعي و البرمجة المنطقية. كانوا يبرروا إختيار البرمجة المنطقية على أساس أن كل لغات البرمجة السابقة ترمي إلى تقريب المستخدم للحاسوب، بينما حواسيب الجيل الخامس ترمي إلى تقريب الحاسوب للمستخدم البشري، فبالتالي يجب أن تتكلم اللغة البشرية، و هي لغة المنطق. كما كانوا يهتمون بالمعرفة عوضا عن البيانات كالوحدة الأساسية للمعالجة فأصبحوا يتحدثون عن قواعد المعرفة عوضا عن قواعد البيانات, و عن محركات الإستنتاج عوضا عن الخوارزميات، إلخ. ردود فعل على البادرة اليابانية، قام عدد من البلدان الأخرى، مثل الولايات المتحدة و كندا و بريطانيا و أوروبا بمجهودات مماثلة. تعرضت مشاريع الجيل الخامس للعديد من الصعوبات، ناتجة أساسا على تناقضات في متطلبات هذه المشاريع، نذكر منها مثلا أن لغات البرمجة المنطقية لا تتماشى مع معماريات الجيل الخامس، التي تستعمل عددا واسعا من الحواسيب الموازية. ففشلت مجهودات الجيل الخامس في تحقيق أهدافها، و زال الإهتمام بها في أواخر الثمانينات.

12.4.1 التاريخ الحديث و إنتشار العنكبوتية

تواصل تطور المعلوماتية و تطبيقاتها حول تطورات العنكبوتية العالمية. ففي سنة 1991 صدر أول برنامج للملاحة على العنكبوتية، أُلقي عليه إسم غوفر. كما وقع تعويض شبكة آبرانت بشبكة أن أس أف نات، و واصلت العديد من البلدان الربط بهذه الشبكة الجديدة. كانت تونس أول بلاد عربية إرتبطت بالعنكبوتية العالمية، سنة 1991، فتلتها (ضمن البلدان العربية) الكويت سنة 1992 و مصر سنة 1993 و الجزائر و المغرب سنة 1994، إلخ. في نفس السنة، نشأت أول برمجة للتفتيش على العنكبوتية (ياهوو) كما نشات أول برمجة تجارية للبحارة (نتسكاب). نشأت لغة جافا، موضوع هذا الكتاب، كلغة مختصة في البرمجة على العنكبوتية، سنة 1995. في نفس السنة، تنقلت العنكبوتية من نطاق حكومي للخدمات إلى نطاق تجاري، فإنطلقت شركات تقدم خدمات الإنترنت (مثل أمريكا أُنلاين) كما إنطلقت شركات لتوزيع العناوين على العنكبوتية و لإستضافة مواقع العنكبوتية. أصدرت شركة مايكروسوفت أربعة إصدارات متتالية لنظام الإستخدام تعوض نظام دوس، و هي أنظمة وندوس 1995، وندوس 1998، وندوس 2000، وندوس أكس بي و (في سنة 2007) وندوس فيستا. أهتم العالم إهتماما كبيرا في السنوات الأخيرة من القرن العشرين بمشكلة السنة 2000، الناتجة عن أن عددا كبيرا من البرامج المكتوبة في السنين السابقة تمثل التاريخ بعددين إثنين (88 عوضا عن 1988، مثلا)، فكان البعض يخشي أن هذه البرامج قد لا تفرق بين سنة 2000 و سنة 1900 (إذ أن كلاها تتمثل في 00)، مما قد يسفر عن أخطاء لا يمكن النبؤ بها. في نطاق العتاد، تتمثل أهم التطورات الحديثة في ميدان الحاسوب في إنتشار الحاسوب المحمول (أو الحاسوب المتجول): فأصدرت شركة آبل أول حاسوب متجول (بالمعنى الحديث) سنة 1991 تحت أسم باوربوك 100، تلتها شركة أي بي أم بحاسوب سنة 1992

تحت إسم ثنكباد 700. كما نذكر إنتشار حاسوب اليد (أو حاسوب الجيب)، و هو حاسوب صغير يُحمل في اليد، فصدرت أول آلة من هذا النوع سنة 1996 تحت إسم بالم بايلت.

5.1 تمارين

1. [س] ما هو الفرق بين حاسوب و آلة حساب؟

2. [س] ما هو الفرق بين شبكة الحواسيب و مفهوم سبكة النصوص.

3. [س] أذكروا عددا من تطبيقات شبكة الحواسيب دون العنكبوتية العالمية.

4. [س] ما هي أهم الأفكار/ التطورات التي جاء بها محرك الطرح؟

5. [س] ما هي أهم الأفكار/ التطورات التي جاء بها محرك التحليل؟

6. [س] ما هي أهم الأفكار/ التطورات التي جاء بها مارك الأول ؟

7. [س] ما هي أهم الأفكار/ التطورات التي جاء بها الإينياك ؟

8. [س] ما هي أهم الأفكار/ التطورات التي جاء بها آي بي آم 7090 ؟

9. [س] ما هي أهم الأفكار/ التطورات التي جاء بها آي بي آم 360 ؟

10. [س] ما هي أهم الأفكار/ التطورات التي جاء بها آبل الأول؟

6.1 مصادر و مراجع

إن المواضيع التي طرقناها في هذا الفصل بخصوص تاريخ الحاسوب و المعلوماتية مواضيع عادية توجد في عدت كتب عامة تهتم بالمعلوماتية أو بالبرمجة. أما عن كتب تختص في تاريخ الحاسوب و المعلوماتية، فنحيل القارئ المهتم لكتاب [سيروزي، 2003] و لكتاب [إفراح، 2001].

الفصل الثاني
مقدمة للبرمجة في لغة جافا

تتمثل البرمجة في تحرير تعليمات مهيكلة ينفذها الحاسوب لكي يؤدي وظيفة ما. سواء كانت هذه الوظيفة تتمثل في حل معادلة رياضية أو في أخذ قرار ضمن مجموعة من البديلات أو في تحليل بيانات تجريبية أو في ترجمة آلية من لغة طبيعية إلى أخرى أو في تسيير نظام آلي أو في مراقبة مفاعلة كيميائية أو في القيام بعملية تجارية، فإن نشاط البرمجة هو أساسا لا يختلف. فيتمثل هذا النشاط في تحديد هياكل بيانات تمثل الكائنات التي تهمنا، مع تعيين و تنظيم عمليات تتصرف في هذه البيانات.

نستهل هذا الفصل بتقديم بسطة على معمارية الحاسوب، ثم نبين بإيجاز تطور البرنامج عبر الدورة الحياتية من إدخاله للحاسوب إلى تنفيذه.

1.2 الدورة الحياتية للبرنامج

1.1.2 معمارية الحاسوب

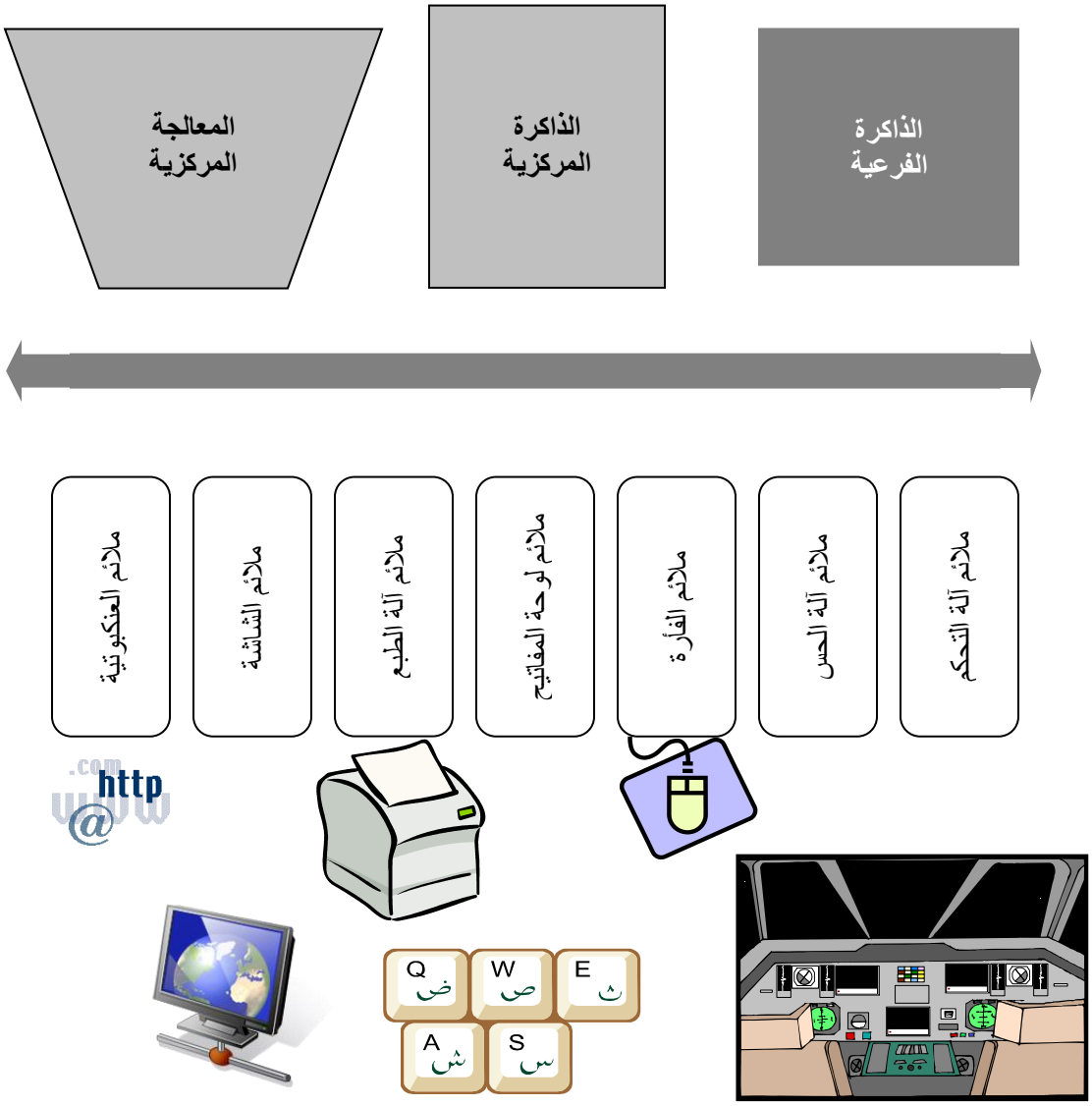
تطور الحاسوب تطورا فائقا منذ الأربعينات، فإزدادت سرعته، و إرتفعت طاقة ذاكرته، و إنخفض سعر إنتاجه، و خف وزنه، و إنخفض إستهلاكه الكهربائي، إلخ... لكن خلال كل هذه الأطوار، لم تتغير معماريته، فبقيت على منوال معمارية فون نويمان، التي نصفها فيما يلي. تتكون معمارية الحاسوب من العناصر التالية:

- وحدة معالجة مركزية، تقوم بعمليات عددية و منطقية، و بعمليات تحكم.
- وحدة ذاكرة مركزية، أو مخزن مركزي، تقوم بخزن البرامج و البيانات.
- وحدات الإصدار و التوريد، أو وحدات الإخراج و الإدخال، تقوم بتحويل البيانات و البرامج بين الحاسوب و محيطه. تشمل وحدات الإصدار، مثلا: جهاز الطبع، الشاشة، القرص الصلب، آلة التحكم، مضخم الصوت، قناة تربط الحاسوب مع شبكة ما (الإنترنت، شبكة محلية، إلخ)؛ يقع ربط كل جهاز إصدار ببرنامج خاص بهذا الجهاز، يختص في تسييره. تشمل وحدات التوريد، مثلا: لوحة المفاتيح، الفأرة، آلة الحسّ، قناة تربط الحاسوب مع شبكة ما (الإنترنت، شبكة محلية، إلخ)، آلة تصوير، إلخ.
- وحدة الذاكرة الفرعية، و هي تشير لوسائل خزن البيانات تكون عادة أكبر حجما و أقل سرعة و أقل تكلفة نسبية (أي حسب الخانة الثنائية) من الذاكرة المركزية. يشمل هذا المفهوم الأقراص الصلبة، لكن قد يشمل أيضا وسائل أخرى مثل وحدات القرص اللين أو وحدات القرص المدمج، أو غيرها.
- هياكل التنقل، تقوم بنقل البيانات و إشارات التحكم بين مكونات النظام.

أم وحدة المعالجة المركزية، فتتكون من ثلاثة عناصر، و هي:

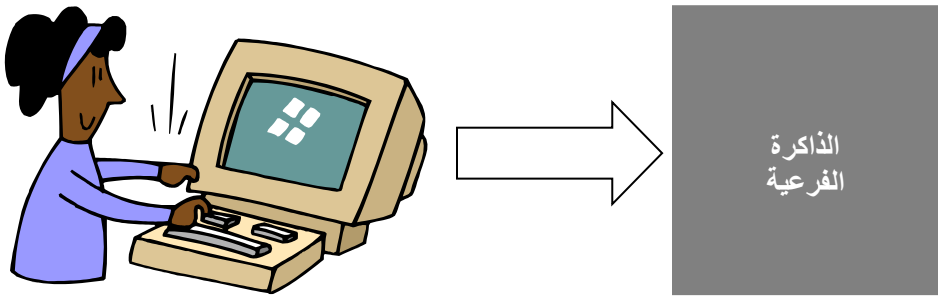
- وحدة الحساب العددي و المنطقي، التي تقوم بعمليات بعمليات عددية و منطقية.
- وحدة التحكم، التي تقوم بتسلسل و تسيير و تنسيق عمليات وحدة المعالجة المركزية.
- لوحة السجلات، التي تقوم بخزن نتائج وقتية خلال عمليات وحدة المعالجة المركزية.

في حالة حاسوب عادي، يقع تركيب كل هذه المكونات على هيكل مركزي نلقي عليه إسم اللوحة الأم. نحرض القارئ أن يتأمل الرسم رقم 1.2 للمزيد من التوضيح. يبين هذا الرسم نوعين من الذاكرة: الذاكرة المركزية، التي تتسم بأنها سريعة لكنها صغيرة / محدودة الحجم؛ و الذاكرة الفرعية، التي تتسم بأنها كبيرة الحجم، لكنها بطيئة نسبيا. على أساس هذه الأوصاف، نخصص الذاكرة القصيرة المدى للبرامج و البيانات المستعدة للتنفيذ فورا، و نستعمل الذاكرة الفرعية للخزن الطويل المدى.



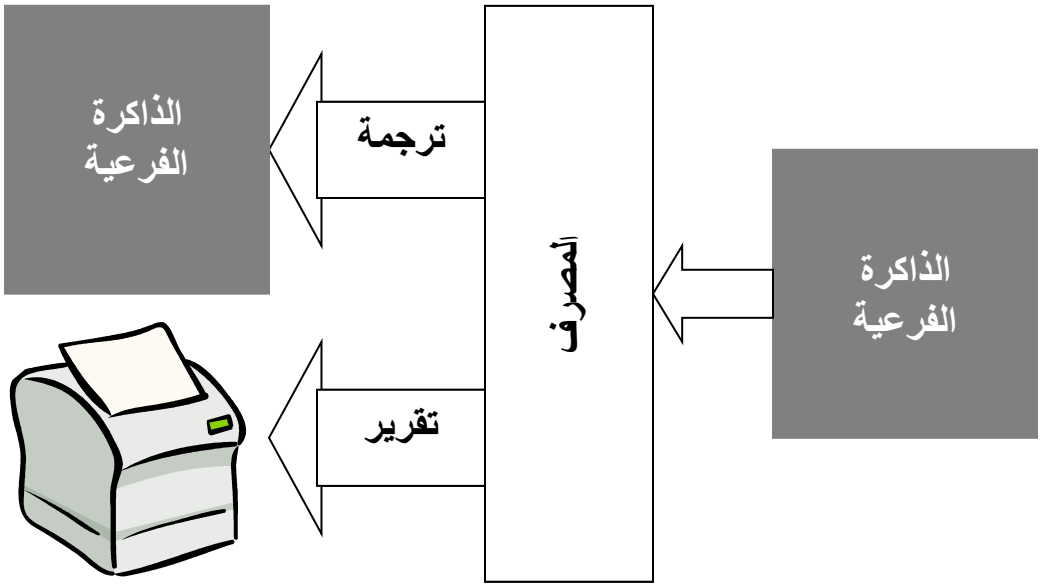
2.1.2 تأليف البرنامج

يتمثل البرنامج في نص مهيكل يأمر الحاسوب بالقيام بعمليات معينة، و يقع تأليف البرنامج من طرف مبرمج مختص، كما يكتب الشاعر قصيدة و يكتب المؤلف قصة و يكتب الإمام خطبة، إلخ. أي أن كتابة البرنامج نشاط إبداعي يجتهد فيه المبرمج بتلبية حاجة ما. فأول طور في الدورة الحياتية للبرنامج هي أن يقع تأليف هذا البرنامج و إدخاله للحاسوب بواسطة بواسطة لوحة المفاتيح، مستعملين في ذلك برنامج نسميه معالج النصوص. فعندما يتم إدخال البرنامج بواسطة معالج النصوص، يقع خزنه عادة في الذاكرة الفرعية (نستعمل الذاكرة الفرعية لغرض الخزن طويل المدى، بينما نستعمل الذاكرة المركزية لخزن البرامج المستعدة للتنفيذ و لخزن البيانات المعرضة للتنفيذ).



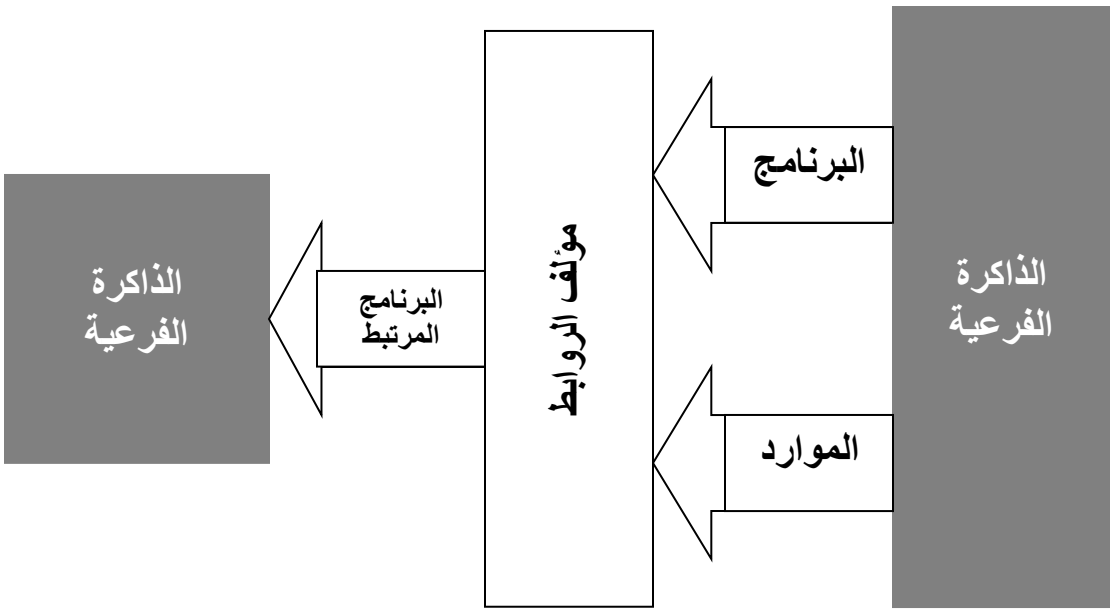
2.1.2 صرف البرنامج

بعد أن يتم تأليف البرنامج و خزنه، قد يعتزم المبرمج أن يحلل برنامجه قصد تصليح الأغلاط أو التأكد من دقته إلخ. فبعد ذلك نقوم بترجمة البرنامج من نصه الأصلي إلى شكل يفهمه الحاسوب، و يتم ذلك عن طريق برنامج يسمى المصرف. فيتمثل دور المصرف في أن يحلل نص البرنامج ليتأكد من أنه يطابق القواعد النحوية التابعة للغة البرمجة؛ ثم، إذا تبين أن البرنامج صحيح نحويا، يقوم المصرف بترجمته، ثم يضع البرنامج المترجم في الذاكرة الفرعية، لأن ترجمة البرنامج و تنفيذه عمليتان مختلفتان. أما إذا لم يكن نص البرنامج يطابق القواعد النحوية التابعة للغة البرمجة، فيقوم المصرف بتقديم تقرير مفصل عن الأخطاء النحوية و لا يترجم نص البرنامج.



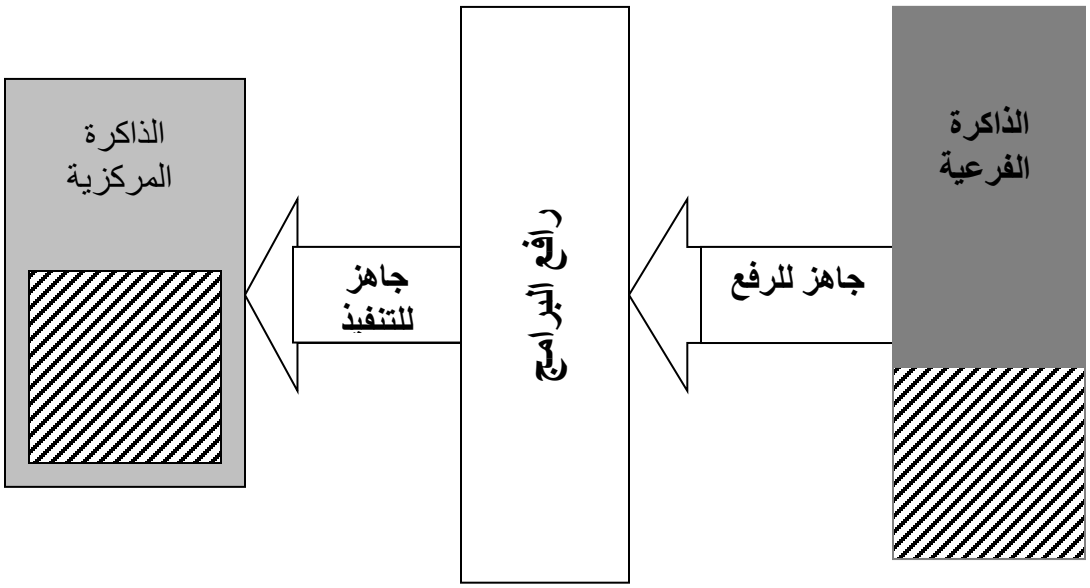
3.1.2 ربط البرنامج

عندما نكتب برنامجا، غالبا ما نحتاج لكائنات أو وضائف أو أنواع بيانات تخرج عن نطاق لغة البرنامج، لكن يضعها البرنامج تحت تصرفنا عند الحاجة. مثلا، إذا كتبنا برنامجا يتعلق بحساب المثلثات، قد نحتاج لأساليب تحسب وضيفة الجيب و مرافق الجيب. فيأتي مؤلف الروابط ليألف صلات الربط بين البرنامج المعني و الموارد التي يحتاجها (من وضائف و كائنات و أنواع بيانات، إلخ). فيكون من البرنامج المعني و الموارد المطلوبة كتلة موحدة جاهزة للتحويل (للذاكرة المركزية) و للتنفيذ. طبعا، إن الموارد التي يضعها الحاسوب تحت تصرف المبرمج غالبا ما تكون مخزونة في الذاكرة الفرعية، لأنها لا تُستعمل إلا قليلا في العادة. هذا، و علما أن عملية الربط منفصلة عن عملية الرفع (التحويل للذاكرة المركزية) و التنفيذ، فإن مؤلف الروابط غالبا ما يعيد البرنامج المرتبط إلى الذاكرة الفرعية من جديد، حسب ما يبين الرسم التالي:



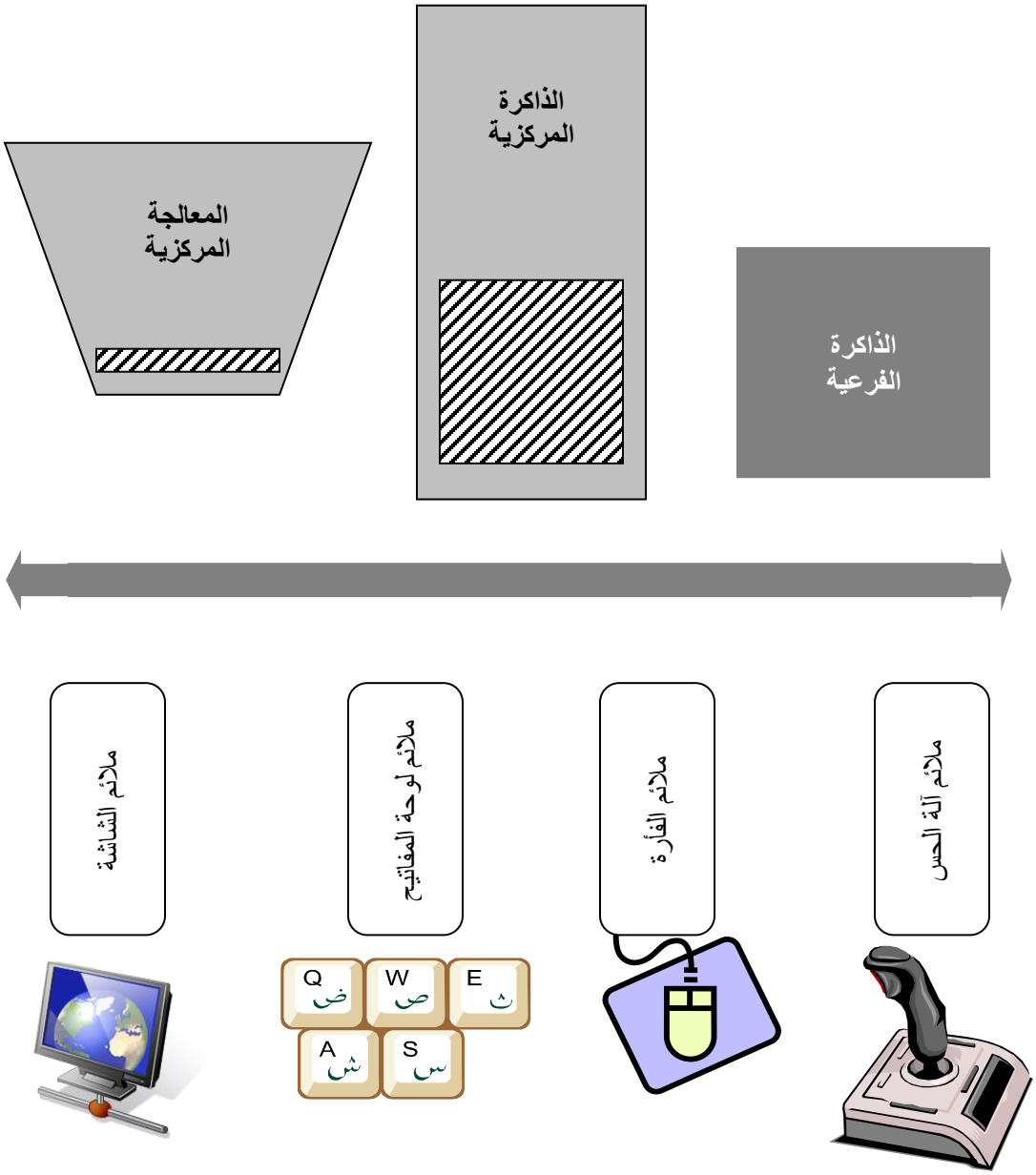
4.1.2 رفع البرنامج

يتمثل هذا الطور في حجز فضاء للبرنامج و ما يرافقه من موارد في الذاكرة المركزية ثم تحويله من الذاكرة الفرعية إلى الذاكرة المركزية. نظرا لإكتظاظ الذاكرة المركزية و صغر حجمها، لا نضع برنامجا فيها إلا إذا كان في طور التنفيذ أو على وشك التنفيذ. بالتالي، فغالبا ما ندمج طور رفع البرنامج مع طور تنفيذه، لكننا نفرق بينهما في نقاشنا هذا، لأسباب توضيحية.



5.1.2 تنفيذ البرنامج

عندما يكون البرنامج موجودا في الذاكرة المركزية و تكون نداءاته للموارد المطلوبة كلها مربوطة، عندئذ يكون جاهزا للتنفيذ. يتمثل التنفيذ في تحويل تعليماته الواحدة بعد الأخرى من الذاكرة المركزية إلى وحدة المعالجة المركزية. فتنطبق هذه التعليمات على متغيرات موجودة في الذاكرة المركزية أو على حاملات بيانات موجودة في الذاكرة الفرعية. كما قد يستعمل البرنامج قنوات أخرى لتصدير و توريد البيانات، فقد يستورد بيانات من لوحة المفاتيح أو من الفأرة، أو من آلة الحس، مثلا؛ و قد يصدر بيانات إلى الشاشة أو إلى مضخم الصوت، مثلا.



إستعرضنا فيما سلف أطوار الدورة الحياتية لبرنامج جافا، التي تشمل التأليف و الصرف و الربط و الرفع و التنفيذ. تطبيقيا، لا تتمثل هذه الأطوار ضرورة في خطوات زمنية مختلفة. فمثلا، غالبا ما يقوم المصرف بتأليف الروابط، كما غالبا ما يقوم رافع البرنامج بتنفيذه أيضا، فتكون خطوتي الرفع و التنفيذ مندمجة تقع في حدث واحد. هذا كما كانت بعض المصرّفات القديمة تقوم بكل من الصرف و تأليف الروابط و الرفع و التنفيذ في آن واحد. عكسا لذلك، فإن بعض الأنظمة العصرية تفصل بين كل هذه الخطى للمزينة من المرونة. فغالبا ما يقع صرف البرنامج على حاسوب و يقع تأليف الروابط على حاسوب آخر، بحيث لا يمكن للمصرف أن يقوم بتأليف الروابط لأنها تشير إلى حاسوب آخر.

بينما نظرنا، في هذا الجزء، في مسيرة البرنامج منذ تأليفه إلى تنفيذه، ننكب في الجزء الموالي على عملية تأليف البرنامج بالذات، فنقترب لهذه المسألة في إطار نظام تعليمي بسيط.

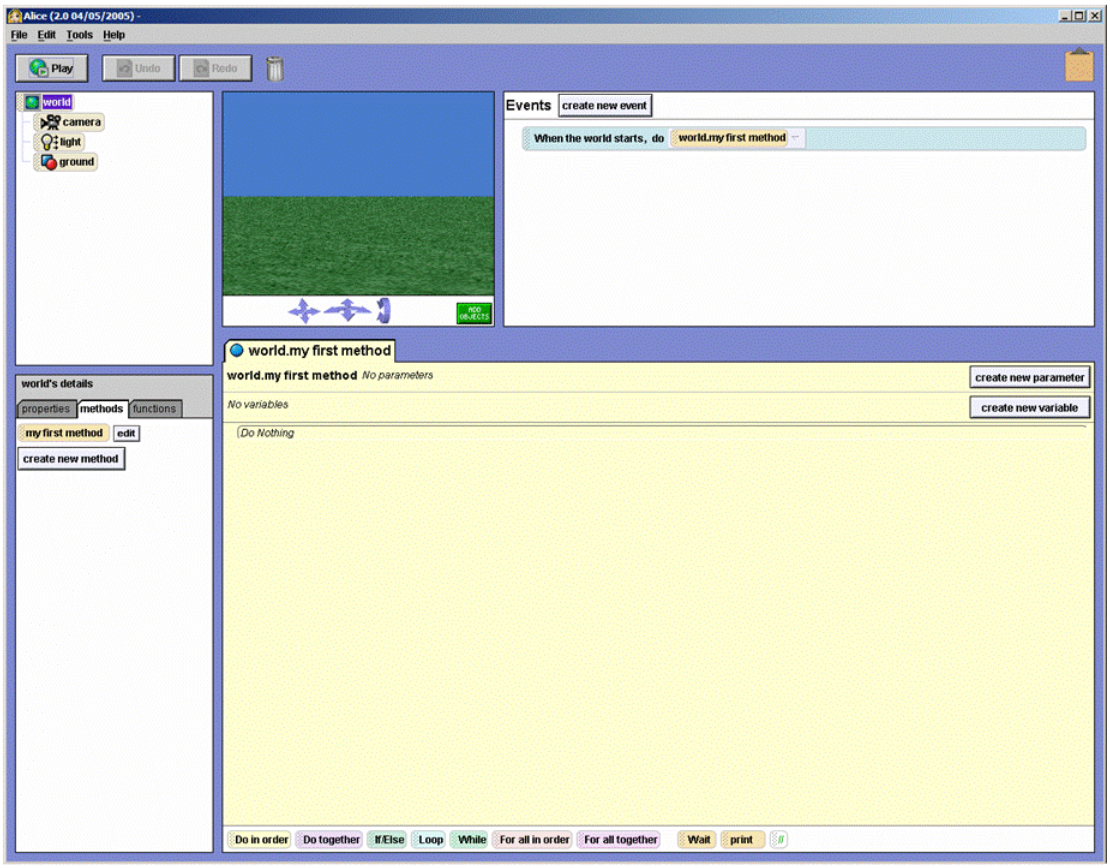
2.2 نظام آليس

نقدم في هذا النص إقترابا تدريجيا للبرمجة بواسطة وسيلة تربوية تسمى آليس صممها باحثون من جامعة كارنيغي ماللون خصيصا لغرض تعليم لغة جافا.

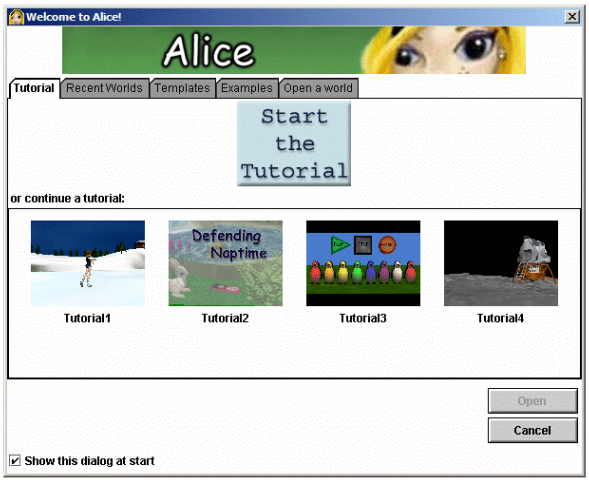
نشجع التلميذ / القارئ على أن يتحصل على نسخة من نظام آليس، التي يمكن التحصل عليها مجانا من العنكبوتية العالمية في العنوان التالي:

http://www.alice.org/

بعد أن تخزنوا نظام آليس على حاسوبكم، المطلوب منكم أن تنفذوه فتخرج النافذة التالية على شاشتكم. تمثل هذه الشاشة نقطة الإنطلاق لعدد من النشاطات التعليمية التي سنقوم بها خلال هذا الفصل.



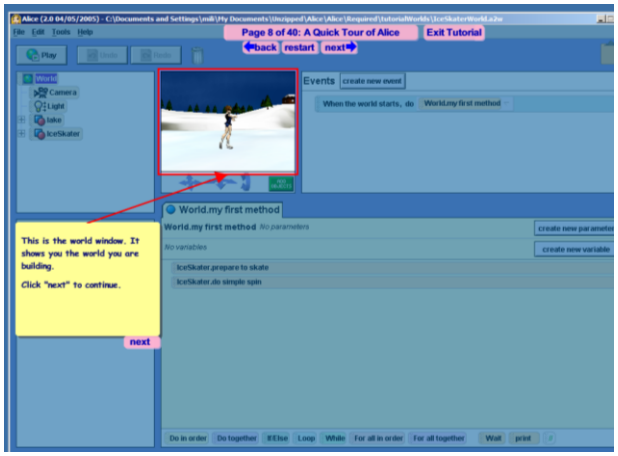
أول ما نقوم به هو التعرف على إمكانيات هذا النظام من خلال الأربعة دروس بدائية التي يعرضها النظام، حسب النافذة التالية:



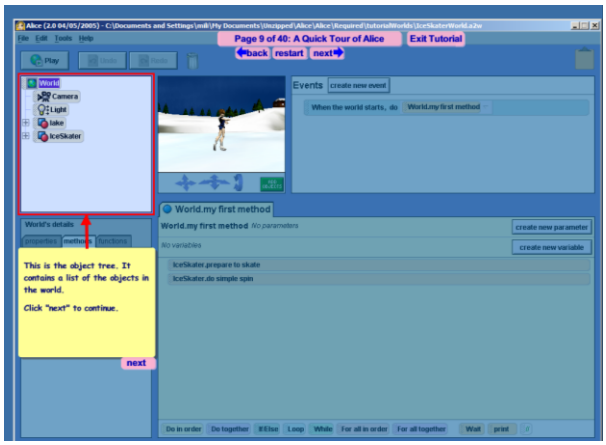
إن لم تظهر هذه النافذة تلقائيا عندما ينطلق النظام، فيمكن لكم أن تحضروحا بالضغط على رز المساعدة في السطر الأعلى للنافذة السابقة، ثم إختيار البديل الأوسط. نستعرض هذه الدروس الأربعة على التوالي، فيما يلي.

1.1.2 الدرس الأول: التحكم في الكائنات

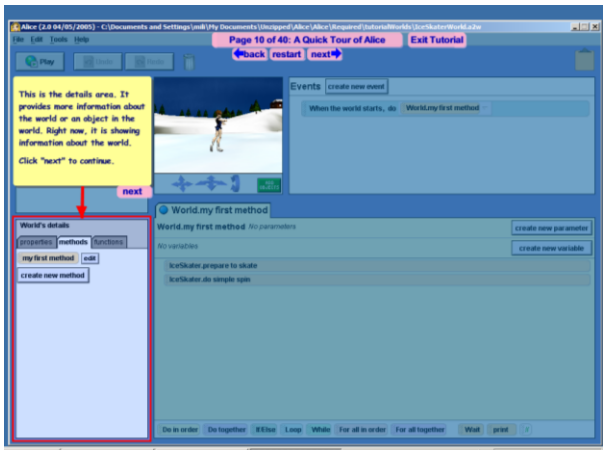
يمكننا نظام آليس من بناء عالم خيالي تسكنه كائنات متنوعة؛ تتمكن هذه الكائنات من التنقل حول هذا العالم الخيالي، و من الدوران فيه و من تغيير ألوانها و من ردود الفعل لأحداث خارجية، إلخ. تتكون واجهة أليس من خمسة مناطق، و هي:



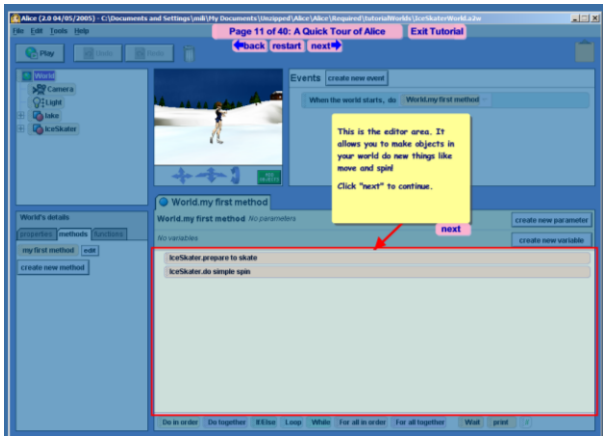
منطقة العالم الخيالي: تبين هذه المنطقة العالم الذي نكونه تدريجيا في نطاق البرمجة؛ كما تمثل الشاشة التي نقدم فيها تنفيذ البرامج التجريبية.



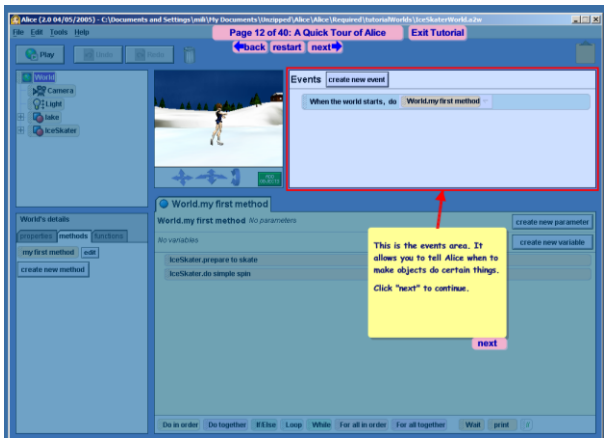
منطقة شجرة الكائنات: تمثل هذه المنطقة مجموعة الكائنات الموضوعة تحت تصرف المبرمج. في هذه الصورة مثلا، تتمثل الكائنات في المصورة (حيث نتحكم في تحركاتها) و في الضوء (حيث نتحكم في إتجاه الضوء و درجة إشعاعه) و في البحيرة (حيث قد نتحكم في موقعها) و في المتزحلقة على الثلج (حيث قد نتحكم في موقعها و حركاتها). علما أننا نتحدث عن شجرة كائنات عوضا عن قائمة كائنات لأن كل كائنة قد تتفرع إلى أعضاء يمكن التحكم فيها بمفردها: مثلا، إذا وجهنا إهتمامنا لكائنة المتزحلقة على الثلج، تبرز كائنات جزئية لها مثل يديها أو رأسها أو ساقيها، إلخ. فنتكمن من التحكم فيها مفردا، فنرفع يديها أو نحرك ساقيها أو ندور رأسها، إلخ.



منطقة التفاصيل: تسجل هذه المنطقة تفاصيل متعلقة بالعالم خلال تطوره، فتبين معلومات تخص أوصاف الكائنات و أساليب الكائنات و وظائف الكائنات. عندما نختار كائنة مثلا، نجد في هذه المنطقة كل الأساليب التي يمكن ذكرها في هذه الكائنة، وكل الأوصاف التي تتسم بها هذه الكائنة، و كل الوظائف التي تنطبق على هذه الكائنة.



منطقة التحرير: نستعمل هذه المنطقة لتحرير البرنامج، حيث نألف تعليمات نوجه من خلالها الكائنات المختارة. فنتمكن من إدخال معلمات و متغيرات مختلفة تمثل أوجه هامة من العالم الخيالي، كما نتمكن من ذكر أساليب تنطبق على الكائنات، و من تنظيم هذه الأساليب في نطاق برامج. فيمكن ترتيب الأساليب بصفة تسلسلية (حيث نطبق الأساليب الواحد بعد الآخر)، أو بصفة متزامنة (حيث تنطبق الأساليب في نفس الوقت)، أو بصفة خيارية (حيث نطبق أسلوبا أو آخر)، أو بصفة تكرارية (حيث نطبق أسلوبا عددا معينا من الأضعاف)، أو بصفة إستقرائية (حيث نطبق أسلوبا حتى يتوفر شرطا ما).



منطقة الأحداث: تمكننا هذه المنطقة من تعيين أحداث تطرأ في العالم الخيالي، مع ظبط ردود الفعل الذي نبرمجه بالنسبة لكل طرف من هذا العالم. نجد في هذه المنطقة حدثا ضمنيا يوجد في كل الحالات، و هو بداية العالم.

إذا تأملنا في منطقة التحرير نجد أسلوبين يتطبقان على المتزحلقة، فينصان على التوالي أنها تتهيأ للتزحلق ثم تدور دورة واحدة. فعلا، لو ضغطنا على علامة التنفيذ،

لقامت المتزحلقة بتنفيذ تعليمات منطقة التحرير.

أما إذا أردنا أن نغير ما تقوم به المتزحلقة، فذلك يتطلب أن نتطلع على العمليات التي تتمكن المتزحلقة من القيام بها. فلهذا الغرض، نضغط على رمز المتزحلقة في منطقة الكائنات، مما يتسبب في إبراز عدد من الأساليب التي تنطبق عليها في منطقة التفاصيل، بما فيها التزحلق الخطي و الدوران و الدوران البسيط و التزحلق الخلفي و القفز و التهيأ للتزحلق. بالإضافة إلى هذه الأساليب الكاملة، نتمكن (حسب ما جاء في أسفل نافذة التفاصيل) في تعيين أساليب جديدة بواسطة عمليات بدائية.

حسب ما يوصي به الدرس في صفحة 24، نرفع عملية التزحلق من منطقة التفاصيل إلى منطقة التحرير، فنختار قيمة 2 بالنسبة للمعلمة التي تضبط عدد الخطوات. عندئذ نضغط على علامة التنفيذ فتتهيأ المتزحلقة ثم تدور ثم تتزحلق خطوتين في إتجاه خطي، حسب ما جاء به البرنامج.

أما إذا رفعنا الأسلوب الذي ينص على الدوران ووضعناه بعد أسلوب التزحلق، ثم ضغطنا على علامة التنفيذ (حسب ما يوصي به الدرس في صفحات 27، 28)، ثم ضغطنا على علامة التنفيذ، لقامت المتزحلقة بعملياتها حسب الترتيب الجديد، أي أنها تتهيء ثم تتزحلق خطوتين خطيا ثم تدور.

تنص الصفحات الموالية من الدرس الأول، إلى غاية الصفحة رقم 36، عن كيفية برمجة المتزحلقة بحيث أنها على إثر الدوران المذكور أعلاه تتزحلق خطوة واحدة نحو الوراء ثم تقفز. أما عن الصفحات الأخيرة، فتطلب منكم أن تغيروا ترتيب العمليات التي تقوم بها المتزحلقة، ثم تنفيذ البرنامج الجديد.

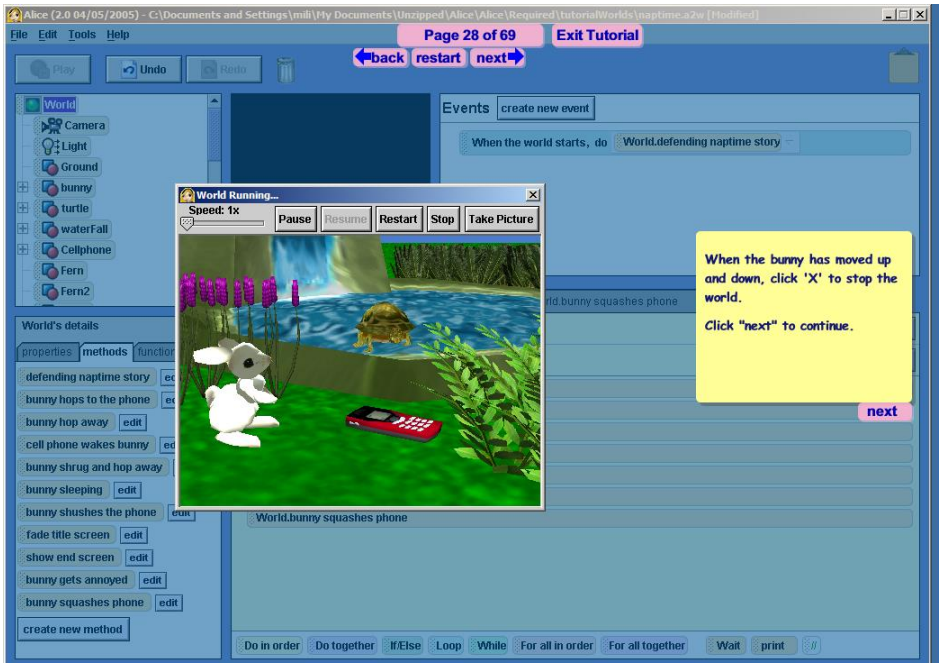
2.1.2 الدرس الثاني: تعيين اساليب و عوالم

نستعمل كلمة عوالم كجمع كلمة عالم، حيث أن نظام آليس يمكننا من ضبط العالم الذي نبرمج في نطاقه ضمن مجموعة من البديلات. في إفتتاح الدرس الثاني نجد قصة مبرمجة تتمثل في أرنب نائم يبقضه جرس هاتف متجول، فينزعج و يعتزم أن يتخذ تدابير ضد هذا الهاتف. لهذا الغرض، يمَكّننا نظام آليس من تعيين أساليب جديدة نلقي عليها

أسماء من إختيارنا و نعين تفاصيلها كما شئنا. ففي الصفحة الثامنة من هذا الدرس نعتزم أن نعين أسلوبا جديدا يختص في تكسير الهاتف المتجول، فنوجه إهتمامنا لمنطقة التفاصيل، حيث نضغط على الرزّ الموجود أسفل هذه المنطقة. عندئذ يطلب منّا نظام آليس إلقاء إسم على هذا الأسلوب الجديد؛ و عندما نلقي عليه الإسم الذي يقترحه الدرس (الأرنب تكسر الهاتف)، فيضعة نظام آليس ضمن الأساليب المعروفة في منطقة التفاصيل. نحن طبعا لم نحدد معنى هذا الأسلوب؛ لكن كما نرى، هناك رزّ بجانب هذا الأسلوب يسمح لنا أن نؤلف محتواه.

يفتح نظام آليس منطقة التحرير ليمكننا من إدخال تفاصيل حول كيفية تكسير الهاتف؛ فلهذا الغرض، ننظر لمنطقة شجرة الكائنات لنتعرف عن مختلف الكائنات و ما تتمكن من القيام به. عندما نضغط على الرزّ الذي يمثل الأرنب، تبين منطقة التفاصيل كل العمليات التي يتمكن الأرنب من القيام بها (صفحة 17)؛ فنجد ضمن هذه العمليات عملية التحول، فنلتقطها من منطقة التفاصيل و نضعها في منطقة التأليف مع ضبط إتجاه التحول (إلى الأعلى) و مسافته (مترا واحدا). نكرر نفس العملية مرة ثانية، مع ضبط إتجاه التحول إلى الأسفل، بنفس المسافة، فيكون الأرنب قفز ثم سقط على الهاتف (صفحة 19).

بعد ان تتم برمجة تكسير الهاتف (بالقفز عليه) نعود لتحرير القصة الأولى (الدفاع عن وقت النوم)، فنجد الأساليب التي تعين هذه القصة (نوم الأرنب و ضرب الجرس و إحتجاج الأرنب و تحول الأرنب إلى الجرس). لكي نظيف لهذا البرنامج عملية تكسير الهاتف، يجب أن نتطلع على الأساليب التي تنطبق على العالم، فنجد عددا من الأساليب، منها أسلوب تكسير الهاتف، فنحول هذا الأسلوب إلى منطقة التحرير، و نظعه في آخر البرنامج (صفحة 27). فنحن الآن مستعدون لتنفيذ هذا البرنامج، حيث نضغط على رز التنفيذ.



نلاحظ من خلال هذا التنفيذ أن الأرنب قفز ببطئ كبير، بحيث قد لا يتسبب قفزه في تكسير الهاتف، فنعتزم أن نسرع في قفزه؛ لهذا الغرض نعود إلى منطقة التأليف و نختار أسلوب تكسير الهاتف. في نطاق هذا الأسلوب، ننظر على التوالي لعمليتي القفز صاعدا ثم نازلا، فنحول مدتهما من ثانية إلى ربع ثانية. فعندما ننفذ العالم نلاحظ أن الأرنب يقفز بسرعة أكبر، مع أن الهاتف لم يتأثر من هذه القفزة العنيفة (صفحة 39)...

فنعتزم أن نظيف عملية تمثل تكسير الهاتف. لهذا الغرض نعود لمنطقة شجة الكائنات و نهتم بكائنة الهاتف، فتبرز في منطقة التفاصيل كل العمليات التي تتمكن كائنة الهاتف من القيام بها. فنجد ضمن هذه الأساليب عملية الإنكسار،

فنحولها إلى نافذة التأليف و نضعها بعد قفز الأرنب. فعندما ننفذ هذا العالم الجديد نرى أن الهاتف يرد الفعل الآن على قفزة الأرنب، فيتفرطح و ينغرس في الأرض (صفحة 43).

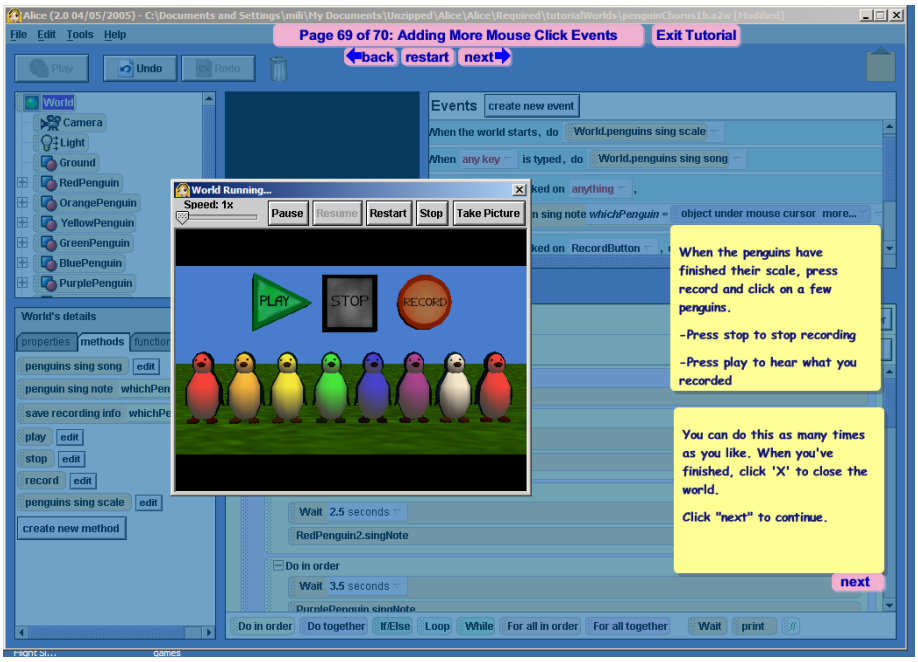
في بقية هذا الدرس نضيف بعض التفاصيل إلى قصتنا، منها أن الأرنب يهز أكتافه بعد القفز على الهاتف، و أنه يقوم بتحريك يديه عندما يأمر الهاتف بالصمت (في أول القصة)، و أن القصة تختم بصورة خاصة صُورت مسبقا. تتطلب عكلية تحريك يدي الأرنب أن نهتم بالأرنب، ثم بجسمه ثم بصدره ثم بيديه، و أن نحرك يده اليمنى إلى الأمام ربع دائرة (قبل أن يتكلم) ثم إلى الوراء ربع دائرة (بعد أن يتكلم).

نحرض القارئ المهتم أن يتابع تفاصيل هذا الدرس و أن يتأكد من أنه يفهم تفاصيله بالتدقيق، فنتحول إلى الدرس الموالي.

3.1.2 الدرس الثالث: التصرف في الأحداث

في هذا الدرس نتعرف على مفهوم الحدث و على التصرف في الأحداث. إذا نفذنا البرنامج الموجود حاليا في نطاق هذا الدرس لشاهدنا البطاريق يعزفون الموسيقى. فإذا تأملنا في منطقة الأحداث لوجدنا أن هناك تعليمة تنص على أن البطاريق يعزفون عندما ينطلق العالم (صفحات 1 إلى 9). فنعتزم على نغيير هذا البرنامج، بحيث أن إنطلاق العالم يؤدي إلى تنفيذ أسلوب آخر دون اسلوب عزف البطاريق. لهذا الغرض، نتأمل في منطقة شجرة الكائنات لنتعرف على الأساليب التي تنطبق على العالم الحالي، فتبرز في منطقة التفاصيل الأساليب المعنية. نختار منها الأسلوب الذي يتمثل في عزف السلم الموسيقي، و نضعه في منطقة الأحداث ليعوض أسلوب العزف الحالي المرتبط بحدث بداية العالم؛ عندما ننفذ البرنامج الجديد، نلاحظ أن البطاريق يعزفون السلم الموسيقي عوضا عن اللحن السابق (صفحات 10 إلى 16).

في التجربة الثانية، نقمو بإدخال حدث جديد يتمثل في الضغط على أي مفتاح من لوحة المفاتيح، فلهذا الغرض نهتم بمنطقة الأحداث و نختار فيها رزّ الأحداث الجديدة، فنعين الحدث الجديد بأنه يتمثل في الضغط على أي مفتاح من لوحة المفاتيح، و نربط بهذا الحدث أسلوب غناء البطاريق. أما تنفيذ هذا البرنامج، فيتمثل في أن البطاريق يعزفون السلم الموسيقي عند إنطلاق التنفيذ، ثم يعزفون لحنهم الأصلي كلما ضغطنا على أي مفتاح من لوحة المفاتيح (صفحات 17 إلى 24). في التجربة الثالثة، نكون حدثا آخر يتمثل في توجيه الفأرة نحو أي كائنة في الصورة و الضغط عليها، هذا و نربط هذا الحدث بأسلوب يتمثل في أن البطريق الأزرق يعزف لحنا (صفحات 25 إلى 33). نلاحض في هذا الخصوص أن ترتيب الأحداث في منطقة الأحداث لا يؤثر على تصرف البرنامج، فبالرغم من أن حدث ضغط المفاتيح يسبق حدث ضغط الفأرة، فيمكن وقوع هدث الأحداث في أي ترتيب شئنا عند التنفيذ، و في أي عدد من الأضعاف.



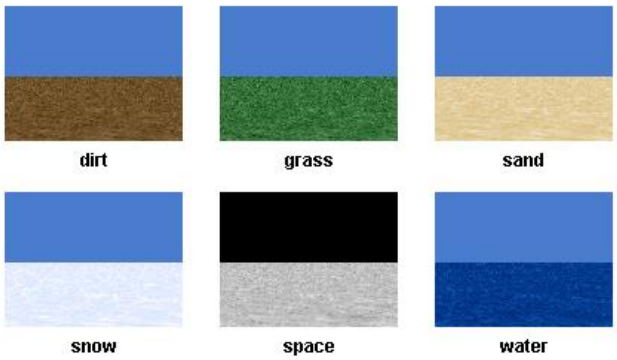
في بقية هذا الدرس، نعتزم أن نجعل كل بطريق يعزف لحنه عندما يقع الضغط عليه، عوضا أن يعزف البطريق الأصفر مهما كانت الكائنة التي نعينها بالفأرة. لهذا الغرض، نغير الحدث الذي أدخلناه سابقا فنربطه بأسلوب ينص على أن يعزف البطريق الذي نعينه بالفأرة؛ أما إذا وجهنا الفأرة لأي كائنة أخرى، فلا نسجل ردود فعل منها. بفضل هذا الحدث الجديد، أصبح هذا العالم (البرنامج) آلة عزف موسيقي، حيث يمكننا العزف عليه بمجرد أن نضغط على مختلف البطاريق. هذا، و نعين ثلاثة أحداث أخرى، تتمثل على التوالي في الضغط على رزّ التسجيل و رزّ العزف و رزّ الإنقطاع عن التسجيل، و نربط كل حدث بالعملية المناسبة. بحيث إذا نفذنا هذا البرنامج ثم ضغطنا على رزّ التسجيل فيسجل البرنامج كل ما نعزفه حتى نضغط على رزّ الإنقطاع؛ فكلما ضغطنا على رزّ العزف إلا و عزف البرنامج كل ما تم تسجيله.

4.1.2 الدرس الرابع: صنع المشاهد

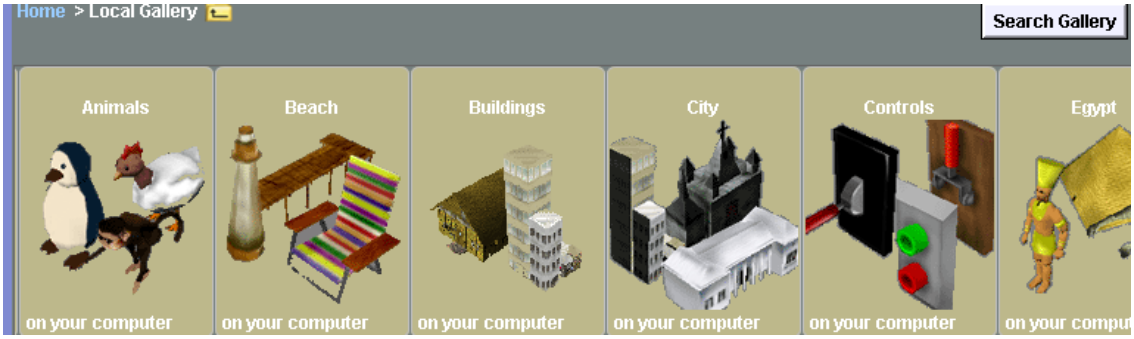
في هذا الدرس نهتم بعملية تكوين المشاهد بواسطة عناصر بدائية. فأول ما نتمرن عليه هو تحويل الكائنات في مشهد ما، حيث أن الصفحات الأولى من الدرس تطلب منكم أن تحولوا الكوكب القمري من موقعه الأصلي إلى وسط المشهد. ثم يمرننا الدرس على تحويل المصورة في المشهد، فيبين كيف نتقرب و نبتعد في المشهد (بواسطة المقود الأوسط تحت الشاشة)، و كيف نصعد و ننزل (بواسطة المقود اليساري تحت الشاشة)، و كيف نغير توجيه المصورة بالنسبة للمحور الأفقي (بواسطة المقود اليميني تحت الشاشة).

يسمح لنا نظام آليس أن نتراجع في أي قرار أخذناه، و ذلك بالضغط على رزّ التراجع (الموجود في يسار الصورة أسفله)، كما يسمح لنا بالتراجع في التراجع نفسه بالضغط على رزّ تكرار العملية (الموجود في يمين الصورة أسفله).

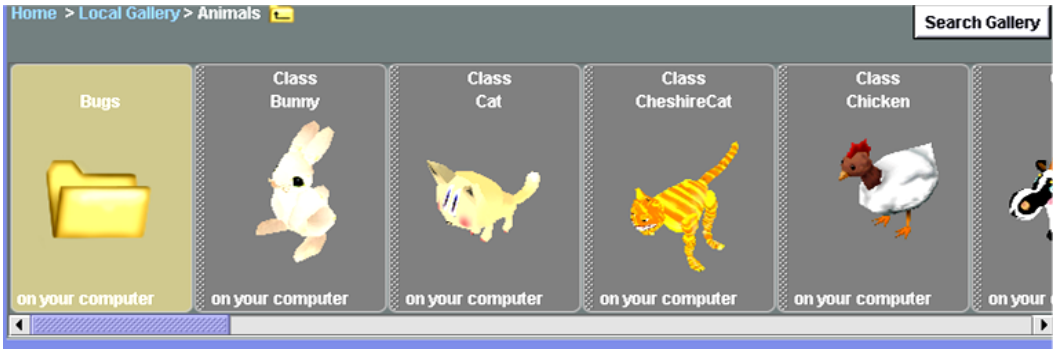
هذا، و يسمح لنا نظام آليس تكوين عوالم جديدة إنطلاقا من عناصر أولية منها البيئة المحيطة، حيث يمدنا النظام بستة إختيارات في إصداره التالي، منها مشهد صحرائي (رمال و سماء) و مشهد أخظر (أعشاب و سماء) و مشهد برّي (أرض و سماء) و مشهد بحري (البحر و السماء) و مشهد فضائي (القمر و سماء القمر، مثلا) و مشهد شتائي (الثلج و السماء).



عندما نختار محيطا، يسمح لنا نظام آليس أن نعمر هذا المحيط بكائنات من إختيارنا، فنختار هذه الكائنات ضمن قائمات مخزونة مسبقا، و مبوبة، كما يلي:



حيث نجد أقساما لكل من: الحيونات و كائنات الشاطئ و العمارات و المدن و أدوات التحكم الكهربائية، إلى غير ذلك. أما إذا عينّا قسما من هذه الأقسام و ضغطنا عليه، فنجد مكزنات هذا القسم، التي قد تتمثل في أفراد أو في أقسام هي بنفسها. على باب المثال، إذا ضغطنا على قسم الحيونات في القائمة السابقة، نجد القائمة التالية، التي تتكون من قسم خاص بالحشرات، ثم الأرنب، و نوعين من القطط، و الدجاج، إلى غير ذلك.



يمكننا نظام آليس من رفع عناصر من هذه القائمات لنضعها في المحيط الذي إخترناه سابقا، كما يمكننا من نسخ العناصر و تعيين موقعها و إتجاهها، و حجمها، و إرتفاعها، إلى غير ذلك. يمكن لنا الملاحة عبر قاعدة الكائنات بحيث ننزل في الأقسام الجزئية بالضغط على رمزها و نصعد من أقسام جزئية بالضغط على رمز الملف الموجود في الركن الأعلى اليساري أو بالعودة إلى رواق البحث الأصلي، عن طريق الرمز الموجود في الركن الأعلى اليميني.
عندما يتم تعيين العالم المعني، نضغط عاى الرزّ التالي

الموجود في أقصى اليمين، فتعود واجهة النظام لما كانت عليه حسب ما جاء في الجزء رقم 1.1.2، ويستعد النظام للبرمجة.

هكذا تتم دراسة الدروس الأربعة الموجودة ضمن نظام آليس. نحرض القارئ أن يتأكد من أنه يتقن الأوجه التي طرقناها في هذا الجزء، و أن يقوم بالتمارين المعنية في نهاية هذا الفصل.

في هذا الجزء تعرضنا لعدد من المفاهيم الهامة المتعلقة بالبرمجة في لغة جافا، نذكر منها بالخصوص:

- مفهوم الكائنات و الأقسام، و كيفية التصريح بالكائنات و إستعمالها.
- مفهوم علاقة الإنتماء، التي تربط بين كائنة و قسم (مثلا: الأرنب أ تنتمي إلى قسم الأرانب).
- مفهوم علاقة التفاوت، التي تربط بين قسمين (مثلا: قسم الأرانب قسم جزئي من قسم الحيونات).
- مفهوم علاقة التجزأ، التي تربط بين قسمين (مثلا: الرأس جزء من الأرنب).
- مفهوم الأسلوب، حيث يقع التصريح به في نطاق قسم و ينطبق على كل كائنة من هذا القسم (مثلا:

bunny.move).

- مفهوم المعلمات، يقع التصريح به في نطاق قسم و ينطبق على كل كائنة من هذا القسم (مثلا:

bunny. move. forward 0.5 meters. duration = 0.5 secs. Style= begin gently…).

حيث أن إتجاه الحركة و مسافتها و مدتها و نوعيتها، كلها تصف تدريجيا تطبيق الأسلوب المذكور.

تعرضنا لكل هذه المفاهيم بصفة طبيعية جدا، كدنا لا نشعر بهم تماما. لكن هذه المفاهيم في غاية من الأهمية، سننكب عليها من جديد، بأكثر عناية و إهتمام، في الفصول القادمة.

3.2 تمارين

1. [س] في الجزء الأول من هذا الفصل نظرنا في الدورة الحياتية للبرنامج، فإستعرضنا عدد من الأطوار؛ و في الجزء الثاني، رأينا كيف نألف و نغير البرامج. ماذا تمثل عملية

بالنسبة للدورة الحياتية المذكورة؟

2 [م] المطلوب منكم أن تكوّنوا عالما فيه أرنب و آليس يمشيان في إتجاه بعضهما بعضا خطوة واحدة ثم يتوجهان للمصورة و يقولان "مرحبا".

3 [م] المطلوب منكم أن تضعوا أربعة أرانب في العالم، يقفزون مترين في نصف دقيقة كلما وجهتم لهم إشارة الفأرة و ضغطتم.

4 [ص] المطلوب منكم أن تضعوا طائرة في العالم فتجري هذه الطائر مدة 5 أمتار ثم تقلع لإرتفاع 10 أمتار ثم تدور و تنزل على الأرض.

4.2 مصادر و مراجع

للمزيد من المعلومات حول معمارية الحواسيب، يمكن النظر لمقالة الأستاذ نبيل عباس [عباس، 2003]. للمزيد من المعلومات عن نظاو آليس، يمكن النظر في موقع العنكبوتية المخصص لهذا النظام، كما يمكن أيضا النظر في الكتاب [أدامس، 2008].

الفصل الثالث

هندسة البرمجيات

في الفصل السابق، درسنا مسار البرنامج من تأليفه إلى تنفيذه عبر أطوار مختلفة كالصرف، و تأليف الروابط، و الرفع، و التنفيذ، إلى غير ذلك. إن كل هذه الأطوار آلية، فيمكن التأمل فيها و إختبارها لنتأكد من دقتها و سدادها. أما طور تأليف البرنامج، فهو أمر عسير جدا في العادة، يتطلب إنظباطا كبيرا و تمسكا متينا بطرق عمل دقيقة و سديدة. فننكب في هذا الكتاب على شرح الأساليب اللائقة لهذا الطور.

1.3 إنزال مصرّف جافا

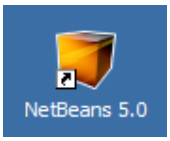
كما ناقشنا في الفصل الأول، فإن قطاع المعلوماتية شهد عددا كبيرا من لغات البرمجة، نجح البعض منها و إزدهر، بينما زال أغلبها. نهتم في هذا الفصل بلغة جافا، التي نشأت في التسعينات من القرن السابق في مخابر شركة "سُن مايكروسيستمس". شهدت هذه اللغة رواجا كبيرا، قد يرجع الفضل فيه لعدة صفات، نذكر منها بالخصوص: أن اللغة موجهة للكائنات؛ أن برامج جافا قادرة على التنفيذ في نطاق العنكبوتية العالمية؛ أن هذه اللغة تتمتع بنظام يؤيد صنع البرامج؛ أن مصرفات هذه اللغة موضوعة مجانا على ذمة المستعملين. على كل حال، أصبحت هذه اللغة رائجة الإستعمال كلغة يستعملها المدرسون لتعليم البرمجة و يستعملها المبرمجون لإنتاج أنظمة تُنفذ على العنكبوتية العالمية. هناك عدد من نسخ هذه اللغة؛ في هذا الكتاب، نستعمل النسخة المعرفة كما يلي:

J2SE and NetBeans IDE Cobundle (NB 5.0 / J2SE 1.4.2_13) .

و هي النسخة العادية لهذه اللغة، و توجد في مقام شركة "سُن مايكروسيستمس"

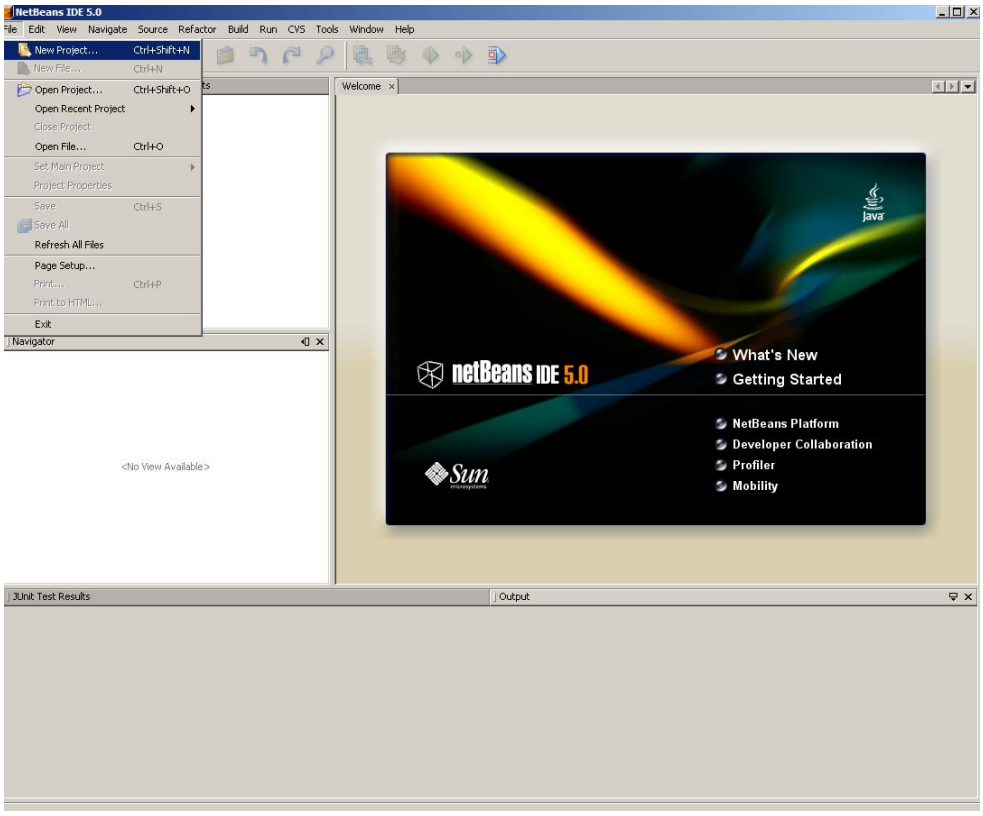
http://www.sun.com.

هناك عدة نسخ من هذا المصرف، تتغير حسب نظام التسيير التابع لحاسوبكم. نستعمل في هذا الكتاب نسخة المصرف التابعة لنظام "وندوز". عندما يتم إنزال هذا النظام، نجد الرمز التالي على واجهة الحاسوب.



2.3 نداء مصرّف جافا

إذا نادينا هذا النظام، تبرز لنا واجهة نظام يمكننا من إنتاج و تطوير برامج في لغة جافا، حسب ما جاء في الرسم أسفله. إذا إخترنا البديل الأول (على أقصى اليسار) في مائدة الإختيارات، حسب ما يدل الرسم أسفله، ثم إخترنا ضمنه بديل "مشروع جديد"، يطلب منا النظام إلقاء إسم على هذا المشروع؛ إثر ذلك تنفتح النافذة في يمين الواجهة فيبرز فيها هيكل برنامج في جافا، يجب علينا أن نكمّل تفاصيله حسب ما ننوي القيام به. في نطاق هذا الفصل نهتم بإنجاز برنامج بسيط يحول قيمة مالية من الدينارات العراقية إلى الدينارات التونسية.



فنتأمل في هذا البرنامج (الموجود أسفله) مع إضافة أعداد الأسطر له، لنتمكن من التحدث عنه. نهتم خصيصا بالأسطر 28 إلى 48، و هي الأسطر التي أضفناها لهيكل جافا الأصلي (كما أنها تمثل جوهر البرنامج المعني). فأما عن الأسطر 28 إلى 31، فهي مجرد تعليق نكتبه ضد البرنامج، توثيقا لنا (إذا قرأنا برنامجنا في تاريخ لاحق) و غيرنا مما قد يضطر لقراءة هذا البرنامج (لتحسينه أو تغييره أو تصليحه، إلخ). تُمكننا لغة جافا من إدخال تعليقات أينما شئنا خلال نص البرنامج. نعرف على هذه التعليقات بوسيلتين إثنين: إما أن نضع علامة // في سطر ما، و نكتب تعليقنا على يمين هذه العلامة. على باب المثال، نكتب:

تصريحات للمتغيرات		//

قيمة الدينار العراقي حسب الدينار التونسي في الشراء.	//	`buyrate  = 0.00115;`
قيمة الدينار العراقي حسب الدينار التونسي في البيع.	//	`sellrate = 0.000985;`

الوسيلة الثانية لإدخال التعليقات تتمثل في إدخالها بين رمزين، رمزا للبداية و رمزا للنهاية، كما يبين المثال التالي:

	نص التعليق	/*
	قد يمتد على	
	عدة أسطر	*/

فالفرق بين هذين النوعين من التعليقات هو أن النوع الأول لا ينطبق إلا على سطر أو بقية سطر، فيبدأ التعليق برمز // و ينتهي بآخر السطر. بينما النوع الثاني يبدأ برمز البداية و ينتهي برمز النهاية (حسب الرسم أعلاه)، و قد يمتد على أي عدد من الأسطر.

ماذا يفعل المصرف عندما يعثر على تعليق؟ فهو يجهله تماما، حتى أنه لا يبالي بأي لغة كُتب هذا التعليق. نستعمل إمكانية التعليق لنضع ضمن البرنامج ملاحضات تبين تفكير المبرمج و تفسر قراراته.

```
1       /*
2        * Main.java
3        *
4        * Created on October 5, 2007, 10:30 PM
5        *
6        * To change this template, choose Tools | Template Manager
7        * and open the template in the editor.
8        */
9
10      package exchangerate;
11
12      /**
13       *
14       * @author mili
15       */
16      public class Main {
17
18          /** Creates a new instance of Main */
19          public Main() {
20          }
21
22          /**
23           * @param args the command line arguments
24           */
25          public static void main(String[] args) {
26              // TODO code application logic here
27
28              /*
                    يقوم هذا البرنامج بترجمة قيمة معينة من العملة العراقية إلى عملة تونسية
                                                حسب أسعار البيع و الشراء بالنسبة ليوم معين*.
31               */
32
33              double buyrate, sellrate;
34              double tndamount, irqamount;
35
36              buyrate  = 0.00115;
37              sellrate = 0.000985;
38              irqamount = 14079.00;
39
40              tndamount = irqamount*buyrate;
41              System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
42              System.out.println (tndamount+" Tunisian Dinars.");
43
44              tndamount = irqamount*sellrate;
45              System.out.print ("when you sell "+irqamount+" iraqi dinars, you get: ");
46              System.out.println (tndamount+" Tunisian Dinars.");
```

```
47
48              System.out.println ("today's date:  Ramadan 22, 1428.");
49
50          }
51
52      }
```

يمثل السطر رقم 33 تصريحا لمتغيرتين إثنين: متغيرة تمثل نسبة شراء الدينار العراقي بحساب الدينار التونسي، و متغيرة تمثل نسبة بيع الدينار العراقي بحساب الدينار التونسي. و في السطر الموالي (34)، نقدم تصريحين لمتغيرتين نخزن فيهما قيمة عمليات التحويل بحساب الدينا التونسي و بحساب الدينار العراقي. في الأسطر رقم 36، 37 و 38 نقوم بإسناد قيمات عددية لكل من: نسبة شراء الدينار العراقي بحساب الدينار التونسي؛ نسبة بيع الدينار العراقي بحساب الدينار التونسي؛ و القيمة بحساب الدينار العراقي التي نرغب في تحويلها.

في السطر رقم 40 نحول القيمة المعينة بالدينار العراقي إلى الدينار التونسي عند الشراء: فنحسب هذه القيمة بضرب القيمة المعينة بالدينار العراقي بنسبة التحويل التابعة للشراء. فنعلن على هذه النتيجة في السطرين 41 و 42 حيث نطبع النص التالي: "لشراء قيمة ... دينار عراقي، يجب إحضار ... دينار تونسي"، حيث نعوض سلسلة النقط الأولى بقيمة الدنانير العراقية و سلسلة النقط الثانية بالقيمة المناسبة في الدنانير التونسية.

أما في السطر 44 فنحول القيمة المعينة بالدينار العراقي إلى الدينار التونسي عند البيع: فنحسب هذه القيمة بضرب القيمة المعينة بالدينار العراقي بنسبة التحويل التابعة للبيع. فنعلن على هذه النتيجة في السطرين 45 و 46 حيث نطبع النص التالي: "عندما نبيع قيمة ... دينار عراقي، نتحصل على... دينار تونسي"، حيث نعوض سلسلة النقط الأولى بقيمة الدنانير العراقية و سلسلة النقط الثانية بالقيمة المناسبة في الدنانير التونسية.

نختم البرنامج بالتصريح عن التاريخ الحالي، و هو الثاني و العشرين من رمضان سنة 1428 هجري.

ننفذ هذا البرنامج بالضغط عن علامة التنفيذ في مائدة الإختيارات الموجودة في أعلى النافذة، فيبرز النص التالي في منطقة النتائج الموجودة في أسفل النافذة.

```
to buy 14079.0 iraqi dinars, you need: 16.19085 Tunisian Dinars.

when you sell 14079.0 iraqi dinars, you get: 13.867815 Tunisian Dinars.

today's date:  Ramadan 22, 1428.
```

يبين هذا النص أن إشتراء قيمة 14079 دينار عراقي يتطلب قيمة 16.19085 دينار تونسي؛ و أن عندما نبيع 14079 دينار عراقي، نتحصل على 13.867815 دينار تونسي.

3.3 أطوار البرمجة

إن تأليف برنامج عملية إبداعية تتمثل في تحويل فكرة ما إلى نص في لغة برمجة معينة. بالتالي، فإن البرمجة قد تشبه تأليف قصة أ تالأيف قصيدة، إلخ. لكن البرمجة تختلف عن هذه النشاطات الأدبية في أوجه هامة جوهرية، نذكر منها: أن لغة البرمجة لغة شكلية تخضع لقواعد نحوية و قواعد دلالية دقيقة؛ بينما تسهدف القصيدة تحريك شعور القارئ و تستهدف القصة تسلية القارئ أو إعلامه، فإن البرنامج يستهدف تسيير الحاسوب للقيام بعمليات معينة؛ بينما تُوجه النصوص الأدبية للقارئ البشري، فإن البرامج للقارئ البشري و أيضا للحاسوب. بسبب هذه الفروق، يختلف تأليف البرنامج عن تأليف نصوص أدبية من حيث المنهج. فيمر تأليف البرامج بالأطوار العريضة التالية:

- ضبط و تحليل مواصفات البرنامج،
- تصميم البرنامج،
- تحرير البرنامج،
- إختبار البرنامج،

- تسيير البرنامج،
- تطوير و صيانة البرنامج.

ننظر في هذه الأطوار، على التوالي، في ما يلي.

1.3.3 ضبط و تحليل مواصفات البرنامج

تتمثل مواصفات البرنامج في مجموعة الشروط التي يجب أن يلبيها البرنامج.

إن تطبيقات البرمجة واسعة و متنوعة، فنستعمل البرامج لحل معادلات رياضية، مثلا؛ أو للقيام بعمليات حسابية؛ أو لتسيير أنظمة صناعية (كمفاعل نووي أو معاملة كيميائية، إلخ)؛ أو لتطبيق نماذج طبيعية (مثل التكهن بتطورات المناخ، أو التكهن بنشاط بركان، أو التكهن بنشاط زلزال، إلخ)؛ أو للمساعدة على التصرف في المؤسسات (التصرف في العمليات التجارية و التصرف في مكاسب المؤسسة و التصرف في الموارد البشرية و التصرف في ملفات الحرفاء، إلخ).

فأول ما نقوم به في نطاق تأليف البرنامج هو ضبط و تحليل الشروط التي يجب أن يلبيها البرنامج المطلوب. كما يستحيل علينا حل مشكلة قبل أن نفهمها، فيستحيل علينا تأليف برنامج قبل أن نضبط مواصفاته و أن نحللها و نفهمها. فإذا نظرنا في البرنامج البسيط الذي قدمناه أعلاه، نتخيل أن تنص مواصفاته كما يلي:

> المطلوب تحويل قيمة 14079 دينار عراقي إلى دينارات تونسية في الشراء ثم في البيع، علما أن نسبة الشراء هي 0.00115 و نسبة البيع هي 0.000985، بالنسبة ليوم 22 رمضان 1428.

إن هذا البرنامج محدودا، في حيث أنه لا يتمكن من تحويل إلا قيمة معينة، حسب نسبة بيع معينة و حسب نسبة شراء معينة، في تاريخ معين. قصد التوسيع في صلاحيته، نغير هذه المواصفات كما يلي:

> المطلوب تحويل قيمة بالدينار العراقي يقدمها المستخدم إلى دينارات تونسية في الشراء ثم في البيع، علما أن نسبة الشراء هي 0.00115 و نسبة البيع هي 0.000985، بالنسبة ليوم 22 رمضان 1428.

طبعا، إن البرنامج الذي كتبناه سابقا لا يلبي هذه المواصفة الجديدة؛ سنكتب برنامجا يلبيها في فصل لاحق، عندما نتعلم عمليات إصدار و توريد البيانات. إن هذه المواصفة أنفع من المواصفة السابقة، لأنها قادرة على تحويل أي قيمة في الدنانير العراقية، عوضا عن قيمة واحدة. لكنها لا تنفعنا إلا يوما واحدا، و هو اليوم المذكور، حيث تكون نسبة الشراء 0.00115 و تكون نسبة البيع 0.00985. قصد المزيد من توسيع صلاحية هذه المواصفة، نغيرها كما يلي:

> المطلوب تحويل قيمة بالدينار العراقي يقدمها عون البنك إلى دينارات تونسية في الشراء ثم في البيع، بناء على نسبة بيع و نسبة شراء و تاريخ قدمها سابقا المسؤول عن فرع البنك.

تعتبر هذه المواصفة أن هناك نوعان من مستخدمي هذا النظام: النوع الأول يتمثل في مسؤول الفرع البنكي، الذي يُدخل كل يوم تاريخ اليوم و نسبات الشراء و البيع بالنسبة للدينار العراقي بحساب الدينار التونسي، الجاري بها العمل في اليوم المذكور. و النوع الثاني هو عون البنك الذي يُدخل قيمات مختلفة من الدنانير العراقية، فيمده البرنامج بالقيمات المطابقة للبيع و الشراء حسب الدينار التونسي. يمكن تطوير هذه المواصفة بالإستغناء عن المسؤول البنكي. فيما يخص تاريخ اليوم، يمكن التحصل عليه من يومية الحاسوب. أما عن نسبات البيع و الشراء، فيمكن التحصل عليهما من موقع البنك المركزي العراقي، سواء كل يوم (عندما نفتح الفرع البنكي) أو كلما قمنا بعملية تحويل (إذا توقعنا أن هذه النسبات قد تتغير خلال اليوم).

المطلوب تحويل قيمة بالدينار العراقي يقدمها عون البنك إلى دينارات تونسية في الشراء ثم في البيع، بناء على نسبة البيع و نسبة الشراء الجاري بهما العمل في ذلك اليوم لدى البنك المركزي العراقي.

يمكن تطوير هذه المواصفة (و الترفيع في دخل البنك) بفرض ضريبة على كل عملية تحويل (كما يجرى به العمل في الواقع).

المطلوب تحويل قيمة بالدينار العراقي يقدمها عون البنك إلى دينارات تونسية في الشراء ثم في البيع، بناء على نسبة البيع و نسبة الشراء الجاري بهما العمل في ذلك اليوم لدى البنك المركزي العراقي، مع فرض ضريبة قدرها خمسة دنانير تونسية بالنسبة لكل عملية تحويل.

يمكن مواصلة تطوير هذه المواصفة بجعل الضريبة التي نفرضها على كل عملية تحويل قابلة للتغيير: عوضا عن أن تكون قيمتها مستقرة، قد نجعل مسؤول الفرع البنكي يضبط قيمتها بواسطة عملية معينة.

المطلوب تحويل قيمة بالدينار العراقي يقدمها عون البنك إلى دينارات تونسية في الشراء ثم في البيع، بناء على نسبة البيع و نسبة الشراء الجاري بهما العمل في ذلك اليوم لدى البنك المركزي العراقي، مع فرض ضريبة بالنسبة لكل عملية تحويل، يضبط قيمتها المسؤول على الفرع البنكي.

يدل هذا المثال البسيط على أهمية الدقة في ضبط المواصفات التابعة للبرامج. فلو إكتفينا بقول أننا نريد برنامجا يحول العملة العراقية إلى عملة تونسية، فتكون المواصفة ناقصة تفاصيل عديدة، إذ أنها تكون قابلة لتأويلات عديد مختلفة.

2.3.3 تصميم البرنامج

عندما نقوم بضبط مواصفات البرنامج، و نتأكد من أن كل الأطراف المعنية (مثل المصمم و المبرمج و المستخدم) تفهمها و توافق عليها، نبادر بتصميم البرنامج. تتمثل عملية التصميم في الأطوار التالية:

- إستعراض الحلول الممكنة لتلبية المواصفات، حيث يتسم كل حل بكيفية تمثيل البيانات و كيفية تسلسل العمليات التي يجب أن نقوم بها على هذه البيانات.
- تقييم الحلول المعروضة حسب مقاييس معينة، فتشمل هذه المقاييس إعتبارات مثل البساطة و النجاعة و يسر

التنفيذ و يسر الإستعمال، إلخ.
- إختيار حلا ضمن الحلول المعروضة، بناء على المقاييس المعتبرة.

عادة، تسير عملية التصميم بدون إعتبار لغة البرمجة، بالرغم من أن، تطبيقيا، قد تؤثر لغة البرمجة على سير عملية التصميم.

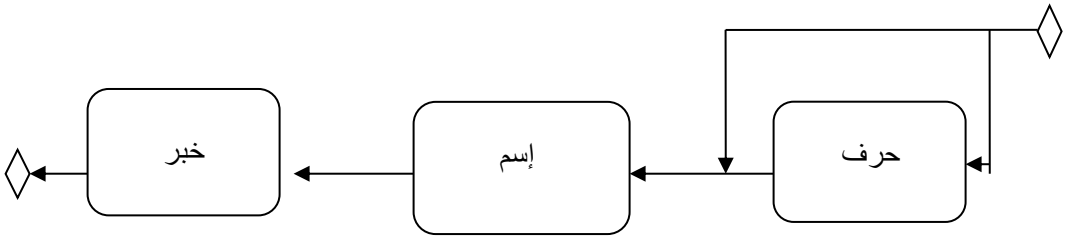
3.3.3 تحرير البرنامج

تتمثل عملية تحرير البرنامج في تحويل الحل الذي وقع إختياره في طور التصميم و تمثيله في لغة البرمجة. تتسم لغات البرمجة (كما تتسم اللغات الطبيعية) بصفتين جوهريتين، و هما:

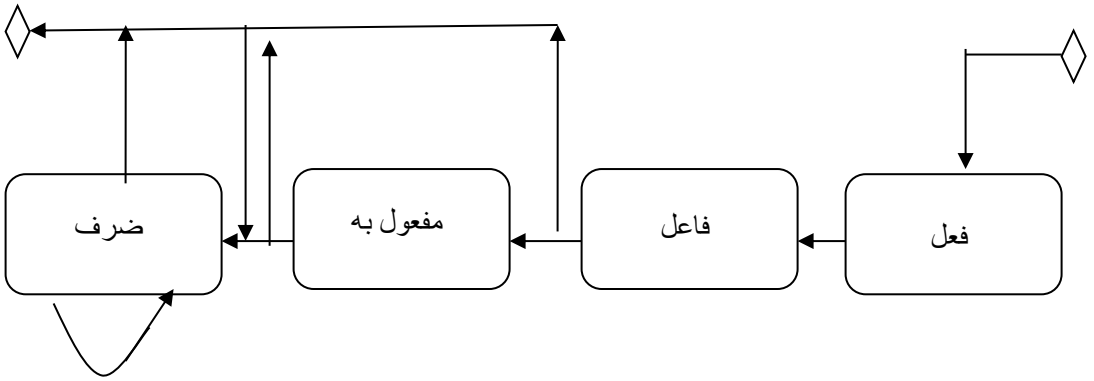
- النحو، أي مجموعة القواعد النحوية التي تعين الجمل الشرعية في اللغة.
- الدلالة، أي مجموعة القواعد المنطقية التي تسند معنى معينا لكل جملة شرعية في اللغة.

قصد ضبط و تفسير القواعد النحوية للغة ما، نستعمل ما نسميه المخططات النحوية. على باب المثال، نستعمل مخططات نحوية لنعين الشكل النحوي للجمل الإسمية و الجمل الفعلية في اللغة العربية:

جملة إسمية



جملة فعاية



نتعرف على هيكل جملة ما بالتجول على المخطط النحوي من رمز الإنطلاق إلى رمز الوصول، متابعين الأسهام. حسب المخطط النحوي التابع للجملة الإسمية، تتكون الجملة الإسمية، إما من حرف يليه إسم يليه خبر أو من إسم يليه خبر. حسب المخطط النحوي للجملة الفعلية، تتكون الجملة الفعلية إما من فعل يليه فاعل، أم من فعل يليه فاعل و يليه مفعول به، أو من فعل يليه فاعل يليه عدد من الضروف (كضرف المكان و ضرف الزمان و غيره)، أو من فعل يليه فاعل يليه مفعول به يليه عدد من الضروف. نقدم في المائدة التالية أمثلة من كل نوع من هذه المفاهيم النحوية:

الرمز النحوي	أمثلة
حرف	إن
إسم	الولد، الكتاب، البيت، الجبل، تفاحة، طعاما
خبر	نجيب، نافع، دافئ، شامخ
فعل	أكل، قرأ، أخذ
فاعل	إسم
مفعول به	إسم
ضرف	في البيت، في الصباح، فوق الجبل، بعد الزوال، إلخ

بناء على هذه الأمثلة، يمكن أن نكتب الجمل الإسمية التالية:

الولد نجيب
إن الولد نجيب
الجبل شامخ.

كما يمكن أن نكتب الجمل الفعلية التالية:

قرأ الولد الكتاب
أكل الولد تفاحة في البيت بعد الزوال
أخذ الولد طعاما.

طبعا، هذه القواعد النحوية في حد ذاتها لا تمنعنا من أن نكتب جملا مثل

الجبل نجيب،
الولد شامخ،
أخذ الولد البيت فوق الجبل،
أكل الكتاب تفاحة،
أكل الولد طعاما في الصباح بعد الزوال،
إلخ...

أي أن القواعد النحوية تضمن سداد هيكل الجمل، لكن لا تضمن أن الجمل صحيحة منطقيا، أو حتي أنها ممكنة تماما. خلال الفصول الموالية، سنستعمل مثل هذه القواعد لكي نضبط و نوضح التركيب النحوي للغة جافا. عندما نكتب برامج في لغة جافا، يجب أن نستعمل مثل هذه القواعد النحوية (سواء مثلناها بالمخططات النحوية أو بغيرها من المسائل) لنتأكد من أن البرنامج الذي نكتبه يطابق القواعد المعنية.

بالإضافة تتطلب ممارسة البرمجة عددا من الإجراءات التي من شأنها أن تحسن جودة البرامج، و أن تيسر إستعمالها و تطويرها:

- إستعمال أسماء متغيرات تعكس محتواها و دورها: مثلا، نمثل القيمة المالية في الدنانير التونسية و القيمة المالية بالدنانير العراقية بالمتغيرتين التاليتين:

```
tndamount, irqamount.
```

كما نمثل المتغيرتين التين تمثلان نسبة التحويل في البيع و الشراء بالمتغيرتان التاليتان:

```
buyrate, sellrate;
```

- توثيق البرنامج، بواسطة تعليقات تفسر تفكير المبرمج، علما أن البرنامج ليس موجها للحاسوب فقط، بل هو موجه لمبرمجين آخرين قد يحتاجوا لتحليل البرنامج قصد صيانته أو تغييره أو تصليحه أو تحليله، إلخ.

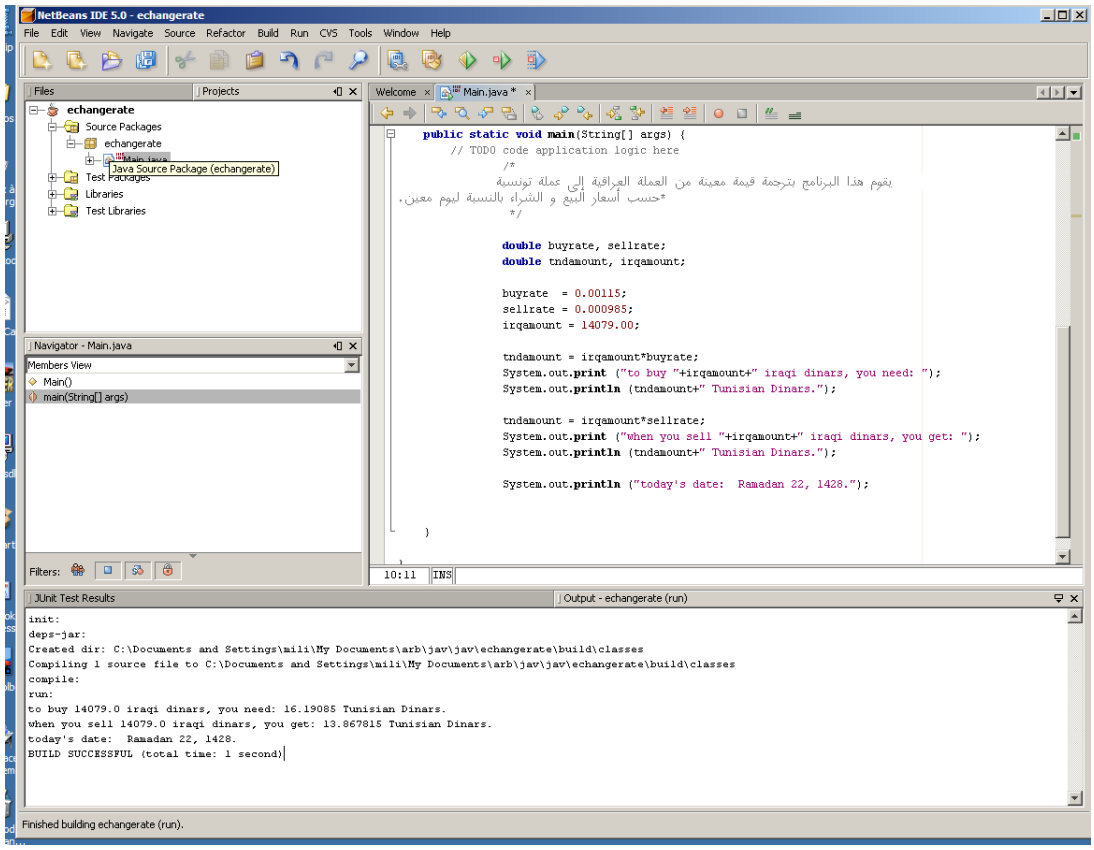
- تقديم البرنامج، الذي يتعلق بكيفية عرض نص البرنامج، فنطبق قواعد مثل: وضع تعليمة في كل سطر، إستعمال الحاشية لإبراز العلاقات بين التعليمات، وضع أسطر بيضاء بين أجزاء مختلفة من البرنامج، إلخ.

- هيكلة البرنامج، حيث نقسم البرنامج إلى أجزاء يتكلف كل جزء بوظيفة مختلفة أم بطور مختلف من أطوار البرنامج.

تختلف لغة البرمجة عن اللغات الطبيعية في أنها لغة شكلية، أي أن كل جملة و كل نص فيها لا يقبل إلا تأويلا واحدا، فلا تتحمل الغموض و لا الإلتباس، مما يحتم علينا دقة كبيرة في المعاملة مع البرمجة.

4.3.3 تسيير البرنامج

لتسيير البرنامج، نفتح المشروع المطلوب ثم نضغط على رمز التنفيذ، فيقوم البرنامج بردود الفعل حسب الرسم التالي:



إذا تأملنا في نتيجة التنفيذ، الآتية في النافذة السفلى من الواجهة، فنجد:

النتيجة	التعليق

`init:`
`deps-jar:`
`Created dir: C:\Documents and Settings\mili\My Documents\arb\jav\jav\echangerate\build\classes`
	تهيئة التنفيذ

`Compiling 1 source file to C:\Documents and Settings\mili\My Documents\arb\jav\jav\echangerate\build\classes`
 	
`compile:`
	إنطلاق المصرف

`run:`	إنطلاق التنفيذ

`to buy 14079.0 iraqi dinars, you need: 16.19085 Tunisian Dinars.`
`when you sell 14079.0 iraqi dinars, you get: 13.867815 Tunisian Dinars.`
 	
`today's date: Ramadan 22, 1428.`
	نتيجة التنفيذ

`BUILD SUCCESSFUL`	تأليف الروابط تم بنجاح
`(total time: 1 second)`	مدة الصرف و الربط و التنفيذ: ثانية واحدة

لو كان البرنامج لامطابق لقواعد النحو التابعة للغة جافا، لفشل طور الصرف، و بالتالي يقع إلغاء التنفيذ لأن المصرف لا يقدر على ترجمة برامج لا تطابق القواعد النحوية. مثلا، إذا نسينا النقطة الفاصلة في السطر رقم 40، بحيث يكون البرنامج على النحو التالي،

```
tndamount = irqamount*buyrate
System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
System.out.println (tndamount+" Tunisian Dinars.");
```

فيفشل المصرف في صرف البرنامج، و يرد الفعل كما يلي:

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\mili\My Documents\arb\jav\jav\echangerate\build\classes
C:\Documents and Settings\mili\My Documents\arb\jav\jav\echangerate\src\echangerate\Main.java:40: ';' expected
        System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
1 error
BUILD FAILED (total time: 1 second)
```

فيعلن المصرف أنه تعرض لخطإ في السطر 41: فعلا فإنه لم يتفطن إلى وجود الخطأ إلا عندما وصل إلى السطر رقم 41؛ لو كان سطر 41 يحتوي على بقية العملية البادئة في السطر 40، لا يكون هناك خطأ. نلاحض أن المصرف أعلن أنه ينتظر نقطة فاصلة، لكن هذا لا يعني ضرورة أن إضافة النقطة الفاصلة هو الحل الوحيد، و قد لا يكون حلا تماما. إذ أن المصرف يجتهد لتوجيه المبرمج، لكنه لا يعرف قصده. فإذا كان المبرمج يرغب في ضرب قيمة العملة العراقية بنسبة الشراء، فيكتب

```
tndamount = irqamount*buyrate;
System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
System.out.println (tndamount+" Tunisian Dinars.");
```

أو حتى

```
tndamount = irqamount*buyrate
;System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
System.out.println (tndamount+" Tunisian Dinars.");
```

(لا فرق بين البرنامجين بالنسبة للمصرف). أما إذا أردنا أن نفرض ضريبة قدرها 5 دنانير تونسية، فنكتب:

```
tndamount = (irqamount*buyrate)-5;
System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
System.out.println (tndamount+" Tunisian Dinars.");
```

و إذا أردنا أن نفرض ضريبة قدرها 5 دنانير عراقية، فنكتب:

```
tndamount = (irqamount-5)*buyrate;
System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
System.out.println (tndamount+" Tunisian Dinars.");
```

5.3.3 إختبار البرنامج

بعد أن نقوم بتحرير البرنامج، نعتزم بإختباره، أي بالتأكد من أنه سديدا أو كشف أخطاء فيه. نفرق بين نوعين من الأخطاء:

- الأخطاء النحوية، المتمثلة في أن البرنامج لا يطابق القواعد النحوية للغة البرمجة. عادة، يتكلف المصرف

بالكشف عن هذه الأخطاء و في بعض الأحيان بإقتراح حل (قد يكون أو لا يكون على صواب). بينّا مثالا من هذه الحالة في الجزء السابق.

- الأخطاء المنطقية، المتمثلة في أن البرنامج لا يؤدي الوظيفة المنتظرة أو لا يقدم النتيجة السديدة. هذا النوع من الأخطاء يصعب الكشف عنه و تصليحه، إذ أنه يتطلب الرجوع إلى مواصفات البرنامج و تحليل دقيقا لكامل البرنامج.

مثلا، إذا كتبنا برنامج تحويل العملة كما يلي،

```
33              double buyrate, sellrate;
34              double tndamount, irqamount;
35
36              buyrate  = 0.00115;
37              sellrate = 0.000985;
38              irqamount = 14079.00;
39
40              tndamount = irqamount*sellrate;
41              System.out.print ("to buy "+irqamount+" iraqi dinars, you need: ");
42              System.out.println (tndamount+" Tunisian Dinars.");
43
44              tndamount = irqamount*buyrate;
45              System.out.print ("when you sell "+irqamount+" iraqi dinars, you get: ");
46              System.out.println (tndamount+" Tunisian Dinars.");
47
48              System.out.println ("today's date:  Ramadan 22, 1428.");
49
```

و نفذنا هذا البرنامج لوجدنا النتيجة التالية:

```
to buy 14079.0 iraqi dinars, you need: 13.867815 Tunisian Dinars.
when you sell 14079.0 iraqi dinars, you get: 16.19085 Tunisian Dinars.
today's date:  Ramadan 22, 1428.
```

إن هذه النتيجة توحينا فكرة لكسب ثروة كبيرة (على حساب البنك) برأس مال قدره 14079 دينار عراقي: كلما بعنا هذه القيمة و إشترياناها من جديد، نربح قيمة قدرها 2.239 دينار تونسي. هذا طبعا ناتج عن خطإ منطقي في البرنامج: إن البرنامج يقيم قيمة الشراء بواسطة نسبة البيع و قيمة البيع بواسطة نسبة الشراء. بما أن هذا خطأ منطقي، و ليس خطأ نحوي، فلا يمكن للمصرف أن يكشفه.

6.3.3 تطوير و صيانة البرنامج

عكسا لكل المنتوجات الصناعية الأخرى، التي لا تتغير بعد إنتاجها، فإن البرامج تتغير و تتطور خلال دورتها الحياتية، حسب تطور حاجيات المستخدم و حسب تطور تقنيات الحاسوب، و حسب الأخطاء التي قد يقع كشفها خلال إستخدام البرنامج، إلخ. بالتالي، يتواكب طور إستخدام البرنامج بتطوير مستمر للبرنامج قصد تكييفه مع مواصفات تتغير بصفة مستمرة و تقنيات في حالة تطور مستمر. يحتم علينا هذا أن نهيئ البرنامج للتطور، فنلتزم بمقاييس في التوثيق و في الهيكلة و في البساطة، إلخ.

4.3 تمارين

1 [س] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليحول قيمة 7854 دينار عراقي إلى دنانير تونسية، مستعملين نفس نسبات التغيير و نفس التاريخ. نفذوا البرنامج و تأكدوا من سداده.

2 [م] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليحول قيمة 7854 دينار عراقي إلى دنانير تونسية، مستعملين نسبات التغيير التابعة لليوم الحالي (قد تجدونها بواسطة بحث على العنكبوتية)، و التاريخ الجاري (لهذا اليوم). نفذوا البرنامج و تأكدوا من سداده.

3 [م] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليحول قيمة 7854 دينار جزائري إلى دنانير تونسية، مستعملين نسبات التغيير التابعة لليوم الحالي (قد تجدونها بواسطة بحث على العنكبوتية)، و التاريخ الجاري (لهذا اليوم). نفذوا البرنامج و تأكدوا من سداده.

4 [م] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليحول قيمة 7854 جنيه إلى ليرات لبنانية، مستعملين نسبات التغيير التابعة لليوم الحالي (قد تجدونها بواسطة بحث على العنكبوتية)، و التاريخ الجاري (لهذا اليوم). نفذوا البرنامج و تأكدوا من سداده.

5 [م] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليفرض ضريبة قدرها خمسة دنانير تونسية على كل العمليات.

6 [م] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليفرض ضريبة قدرها خمسة دنانير تونسية على عملية البيع و ضريبة قدرها 500 دينا عراقي على عملية الشراء.

7 [م] المطلوب منكم تغيير برنامج تحويل العملة الصادر في الجزء رقم 2.3 ليفرض ضريبة قدرها ثلاثة دنانير تونسية و 200 دينار عراقي على كل العمليات.

8 [ص] إن الحاسوب يلعب دورا هاما جدا في تسيير الطائرات العصرية، فيقال عن الطائرات العصرية أنها تتمثل في حاسوب ذات أجنحة. كما يقال أن سائق الطائرة العصرية أصبح مبرمجا بقدر ما هو طيارا، بحيث أنه يتحكم في سير الطائرة بواسطة تعليمات يقدمها للطائرة عن طريق الحاسوب المدمج. نهتم في هذا التمرين ببرنامج السائق الآلي للطائرة، الذي يتلقى تعليمات من قائد الطائرة و بيانات عن الوضع الحالي للطائرة و ينتج إشارات تحكم لمختلف مكونات الطائرة، مثل المحركات و الأجنحة و الذيل و غيرها. يمثل الرسم التالي موقع القيادة التابع لطائر "بوينغ 800/737" (حسب برنامج محاكات الطيران الصادر عن شركة مايكروسوفت).

و تمثل اللوحة أسفله المفاتيح التي يستعملها القائد ليضبط معلمات الطيران، كالسرعة المطلوبة و الإتجاه المطلوب و الإرتفاع المطلوب و السرعة العمودية المطلوبة، إلخ.

أما الرسم أسفله فهو يمثل البيانات التي تدل على الوضع الحالي للطائرة كالسرعة الحالية و الإتجاه الحالي و الإرتفاع الحالي و السرعة العمودية الحالية و قوة المحركات و حرارة المحركات و هيئة الأجنحة و بيانات الملاحة، إلخ.

إذا إعتزمنا أن نفكر في كتابة مواصفات السائق الآلي، فما هي البيانات الواردة؟ كيف نمثلها؟ ما هي البيانات الصادرة؟ كيف نمثلها؟ ما هو شكل المواصفات؟ كيف نمثلها؟

الباب الثاني
العناصر الأساسية في جافا

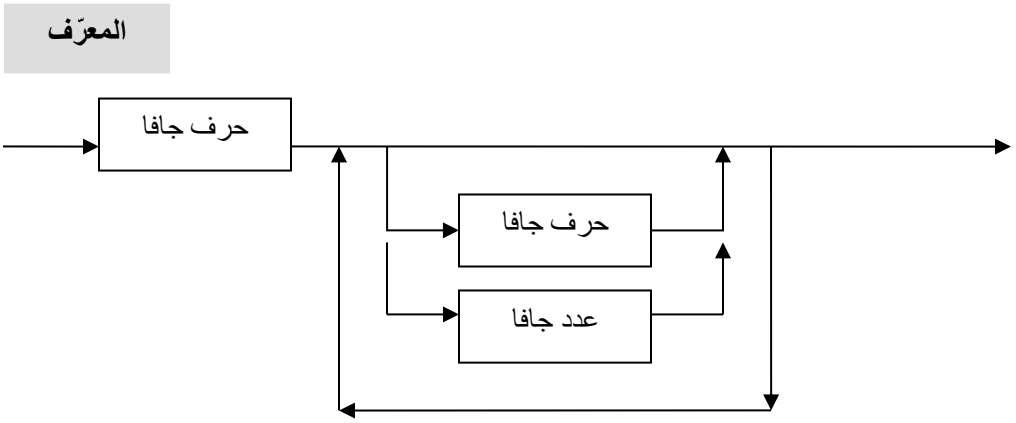
الفصل الرابع
المتغيرات و العبارات

1.4 المتغيرات و المستقرات

من أهم العناصر في لغة جافا، مثل جل اللغات البرمجية، نذكر مفهوم المتغيرات. نستعمل المتغيرات لنمثل كائنات ذات إهتمام بالنسبة لبرنامجنا، مثل:

- درجة الحرارة في مكان ما و زمان ما،
- الوقت الذي طرأ فيه حدث ما،
- إسم شخص ما، إلخ.

نعرف بالمتغيرات في جافا بواسطة ما نسميه المعرّف، و هو سلسلة من الحروف و الرموز و الأعداد، تطابق القواعد النحوية التالية:



حيث يمثل حرف جافا أي حرف لاتيني عادي (a إلى z) أو مضخم (A إلى Z) أو رمز السطر (_) أو رمز الدولار($). كما يمثل عدد جافا أي عدد بين 0 و 9. نقدم أمثلة تطابق تعيين المعرف في جافا:

```
x, x1, birth_date, birthDate, Last$Name.
```

كما نقدم فيما يلي سلاسل لا تطابق تعيين المعرف في جافا:

```
1x, 0to9, me.too, big number.
```

عادة، نستعمل الحروف اللاتينية العادية لنمثل المتغيرات، مع إستعمال الأعداد عند الحاجة لنفرق بين متغيرات مماثلة:

- إذا كانت لنا ثلاثة أعداد تلعب أدوارا متساوية، نسميها: `i1, i2, i3`.
- إذا أردنا أن نمثل أعدادا تتعلق بسنوات (مثلا إنتاج معمل في ثلاثة سنوات)، نسميها: `y2006, y2007, y2008`.
- إذا أردنا أن نمثل أعداد تلميذ في ثلاثة مواد دراسية، نسميها: `maths, physics, history`.

أما إذا كان إسم التغيرة مركبا، فنكوّن المعرف بضم الكلمات المعنية معا، على أن تكون الكلمة الأولى بالأحرف العادية بينما تبتدأ الكلمات الأخرى بالأحرف المضخمة، حسب الأمثلة التالية:

- إذا أردنا أن نمثل تاريخ ولادة، نسميه: `birthDate`.
- إذا أردنا أن نمثل أعداد ثلاثة إمتحانات و معدلها، نكتب: `exam1Grade, exam2Grade, exam3Grade, gradeAverage`.
- إذا أردنا أن نمثل إحصائيات سنوية لمؤسسة ما، نكتب مثلا: `yearlyIncome, yearlyExpenses, yearlyBenefits`.

نستعمل المتغيرات لخزن قيمات ذات أهمية بالنسبة للبرنامج خلال تنفيذ البرنامج. فتتغير هذه المتغيرات لتحمل قيمات إفتتاحية و قينات مؤقتة و قيمات نهائية خلال تنفيذ عمليات البرنامج. هناك حالات حيث نريد خزن قيمات لا تتغير خلال تنفيذ البرنامج، فنستعمل مستقرات لهذا الغرض. عكسا للمتغيرات، فإن المستقرات لا تتغير قيماتها خلال التنفيذ. تنص قواعد الممارسة أن نستعمل الحروف اللاتينية المضخمة لتمثيل المستقرات، مع إستعمال رمز السطر (_) لنفصل بين الكلمات في المعرفات المركبة. على باب المثال، لنا حسابات إدخار في البنك نريد أن نحتسب الرصيد الجديد لهذه الحسابات بعد أن نضيف لها الفائض السنوي، فنمثل بياناتنا كما يلي:

- متغيرة نلقي عليها إسم **accountBalance** تمثل رصيد حسابات حرفاء البنك؛ تختلف رصائد الحسابات من حريف لحريف، فنستعمل متغيرة لهذا الغرض.
- مستقرة نلقي عليها إسم **INTEREST_RATE** تمثل نسبة الفائض الذي نطبقه على هذه الحسابات؛ نطبق نفس النسبة لكل الحرفاء، فنستعمل مستقرة لهذا الغرض.
- نحتسب الرصيد الجديد لكل حساب حسب الصيغة التالية:

```
accountBalance * (1 + INTEREST_RATE).
```

2.4 أنواع البيانات

تشترط لغة جافا أن نصرح بكل المتغيرات و المستقرات التي نستعملها في نطاق برنامج. تتمثل عملية التصريح في:

- إسناد إسم للمتغيرة أو المستقرة.
- إسناد نوع من أنواع البيانات للمتغيرة أو المستقرة. تشمل لغة جافا أربعة أنواع بيانات، و هي: الأعداد الكاملة (**byte, short, int, long**) و الأعداد الحقيقية (**float, double**) و الحروف (**char**) و القيمات المنطقية (**boolean**).
- يمكن إسناد قيمة للمتغيرة أو المستقرة عند التصريح بها. نفرق بين التصريح بمتغيرة و التصريح بمستقرة في أن المستقرة تسبقها كلمة خاصة، و هي **final** و تصحبها عملية إسناد قيمة لهذه المستقرة لا تتغير طوال تنفيذ البرنامج. يحرص المصرف على أن لا نغير قيمة المستقرة خلال تنفيذ البرنامج. نقدم فيما يلي عمليات تصريح بمتغيرات و مستقرات في لغة جافا. بالرغم من أن جافا يسمح لنا إسناد قيمة إفتتاحية للمتغيرات، فنفضل أن لا نستعمل هذه الإمكانية، فتكتفي عملية التصريح بإسناد نوع بيانات للمتغيرة.

نقدم فيما يلي أمثلة بسيطة للتصريح بمتغيرت و مستقرات.

التصريح في جافا		التعليق
`int i;`	//	التصريح بمتغيرة من نوع الأعداد الكاملة
`boolean equalTerms;`	//	التصريح بمتغيرة من نوع القيمات المنطقية
`final int TABLE_SIZE = 41;`	//	التصريح بمستقرة من نوع الأعداد الكاملة و إسنادها قيمة 41
`final float PI = 3.14159;`	//	التصريح بمستقرة من نوع الأعداد الحقيقية و إسنادها قيمة 3.14159
`char answer;`	//	التصريح بمتغيرة من نوع الحروف

عندما نصرح بمستقرة نسميها مثلا و نسندها قيمة 41، نستعملها خلال البرنامج كلما أردنا أن نشير لقيمة 41، فقد يتسائل القارئ: لماذا لا نستعمل 41 عوضا عن إسم المستقرة و نحذف تصريح المستقرة تماما؟ هناك عدة أهداف يجعلون إستعمال المستقرة أفضل:

- هدف توثيقي: كلما نرى معرف **TABLE_SIZE** في نص البرنامج، نعرف أنه يدل على حجم المائدة المعنية. بينما قيمة 41 لا تبين ما تمثله، و قد لا تمثل نفس الشيء كلما ظهرت. مثلا، قد تمثل حجم المائدة في بعض الحالات، و عمر شخص في حالات أخرى، و أجر عامل في حالات أخرى (بحساب الدينار في الساعة، مثلا)، و طول مسافة بين مدينتين في حالة أخرى، إلخ.
- هدف صياني: إذا أردنا أن نغيير حجم المائدة في يوم ما، نكتفي بأن نغير التصريح بالمستقرة و تبقى بقية البرنامج على حالها. لو لم نستعمل هذا التصريح لإضطرينا أن نفحص كامل البرنامج سطرا سطرا نبحث عن كل مكان تظهر فيه قيمة 41 و تكون تشير إلى حجم المائدة، فنغير هذه القيمة إلى القيهة الجديدة.

- هدف وظيفي: إذا وُجد مكان في البرنامج حيث نحتاج لقيمة حجم المائدة إلا 1 مثلا، فتظهر هذه القيمة ك40، بحيث لا نفطن بها إذا كنا نبحث عن قيمة 41 ، مما قد يؤدي إلى أخطاء عندما نغير حجم المائدة.

في الأجزاء التالية نهتم بدراسة أنواع البيانات في جافا، فننكب على التوالي على الأعداد الكاملة، ثم الأعداد الحقيقية ثم الحروف ثم القيمات المنطقية. ينفعنا في هذا المساق أن نعين ما نعني بنوع البيانات، لأن ذلك له تأثير على كيفية دراستهم و تحليلهم.

التعيين رقم 1.4. يتمثل نوع بيانات في عنصرين إثنين:
- مجموعة من القيمات،
- مجموعة من العمليات التي تنطبق على هذه القيمات.

توضيحا لهذا التعيين، نقدم في الجدول التالي عددا من أنواع البيانات، كما نبين بالنسبة لكل نوع مجموعة القيمات و مجموعة العمليات التابعة له.

نوع البيانات	مجموعة القيمات	مجموعة العمليات
الأعداد الكاملة	0، 1، 2، 3، 1-، 2-، 3-، ...	الجمع، الطرح، الضرب، القسمة الكاملة، الباقي، ....
الأعداد الحقيقية	0، 0.5، 0.65-، 1.2، 0.001-، ...	الجمع، الطرح، الضرب، القسمة الحقيقية، الجيب، ....
القيمات المنطقية	غالط، صحيح	العطف، الخيار، النفي، الضمان المنطقي، ....
الحروف	….,C,c,B,b,A,a	القرائة، الكتابة، المقارنة، ....

1.2.4 الأعداد الكاملة

توفر لغة جافا أربعة أنواع من الأعداد الكاملة، تختلف عن بعضها بعضا حسب عدد الثنائيات التي تستغرقها، و بالتالي حسب حجم الأعداد التي تمثلها:

النوع	عدد الثنائيات	أدنى عدد	أقصى عدد
`byte`	8	-128	127
`short`	16	-32768	32767
`int`	32	-2147483648	2147483647
`long`	64	-9223372036854775808	9223372036854775807

أما عن العمليات التي تنطبق على الأعداد الكاملة في جافا، فنذكر عمليات الجمع و الطرح و الضرب و القسمة الكاملة و الباقي. نبين عمليتي القسمة الكاملة و الباقي بالمثال البسيط التالي:

المقسوم	القاسم	القسمة	الباقي
15	4	3	3

0	4	4	16
1	4	4	17
2	4	4	18
3	4	4	19
0	5	4	20
1	5	4	21
2	5	4	22
3	5	4	23
0	6	4	24
1	6	4	25

عندما نقوم بعمليات عددية على متغيرات من نوع الأعداد الكاملة، يجب أن ننتبه لإمكانية الفيضانات، و ذلك بسبب الحدود الموجودة في إمكانية تمثيل الأعداد الكاملة في الحاسوب. نحن نعلم مثلا أن الأعداد الكاملة من نوع **int** تتراوح بين 2147483648- (أدنى قيمة) و 2147483647 (أقصى قيمة). فماذا يصير إذا صرحنا بمتغيرة من نوع **int** و أسندنا لها أقصى قيمة لهذا النوع، ثم زدنا لها 1 أو 2، مثلا؟ و ماذا يصير إذا صرحنا بمتغيرة من نوع **long** عوضا عن **int** و قمنا بنفس العمليات؟ نتأمل بسرعة في برنامج يقوم بهذه التجارب:

```
public static void main(String[] args) {                    //  1.
        // TODO code application logic here                 //  2.
        int i =  2147483647;                                //  3.
        System.out.println(i);                              //  4.
        i = i+1;                                            //  5.
        System.out.println(i);                              //  6.
        i = i+1;                                            //  7.
        System.out.println(i);                              //  8.
        i =  2147483647;                                    //  9.
        i = i*2;                                            // 10.
        System.out.println(i);                              // 11.
        long j =  2147483647;                               // 12.
        System.out.println(j);                              // 13.
        j = j+1;                                            // 14.
        System.out.println(j);                              // 15.
        j = j+1;                                            // 16.
        System.out.println(j);                              // 17.
        j =  2147483647;                                    // 18.
        j = j*2;                                            // 19.
        System.out.println(j);                              // 20.
    }                                                       // 21.
```

في السطر 3، نصرح بمتغيرة **i** و نظع فيها قيمة **2147483647**، و عي قيمة قانونية تستوعبها هذه المتغيرة، مع أنها أقصى قيمة تستوعبها المتغيرة. بالتالي، فإن العمليات التي نقوم بها في كل من ألاسطر 5 و 7 و 10، كلها تؤدي إلى فيضان، لأنها كلها تسند لمتغيرة **i** قيمة تتجاوز وسعها. فإذا قمنا بنفس العمليات في الأسطر 12 إلى 20 على المتغيرة **j** من نوع **long** (الأعداد الطبيعية الطويلة) فإن كل هذه العمليات قد تتنفذ بصفة عادية لأنها تضع في متغيرة **j** قيمات تستوعبها. نتأكد من هذا حسب نتيجة التنفيذ، حيث يسفر تنفيذ هذا البرنامج على النتائج التالية:

```
init:
deps-jar:
Compiling 1 source file to
C:\Documents and Settings\Ali Mili\tryvars\build\classes
compile:
run:
2147483647   <--
```

```
-2147483648     |  <--
-2147483647     |     |  <--
-2              |     |     |  <--
2147483647   <--    |     |     |
2147483648         <--    |     |
2147483649               <--    |
4294967294                     <--
BUILD SUCCESSFUL (total time: 6 seconds)
```

تبين السهام في هذا الرسم النتائج المتناظرة بالنسبة للعمليات التي قمنا بها على متغيرتي و التان لا تختلفان إلا من حيث نوع البيانات. كلما تختلف النتيجتان المتناظرتان، فالسبب في ذلك هو الفيضان.

2.2.4 ألأعداد الحقيقية

توفر لغة جافا نوعان إثنان من الأعداد الحقيقية، و هما الأعداد ذات النقطة العائمة و الأعداد المثنات. يختلف هذان النوعان من حيث حجمهما و بالتالي من حيث جودة تمثيلهما. بصفة عامة، نمثل الأعداد الحقيقية، سواء كانت من نوع الأعداد ذات النقطة العائمة أو من نوع الأعداد المثنات، حسب النموذج التالي:

$$M \times 10^{X}$$

الذي نمثله في لغة جافا بالصيغة التالية: **M e X**. نلقي على **M** إسم جسم العدد و نلقي على **X** إسم داعية العدد.

بالنسبة للأعداد الحقيقية، نهتم بوجهين إثنين من جودة التمثيل، و هما:

- إتساع التمثيل، أي مجال الأعداد الحقيقية التي يمكن تمثيلها بالنسبة لكل نوع. إن قيمة الداعية تعين إتساع التمثيل. نهتم بأكبر عدد يمكن تمثيله في لغة جافا لنعكس إتساع التمثيل.
- دقة التمثيل (أو كثافة التمثيل)، أي المسافة بين عددين متتاليين ضمن الأعداد القابلة للتمثيل في لغة جافا. إن عدد البقاع في جسم العدد يعين دقة التمثيل. نهتم بأصغر عدد إجابي يمكن تمثيله في لغة جافا لنعكس دقة التمثيل.

نبين في الجدول التالي، بالنسبة لكل نوع من أنواع الأعداد الحقيقية، أصغر عدد، أصغر عدد من حيث القيمة النسبية، و أصغر عدد من حيث القيمة المطلقة.

نوع الأعداد	أكبر عدد أصغر عدد نسبي أصغر عدد مطلق	رمز جافا
الأعداد ذات النقطة العائمة	3.40282347e+38	float
	-3.40282347e+38	
	1.40239846e-45	
الأعداد المثنات	1.79769313486231570e+308	double
	-1.79769313486231570e+308	
	4.94065645841246544e-324	

عنما درسنا الأعداد الكاملة أعلاه، أثرنا إمكانية الفيضان، حيث أن عملية جمع عددين من الأعداد القابلة للتمثيل في لغة جافا قد تؤدي إلى عدد غير قابل للتمثيل في جافا. في نطاق دراسة الأعداد الحقيقية، نثير ثلاثة مشاكل:

- الفيضان. يطرأ فيضان كلما نقوم بعملية تسفر عن عدد غير قابل للتمثيل، بحكم كبره (إذا كان أكبر من أكبر عدد قابل للتمثيل).
- الذوبان. يطرأ ذوبان كلما نقوم بعملية تسفر عن عدد غير قابل للتمثيل، بحكم صغره (إذا كان أصغر من أصغر عدد قابل للتمثيل).
- تضاعف التقريب. كلما قامت جافا بعملية على الأعداد الحقيقية فوجدت نتيجة غير قابلة للتمثيل، إلا و مثلتها بأقرب عدد لها ضمن الأعداد الحقيقية القابلة للتمثيل. نقول عندئذ أننا شهدنا ظاهرة خطأ تقريب. إذا شمل برنامج عددا كبيرا من العمليات على الأعداد الحقيقية، فتتضاعف أخطاء التقريب لحد أن النتيجة النهائية قد تختلف إختلافا ملموسا عن النتيجة الصحيحة.

نقدم فيما يلي برنامجا يوضح على التوالي ظواهر الفيضان، الذوبان، و تضاعف التقريب:

```
public static void main(String[] args) {                    //  1.
        // TODO code application logic here                 //  2.
                                                            //  3.
        //  illustrate overflow                             //  4.
        double x = 1.79769313486231570e+308;                //  5.
        System.out.println(x);                              //  6.
        double y = 2*x;                                     //  7.
        System.out.println(y);                              //  8.
                                                            //  9.
        //  illustrate underflow                            // 10.
        double v = 4.94065645841246544e-324;                // 11.
        System.out.println(v);                              // 12.
        double w = v/2;                                     // 13.
        System.out.println(w);                              // 14.
                                                            // 15.
        //  illustrate roundoff errors                      // 16.
        x = Math.sqrt(x);                                   // 17.
        x = x*x;                                            // 18.
        System.out.println(x);                              // 19.
        x = Math.log(x);                                    // 20.
        x = Math.exp(x);                                    // 21.
        System.out.println(x);                              // 22.
    }                                                       // 23.
```

نعلق بإيجاز على هذا البرنامج البسيط:

- الأسطر 5 إلى 8: نخزن أكبر عدد حقيقي تستوعبه لغة جافا، ثم نطبعه ثم نضربه في 2 و نطبع النتيجة. علما أن النتيجة غير قابلة للتمثيل في لغة جافا، نتوقع أن تكون النتيجة غير صحيحة.
- الأسطر 11 إلى 14: نخزن أصغر عدد حقيقي تستوعبه لغة جافا، ثم نطبعه ثم نقسمه في 2 و نطبع النتيجة. علما أن النتيجة غير قابلة للتمثيل في لغة جافا، نتوقع أن تكون النتيجة غير صحيحة.
- الأسطر 17 إلى 22: في هذه الأسطر نبين ظاهرة تضاعف التقريب، حيث نستعمل قيمة **x** الأصلية فنستخرج جذرها الثاني ثم نحتسب مربعها؛ مبدئيا، يجب أن نسترجع القيمة الأصلية لهذه المتغيرة، لكن نتوقع أن تحول ظاهرة تضاعف التقريب دون ذلك. في الأسطر 21 و 22 نقوم بتجربة أخرى حيث نطبق وظيفتين متعاكستين على أن نسترجع القيمة الأصلية، لكن مرة أخرى ت تحول ظاهرة تضاعف التقريب دون ذلك.

نتأمل في نتيجة تنفيذ هذا البرنامج، فنجد أنها تؤكدكل توقعاتنا:

```
1.7976931348623157E308
Infinity
4.9E-324
0.0
1.7976931348623155E308
```

```
1.7976931348622732E308
```

نعلق على هذه النتائج سطرا سطرا فيما يلي:

النتيجة	التعليق
`1.7976931348623157E308`	أكبر قيمة قابلة للتمثيل
`Infinity`	إذا ضربنا أكبر قيمة قابلة للتمثيل في 2، وجدنا عددا لامنتهيا
`4.9E-324`	أصغر قيمة قابلة للتنفيذ
`0.0`	إذا قسمنا أصغر قيمة قابلة للتنفيذ في 2، وجدنا الصفر
`1.7976931348623155E308`	إذا أخذنا القيمة الأصلية لـ`x` ثم إحتسبنا جذرها ثم مربعها، لوجدنا قيمة مختلفة (تأمل في العدد المسطر)
`1.7976931348622732E308`	إذا أخذنا القيمة الجديدة لـ`x` ثم نفذنا عليها وظيفتين متعاكستين، لوجدنا قيمة مختلفة (تأمل في الأعداد المسطرة)

3.2.4 الحروف

توفر لغة جافا نوعا من البيانات تحت إسم `char` يحتوي على جميع الحروف اللاتينية العادية و المضخمة، كما يحتوي على الأرقام 0 إلى 9 و على عدد من الرموز، من بينها الحرف الأبيض الذي يدل على الفضاء بين كلمتين، مثلا. نقدم بعض الأمثلة في تصريحات لمتغيرات و مستقرات من هذا النوع:

```
final char BLANK = ' ';
final char PERIOD = '. ';
char c, d;
```

إذا أردنا أن نمثل حرفا في لغة جافا، فنضع رمز ' قبله و بعده، كما يبين المثال أعلاه. نحذر القارئ أن يفرق بين الحرف المكشوف 'c' و متغيرة من نوع الحروف إسمها `c`. نقدم مثالا بسيطا لبرنامج جافا يعالج حروفا:

```
final char BLANK = ' ';
final char PERIOD = '.';
char c, d;
c = 'd'; d = 'c';
System.out.println("c:"+BLANK+c+PERIOD+BLANK+"d:"+BLANK+d+ PERIOD);
```

يسفر تنفيذ هذا البرنامج البسيط على النتيجة التالية:

```
c: d. d: c.
```

نحرض القارئ المهتم أن يتأكد من أنه يفهم العلاقة بين هذا البرنامج و هذه النتيجة. نشير في هذا الشأن إلى الفرق بين الحروف، التي نمثلها بواسطة علامة ' و سلاسل الحروف، التي نمثلها بواسطة علامة ". كما نشير أن علامة الجمع (+) عندما نطبقها على السلاسل، تربط بينها، فتضع السلسلة بعد الأخرى... ممّا يؤدي للنتيجة التي نجدها أعلاه. نشير أيضا للفرق بين حرف يمثل عددا، ك '0' أو '1' أو '2' و العدد الذي يمثله الحرف، ك0 أو 1 أو 2، إلخ؛ الفرق بينهما هو نوع البيانات (حروف أو أعداد). و ما ينجر عن هذا الفرق هو فرق في العمليات التي يمكن تطبيقها (مثلا، يمكن ضرب عددين في بعضهما و لا يمكن ضرب حرفين).

أما عن العمليات بين الحروف، فنذكر من أهمها عملية المقارنة بين الحروف، التي تعكس الترتيب الأبجدي للحروف اللاتينية، مع التراتيب التالية:

- ترتيب الحروف التي تمثل الأعداد يوازي ترتيب الأعداد نفسها، أي أن حرف '0' أقل من حرف '1' أقل من حرف '2' ... أقل من حرف '9'.
- الحروف التي تمثل الأعداد أقل من الحروف التي تمثل الأبجدية اللاتينية.
- الحروف اللاتينية المضخمة أقل من الحروف اللاتينية العادية.

فنجد الترتيب التالي:

‘0’<’1’<…<’9’<’A’<’B’<…<’Z’<’a’<’b’<…<’z’.

طبقا لهذه التراتيب, يسفر البرنامج التالي

```
System.out.println('a'<'b');
System.out.println('Z'<'a');
System.out.println('9'<'A');
```

على النتيجة التالية:

```
true
true
true
```

و هو ما نتوقعه على ضوء القواعد أعلاه.

أما عن الحروف العربية، فيبدو أن جافا ترتبها حسب الترتيب الأبجدي العربي مع ترتيب الأعدا قبلها، كما يبين البرنامج التالي

```
System.out.println('ب'>'أ');
System.out.println('ت'>'ب');
System.out.println('ث'>'ت');
System.out.println('ج'>'ث');
System.out.println('ح'>'ج');
System.out.println('خ'>'ح');
System.out.println("يبدو أن هذا الترتيب صحيحا");
System.out.println('9'<'أ');
System.out.println('ب'>'ب');
System.out.println('ب'>'ت');
```

الذي يسفر على النتيجة التالية:

```
true
true
true
true
true
true
يبدو أن هذا الترتيب صحيحا
true
false
false
```

4.2.4 القيمات المنطقية

توفر لنا لغة جافا نوع بيانات يتضمن قيمتين إثنتين، و هما صحيح (**true**) و غالط (**false**). نلقي على هذا النوع إسم **boolean**. نستعمل متغيرات من هذا النوع لنخزن شروطا منطقية أو لنحتسب عبارات منطقية. توفر لنا لغة جافا عددا من العمليات المنطقية التي يمكن القيام بها على متغيرات من هذا النوع، نذكر منها بالخصوص:

- العطف المنطقي، الذي نمثله في جافا برمز **&&**.
- الخيار المنطقي، الذي نمثله في جافا برمز **|**.
- المساواة المنطقية، الذي نمثلها في جافا برمز **==**.
- النفي المنطقي، الذي نمثله في جافا برمز **!**.

يبين الجدول التالي قيمات هذه العمليات حسب قيمات أطرافها:

الأطراف		العمليات			
		العطف	الخيار	المساواة	النفي
a	b	a && b	a | b	a == b	!a
صحيح	صحيح	صحيح	صحيح	صحيح	غالط
غالط	صحيح	غالط	صحيح	غالط	صحيح
صحيح	غالط	غالط	صحيح	غالط	غالط
غالط	غالط	غالط	غالط	صحيح	صحيح

يختبر البرنامج التالي عملية المساواة حسب ما جاءت في المائدة أعلاه،

```
System.out.println(true==true);
System.out.println(true==false);
System.out.println(false==true);
System.out.println(false==false);
```

فتكون النتيجة على النحو التالي:

```
true
false
false
true
```

و هو فعلا مطابق للجدول المذكور.

نقدم برنامجا بسيطا يتأكد من أن جافا يرتب الحروف العربية حسب الترتيب الأبجدي، على الأقل بالنسبة للحروف الأولى، فيضع النتيجة في متغيرة منطقية. فنكتب ما يلي:

```
public static void main(String[] args) {
        // TODO code application logic here

        boolean alphaorder;

        alphaorder = ('أ'<'ب') &&
                     ('ب'<'ت') &&
                     ('ت'<'ث') &&
                     ('ث'<'ج') &&
                     ('ج'<'ح') &&
                     ('ح'<'خ');
        if (alphaorder)
        {
            System.out.println("يبدو أن هذا الترتيب صحيحا");
        }
        else
        {
            System.out.println("هذا الترتيب غالط");
        }
    }
```

فيسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:

يبدو أن هذا الترتيب صحيحا

BUILD SUCCESSFUL (total time: 0 seconds)
```

غالبا من نستعمل المتغيرات المنطقية لنعبر على شروط مساواة أو شروط تفاوت بين متغيرات أو عبارات، فنستعمل لهذا الغرض عمليات مقارنة مثل

- المساواة، نمثلها برمز ==.
- اللامساواة، نمثلها برمز =!.
- التفاوت الشديد بين الأعداد، نمثله برموز <، >.
- التفاوت المنحل بين الأعداد، نمثله برموز =<، =>.
- التفاوت الشديد بين الحروف، نمثله برموز <، >.
- التفاوت المنحل بين الحروف، نمثله برموز =<، =>.

3.4 العبارات

تمكننا لغة جافا من تكوين عبارات بواسطة عمليات فردية و عمليات زوجية نطبقها على متغيرات و مستقرات في مختلف أنواع البيانات. مثلا، إذا كانت **a** و **b** متغيرات أو مستقرات من نوع الأعداد الكاملة و كانت **x** و **y** متغيرات أو مستقرات من نوع الأعداد الحقيقية و كانت **p** و **q** متغيرات أو مستقرات من نوع القيمات المنطقية، فنكتب فيما يلي على التوالي عبارة من نوع الأعداد الكاملة و عبارة من نوع الأعداد الحقيقية و عبارة منطقية.

```
((a+b) % (5) ) * ((-b) + a) / 15
Math.log(Math.abs(x-2.0)) + 2.1*y*y
(!p && !q) | (q && !p) && q
```

إن العبارات في جافا تخضع لقواعد نحوية تمكننا من ضبط العبارات الصحيحة كما تخضع لقواعد دلالية تمكننا من تأويلها. على باب المثال، ننظر في القواعد النحوية لعبارات الأعداد الكاملة، فنكتبها كما يلي:

```
<expression> ::= <expression> + <term>
               | <expression> - <term>
               | - <expression>
               | <term>

<term>       ::= <term> * <factor>
               | <term> / <factor>
               | <term> % <factor>
               | <factor>

<factor>     ::= <vname>
               | <literal>
               | (<expression>)

<vname>      ::= a | b | c | d | e …

<literal>    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 .....
```

بالنسبة لكل قاعدة، نلقي على الرمز الموجود على اليسار إسم رأس القاعدة و نلقي على سلسلة الرموز الموجودة على اليمين إسم جسد القاعدة. عندما نضع علامة | بين سلسلتين، فهذا يدل على إمكانيات بديلات في الجسد. بحيث إذا كتبنا قاعدة مثل

Head ::= Body1 | Body2

فهذا يدل على قاعدتين، هما

Head ::= Body1
Head ::= Body2.

نترجم الألفاظ التابعة للقواعد أعلاه كما يلي:

English	عربي
Expression	عبارة
Term	طرف
Factor	عامل
Vname	إسم
literal	عدد

إذا أردنا أن نحلل عبارة عددية حسب هذه القواعد النحوية، نبني شجرة ثنائية نقي عليها إسم شجرة العبارة تخضع للشروط التالية:

- تمثل كل عقدة من عقد الشجرة رمزا من رموز النحو،
- تمثل كل عقدة نهائية (ورقة) من هذه الشجرة رمزا من رموز العبارة.
- تتمثل العبارة في سلسلة أوراق الشجرة إذا قرأناها من اليسار إلى اليمين.
- تمثل كل عقدة داخلية للشجرة قاعدة من القواعد النحوية، حيث تشير العقدة لرأس القاعدة و تشير العقد المربوطة بهذه العقدة إلى جسم القاعدة.
- يمثل جذر الشجرة (نرسمه في قمة الشجرة لأنها مقلوبة) رمز العبارة (<expression>).

على باب المثال، نبين في الرسم رقم 1.4 شجرة العبارة التابعة للعبارة العددية التالية:

```
(a+b) * c / d.
```

يمكن قراءة هذه العبارة إذا تابعنا أوراق الشجرة (في الرسم رقم 1.4) من اليسار إلى اليمين.

الجدير بالذكر بخصوص هذه القواعد النحوية أنها تنص ضمنيا على عدد من التراتيب الخاصة بتأويل العبارات، نذكر منها بالخصوص:

- التجمع على اليسار. أي أننا إذا كتبنا مثلا

```
a - b - c
```

فنأول هذه العبارة ك

```
(a - b) - c
```

عوضا عن

```
a - (b - c).
```

- ترتيب الأولويات. إذا كتبنا مثلا

```
a + b * c
```

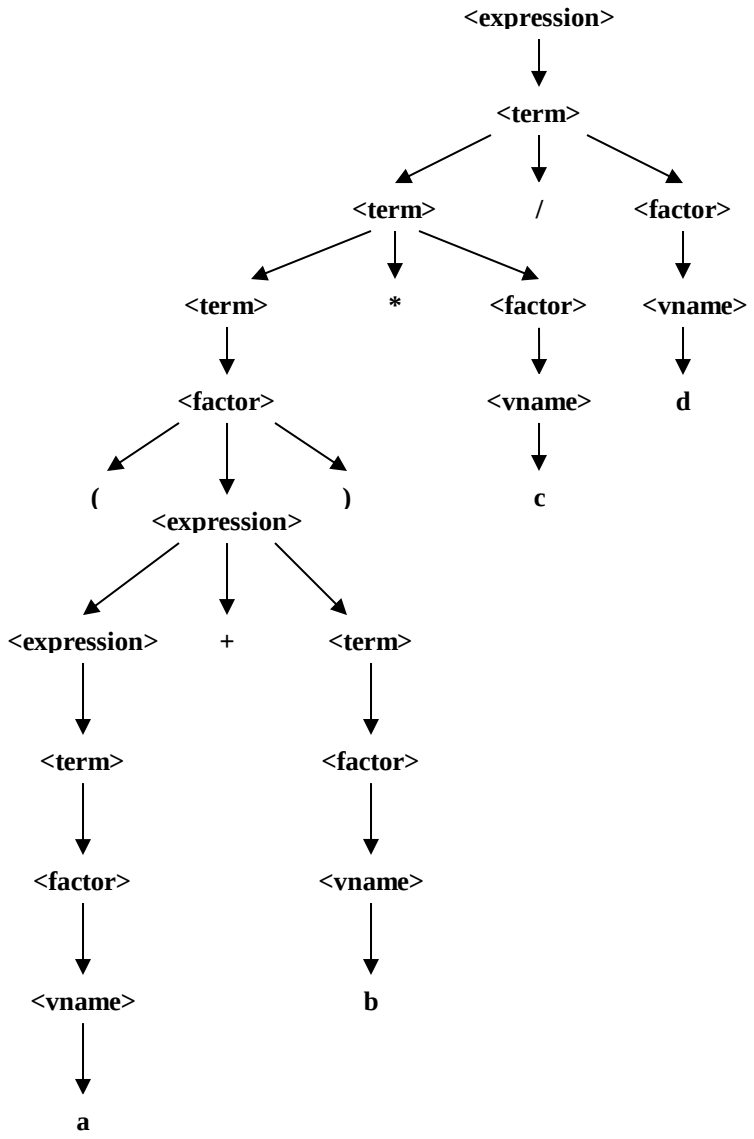
فنأول هذه العبارة ك

```
a + (b * c)
```

عوضا عن

```
(a + b) * c.
```

بصفة عامة، إذا كتبنا عمليات عددية متتالية، تنفذ جافا عمليات الضرب و القسمة و الباقي قبل أن تنفذ عمليات الجمع و الطرح.



الرسم رقم 1.4: شجرة العبارة، `(a+b) * c / d`

يمكن تعيين قواعد نحوية بالنسبة لعبارات الأعداد الحقيقية و القيمات المنطقية في جافا، لكن هذا يتجاوز نطاق هذا الفصل، فنحيله للقارئ في شكل تمارين في آخر الفصل.

4.4 الإسناد و تحويل الأنواع

تتمثل تعليمة الإسناد في لغة جافا في إسناد قيمة عبارة لمتغيرة ما، عوضا عن القيمة الحالية لهذه المتغيرة. نكتب هذه التعليمة في جافا كما يلي:

```
x = E
```

حيث تمثل `x` متغيرة من نوع ما، و تمثل `E` عبارة نوعها يطابق نوع المتغيرة. ليس ضروري أن يكون نوع المتغيرة و نوع العبارة متساويان، بل يجب فقط أن يكون نوع المتغيرة قابل لإستيعاب قيمة العبارة، كما سنناقش فيما يلي. يمثل البرنامج التالي بعض الأمثلة في تعليمات الإسناد.

```
long a;                                                    //  1.
double x;                                                  //  2.
char c;                                                    //  3.
boolean b;                                                 //  4.
final char BLANK = ' ';                                    //  5.
                                                           //  6.
a = 9;                                                     //  7.
a = a+1;                                                   //  8.
x = 3.24e3;                                                //  9.
c = '9';                                                   // 10.
b = (c<'9') && (a<9);                                      // 11.
                                                           // 12.
System.out.println(a);                                     // 13.
System.out.println(x);                                     // 14.
System.out.println(c);                                     // 15.
System.out.println(b);                                     // 16.
```

في السطر السابع نضع قيمة 9 في متغيرة من نوع الأعداد الكاملة الطويل، فنضيف 1 لهذه المتغيرة في السطر الموالي. في السطر 9 نسند قيمة من نوع الأعداد الحقيقية لمتغيرة `x` و في السطر العاشر نسند لمتغيرة `c` حرف 9. في السطر الحادي عشر نضيف لمتغيرة `b` المنطقية نتيجة مقارنة `c` مع حرف 9 و مقارنة `a` مع عدد 9. نتوقع أن تكون النتيجة سلبية لأن كلا الشرطان غالطان. في الأسطر 13 إلى 16 نطبع على التوالي الأربعة متغيرات، فنجد النتيجة التالية.

```
10
3240.0
9
false
```

يتكلف المصرف بمراقبة كل عمليات الإسناد ليتأكد أن المتغيرة التي تمثل هدف الإسناد قابلة فعلا لإستيعاب القيمة المسندة لها. نلقي على هذه الخدمة إسم مراقبة الأنواع. مبدئيا، يحرص المصرف على أن تكون العبارة و المتغيرة ذات نفس النوع، لكنه يتسامح في خرق هذه القاعدة في بعض الحالات:

- يسمح لمتغيرة من نوع الأعداد الكاملة الطويلة أن تستوعب عددا كاملا عاديا (السطر 16) لكن لا يسمح الإسناد المعاكس (سطر 15).
- يسمح لمتغيرة من نوع الأعداد الحقيقية، سواء كانت مثنات أم لا، أن تستوعب عددا كاملا ، سواء كان طويلا أم لا (الأسطر18، 19، 23، 24) ، لكن لا يسمح الإسناد المعاكس (الأسطر 17و 20و 21و 22).
- لا يسمح الإسناد بين الحروف و سلاسل الحروف، حتى لو كانت السلسلة ذات عنصر واحد.

هناك إمكانية لتحويل نوع بيانات متغيرة أو عبارة في نطاق عبارة عند تقييمها، بواسطة وظائف خاصة توفرها لغة جافا. نفترض أن لنا متغيرتان من نوع الأعداد الكاملة، نريد أن نحتسب قيمة قسمة إحداهم بالأخرى، فنكتب ما جاء في السطر 37.

- في هذا السطر، بما أن الطرفان في هذه العملية من نوع الأعداد الكاملة، فيأول جافا رمز القسمة كقسمة كاملة (أي تؤدي إلى نتيجة كاملة بدون باقي).
- يمكن تحويل هذه النتيجة من نوعها كعدد كامل إلى نوع الأعداد الحقيقية، فنطبق وظيفة (`float`) على النتيجة (السطر 38) مما لا يغير شيئا، إذ أن النتيجة تنتقل من 0 إلى 0.0.
- أما إذا أردنا أو يأول جافا رمز القسمة كقسمة حقيقية (نتيجتها عدد حقيقي)، فيجب أن يكون أحد الطرفين في القسمة من نوع الأعداد الحقيقية؛ هذا ما قمنا به في السطر 39، فتكون النتيجة `0.09090909` كما نرى أسفله.

- في السطر 40، نحول كلا العاملين في القسمة من أعداد كاملة إلى أعداد حقيقية، فتكون النتيجة 0.09090909 كما كان الحال في السطر السابق.

```
    int a;                                                          //  1.
    long aa;                                                        //  2.
    double x;                                                       //  3.
    float y;                                                        //  4.
    char c;                                                         //  5.
    String s;                                                       //  6.
                                                                    //  7.
    a = 9;                                                          //  8.
    aa = 99;                                                        //  9.
    x = 3.24e3;                                                     // 10.
    y = 453.2f;                                                     // 11.
    s = new String("99");                                           // 12.
    c = '9';                                                        // 13.
                                                                    // 14.
    // a = aa;        //  possible loss of precision               // 15.
    aa = a;                                                         // 16.
    // a = x;         //  possible loss of precision               // 17.
    x = a;                                                          // 18.
    x = aa;                                                         // 19.
    // a = x;         //  possible loss of precision               // 20.
    // aa = x;        //  possible loss of precision               // 21.
    // a = y;         //  possible loss of precision               // 22.
    y = a;                                                          // 23.
    y = aa;                                                         // 24.
                                                                    // 25.
    //  c = s;        // incompatible types                       // 26.
    //  s = c;        // incompatible types                       // 27.
                                                                    // 28.
    System.out.println(a);                                          // 29.
    System.out.println(aa);                                         // 30.
    System.out.println(x);                                          // 31.
    System.out.println(y);                                          // 32.
    System.out.println(c);                                          // 33.
    System.out.println(s);                                          // 34.
    a = 9;                                                          // 35.
    aa = 99;                                                        // 36.
    System.out.println(a/aa);                                       // 37.
    System.out.println((float)(a/aa));                              // 38.
    System.out.println((float)a/aa);                                // 39.
    System.out.println(((float) a)/ ((float) aa));                  // 40.
```

```
init:
deps-jar:
Compiling 1 source file to …
compile:
run:
9
9
9.0
9.0
9
99
0
                                                                    0.0
                                                                  0.09090909
```

```
0.09090909
BUILD SUCCESSFUL (total time: 0 seconds)
```

5.4 تمارين

1 [م] المطلوب منكم أن تتأملوا في البرنامج التالي. ماذا تتوقعون أن تكون النتيجة من تنفيذ هذا البرنامج؟ المطلوب منكم تنفيذ هذا البرنامج و التعليق عن نتيجته.

```
double a = 4.940656458412465;
double b = 5.0e-16;
System.out.println((a+b)+b);
System.out.println(a+(b+b));
```

2 [م] المطلوب منكم أن تتأملوا في البرنامج التالي. ماذا تتوقعون أن تكون النتيجة من تنفيذ هذا البرنامج؟ المطلوب منكم تنفيذ هذا البرنامج و التعليق عن نتيجته.

```
double a = 4.940656458412465;
double b = 5.0e-17;
System.out.println(a);
System.out.println(a+b);
```

3 [م] قدموا قائمة في قواعد في الرياضيات لا تصح في لغة جافا، نظرا لإمكانية الفيضان و الذوبان و تضاعف التقريب، و غير ذلك.

4 [م] المطلوب منكم تأليف قواعد نحوية لعبارات الأعداد الكاملة على منوال القواعد التي كتبناها في الجزء الثالث لعبارات الأعداد الحقيقية؛ ثم المطلوب منكم تحليل العبارة التالية حسب هذه القواعد ببناء شجرة العبارة التابعة لها.

```
Math.log(Math.abs(x-2.0)) + 2.1*y*y
```

5 [م] المطلوب منكم تأليف قواعد نحوية للعبارات المنطقية على منوال القواعد التي كتبناها في الجزء الثالث لعبارات الأعداد الحقيقية؛ ثم المطلوب منكم تحليل العبارة التالية حسب هذه القواعد ببناء شجرة العبارة التابعة لها.

```
(!p && !q) | (q && !p) && q
```

6 [س] المطلوب منكم بناء شجرة العبارة للعبارة العددية التالية:

```
((a+b) % (5) ) * ((-b) + a) / 15
```

7 [م] المطلوب منكم تغيير القواعد النحوية التي قدمناها في الجزء الثالث ليصبح التجمع على اليمين عوضا من أن يكون على اليسار.

8 [م] المطلوب منكم تغيير القواعد النحوية التي قدمناها في الجزء الثالث لتنقلب أولوية العمليات، حيث يقع تنفيذ عمليات الجمع و الطرح قبل تنفيذ عمليات الضرب و القسمة و الباقي.

الفصل الخامس

تعليمات التوريد و الإصدار

كل البرامج التي نكتبها لا جدوى لها لو لا تتبادل بيانات مع محيطها، سواء كان هذا المحيط مستخدمون أو قواعد بيانات أو آلات تحليل أو آلات تحكم أو غيرها. في هذا الفصل نهتم بتعليمات التوريد و الإصدار، التي تسمح لبرامج جافا أن تتبادل بيانات مع محيطها. ننظر على التوالي في عمليات التوريد ثم الإصدار؛ و ضمن تعليمات الإصدار ننظر على التوالي في تعليمات إصدار النصوص ثم إصدار الرسوم.

1.5 تعليمات التوريد

نفترض برنامجا يتعامل مع المستخدم بصفة تفاعلية حيث يطلب من المستخدم أن يقدم بيانات يعالجها فوريا؛ تقدم لنا لغة جافا وسائل تمكننا من إلتقاط بيانات من المستخدم. بصفة أدق، توفر لغة جافا مفهوم الناسخ، الذي يمثل منبع بيانات يمكننا إلتقاط بياناته عند الحاجة. يقع التصريح بالناسخ و إستعماله حسب التراتيب التالية:

- التصريح

```
Scanner inputWindow = new Scanner(System.in)
```

يقدم هذا التصريح ناسخ **inputWindow** كمصدر بيانات و يربطه بوسيط **System.in** الذي يمثل البيانات الصادرة عن لوحة المفاتيح. يمكن ربط الناسخ بوسائط أخرى، مثل حاملة بيانات (**File**) أو سلسلة حروف (**String**).

- الإستعمال. بعد أن نصرح بناسخ، يمكن لنا أن نستعمله لقراءة عناصر فردية من هذا الناسخ. تتغير عملية القراءة حسب نوع البيانات التي نريد قراءتها.

  - سلسلة. إذا أردنا أن نقرأ سلسلة، فنكتب إسم الناسخ تتبعه وظيفة **next()**. مثلا، إذا نكتب

```
inputWindow.next()
```

نستخرج السلسلة الموالية في الناسخ، إلى غاية الفاصل المقبل في الناسخ، سواء كان الفاصل فضاء أبيض أو نهاية سطر أو نهاية الناسخ، إلخ. إذا كان الناسخ يحمل النص التالي، مثلا:

```
Hello world.  This is a test of
A Java scanner.
```

فتسفر مناداة وظيفة **next()** على إستخراج السلاسل التالية، على التوالي:

```
Hello
world.
This
is
a
test
of
A
Java
Scanner.
```

  - سطر. إذا أردنا أن نقرأ سطرا كاملا من ناسخ، فنكتب إسم الناسخ تتبعه وظيفة **nextLine()**. مثلا، إذا نكتب

```
inputWindow.nextLine()
```

نستخرج السلسلة الموالية في الناسخ، إلى غاية نهاية السطر الحالي. إذا كان الناسخ يحمل النص التالي، مثلا:

```
Hello world.  This is a test of
A Java scanner.
```

فتسفر مناداة وظيفة **next()** على إستخراج السلاسل التالية، على التوالي:

```
Hello world.  This is a test of
A Java scanner.
```

  - عدد كامل. **nextInt()**
  - عدد كامل قصير. **nextShort()**
  - عدد كامل طويل. **nextLong()**

- عدد حقيقي مثنى. **.nextDouble()**
- عدد حقيقي عائم. **.nextFloat()**
- قيمة منطقية. **.nextBoolean()**

- كما يمكننا أن نختبر إن وصلنا لنهاية الناسخ أم لا، بواسطة الوظيفة التالية ذات القيمة المنطقية: **.hasNext()**

1.1.5 ناسخة لوحة المفاتيح

توضيحا لمفهوم الناسخ و تطبيقاته، نقدم مثال برنامج بسيط يسأل المستخدم عن إسمه و تاريخ ميلاده ثم يستعمل هذه البيانات لتأليف رسالة ودّية.

```
package io;                                                              //  1.
                                                                         //  2.
/**                                                                      //  3.
 *                                                                       //  4.
 * @author Ali Mili and Ahmed Ferchichi                                  //  5.
 */                                                                      //  6.
                                                                         //  7.
import java.util.Scanner;                                                //  8.
                                                                         //  9.
public class Main {                                                      // 10.
                                                                         // 11.
    /**                                                                  // 12.
     * @param args the command line arguments                           // 13.
     */                                                                 // 14.
                                                                         // 15.
    public static void main(String[] args) {                            // 16.
        // TODO code application logic here                              // 17.
                                                                         // 18.
        Scanner inputWindow = new Scanner(System.in);                    // 19.
        System.out.println("What is your name and Birth Date?");         // 20.
        String name = new String();                                      // 21.
        String month = new String();                                     // 22.
        int bday;  int byear;                                            // 23.
        name = inputWindow.next();                                       // 24.
        month = inputWindow.next();                                      // 25.
        bday = inputWindow.nextInt();                                    // 26.
        byear = inputWindow.nextInt();                                   // 27.
        int age = 2010 - byear;                                          // 28.
                                                                         // 29.
        System.out.println("Hello "+name+".  On "+month+" "+bday+        // 30.
        " 2010 you will be ");                                           // 31.
        System.out.println(age+" years old.  I will remember to wish you");
        System.out.println("Happy "+age+"th Birthday.");                 // 33.
    }                                                                    // 34.
}                                                                        // 35.
```

نقدم بعض التعليقات لنوضح هذا البرنامج:

- السطر 8: نعلن عن رغبتنا في إستعمال وظيفة الناسخ في هذا البرنامج.
- السطر 19: نكون ناسخا إسمه **inputWindow** و نربطه بلوحة المفاتيح.
- السطر 20: في هذا السطر، نطلب من المستخدم أن يقدم إسمه و تاريخ ولادته. بالرغم من أننا لم نوضح هذه التفاصيل في هذا الإعلان، فننتظر أن يقدم المستخدم إسمه (كسلسلة) ثم شهر ولادته (كسلسلة) ثم يوم ولادته (في الشهر) ثم سنة الملادة. تفصل فضاءات بيضاء بين كل هذه العناصر.

- الأسطر 21 إلى 23: نصرح بالأربعة متغيرات التي نحتاج لها لخزن الإسم (سلسلة) و شهر الولادة (سلسلة) و يوم الولادة (عدد كامل) و سنة الولادة (عدد كامل).
- الأسطر 24 إلى 27: نقرأ البيانات التي طلبناها من المستخدم، مستعملين الوظائف المناسبة بالنسبة لكل عنصر.
- السطر 28: نحتسب العمر الذي يصله المستخدم سنة 2010 و نخزنه في متغيرة خاصة بهذا الغرض.
- الأسطر 30 إلى 33: نكتب رسالة ودية للمستخدم حسب البيانات التي قدمها للبرنامج.

نقدم أسفله نموذجا من تنفيذ هذا البرنامج:

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\Ali Mili\My
Documents\NetBeansProjects\io\build\classes
compile:
run:
What is your name and Birth Date?
beya february 24 2003
Hello beya.  On february 24 2010 you will be
7 years old.  I will remember to wish you
Happy 7th Birthday.
BUILD SUCCESSFUL (total time: 30 seconds)
```

رغبة في التوضيح، وضعنا سطرا تحت النص الذي يكتبه المستخدم. إختبرنا إن كانت جافا تعالج النصوص العربية كما ينبغي، فغيرنا نص البرنامج لتكون الرسائل مكتوبة بالعربية و أدخلنا اليانات بالعربية، فأصبح البرنامج على النحو التالي:

```
  System.out.println("ما هو إسمك و تاريخ ولادتك؟");                    // 20.

  System.out.println("أهلا "+name+". في "+month+" "+bday+" 2010 سيكون عمرك ");
  System.out.println(age+"سنين.  سأتذكر لأتمنى لك ");                    //
32.
  System.out.println("عيد ميلاد سعيد "+age);                            // 33.
```

فتكون نتيجة تنفيذ البرنامج كما يلي:

```
ما هو إسمك و تاريخ ولادتك؟
بيتة فيفرى 24 2003
أهلا بيتة. في فيفري 24 2010 سيكون عمرك
7سنين.  سأتذكر لأتمنى لك
عيد ميلاد سعيد 7.
```

2.1.5 ناسخة سلسلة حروف

كما ذكرنا أعلاه، يمكن إستعمال الناسخ لقراءة، لا فقط البيانات الواردة من لوحة المفاتيح، بل أيضا البيانات الموجودة في سلسلة حروف أو في حاملة بيانات. سندرس معالجة حاملات البيانات في فصل لاحق؛ في هذا الفصل، ننظر بإختصار في مثال توضيحي عن إستعمال ناسخ لقراءة سلسلة من الحروف. فنعتبر البرنامج التالي:

```
        String dataString = new String();                                  //  1.
        dataString = "2008  أوت  28  أي  1429  شعبان  27  الخميس";        //
2.
        Scanner inputWindow = new Scanner(dataString);                     //  3.
```

```
                                                                    //  4.
        String day = new String();                                  //  5.
        String islamicMonth = new String();                         //  6.
        String gregorianMonth = new String();                       //  7.
        String preposition = new String();                          //  8.
                                                                    //  9.
        int islamicDom, gregorianDom;                               // 10.
        int islamicYear, gregorianYear;                             // 11.
                                                                    // 12.
        day = inputWindow.next();                                   // 13.
                                                                    // 14.
        islamicDom = inputWindow.nextInt();                         // 15.
        islamicMonth = inputWindow.next();                          // 16.
        islamicYear = inputWindow.nextInt();                        // 17.
                                                                    // 18.
        preposition = inputWindow.next();                           // 19.
                                                                    // 20.
        gregorianDom = inputWindow.nextInt();                       // 21.
        gregorianMonth = inputWindow.next();                        // 22.
        gregorianYear = inputWindow.nextInt();                      // 23.
                                                                    // 24.
        System.out.println("      الـهجري      الـميلادي");       //
25.
        System.out.println("السنة:  "+islamicYear+"     "+gregorianYear);// 26.
        System.out.println("الشهر:  "+islamicMonth+"   "+gregorianMonth);// 27.
        System.out.println("اليوم:   "+islamicDom+"      "+gregorianDom);// 28.
```

نقدم بعض التعليقات لنوضح هذا البرنامج:

- الأسطر 1 إلى 3: نصرح بسلسلة **dataString**، نخزن فيها سلسلة (الـخميس 27 شعبـان 1429 أي 28 أوت 2008) و نربطها بناسخ **inputWindow** حيث سنقرأها بواسطته.
- الأسطر 5 إلى 11: نصرح بالمتغيرات التي نخزن فيها عناصر السلسلة.
- الأسطر 13 إلى 17: نقرأ عناصر التاريخ الهجري فردا فردا، حسب نوع بيانات كل عنصر.
- السطر 19: نقرأ الفاصل بين التاريخين (مع أننا لا نستعمله).
- الأسطر 21 إلى 23: نقرأ عناصر التاريخ الميلادي فردا فردا، حسب نوع بيانات كل عنصر.
- الأسطر 25 إلى 28: نقدم مائدة تبين موافقة التاريخين الهجري و الميلادي.

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
الـهجري      الـميلادي
السنة: 1429  2008
الشهر: شعبـان  أوت
اليوم: 27  28
BUILD SUCCESSFUL (total time: 0 seconds)
```

في هذا البرنامج، قرأنا السنين و الأيام كأعداد كاملة عوضا عن سلاسل. ما ينجر عن هذا الإختيار؟ ينجر أننا إذا أردنا أن نقوم بعمليات عددية على متغيرات السنة أو اليوم، فيكون ذلك ممكنا على المتغيرات العددية لكنه مستحيلا على المتغيرات من نوع السلاسل.

- مثلا، إذا أردنا أن نحتسب اليوم الموالي للتاريخ الجاري، يكفي أن نظيف 1 لعدد اليوم الحالي (مع بعض الصعوبة إذا كان اليوم الحالي آخر يوم في الشهر).
- إذا أردنا أن نحتسب السنة الموالية للتاريخ الجاري، يكفي أن نظيف 1 لعدد السنة الحالية.

أنظروا للتمرين رقم 2.

2.5 تعليمات الإصدار

1.2.5 طبع السلاسل و الأسطر

بصفة عامة، توفر لغة جافا عناوين مختلفة لإصدار البيانات، اهمها عنوان الشاشة، التي تعينها لغة جافا بإسم `System.out`. توفر لغة جافا تعليمتين بسيطتين لطبع النصوص، و هما `println()` و `print()`. كلاهما يطبعان النص الذي نوفره بين قوسين، مع أن `println()` تختم هذا النص بعودة إلى السطر الموالي. على باب المثال، نقدم فيما يلي برنامجا بسيطا توضيحيا فنعلق على تنفيذه.

```
System.out.print("هذا جزء سطر");                    //  1.
System.out.println(" و هذه بقية السطر");          //  2.
System.out.println("أما هذا فهو سطر كامل.");      //  3.
System.out.println("");                            //  4.
System.out.println("بعد السطر الفارغ");            //  5.
```

يكتب السطر رقم 1 جملة "هذا جزء سطر" و يبقى مستعدا للمواصلة الكتابة على نفس السطر. أما السطر رقم 2 فيكتب في مساق النص السابق أي يكتب "و هذه بقية السطر" ثم يضع نهاية السطر الحالي. بحيث أن تعليمة الطبع الموجودة في السطر رقم 3 تطبع نصها في السطر الثاني من الشاشة، ثم أنها تختم هذا السطر ما إن طبعت نص "أما هذا فهم سطر كامل". أما التعليمة الموجودة في السطر رقم 4 فهي لا تكتب شيئا بل تكتفي بالإنتقال للسطر الموالي؛ فيأتي السطر رقم 5 ليكتب النص "بعد السطر الفارغ" و يختم السطر الحالي. يسفر تنفيذ هذا البرنامج على النتيجة التالية، و هي ما ننتظر:

```
هذا جزء سطر و هذه بقية السطر
أما هذا فهو سطر كامل.

بعد السطر الفارغ
```

تمكننا لغة جافا من كتابة نصوص مركبة بواسطة عملية الربط بين السلاسل، التي نمثلها بعملية +؛ تضبط لغة جافا تأويل هذا الرمز (كعملية ربط بين سلاسل أو كعملية جمع بين أعداد) حسب نوع بيانات أطراف العملية. إذا كان أحد الطرفين سلسلة، فتأول لغة جافا هذا الرمز (+) كعملية ربط بين سلاسل؛ إذا كان الطرف الآخر ذات نوع بيانات دون السلسة، فتحوله لغة جافا لسلسلة قبل أن تقوم بعملية الربط. أما إذا كان الطرفان من أنواع الأعداد، فتأول لغة جافا هذا الرمز كعملية جمع، فتقوم بالجمع ثم تحول النتيجة من نوع الأعداد إلى نوع السلاسل ثم تطبع السلسلة الناتجة عن هذا التحويل. على باب المثال، ننظر في المثال التالي:

```
int i;  int j;                                                  //  1.
i = 213;  j = 315;                                              //  2.
System.out.println("separated integers:" + i + " " + j);       //  3.
System.out.println("how about these integers:" + i + j);       //  4.
System.out.println(i + j + "added integers:");                  //  5.
System.out.println(i + j);                                      //  6.
System.out.println("added integers:" + (i + j));               //  7.
```

في السطرين الأول و الثاني نصرح بالمتغيرتين من نوع الأعداد الكاملة و نسند لهما قيمتين من نوعهما.

- في السطر الثالث نطبع على التوالي سلسلة ثم عددا ثم سلسلة ثم عددا. في هذه العبارة، يوجد كل رمز "+" بجانب سلسلة، فيقع تأويل الرمز كعملية ربط عوضا عن عملية جمع عددي. فيطبع هذا السطر العددين منفصلين.

- في السطر الرابع، يوجد العددان بجانب بعضهما، لكن مصرف جافا يطبق رمز "+" الموجود على اليسار قبل أن يطبق رمز "+" الموجود على اليمين. فيجد أن رمز "+" الموجود على اليسار يفصل بين عدد و سلسلة، فيؤول هذا الرمز كعملية ربط بين سلسلتين، حيث يحول عدد إلى سلسلة و يربط بين السلسلتين ليجد سلسلة. فيجد أن رمز "+" الموجود على اليمين هو أيضا يفصل بين سلسلة و عدد، فيؤوله كعملية ربط عوضا عن عملية جمع. فيكون العددان منفصلان في هذه الحال أيضا.
- في السطر الخامس وضعنا العددين قبل سلسلة الحروف (إذا قرأناها من اليسار إلى اليمين)، بحيث عندما يطبق المصرف رمز "+" الموجود على اليسار يجده يفصل بين عددين فيؤوله كعملية جمع عوضا عن عملية ربط.
- في السطر السادس نضع رمز "+" بين عددين فلا تُطرح مسألة تأويله حيث يقوم البرنامج بجمع العددين و طبعهما و الرجوع إلى السطر.
- أما في السطر السابع فنكتب نفس التعلمية كالسطر الرابع، لكن نضع قوسين حول عملية الجمع فنفرض على البرنامج أن يطبق رمز "+" داخل القوسين قبل أن يطبق رمز "+" خارج القوسين. علما أن رمز "+" داخل القوسين يفصل بين عددين، يقع تأويل هذا الرمز كعملية جمع.

يسفر هذا البرنامج على النتيجة التالية:

```
separated integers:213 315
how about these integers:213315
528added integers:
528
added integers:528
```

و هي ما كنا نتوقع. ننظر بإيجاز في مثال يحتوي على عمليات ضرب بالإضافة لعمليات الجمع. يتمثل تأثير عملية الضرب في أنها تتنفذ قبل عملية "+"، مهما كان تأويلها.

```
int i;  int j;                                                  //  1.
i = 213;  j = 315;                                              //  2.
System.out.println("ما هي النتيجة:" + 2*i  + " " + j);         //  3.
System.out.println("ما هي النتيجة:" + i * j);                  //  4.
System.out.println("ما هي النتيجة:" + i+j * j);                //  5.
System.out.println(i+j * j + "ما هي النتيجة:");                //  6.
```

في السطرين الأول و الثاني نصرح بالمتغيرات و نسندها قيمات عددية. في السطر الثالث تتنفذ عملية الضرب قبل عمليات "+"، و يقع تأويل ال"+" كعملية ربط بين سلاسل. في السطر الرابع تتنفذ عملية الضرب قبل عملية "+"، و يقع تأويل ال"+" كعملية ربط بين سلسلتين. في السطر الخامس يقع تأويل كلا الرمزان "+" كعمليات ربط فنجد سلسلة تتمثل في ربط سلسلة **i** (و هي **213**) بسلسلة تمثل نتيجة **j * j** (و هي **99225**) ، فنجد **21399225**. في السطر السادس نؤول ال"+" الأول كعملية جمع عددي و ال"+" الثاني كعملية ربط، بحيث يقع تقييم الصيغة العددية **i+j * j** قبل ربطها بالنص الذي يتبعها، و تكون قيمتها: **99438.** يسفر تنفيذ هذا البرنامج على النتيجة التالية.

```
ما هي النتيجة:426 315
ما هي النتيجة:67095
ما هي النتيجة:21399225
99438ما هي النتيجة:
```

2.2.5 الطبع بالأنماط

عندما نكتب أعدادا في الأمثلة السابقة، تكتفي جافا بكتابتها في مساق النص، بدون أي إجراءات خاصة. نفترض أننا نريد أن نطبع قائمة بضائع، مع أثمانها، فنكتب مثلا:

```
double seprice, dmprice, jpprice;                              //  1.
seprice = 49.75;  dmprice = 89; jpprice = 53.525;              //  2.
System.out.println("Software Engineering:  $" + seprice );     //  3.
System.out.println("Discrete Maths:  $" + dmprice );           //  4.
System.out.println("Java Programming:  $" +  jpprice);         //  5.
```

تكون نتيجة هذا البرنامج على النحو التالي:

```
Software Engineering:  $49.75
Discrete Maths:  $89.0
Java Programming:  $53.525
```

إن هذه النتيجة غير مرضية، لعدد من الأسباب: أولا، الأثمان ليست على عمود واحد؛ ثانيا، ليست الأعداد في نفس النمط (عددين بعد الصفر، ثم عدد واحد بعد الصفر، ثم ثلاثة أعداد بعد الصفر). نعتبر مثالا أكثر تعقيدا، حيث نطبع قائمة كتب مع أسعارها و كميات (في نطاق فاتورة مثلا) ثم سعر جملي، فنكتب ما يلي:

```
double seprice, dmprice, jpprice;
seprice = 49.75;  dmprice = 89; jpprice = 53.525;
int seqty, dmqty, jpqty;
seqty = 6; dmqty = 7; jpqty = 5;
System.out.println
("Software Engineering:  $"+seprice+" qty "+seqty+" tot $"+seqty*seprice );
System.out.println
("Discrete Maths:  $"+dmprice+" qty "+dmqty+" tot $"+dmqty*dmprice );
System.out.println
("Java Programming:  $"+jpprice+" qty "+jpqty+" tot $"+jpqty*jpprice);
```

مما يسفر على النتيجة التالية

```
Software Engineering:  $49.75 qty 6 tot $298.5
Discrete Maths:  $89.0 qty 7 tot $623.0
Java Programming:  $53.525 qty 5 tot $267.625
```

فتصعب علينا قراءة هذه البيانات لأنها ليست مصفوفة و ليست في نمط موحد. توفر لنا لغة جافا أنماط بيانات تمكننا من طبع البيانات في أشكال قابلة للقراءة، ننظر فيها في هذا الجزء. أول ما ننظر فيه هي أنماط يضبطها المبرمج يدويا في نطاق سلسلة رمزية تضبط حجما و شكلا بالنسبة لكل بيان، حسب نوعه. نقدم في المائدة التالية أهم البيانات مع كيفية ضبط شكل لطبعها.

نوع البيانات	النمط	التأويل	مثال
قيمة منطقية	%Qb	Q: يمثل عدد البقاع المخصصة لهذه القيمة	%9b
عدد كامل قصير	%Qd	Q: يمثل عدد البقاع المخصصة لهذا العدد	%9d
عدد كامل	%Qd	Q: يمثل عدد البقاع المخصصة لهذا العدد	%9d
عدد كامل طويل	%Qd	Q: يمثل عدد البقاع المخصصة لهذا العدد	%9d
عدد حقيقي عائم	%Q.Rf	Q: يمثل عدد البقاع المخصصة لهذا العدد R: عدد البقاع المخصصة للجزء الكسري	%9.3f
عدد حقيقي مثنى	%Q.Re	Q: يمثل عدد البقاع المخصصة لهذا العدد R: عدد البقاع المخصصة للجزء الكسري	%9.3e

توضيحا لهذه الأنماط، نقدم في ما يلي برنامجا قصيرا يجربها على التوالي، و نبين نتيجته.

```
package form;

import java.io.*;
import java.util.*;

public class Main
{
  public static void main (String arg[]) {

    Formatter formatter  =
      new Formatter ((OutputStream)System.out);

    boolean b =  false;
    short   s =  349;
    int     i =  32667;
    long    l =  99765678;
    float   f =  879.5f;
    double  d = -5.978e-12;
    formatter.format ("boolean = %9b %n",   b);
    formatter.format ("short   = %9d %n",   s);
    formatter.format ("int     = %9d %n",   i);
    formatter.format ("long    = %9d %n",   l);
    formatter.format ("float   = %9.3f %n", f);
    formatter.format ("double  = %9.2e %n", d);

    formatter.flush ();
    formatter.close ();


  } // main

} //
```

فتكون نتيجة هذا البرنامج كما يلي:

```
boolean =     false
short   =       349
int     =     32667
long    =  99765678
float   =   879.500
double  = -5.98e-12
```

في كل هذه الحالات، تخصص جافا 9 فضاءات لكتابة هذه القيمات (حسب طلبنا) فتكون الأعداد مصفوفة على اليمين. نلاحظ أن العدد المثنى (الأخير في القائمة) وقع تقريبه لقيمة 5.98 (من قيمته الأصلية 5.978) لأن النمط إشترط علينا أن نخصص سواء عددين على يمين النقطة العشرية. تطبيقا لهذه الأنماط، نقدم فيما يلي برنامجا يقيم فاتورة تتمثل في قائمة من البضائع مع سعرها الفردي و عددها و سعرها الجملي، إلخ. فنكتب:

```
package form;                                                    //  1.
                                                                 //  2.
import java.io.*;                                                //  3.
import java.util.*;                                              //  4.
                                                                 //  5.
public class Main                                                //  6.
{                                                                //  7.
  public static void main (String arg[]) {                       //  8.
```

```
                                                                    //  9.
    // Send formatted output to the System.out stream.              // 10.
                                                                    // 11.
    Formatter formatter  =                                          // 12.
      new Formatter ((OutputStream)System.out);                     // 13.
                                                                    // 14.
    double seprice, dmprice, jpprice;                               // 15.
    seprice = 49.75;  dmprice = 89; jpprice = 53.525;               // 16.
    int seqty, dmqty, jpqty, totqty;                                // 17.
    seqty = 6; dmqty = 7; jpqty = 5;                                // 18.
    double total, linetotal;                                        // 19.
    total = 0;  totqty = 0;                                         // 20.
    String header = new String                                      // 21.
    ("Title                  Unit price  Quantity  Total %n");      // 22.
    String layout = new String ("%s \t %9.2f  %5d  %9.2f  %n");     // 23.
    String footer = new String ("%s \t %17d %9.2f %n");             // 24.
                                                                    // 25.
    formatter.format (header);                                      // 26.
    linetotal = seprice*seqty;                                      // 27.
    total = total + linetotal; totqty = totqty + seqty;             // 28.
    formatter.format                                                // 29.
    (layout, "Software Engineering", seprice, seqty, linetotal);    // 30.
    linetotal = dmprice*dmqty;                                      // 31.
    total = total + linetotal; totqty = totqty + dmqty;             // 32.
    formatter.format                                                // 33.
    (layout, "Discrete Mathematics", dmprice, dmqty, linetotal);    // 34.
    linetotal = jpprice*jpqty;                                      // 35.
    total = total + linetotal; totqty = totqty + jpqty;             // 36.
    formatter.format(layout, "Java Programming", jpprice, jpqty, linetotal);
    formatter.format (footer, "Invoice total: ", totqty, total);   // 38.
    formatter.flush ();                                             // 39.
    formatter.close ();                                             // 40.
   } // main                                                        // 41.
} // class                                                         // 42.
```

نعلق على هذا البرنامج في ما يلي:

- السطرين 3 و 4: نصرح بباقتين نحتاج لها في هذا البرامج لتعيين الأنماط.
- السطرين 12 و 13: يمكننا مؤلف الأنماط (**Formatter**) من طبع بيانات حسب أنماط معينة. نصرح في هذين السطرين بمؤلف أنماط نلقي عليه إسم "**formatter**".
- الأسطر 15 إلى 18: نصرح بمتغيرات تخزن أسعار الكتب و كمياتها.
- السطرين 19 و 20: نسند للمتغيرات قيمات إفتتاحية.
- الأسطر 21 إلى 24: نعين الأنماط التي نستعملها في طبع الوثيقة المطلوبة، بما فيها نمط السطر الأول، نمط السطر الأخير، و أنماط السطر الجاري.
- السطر 26: نطبع السطر الأول للوثيقة.
- الأسطر 27 إلى 30: نحتسب البيانات الخاصة بالكتاب الأول ثم نطبعها حسب النمط الجاري. نلاحظ أن وظيفة **format** تنتظر أن نمدها بالنمط (**layout**) ثم بالبيانات التي تندرج في هذا النمط (العنوان، السعر الفردي، الكمية، السعر الجملي).
- الأسطر 31 إلى 34: نفس العملية بالنسبة للكتاب الثاني.
- الأسطر 35 إلى 38: نفس العملية بالنسبة للكتاب الثالث.
- السطرين 39 و 40: غلق مؤلف الأنماط و ختام عملياته.

تكون نتيجة هذا البرنامج على النحو التالي:

```
Title                 Unit price  Quantity  Total
Software Engineering       49.75         6    298.50
Discrete Mathematics       89.00         7    623.00
Java Programming           53.53         5    267.63
Invoice total:                          18   1189.13
```

3.2.5 الطبع بالأنماط الآلية

توفر لغة جافا عددا من الأنماط الآلية لطبع العديد ن أنواع البيانات، بما فيها الأعداد و التواريخ و الأوقات و القيمات المالية و الإعلانات و غيرها. ننظر في بعض انواع البيانات و في وظيفة الأنماط بالنسبة لها.

- الأعداد الحقيقية. نعتبر العدد الحقيقي التالي: 65747653432.679. إذا طلبنا من جافا أن يطبع هذا العدد، بدون نمط، فسيطبعه كما هو، مما يجعل قراءته صعبة. أما إذا أردنا أن نسند نمطا لهذا العدد، فنجد أن هذا النمط يتغير من بلد إلى آخر، حسب المائدة التالية:

البلد	النمط
فرنسا	65 747 653 432,679
ألمانيا	65.747.653.432,679
ال و م أ	65,747,653,432.679

- التواريخ. كما نعلم أيضا، فإن تمثيل التواريخ يختلف كثيرا من بلد إلى آخر، أو من لغة إلى أخرى. ففي اللغة الفرنسية نضع اليوم في الشهر قبل الشهر، بينما في الأنجليزية غالبا ما نضع الشهر قبل اليوم. ثم في الأنجليزية غالبا ما نشير لليوم في الشهر لا كعدد بل كرتبة عددية (لا نقول إثنين سبتمبر بل نقول الثاني من سبتمبر).

- القيمات المالية. يتغير تمثيل القيمات المالية، لا فقط من حيث رمز العملة (الدينار، الريال، الدرهم، الجنيه، الدولار، اليورو، الين، إلخ) بل أيضا من حيث وضع هذا الرمز بالنسبة للعدد الذي يمثل القيمة. بالنسبة للدولار مثلا، نضع رمز الدولار على يسار العدد، بينما في العديد من العملات الأخرى يوضع على اليمين.

- الأوقات. يختلف تمثيل الأوقات من بلد إلى آخر إذ أن البعض يمثلون الساعات بين الصفر و ال23 (إلى غاية 24 ساعة) بينما الآخرون يمثلون الساعلا بين الصفر و 11 (غلى غاية 12 ساعة) مع التفريق بين الساعة قبل الظهر و بعد الظهر.

كيف يقرر مصرف جافا أي نمط يستعمل؟ يتصل المصرف بنظام الإستغلال (نظام التشغيل) ليطلع عن هوية مستعمل الحاسوب، فيألف الأنماط بناء عن هذه المعلومات. توضيحا للأنماط الآلية، نقدم البرنامج التالي:

```
package form;                                                    //  1.
                                                                 //  2.
import java.util.*;                                              //  3.
import java.text.NumberFormat;                                   //  4.
import java.text.DateFormat;                                     //  5.
                                                                 //  6.
public class Main                                                //  7.
{                                                                //  8.
  public static void main (String arg[]) {                       //  9.
                                                                 // 10.
    NumberFormat nf = NumberFormat.getNumberInstance();          // 11.
    NumberFormat cf = NumberFormat.getCurrencyInstance();        // 12.
```

```
    NumberFormat pf = NumberFormat.getPercentInstance();          // 13.
    DateFormat   df = DateFormat.getDateInstance();               // 14.
                                                                  // 15.
    double price = 49.70;                                         // 16.
    double percentage = 0.258;                                    // 17.
    double quantity = 65747653432.679;                            // 18.
    Date today = new Date();                                      // 19.
                                                                  // 20.
    System.out.println(cf.format(price));                         // 21.
    System.out.println(pf.format(percentage));                    // 22.
    System.out.println(df.format(today));                         // 23.
    System.out.println(nf.format(quantity));                      // 24.
                                                                  // 25.
  } // main                                                       // 26.
} // class                                                        // 27.
```

نعلق على هذا البرنامج:

- السطر 4: نصرح أننا نحتاج لأنماط الأعداد.
- السطر 5: نصرح أننا نحتاج لأنماط التواريخ.
- الأسطر 11 إلى 13: نصرح بأنماط عددية بالنسبة للأعداد الحقيقية، و النسبات المائوية و القيمات المالية. تعين هذه التصريحات الأنماط المذكورة حسب هوية الحاسوب.
- السطر 14: نصرح بنمط يضبط شكل التواريخ حسب هوية الحاسوب.
- الأسطر 16 إلى 18: نصرح بمتغيرات من نوع الأعداد الحقيقية، سنستعملها لأغراض مختلفة؛ و نسند لها قيمات إفتتاحية.
- السطر 19: نصرح بمتغيرة من نوع التاريخ و نسند لها تاريخ اليوم (حسب رزنامة الحاسوب).
- الأسطر 21 إلى 24: نطبع المتغيرات المعنية حسب الأنماط المعينة.

نفذنا هذا البرنامج فكانت نتيجته على النحو التالي:

```
49.7 د.ت.
26%
02/09/2008
65,747,653,432.679
```

علما أن الحاسوب الذي نفذنا عليه هذا البرنامج ذات هوية تونسية، فقد طبع القيمة المالية بجساب الدينار التونسي (د.ت.) لكنه طبع التاريخ حسب النمط الفرنسي عوضا عن نمط عربي تونسي، كما طبع العدد في السطر الأخير حسب نمط أمركي عوضا عن نمط أوروبي، حسب الإستعمال الجاري في تونس.

3. عناصر في البرمجة الرسومية في جافا

إن لغة جافا لا تكتفي بإنتاج بيانات في شكل نصوص، بل تنتج أيضا بيانات في شكل رسوم و صور. إن جافا تتسم بوظائف دقيقة و متطورة للغاية لمعالجة الرسوم، تتجاوز نطاق هذا الكتاب؛ فنكتفي في هذا الجزء بعرض توضيحي بسيط لبعض هذه الوضائف. نحتاج في نطاق هذا العرض لباقتين من برامج جافا تختص في إنتاج الرسوم، و هي

```
java.awt.*;
javax.swing.*;
```

و نحتاج لعنصرين إثنين لتركيب الرسم الذي نريد تكوينه، و هما عنصر الإطار (**JFrame**) و عنصر العلامة (**JLabel**). يمكننا عنصر الإطار من رسم إطر على الشاشة، قابل لإستيعاب بيانات في شكل صور و غيرها؛ و يمكننا عنصر العلامة من تركيب مكونات الإطار. نقدم البرنامج التالي كمثال:

و نحتاج لعدد من العناصر لتركيب الرسوم في جافا، نذكر منها بالخصوص:

- عنصر الإطار (**JFrame**). يمثل الإطار نافذة مستطيلة تبرز في الشاشة لتستوعب بيانات و أشكال مختلفة. تحمل هذه النافذة في الركن اليميني الأعلى العلامات التي تعودنا بها في نظام "وندوز"، و هي

حيث تمكننا هذه العلامات من غلق النافذة أو من تصغيرها وقتيا أو تكبيرها لتغطي كامل الشاشة. كما يمكن لنا أن نسند إسما لهذه النافذة فيبرز هذا الإسم في أعلى النافذة.
- عنصر اللوحة (**JPanel**). تمثل اللوحة الفضاء داخل الإطار حيث نضع البيانات التي نريد إبلاغها، فنضبط لون اللوحة و نضبط حتواها حسب تعليمات دقيقة سننظر في عينة منها فيها فيما يلي.
- عنصر العلامة (**JLabel**). تمثل العلامة وحدة نستعملها لتركيب اللوحات، فتتركب من نصوص و صور يمكن ضبط مقاهمها بالنسبة لبعضها بعضا حسب تعليمات دقيقة سننظر في عينة منها فيها فيما يلي.
- عنصر الصورة الرمزية (**ImageIcon**). تمثل الصورة الرمزية صورة نستوردها من الحاسوب فيستعملها البرنامج لإدخالها في علامة، و من هناك في لوحة، و من هناك في إطار.

توضيحا لهذه العناصر، نقدم فيما يلي برنامجا ينتج إطارا بسيطا.

```
package rosoom;                                                    //  1.
                                                                   //  2.
import java.awt.*;                                                 //  3.
import javax.swing.*;                                              //  4.
                                                                   //  5.
public class Main {                                                //  6.
                                                                   //  7.
    public static void main(String[] args) {                      //  8.
        // TODO code application logic here                        //  9.
                                                                   // 10.
        JFrame itaar = new JFrame ("تحية و سلام");                  //
11.
        itaar.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);      // 12.
                                                                   // 13.
        ImageIcon tayrIcon = new ImageIcon("tayr.gif");            // 14.
        ImageIcon nejmaIcon = new ImageIcon("nejma.gif");          // 15.
                                                                   // 16.
        JLabel tayrLabel, nejmaLabel;                              // 17.
                                                                   // 18.
        tayrLabel = new JLabel("تحيات", tayrIcon, SwingConstants.CENTER); //
19.
        nejmaLabel = new JLabel("سلامات",nejmaIcon,SwingConstants.CENTER);// 20.
                                                                   // 21.
        JPanel mainPanel = new JPanel();                           // 22.
                                                                   // 23.
        mainPanel.setBackground(Color.cyan);                       // 24.
        mainPanel.setPreferredSize(new Dimension(300,250));        // 25.
        mainPanel.add(tayrLabel);                                  // 26.
        mainPanel.add(nejmaLabel);                                 // 27.
                                                                   // 28.
        itaar.getContentPane().add(mainPanel);                     // 29.
        itaar.pack();                                              // 30.
        itaar.setVisible(true);                                    // 31.
    }                                                              // 32.
}                                                                  // 33.
```

نعلق على هذا البرنامج في ما يلي:

- الأسطر 3 و 4: يجب أن نستورد هذين الباقتين من البرامج إذا أردنا أن نستعمل وظائف رسومية.
- السطرين 11 و 12: نصرح بالإطار و نلقي عليها إسما يبرز في أعلى الإطار و نعين شرط إغلاق الإطار.
- السطرين 14 و 15: نستورد صورتين موجودتين في حاملات بيانات حيث يوجد البرنامج نفسه. في هذه الحالة

```
C:\Documents and Settings\Ali Mili\My Documents\NetBeansProjects\rosoom
```

- الأسطر 17 إلى 20: نصرح بعلامتين و نكونها من الصور الرمزية و نصوص في شكا سلاسل.
- الأسطر 22 إلى 27: نصرح بلوحة، نضبط لونها و حجمها و نضع فيها العلامتين المكونتين أعلاه.
- الأسطر 29 إلى 31: نضع اللوحة في الإطار المعني و نبرزه على الشاشة.

ننفذ هذا البرنامج فتكون نتيجته على النحو التالي:

يمثل هذا الرسم نموذجا من البيانات الصادرة الرسومية في جافا.

4.5 تمارين

1 [م/ص] المطلوب منكم كتابة برنامج يقرأ سلسلة من الأعداد (بين 0 و 9) و يحولها إلى عدد كامل يوافق هذه السلسلة. مثلا، إذا قرأ سلسلة "2008" يحولها لعدد 2008. يمكن أن نحدد حجم الأعداد (لخمسة حروف مثلا).

2 [م] المطلوب منكم تغيير البرنامج الذي قدمناه في الجزء رقم 2.1.5. بحيث نفصل سلسلة الحروف الواردة إلى سلاسل عوضا عن بعض السلاسل و بعض الأعداد الكاملة. ما ينجر عن قراءة السنين و الأيام كسلاسل عوضا عن أعداد كاملة؟ المطلوب منكم أن تطبعوا تاريخ اليوم الموالي و السنة الموالية للتاريخ الحالي (علما أن السنة الموالية في اليومية الهجرية لا تطابق السنة الموالية في اليومية الميلادية).

3 [س] هل البرنامجان التاليان متساويان؟ لماذا أو لماذا لا؟ البرنامج الأول:

```
System.out.print("أما هذا فهو سطر كامل."); System.out.println("");
```

البرنامج الثاني:

```
System.out.println("أما هذا فهو سطر كامل.");
```

4 [س] المطلوب منكم التكهن بنتيجة البرنامج التالي، مع تفسيره، ثم تنفيذ البرنامج للتأكد.

```
int i;  int j;  int k;
i = 213;  j = 315;  k = 53;
System.out.println("what output:" + i + " " + j*k);
System.out.println("what output:" + i * j + k);
System.out.println(k + "what output:" + i+j * j);
System.out.println(i+j * j + k + "what output:");
```

5 [س] المطلوب منكم كتابة برنامج يقرأ تاريخا من الشاشة يتمثل في يوم في الشهر (بين 1 و 31) و شهر (بين 1 و 12) و سنة (كعدد كامل قصير)، ثم كتابه هذا التاريخ حسب النمط الفرنسي و حسب النمط الأنجليزي.

6 [س] المطلوب منكم كتابة برنامج يقرأ عددا كاملا بين 0 و 1439 (بحساب الدقائق في يوم) و يطبع الساعة المعنية حسب النمطين الجاري بهما العمل.

7 [م] تأملوا في البرنامج الذي قدمناه في الجزء 3.2.5. زوروا لوحة التحكم في حاسوبكم و غيروا التفاصيل الخاصة بهوية الحاسوب، ثم تأملوا كيف تؤثر هذه التفاصيل على تصرف البرنامج.

8 [م] إستعملوا الوظائف الرسومية في جافا لتطبعوا بطاقة تهاني من تصميمكم.

5.5 مصادر و مراجع

إن إمكانيات جافا من حيث توريد و إصدار البيانات يتجاوز ما يوجد في هذا الفصل بكثير. الرجاء من القارئ المهتم أن يرجع إلى موقع جافا لدى شركة "سُن".

الفصل السادس

العبارات المنطقية و تعليمات الخيار

1.6 العبارات المنطقية

1.1.6 العبارات المنطقية الفردية

تشمل لغة جافا عددا من علاقات التفاوت و المساواة نستعملها لتمثيل العبارات المنطقية الفردية، نقدمها في المائدة التالية:

النوع	الرمز	التأويل
المساواة	==	العبارتان على يمين و يسار الرمز متساويتان
اللامساواة	=!	العبارتان على يمين و يسار الرمز متساويتان
التفاوت الشديد	<	العبارة على اليسار أصغر من العبارة على اليمين
	>	العبارة على اليسار أكبر من العبارة على اليمين
التفاوت اللين	<=	العبارة على اليسار أصغر من أو تساوي العبارة على اليمين
	>=	العبارة على اليسار أكبر من أو تساوي العبارة على اليمين

إن تأويل عمليات المقارنة يختلف حسب نوع البيانات التابع للعبارتين المعنيتين؛ نعتبر، قصد التبسيط، أن العبارتين لهما نفس النوع (و لو عن طريق تحويل الأنواع، الذي ناقشناه في الفصل الرابع). فنتأمل في مختلف أنواع البيانات على التوالي.

القيمات المنطقية. تنطبق عمليات المساواة على القيمات المنطقية (صحيح، غالط)، فتعطي النتيجة البديهية، أي أن كل العمليات التالية تؤدي إلى قيمة صحيح و كل العمليات الأخرى تؤدي إلى قيمة غالط.

```
true==true, false==false, true!=false, false!=true.
```

يجدر بالذكر أن نقارنة متغيرة بقيمة منطقية يمكن عادة تبسيطه بعبارة لا تشمل القيمة المنقية بتاتا، حسب ما نكتب في ما يلي (حيث تمثل **b** متغيرة منطقية) :

```
(b==true) = (b!=false) = b.
```

من ناحية أخرى، نكتب

```
(b==false) = (b!=true) = !b.
```

الأعداد الحقيقية. نظرا لأن ذاكرة الحاسوب محدودة، فإنه غير قادر على تمثيل كل الأعداد الحقيقية. تبرز حدود الذاكرة في ظاهرتين إثنتين:

- تحديد الإتساع. يمكن أن نجمع عددين قابلين للتمثيل، فنجد عددا غير قابل للتمثيل. توضيحا لهذه الظاهرة، نكتب البرنامج الوجيز التالي:

```
double x, y, z;
x = 1.25e308; y = 8.9e307;
System.out.println(x);
System.out.println(y);
z = x+y;
System.out.println(z);
```

إذا نفذنا هذا البرنامج، تكون نتيجته على النحو التالي:

```
1.25E308
8.9E307
Infinity
```

```
BUILD SUCCESSFUL (total time: 8 seconds)
```

إن نتيجة عملية الجمع هي في الواقع **1.25E308**، لكن هذه القيمة تتجاوز الأعداد القابلة للتمثيل، فتصرح جافا بأنها قيمة لامنتهية.

- تحديد الدقة. يمكن أن نجد عددين حقيقيين قابلين للتمثيل فنجد أن معدلهما غير قابل للتمثيل، كما يبين البرنامج التوضيحي التالي.

```
float v = 1.4023983f;
float w = 1.4023987f;
float u = (v+w)/2;
System.out.println(v+" "+w+" "+u);
```

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
1.4023983 1.4023987 1.4023986
BUILD SUCCESSFUL (total time: 0 seconds)
```

كما نلاحظ، فإن النتيجة غالطة، لأن معدل العددين **1.4023983** و **1.4023987** هو **1.4023985** و ليس **1.4023986** كما نرى من هذا البرنامج. و السبب في ذلك هو أن معدل العددين**v** و **w** ليس قابل للتمثيل في الحاسوب، فيعوضه الحاسوب بأقرب عدد قابل للتمثيل.

بسبب الطبيعة التقريبية للحساب بالأعداد الحقيقية في الحاسوب، نمتنع عادة من إختبار مساواة بين عددين حقيقيين، فنستعمل إختبارا تقريبيا. لا نختبر إن كان العددان متساويان، بل نختبر إن كان الفرق بين العددين ضئيلا بالنسبة لمقياس معين. نتأمل في مثال بسيط:

```
double x = 6.79769313486231570E307;                    //  1.
double y;                                              //  2.
y = Math.log(x);                                       //  3.
y = Math.exp(y);                                       //  4.
System.out.println(x +" "+ y);                         //  5.
double highprecision = 0.0000000000001E307;            //  6.
double medprecision  = 0.000000000001E307;             //  7.
double lowprecision  = 0.00000000001E307;              //  8.
System.out.println(x==y);                              //  9.
System.out.println(Math.abs(x-y)<highprecision);       // 10.
System.out.println(Math.abs(x-y)<medprecision);        // 11.
System.out.println(Math.abs(x-y)<lowprecision);        // 12.
```

نعلق على هذا البرنامج فيما يلي:

- السطرين 1 و 2: نصرح بمتغيرتين من نوع الأعداد الحقيقية المثنات و نسند لإحداها قيمة إفتتاحية.
- السطرين 3 و 4: نسند لمتغيرة **y** نتيجة تطبيق وظيفتين متعاكستين لمتغيرة **x**. مبدئيا، ننتظر أن تصبح قيمتا **x** و **y** متساويتان.
- السطر 5: نطبع قيمتي **x** و **y** لنختبر إن كانتا متساويتان أم لا.
- الأسطر 6، 7، 8: نصرح بثلاثة قيمات نستعملها لغرض إختبار تقريبي إن كانت قيمتا **x** و **y** متساويتان أم لا.

- السطر التاسع: نختبر المساواة المطلقة بين قيمتي **x** و **y**. بالرغم من أن نتيجة هذا الإختبار كانت بالضرورة إجابية، فنتوقع أن تكون نتيجته سلبية بسبب أخطاء التقريب. بالتالي، نلتجئ في الأسطر الموالية لإختبارات تقريبية.
- الأسطر 10، 11، 12: إن لم تكن قيمتا **x** و **y** متساويتان مطلقا، فكم تقترب من بعضها بعضا؟ نستعمل ثلاثة مقاييس متفاوتة الدقة لهذا الغرض.

ننفذ هذا البرنامج، فيسفر على النتيجة التالية:

```
6.797693134862316E307 6.797693134861985E307
false
false
true
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

نلاحض أت العددان يختلفان عن بعضهما، بأربعة أعداد (مسطرة). بالتالي، فإن إختبار المساواة المطلقة يسفر ضرورة على نتيجة سلبية. نتحصل على نفس النتيجة إذا إختبرنا المساواة التقريبية بواسطة مقياس دقيق (**highprecision**). أما إذا إختبرنا المساواة التقريبية مستعملين مقياس متوسط الدقة (**medprecision**) أو منخفض الدقة (**lowprecision**)، فنجد أنهما متساويان.

> بصفة عامة، لا نختبر المساواة بين عددين حقيقيين بصفة مطلقة، بل بصفة تقريبية، مستعملين مقياس يعكس مستوى الدقة المستهدفة.

الحروف. ترتب لغة جافا الحروف و الأعداد و الرموز حسب التشفير الدولي المسمى بيونيكود (unicode). يسند هذا التشفير عددا لكل حرف أو عدد أو رمز، فيرتب الحروف حسب الترتيب العددي لتشفيرها. يجد القارئ تشفير الحروف العربية في الملحق أ و يجد تشفير الحروف اللاتينية في الملحق ب. ضمن القواعد العامة الناتجة عن هذا الترتيب، نذكر ما يلي:

- الحروف اللاتينية مرتبة حسب الترتيب الأبجدي اللاتيني.
- تاحروف العربية مرتبة حسب الترتيب الأبجدي العربي.
- الأعداد العربية (0، 1، 2، ... 9) تسبق الحروف اللاتينية؛
- الحروف اللاتينية تسبق الحروف العربية؛
- ضمن الحروف اللاتينية، الحروف المضخمة تسبق الحروف العادية؛
- الأعداد العربية (0، 1، 2، ... 9) تسبق الأعداد الهندية (٠، ١، ٢، **...** ٩).

كلما يسبق حرف c حرف d، إلا و نجد أن الشرط التالي قيمته إجابية:

```
(c<d) .
```

2.1.6 العبارات المنطقية المركبة

نستعمل العبارات المنطقية الفردية التي درسناها في الجزء السابق لتكوين عبارات منطقية مركبة، بواسطة عمليات منطقية فردية و ثنائية و ثلاثية، نذكر منها بالخصوص:

- عملية النفي، التي نمثلها برمز ! و هي عملية فردية تحول قيمة صحيح إلى غالط و قيمة غالط إلى صحيح.

- عملية العطف، التي نمثلها برمز && و هي عملية ثنائية تنتج قيمة صحيح كلما كان الطرفان في العملية قيمتهما صحيح.
- عملية الخيار، التي نمثلها برمز || و هي عملية ثنائية تنتج قيمة صحيح كلما كان أحد الطرفين في العملية أو كلاهما قيمته صحيح.
- عملية العبارة المشترطة، التي نمثلها على النحو التالي **(c?p:q)** و التي تساوي **p** إذا كانت قيمة الشرط **c** صحيح و تساوي **q** إذا كانت قيمة الشرط **c** غالط.

نقدم في ما يلي أمثلة توضيحية لهذه العمليات:

- ينجح التلميذ إذا لم يكن معدله دون ال12. فنكتب شرط النجاح كما يلي (حيث تمثل **a** معدله):

```
!(a<12).
```

- لا يفوز التلميذ بجائزة من درجة أولى إلا إذا كان معدله في الرياضيات (**m**) أكبر من 14 و معدله في الإعلامية (**c**) أكبر من 16. فنكتب شرط الفوز كما يلي:

```
(m>14) && (c>16).
```

- يفوز التلميذ بجائزة تشجيعية إذا كان معدله في الرياضيات (**m**) أكبر من 12 أو كان معدله في الإعلامية (**c**) أكبر من 14. فنكتب شرط الفوز كما يلي:

```
(m>12) || (c>14).
```

- إذا كان التلميذ في المرحلة الأولى أو الثانية من التعليم العالي (نمثل هذا الشرط ب **ug**)، فلا يرتقي إلا إذا كان معدله (**a**) يتجاوز ال14، و إذا كان في المرحلة الثالثة، فلا يرتقي إلا إذا كان معدله (**a**) يتجاوز ال16. فنكتب شرط الإرتقاء كما يلي:

```
(ug?m>14:m>16).
```

سندرس تطبيقات أخرى للعبارات المشترطة في فصول لاحقة.

2.6 تعليمات الخيار و الشرط

توفر لنا لغة جافا تعليمات تمكننا من تنفيذ مجموعة من العمليات أو مجموعة أخرى حسب نتيجة شرط نركبه كما ذكرنا في الجزء السابق. نفرق أساسا بين نوعين من التعليمات، ندرسها على التوالي في ما يلي.

1.2.6 تعليمة الشرط

نكتب تعليمة الشرط على النحو التالي:

```
if (condition)
   {
    thenclause
   }
```

إذا كان الشرط **(condition)** صحيحا، فتنفذ جافا التعليمات الموجودة بين قوسين، و إن لم يكن صحيحا، فينتهي تنفيذ هذه التعليمة. إذا فشلت جافا في تقييم الشرط، فيفشل تنفيذ التعليمة. مثلا، إذا أردنا أن نحول عددا كاملا إلى قيمته المطلقة، فنتأمل في قيمته الحالية: إن كانت سلبية، فنحولها إلى عكسها، و إن كانت إجابية، لا نغيرها. فنكتب ما يلي:

```
int i;
```

```
… … …
if (i<0) {i=-i;}
```

عندما يتم تنفيذ هذه التعليمة تصبح قيمة `i` القيمة المطلقة لما كانت عليه قبل هذه التعليمة. توضيحا لتعليمة الشرط، نقدم البرنامج التالي الذي يحتسب جذور معادلة ذات درجة ثانية. نطلب ضوارب المعادلة ثم نحتسب مميز المعادلة. إذا كان هذا المميز إجابيا، فنحتسب الجذرين و نطبعهما، مع طبع قيمة صيغة المعادلة، حيث نعوض المتغيرة بالجذر، للتأكد من أنها تساوي صفرا.

```
import java.util.Scanner;                                          //  1.
public class Main {                                                //  2.
    /**                                                            //  3.
     * @param args the command line arguments                     //  4.
     */                                                            //  5.
    public static void main(String[] args) {                       //  6.
        // TODO code application logic here                        //  7.
        Scanner inputWindow = new Scanner(System.in);              //  8.
        System.out.println("Give coefficients of quadratic equation");
        double a, b, c;                                            // 10.
        a = inputWindow.nextDouble();                              // 11.
        b = inputWindow.nextDouble();                              // 12.
        c = inputWindow.nextDouble();                              // 13.
        System.out.println(a + " " +b+ " " + c);                   // 14.
        double delta = b*b-4*a*c;                                  // 15.
                                                                   // 16.
        if (delta>=0)                                              // 17.
        {                                                          // 18.
            double root1, root2;                                   // 19.
            root1 = (-b-Math.sqrt(delta))/(2*a);                   // 20.
            System.out.println(root1 +" "+ (a*root1*root1+b*root1+c));
            root2 = (-b+Math.sqrt(delta))/(2*a);                   // 22.
            System.out.println(root2 +" "+ (a*root2*root2+b*root2+c));
        }                                                          // 24.
    }                                                              // 25.
}                                                                  // 26.
```

نعلق على هذا البرنامج في ما يلي:

- السطر 1: نصرح أننا نحتاج لناسخ لقراءة ضوارب المعادلة.
- السطر 8: نربط الناسخ بلوحة المفاتيح.
- السطر 9: نطلب من المستخدم أن يزود البرنامج بالمعادلة التي يريد حلها.
- الأسطر 10 إلى 14: قراءة تفاصيل المعادلة و رد الصدى.
- السطر 15: إحتساب المميز.
- السطر 17: إذا كان المميز إجابيا، فنهتم بإحتساب جذور المعادلة.
- الأسطر 19 إلى 23: نحتسب الجذرين و نطبعهما، مع طبع قيمة الصيغة المعنية لنتأكد من أن قيمتها (عندما نعوض المتغيرة بالجذر) صفرا.

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
Give coefficients of quadratic equation
1 -7 12
1.0 -7.0 12.0
3.0 0.0
```

```
4.0 0.0
BUILD SUCCESSFUL (total time: 3 minutes 15 seconds)
```

و هي النتيجة التي ننتظرها في هذه الحالة.

التعليمات المتداخلة. طبعا، لا يمنعنا شيء من أن نكتب تعليمة شرط ضمن تعليمة شرط. نقدم في ما يلي مثال برنامج يتضمن عددا من تعليمات الشرط المتداخلة. يسنهدف هذا البرنامج إحتساب اللوغاريثما لعدد حقيقي حسب دقة معينة، ثم التأكد من أن النتيجة تلبي مقياس الدقة. و إن لم تفعل فنغير المقياس. نكتب البرنامج التالي:

```
double x = 3.40282347e+35f;
double y;
double epsilon = 0.000000000000001e+35f;

y = Math.log(x);

System.out.println(x+" "+Math.exp(y));

System.out.println("Checking for precision: "+epsilon);
if (Math.abs(x-Math.exp(y))>epsilon)
{
   System.out.println("Fails to meet given precision. ");
   epsilon = 10*epsilon;
   System.out.println("Checking for a new precision: "+epsilon);

   if (Math.abs(x-Math.exp(y))>epsilon)
   {
      System.out.println("Fails to meet given precision. ");
      epsilon = 10*epsilon;
      System.out.println("Checking for a new precision: "+epsilon);
      if (Math.abs(x-Math.exp(y))>epsilon)
      {
      System.out.println("Fails to meet given precision. ");
      epsilon = 10*epsilon;
      System.out.println("Checking for a new precision: "+epsilon);
      }
    }
 }
```

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
3.4028234346940236E35 3.402823434694022E35
Checking for precision: 1.0000000200408773E20
Fails to meet given precision.
Checking for a new precision: 1.0000000200408773E21
BUILD SUCCESSFUL (total time: 0 seconds)
```

كما نلاحظ، فإن القيمتان المطبوعتان ليست متساويتان؛ نقيم سطرا تحت الجزئين المختلفين. كان من الضروري ان تكون القيمتان متساويتان، لأن الأولى قيمة **x** الأصلية و الثانية نتيجة تطبيق وظيفتين متعاكستين على **x**، لكن أخطاء

التقريب حالت دون ذلك. نلاحظ أيضا أن عملية الإحتساب لم تلبي مقياس الدقة الأصلي (**epsilon = 0.000000000000001e+35f**) لكنها تلبي المقياس الجديد الذي تحصلنا عليه عندما ضربنا **epsilon** ب10.

2.2.6 تعليمة الخيار

نكتب تعليمة الخيار على النحو التالي:

```
if (condition)
   {
    thenclause
   }
else
   {
    elseclause
   }
```

إذا كان الشرط **(condition)** صحيحا، فتنفذ جافا التعليمات الموجودة بين قوسين تحت إسم **thenclause** ، و إن لم يكن صحيحا، فتنفذ جافا التعليمات الموجودة بين قوسين تحت إسم **elseclause**. إذا فشلت جافا في تقييم الشرط، فيفشل تنفيذ التعليمة. مثلا، إذا أردنا أن نضع في متغيرة **z** أكبر قيمة ضمن قيمتي **x** و **y** ، فنقارن **x** و **y** و نضع أكبرهما في **z**. فنكتب ما يلي:

```
int x, y, z;
… … …
if (x>y) {z=x;}
else {z=y;}
```

لنبين إستعمال هذه التعليمة، نتأمل في برنامج معادلة الدرجة الثانية الذي قدمناه في الجزء السابق، و ننفذه ببيانات جديدة، فنجد.

```
Give coefficients of quadratic equation
1 -7 21
1.0 -7.0 21.0
BUILD SUCCESSFUL (total time: 16 seconds)
```

قد يتسائل مستخدم البرنامج: ماذا صار؟ لماذا بقي البرنامج صامتا (و إكتفى بطبع البيانات التي زودناها له)؟ هو بالطبع بقي صامتا لأن مميز المعادلة سلبي (35-=21*4-7*7)، فبالتالي يجد أن هذه المعادلة لا تقبل جذور. لكن المستخدم قد يفضل تصريحا واضحا في هذا المعنى. فنغير البرنامج كما يلي، حيث نعوض تعليمة الشرط الموجودة حاليا بتعليمة خيار نكتبها كما يلي:

```
if (delta>=0)
   {
      double root1, root2;
      root1 = (-b-Math.sqrt(delta))/(2*a);
      System.out.println(root1 +" "+ (a*root1*root1+b*root1+c));
      root2 = (-b+Math.sqrt(delta))/(2*a);
      System.out.println(root2 +" "+ (a*root2*root2+b*root2+c));
    }
    else
    {
       System.out.println("The equation has no solutions.");
    }
```

إذا نفذنا هذا البرنامج على نفس البيانات التي قدمناها أعلاه، يرد البرنامج النتيجة التالية:

```
Give coefficients of quadratic equation
1 -7 21
1.0 -7.0 21.0
The equation has no solutions.
BUILD SUCCESSFUL (total time: 8 seconds)
```

و هي نتيجة يطمئن لها المستخدم أكثر من الصمت.

التعليمات المتداخلة. إذا نفذنا برنامج معادلة الدرجة الثانية بالبيانات التالية (1، -6، 9)، يرد البرنامج النتيجة التالية:

```
Give coefficients of quadratic equation
1 -6 9
1.0 -6.0 9.0
3.0 0.0
3.0 0.0
BUILD SUCCESSFUL (total time: 24 seconds)
```

نريد أن يتفطن البرنامج أن الجذرين في هذه الجالة متساويان، و أن يصرح بأن في هذه الحالة تقبل المعادلة جذرا واحدا عوضا عن جذرين. كما نتذكر من دراسة المعادلات ذات الدرجة الثانية، فإن هذه الحالة تطرأ عندما يكون مميز المعادلة يساوي صفرا. فنحور البرنامج كما يلي:

```
if (delta==0)
  {
      System.out.println("Equation has one solution: ");
      double root = -b/(2*a);
      System.out.println(root +" "+ (a*root*root+b*root+c));
  }
  else if (delta>0)
  {
      System.out.println("Equation has two solutions: ");
      double root1, root2;
      root1 = (-b-Math.sqrt(delta))/(2*a);
      System.out.println(root1 +" "+ (a*root1*root1+b*root1+c));
      root2 = (-b+Math.sqrt(delta))/(2*a);
      System.out.println(root2 +" "+ (a*root2*root2+b*root2+c));
  }
  else
  {
      System.out.println("The equation has no solutions.");
  }
```

بحيث أن البرنامج يفرق الآن بين ثلاثة حالات مختلفة، حسب إن كان مميز المعادلة صفرا (وجود جذر واحد)، أو أكبر من الصفر (وجود جذرين)، أو أصغر من الصفر (لا يوجد أي جذر). فإذا نفذنا هذا البرنامج على نفس البيانات التي قدمناها أعلاه، يرد البرنامج النتيجة التالية:

```
Give coefficients of quadratic equation
1 -6 9
1.0 -6.0 9.0
Equation has one solution:
3.0 0.0
BUILD SUCCESSFUL (total time: 20 seconds)
```

و هي النتيجة المنتظرة في هذه الحالة. نرى أن هذا الحل يتضمن تعليمات خيار متداخلة، فنصرح بالقاعدة العامة التالية بخصوص تعليمات الخيار و الشرط.

بصفة عامة، ترتبط كل تعليمة **(else)** بأقرب شرط **(if)** لها.

تنطبق هذه القاعدة على أي تركيب يشمل تعليمات الخيار و تعليمات الشرط. مثلا، إذا كتبنا النص التالي في جافا،

```
if (c) if (d) {A} else {B}
```

فيأوله مصرف جافا بهذه الصفة

```
if (c) {if (d) {A} else {B}}
```

عوضا عن هذه الصفة،

```
if (c) {if (d) {A}} else {B}.
```

توضيحا لإستعمال تعليمات الخيار، نهتم ببرنامج يحتسب أجر عامل حسب عدد الساعات التي عملها في الأسبوع. ينص قانون المؤسسة أن أجر العامل بالنسبة للساعة الواحدة يتغير حسب عدد الساعات:

- إذا كان عدد الساعات 40 أو أقل، فيقبض العامل 10.2 دينار في الساعة.
- بالنسبة لكل ساعة تفوق ال40 و دون ال50، يقبض العامل 12.4 دينارا في الساعة.
- بالنسبة لكل ساعة تفوق ال50 و دون ال60، يقبض العامل 15.6 دينارا في الساعة.
- بالنسبة لكل ساعة تفوق ال60 و دون ال70، يقبض العامل 20.8 دينارا في الساعة.
- إذا عمل عامل أكثر من 70 ساعة، فيقبض أجر 70 ساعة.

نكتب البرنامج التالي لهذا الغرض:

```
import java.util.Scanner;                                          //  1.
import java.util.*;                                                //  2.
import java.text.NumberFormat;                                     //  3.
import java.text.DateFormat;                                       //  4.
                                                                   //  5.
public class Main {                                                //  6.
    /**                                                            //  7.
     * @param args the command line arguments                     //  8.
     */                                                           //  9.
    public static void main(String[] args) {                      // 10.
        // TODO code application logic here                       // 11.
        Scanner inputWindow = new Scanner(System.in);              // 12.
        NumberFormat nf = NumberFormat.getNumberInstance();        // 13.
        NumberFormat cf = NumberFormat.getCurrencyInstance();      // 14.
                                                                   // 15.
        int nbhours;                                               // 16.
        double salary;                                             // 17.
                                                                   // 18.
        final double upto40 = 10.2;                                // 19.
        final double upto50 = 12.4;                                // 20.
        final double upto60 = 15.6;                                // 21.
        final double upto70 = 20.8;                                // 22.
                                                                   // 23.
```

```
        System.out.println("Give number of hours of work: ");     // 24.
        nbhours = inputWindow.nextInt();                          // 25.
                                                                  // 26.
        if (nbhours <= 40)                                        // 27.
        {// nbhours <=40                                          // 28.
            salary = nbhours*upto40;                              // 29.
        }                                                         // 30.
        else if (nbhours <= 50)                                   // 31.
        {// nbhours > 40 && nbhours <= 50                         // 32.
            salary = 40*upto40 + (nbhours-40)*upto50;             // 33.
        }                                                         // 34.
        else if (nbhours <= 60)                                   // 35.
        {// nbhours > 50 && nbhours <= 60                         // 36.
            salary = 40*upto40 + 10*upto50 +                      // 37.
                    (nbhours-50)*upto60;                          // 38.
        }                                                         // 39.
        else if (nbhours <=70)                                    // 40.
        {// nbhours > 60 && nbhours <= 70                         // 41.
            salary = 40*upto40 + 10*upto50 +                      // 42.
                    10*upto60 + (nbhours-60)*upto70;              // 43.
        }                                                         // 44.
        Else                                                      // 45.
        {// nbhours > 70                                          // 46.
            salary = 40*upto40 + 10*upto50+                       // 47.
                    10*upto60 + 10*upto70;                        // 48.
        }                                                         // 49.
        System.out.println(cf.format(salary));                    // 50.
    }                                                             // 51.
}                                                                 // 52.
```

نعلق على هذا البرنامج في ما يلي:

- الأسطر 1 إلى 4: الباقات اللازمة للتوريد و إصدار البيانات.
- الأسطر 12 إلى 14: التصريحات اللازمة للتوريد و إصدار البيانات.
- الأسطر 16 و 17: متغيرات لخزن عدد الساعات و الأجر الأسبوعي.
- الأسطر 19 إلى 22: نصرح بمستقرات تخزن نسبات الأجر بحساب الساعة.
- السطرين 24 و 25: نطلب و نقتطف عدد الساعات.
- السطر 29: أجر من إشتغل 40 ساعة أو أقل.
- السطر 33: أجر من إشتغل أكثر من 40 و حتي 50 ساعة في الأسبوع.
- السطرين 37 و 38: أجر من إشتغل أكثر من 50 و حتي 60 ساعة في الأسبوع.
- السطرين 42 و 43: أجر من إشتغل أكثر من 60 و حتي 70 ساعة في الأسبوع.
- السطرين 47 و 48: أجر من إشتغل أكثر من 70 ساعة.

ننفد هذا البرنامج و نجربه ب56 ساعة، فيسفر على النتيجة التالية:

```
Give number of hours of work:
56
625.6 د.ت.
BUILD SUCCESSFUL (total time: 11 seconds)
```

نحتسب أجر من عمل 56 ساعة، فنجد:

$40\times10.2 + 10\times12.4+6\times15.6=625.6.$

و هي النتيجة التي يجدها البرنامج.

3.6 تعليمات التوجيه

نفترض أننا نشغل حارسا أجره اليومي 18 دينارا، نريد أن نحتسب أجره الشهري بالنسبة لكل شهر. لهذا الغرض، يجب أن نضرب الأجر اليومي بعدد الأيام في الشهر. كما نعلم، فإن عدد الأيام في الشهر يتغير حسب الشهر، و بالنسبة لشهر فيفري يتغير حسب السنة. يمكن إستعمال تعليمات الخيار المتداخلة، كما يلي:

```
int month; int year;
int nbdays;
double dailySalary;
double Monthlysalary;
if (month==12)
   {nbdays = 31;}
else
   if (month==2)
      {if (year%4==0)
          {nbdays = 29;}
       else
          {nbdays = 28;}}
   else
      if (month == 3)
         {nbdays = 31;}
      else
         if (month==4)
            {nbdays = 30;}
         else
.. .. .. ..
         if (month == 12)
            {nbdays = 31;}
         else
            {errorMessage (illegalMonthNumber);}
```

بالرغم من أن المشكلة التي نريد حلها في هذا البرنامج بسيطة، فإن البرنامج معقد و تحليله صعب. لهذه الأسباب، توفر لنا لغة جافا تعليمة مهيكلة، تنطبق على مثل هذه الحالات. نلقي على هذه التعليمة إسم تعليمة التوجيه أو تعليمة التوزيع، و نقدم قاعدة نحوية تبين شكلها:

<switchStatement> ::= switch (<expression>)
{ <switchCase>
<switchCase>
<switchCase>
.. ..
<switchCase> }

<switchCase> ::= case <expression> : <statement>

<switchCase> ::= default : <statement>

تتمثل تعليمة التوجيه في تقييم عبارة ما، من نوع الحرف أو العدد الكامل (القصير أو العادي)، و تنفيذ مجموعة من التعليمات حسب قيمة العبارة. يمثل رمز العبارة (<expression>) في القاعدة الأولى أعلاه العبارة التي نستعمل قيمتها لتعيين الحالة (<switchCase>) التي نختار تنفيذها؛ نلقي على هذه إسم عبارة التوجيه. و يمثل رمز العبارة

(<expression>) في القاعدة الثانية أعلاه القيمة التي توجه التنفيذ إلى مجموعة التعلميات الحالية؛ نلقي على هذه إسم عبارة البديل. إن عبارة البديل يجب أن تكون خالية من المتغيرات، أي أن قيمتها تكون معروفة في طور الصرف عوضا عن طور التنفيذ. أما عن الحالة المهملة (default)، فيقع إختيارها إذا فشل جافا في مقارنة عبارة التوجيه مع كل عبارات البدائل.

```
import java.util.Scanner;                                                    //  1.
import java.util.*;                                                          //  2.
import java.text.NumberFormat;                                               //  3.
                                                                             //  4.
public class Main {                                                          //  5.
    /**                                                                      //  6.
     * @param args the command line arguments                               //  7.
     */                                                                     //  8.
    public static void main(String[] args) {                                 //  9.
        // TODO code application logic here                                  // 10.
        Scanner inputWindow = new Scanner(System.in);                        // 11.
        NumberFormat nf = NumberFormat.getNumberInstance();                  // 12.
        NumberFormat cf = NumberFormat.getCurrencyInstance();                // 13.
                                                                             // 14.
        int month; int year;                                                 // 15.
        int nbdays=0;                                                        // 16.
        double monthlySalary;                                                // 17.
                                                                             // 18.
        final double dailySalary = 18.4;                                     // 19.
                                                                             // 20.
        System.out.println                                                   // 21.
        ("Give Month and Year for which you want Salary Data: ");            // 21.
        month = inputWindow.nextInt();                                       // 22.
        year = inputWindow.nextInt();                                        // 23.
                                                                             // 24.
        switch (month)                                                       // 25.
        {                                                                    // 26.
            case 1:  nbdays = 31;  break;                                    // 27.
            case 2:  if (year%4==0) {nbdays=29;} else {nbdays=28;}           // 28.
                     break;                                                  // 29.
            case 3:  nbdays = 31;  break;                                    // 30.
            case 4:  nbdays = 30;  break;                                    // 31.
            case 5:  nbdays = 31;  break;                                    // 32.
            case 6:  nbdays = 30;  break;                                    // 33.
            case 7:  nbdays = 31;  break;                                    // 34.
            case 8:  nbdays = 31;  break;                                    // 35.
            case 9:  nbdays = 30;  break;                                    // 36.
            case 10: nbdays = 31;  break;                                    // 37.
            case 11: nbdays = 30;  break;                                    // 38.
            case 12: nbdays = 31;  break;                                    // 39.
            default: System.out.println("Not a Month Number.");              // 40.
        }                                                                    // 41.
        monthlySalary = dailySalary * nbdays;                                // 42.
        System.out.println("This Month's Salary:  ");                        // 43.
        System.out.println(cf.format(monthlySalary));                        // 44.
    }                                                                        // 45.
}                                                                            // 46.
```

نعلق على هذا البرنامج في ما يلي:

- الأسطر 1 إلى 3: الباقات اللازمة للتوريد و إصدار البيانات.
- الأسطر 11 إلى 13: التصريحات اللازمة للتوريد و إصدار البيانات.
- الأسطر 15 إلى 17: متغيرات لخزن الشهر و السنة و عدد الأيام في الشهر و الأجر الشهري.
- السطر 19: مستقرة تحمل أجر الحارس في ساعة.

- الأسطر 21 إلى 23: نطلب من المستخدم أن يزودنا بعدد الشهر و السنة و نخزنهما في المتغيرتين المخصصتين لهما.
- السطر 25: تختصر عبارة التوجيه في عدد الشهر.
- الأسطر 27 إلى 39: نخصص بديلا لكل شهر. نختم كل بديل بتعليمة **break** التي تحول تنفيذ البرنامج لنهاية تعليمة التوجه.
- السطرين 28 و 29: نعين عدد الأيام في شهر فيفري حسب إن كان عدد السنين ينقسم إلى 4 أم لا.
- السطر 40: إذا لم يكن رقم الشهر بين 1 و 12، فهو لا يشير لشهر.
- الأسطر 42 إلى 44: نحتسب الأجر الشهري حسب الأجر اليومي و عدد الأيام في الشهر، ثم نطبعه.

ننفذ هدا البرنامج فنقدم له شهر فيفري 2008, فيسفر على النتيجة التالية.

```
run:
Give Month and Year for which you want Salary Data:
2 2008
This Month's Salary:
533.6 د.ت.
BUILD SUCCESSFUL (total time: 14 seconds)
```

فعلا، إذا ضربنا 18.4 في 29، لوجدنا 533.6.

4.6 تمارين

1 [س] أكتبوا برنامجا بسيطا يتأكد من القواعد التالية بخصوص ترتيب الحروف في جافا: الأعداد تسبق الحروف؛ الحروف اللاتينية تسبق الحروف العربية؛ ضمن الحروف اللاتينية، الحروف المضخمة تسبق الحروف العادية؛ الأعداد العربية تسبق الأعداد الهندية.

2 [م] نبيع كتبا بالجملة ثمنها 85.7 دينارا. إذا إشترى حريف من 1 إلى 10 كتب، فيدفع الثمن كاملا. إذ إشترى أكثر من 10 كتب و دون ال20 كتاب، فنمنحه إنخفاض بنسبة 12 بالمائة في الكتب بعد العاشر. إذ إشترى أكثر من 20 كتاب و دون ال30 كتاب، فنمنحه إنخفاض بنسبة 16 بالمائة في الكتب بعد العشرين. إذ إشترى أكثر من 30 كتاب و دون ال40 كتاب، فنمنحه إنخفاض بنسبة 22 بالمائة في الكتب بعد الثلاثين. إذ إشترى أكثر من 40 كتاب و دون ال50 كتاب، فنمنحه إنخفاض بنسبة 28 بالمائة في الكتب بعد الأربعين. إذ إشترى أكثر من 50 كتاب، فنمنحه إنخفاض بنسبة 40 بالمائة في الكتب بعد الخمسين. المطلوب منكم تأليف برنامج يقبل عدد الكتب و يحتسب ثمنها.

3 [م] ضرائب متصاعدة. تفرض حكومة ضرائب على الدخل حسب الجدول التالي: إذا كان الدخل السنوي للمواطن دون العشرة آلاف دينار، فلا يدفع أداء على الدخل. إذا كان دخل المواطن أكثر من عشرة و دون العشرين ألف دينار سنويا، يدفع المواطن 6 بالمائة من الدخل الذي يفوق العشرة آلاف دينار. إذا كان دخل المواطن أكثر من عشرين و دون الثلاثين ألف دينار سنويا، يدفع المواطن 16 بالمائة من الدخل الذي يفوق العشرين ألف دينار. إذا كان دخل المواطن أكثر من ثلاثين و دون الأربعين ألف دينار سنويا، يدفع المواطن 25 بالمائة من الدخل الذي يفوق الثلاثين ألف دينار. إذا كان دخل المواطن أكثر من أربعين و دون الخمسين ألف دينار سنويا، يدفع المواطن 35 بالمائة من الدخل الذي يفوق الأربعين ألف دينار. إذا كان دخل المواطن أكثر من خمسين ألف دينار سنويا، يدفع المواطن 50 بالمائة من الدخل الذي يفوق الخمسين ألف دينار. المطلوب منكم تأليف برنامج يتلقى دخل مواطن فيحتسب قيمة الضريبة التي تنطبق على هذا المواطن ثم إحتساب دخله بعد الضريبة.

4 [م] نريد تغيير البرنامج الموجود في الجزء 3.6 لتطبيق الضرائب على الدخل على الأجر الشهري. تفرض الحكومة ضرائب على الدخل حسب الجدول التالي: إذا كان الدخل الأسبوعي للمواطن دون الثمانمائة دينار شهريا، فلا يدفع أداء على الدخل. إذا كان دخل المواطن أكثر من الثمانمائة دينار و دون الألف و ستمائة دينار شهريا، يدفع المواطن 6 بالمائة من الدخل الذي يفوق الثمانمائة دينار. إذا كان دخل المواطن أكثر من الألف و ستمائة دينار و دون الألفين و

أربعمائة دينار شهريا، يدفع المواطن 16 بالمائة من الدخل الذي يفوق الألف و ستمائة دينار. إذا كان دخل المواطن أكثر من الألفين و أربعمائة دينار و دون الثلاثة آلاف و مائتي دينار شهريا، يدفع المواطن 25 بالمائة من الدخل الذي يفوق الألفين و أربعمائة دينار. إذا كان دخل المواطن أكثر من الثلاثة آلاف و مائتي دينار و دون الأربعة آلاف دينار شهريا، يدفع المواطن 35 بالمائة من الدخل الذي يفوق الثلاثة آلاف و مائتي دينار. إذا كان دخل المواطن أكثر من الأربعة آلاف دينار شهريا ، يدفع المواطن 50 بالمائة من الدخل الذي يفوق الأربعة آلاف دينار. المطلوب منكم إضافة تعليمات للبرنامج الموجود في الجزء 3.6 لكي يحتسب الضرائب بعد إحتساب الأجر الشهري، ثم يطرح الضائب من الأجر فيطبع هذه النتائج.

5 [س] المطلوب نمكم تأليف برنامج يسند لكل حرف عربي عدده في الترتيب الأبجدي (يسند 1 للأليف و 2 للباء و 3 للتاء، إلخ). فيطلب من المستخدم أن يمده بحرف ما و يطبع العدد المناسب.

6 [س] المطلوب نمكم تأليف برنامج يسند لكل حرف عربي حرفا لاتينيا يقاربه في النطق (مثلا: أ، A؛ ب، B؛ ت، C؛ ط، T؛ س، s؛ ق، q؛ ك، k؛ إلخ). فيطلب من المستخدم أن يمده بحرف عربي و يطبع الحرف اللاتيني المناسب.

7 [س] المطلوب نمكم تأليف برنامج يسند لكل حرف لاتيني حرفا عربيا يقاربه في النطق (مثلا: أ، A؛ ب، B؛ ت، C؛ ط، T؛ س، s؛ ق، q؛ ك، k؛ إلخ). فيطلب من المستخدم أن يمده بحرف لاتيني و يطبع الحرف العربي المناسب.

.

الباب الثالث

هياكل البرمجة في جافا

الفصل السابع

تعليمات التكرار

إن الحاسوب يجلب جل قوته من مقدرته على تنفيذ عمليات يكررها عددا كبيرا من المرات. بالتالي، تكتسي تعليمات التكرار في جافا، و غيرها من لغات البرمجة، أهمية بالغة. نقدم في هذا الفصل بعض تعليمات التكرار التي توفرها لغة جافا، و نناقش إستعمالها في أمثلة بسيطة.

1.7 التكرار ذات الشرط المسبق

توفر لغة جافا تعليمة تكرار يكون شكلها على النحو التالي:

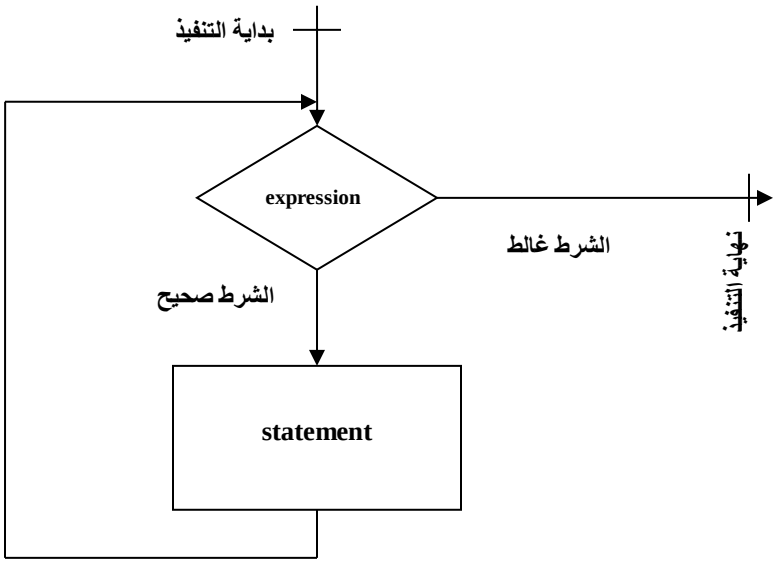
<whileStament> ::= while (<expression>) <statement>

نلقي على تعليمة التكرار إسم المدار. كما نلقي على **<expression>** إسم شرط المدار و نلقي على **<statement>** إسم جسم المدار.

تقوم جافا بتنفيذ هذه التعليمة على النحو التالي:

1. تختبر الشرط المتمثل في العبارة المنطقية **(<expression>).**
2. إذا فشلت في إحتساب القيمة المنطقية لهذه العبارة، فيفشل تنفيذ التعليمة.
3. إذا كانت قيمة هذه العبارة المنطقية سلبية، ينتهي تنفيذ التعليمة فورا بدون أي تغيير.
4. إذا كانت قيمة هذه العبارة المنطقية إجابية، تنفذ جافا تعليمة **<statement>** و تعود للسطر الأول لتختبر العبارة المنطقية **(<expression>)** من جديد. نشير لكل تنفيذ لتعليمة **<statement>** بأنه تكرار.
5. إذا تواصلت جافا في إختبار الشرط **(<expression>)** و تنفيذ التعليمة **<statement>** بدون نهاية (أي أن الشرط كان دائما صحيحا) فنقول أن هذه التعليمة مدار لا منتهي، و نعتبر عندئذ أن تنفيذ هذا المدار فاشل و أنه لا يقبل نتيجة نهائية.

نمثل هذه التعليمة بالرسم التالي رقم 1.7. نقدم في ما يلي أمثلة بسيطة في إستعمال هذه التعليمة.



الرسم رقم 1.7: تنفيذ التكرار ذات الشرط المسبق

1.1.7 عملية الجمع

نعتبر متغيرتين من نوع الأعداد الكاملة (**a** و **b**) و نعتبر البرنامج التالي على هتين المتغيرتين:

```
int a, b;

System.out.println("Give Factors a and b:  ");
a = inputWindow.nextInt();    //  a0
b = inputWindow.nextInt();    //  b0

while (b!=0)
{
   a=a+1; b=b-1;
}

System.out.println(a);
```

كلما نفذنا هذا البرنامج و زودناه بقيمتين لمتغيرتي **a** و **b** إلا و طبع نتيجة جمع **a** و **b**. لنفهم السبب في ذلك نلقي على القيمة الإفتتاحية لهتين المتغيرتين إسمي **a0** و **b0** و نلاحض أننا كلما ننفذ جسم المدار

```
{
   a=a+1; b=b-1;
}
```

لا نغير قيمة (**a+b**) بالرغم من أننا نغير قيمة **a** و **b**. علما أن قيمة (**a+b**) لا تتغير بعد كل تنفيذ لجسم المدار، نستنتج أنها تساوي قيمتها الإفتتاحية عند إنطلاق التنفيذ، فنكتب:

a+b=a0+b0.

نحن نعلم (حسب تأويل تعليمة التكرار، الرسم 1.7) أن تنفيذ التعليمة ينتهي عندما يصبح شرط المدار غالطا، أي في هذه الحالة عندما تصبح قيمة **b** صفرا. من المعادلتين

b=0
a+b=a0+b0

نستنتج

a=a0+b0

و هو ما توقعناه من تجربتنا. تأكيدا لهذه النتيجة، نعود للبرنامج، فنطبع قيمتي **a** و **b** كلما نفذنا جسم البرنامج، كما نطبع قيمة العبارة **a+b**، فتكون النتيجة كما يلي:

```
Give Factors a and b:
2 7
a= 3 b= 6 a+b = 9
a= 4 b= 5 a+b = 9
a= 5 b= 4 a+b = 9
a= 6 b= 3 a+b = 9
a= 7 b= 2 a+b = 9
a= 8 b= 1 a+b = 9
a= 9 b= 0 a+b = 9
9
BUILD SUCCESSFUL (total time: 13 seconds)
```

كما نرى في نتيجة هذا التنفيذ، فإن قيمة المتغيرتين تتغير خلال تنفيذ المدار، بينما قيمة العبارة **(a+b)** تبقى على حالها خلال التنفيذ.

أما إذا كانت القيمة الإفتتاحية لمتغيرة **b** أقل من الصفر، فإن هذا البرنامج يتمثل في تكرار لامنتهي، لأننا نطرح 1 من **b** في كل تكرار، منتظرين أن تصبح قيمة **b** صفرا، و لن تصبح. بالتالي لا نناقش نتيجته لأنه لا يؤدي إلى نتيجة قابلة للتحليل.

2.1.7 عملية الطرح

نعتبر متغيرتين من نوع الأعداد الكاملة **(a و b)** و نعتبر البرنامج التالي على هتين المتغيرتين:

```
int a, b;

System.out.println("Give Factors a and b:  ");
a = inputWindow.nextInt();    //  a0
b = inputWindow.nextInt();    //  b0

while (b!=0)
{
   a=a-1; b=b-1;
}

System.out.println(a);
```

كلما نفذنا هذا البرنامج و زودناه بقيمتين لمتغيرتي **a** و **b** إلا و طبع نتيجة طرح **b** من **a**. . لنفهم السبب في ذلك نلقي على القيمة الإفتتاحية لهتين المتغيرتين إسمي **a0** و **b0** و نلاحض أننا كلما ننفذ جسم المدار

```
{
   a=a+1; b=b-1;
}
```

لا نغير قيمة (a-b) بالرغم من أننا نغير قيمة a و b. علما أن قيمة (a-b) لا تتغير بعد كل تنفيذ لجسم المدار، نستنتج أنها تساوي قيمتها الإفتتاحية عند إنطلاق التنفيذ، فنكتب:

$$a-b=a0-b0.$$

نحن نعلم (حسب تأويل تعليمة التكرار، الرسم 1.7) أن تنفيذ التعليمة ينتهي عندما يصبح شرط المدار غالطا، أي في هذه الحالة عندما تصبح قيمة b صفرا. من المعادلتين

$$b=0$$
$$a-b=a0-b0$$

نستنتج

$$a=a0-b0$$

و هو ما توقعناه من تجربتنا. تأكيدا لهذه النتيجة، نعود للبرنامج، فنطبع قيمتي a و b كلما نفذنا جسم البرنامج، كما نطبع قيمة العبارة a-b، فتكون النتيجة كما يلي:

```
Give Factors a and b:
9 7
a= 8 b= 6 a-b = 2
a= 7 b= 5 a-b = 2
a= 6 b= 4 a-b = 2
a= 5 b= 3 a-b = 2
a= 4 b= 2 a-b = 2
a= 3 b= 1 a-b = 2
a= 2 b= 0 a-b = 2
2
BUILD SUCCESSFUL (total time: 7 seconds)
```

كما نرى في نتيجة هذا التنفيذ، فإن قيمة المتغيرتين تتغير خلال تنفيذ المدار، بينما قيمة العبارة (a-b) تبقى على حالها خلال التنفيذ.

أما إذا كانت القيمة الإفتتاحية لمتغيرة b أقل من الصفر، فإن هذا البرنامج يتمثل في تكرار لامنتهي، لأننا نطرح 1 من b في كل تكرار، منتظرين أن تصبح قيمة b صفرا، و لن تصبح. بالتالي لا نناقش نتيجته لأنه لا يؤدي إلى نتيجة قابلة للتحليل.

3.1.7 عملية الضرب

نعتبر ثلاثة متغيرات من نوع الأعداد الكاملة (a و b و c) و نعتبر البرنامج التالي على هذه المتغيرات:

```
int a, b, c;

System.out.println("Give Factors a and b:  ");
a = inputWindow.nextInt();
b = inputWindow.nextInt();
c = 0;

while (b!=0)
{
   c=c+a; b=b-1;
}
```

```
System.out.println(c);
```

كلما نفذنا هذا البرنامج و زودناه بقيمتين لمتغيرتي **a** و **b** إلا و طبع نتيجة ضرب **b** من **a**. لنفهم السبب في ذلك نلقي على القيمة الإفتتاحية لهتين المتغيرتين إسمي **a0** و **b0** و نلاحض أننا كلما ننفذ جسم المدار

```
{
   c=c+a; b=b-1;
}
```

لا نغير قيمة (**c+a*b**) بالرغم من أننا نغير قيمة **c** و **b**. علما أن قيمة (**c+a*b**) لا تتغير بعد كل تنفيذ لجسم المدار، نستنتج أنها تساوي قيمتها الإفتتاحية عند إنطلاق التنفيذ، فنكتب:

$$c+a*b=c0+a0*b0.$$

نحن نعلم (حسب تأويل تعليمة التكرار، الرسم 1.7) أن تنفيذ التعليمة ينتهي عندما يصبح شرط المدار غالطا، أي في هذه الحالة عندما تصبح قيمة **b** صفرا. من المعادلتين

$$b=0$$
$$c+a*b=c0+a0*b0$$

نستنتج

$$c=c0+a0*b0$$

و هو ما توقعناه من تجربتنا. تأكيدا لهذه النتيجة، نعود للبرنامج، فنطبع قيمات **a** و **b** و **c** كلما نفذنا جسم البرنامج، كما نطبع قيمة العبارة **c+a*b**، فتكون النتيجة كما يلي:

```
Give Factors a and b:
3 6
a= 3 b= 5 c= 3 c+a*b = 18
a= 3 b= 4 c= 6 c+a*b = 18
a= 3 b= 3 c= 9 c+a*b = 18
a= 3 b= 2 c= 12 c+a*b = 18
a= 3 b= 1 c= 15 c+a*b = 18
a= 3 b= 0 c= 18 c+a*b = 18
18
BUILD SUCCESSFUL (total time: 11 seconds)
```

كما نرى في نتيجة هذا التنفيذ، فإن قيمة المتغيرات تتغير خلال تنفيذ المدار، بينما قيمة العبارة (**c+a*b**) تبقى على حالها خلال التنفيذ.

أما إذا كانت القيمة الإفتتاحية لمتغيرة **b** أقل من الصفر، فإن هذا البرنامج يتمثل في تكرار لامنتهي، لأننا نطرح 1 من **b** في كل تكرار، منتظرين أن تصبح قيمة **b** صفرا، و لن تصبح. بالتالي لا نناقش نتيجته لأنه لا يؤدي إلى نتيجة قابلة للتحليل.

4.1.7 عملية القسمة الكاملة

نعتبر عددين كاملين إجابيين **a** و **b** و نهتم بعملية قسمة **b** إلى **a**. نقول أن العدد الكامل q يمثل قسمة **b** إلى **a** إذا و فقط إذا وُجد عدد كامل r يلبي الشرط التالي:

$$b = a \times q + r \wedge 0 \le r < a.$$

مثلا، نقول أن قسمة 27 إلى 4 تساوي 6 لأن هناك عدد (3) يلبي الشرط التالي:

$$27 = 4 \times 6 + 3 \wedge 0 \leq 3 < 4.$$

نكتب البرنامج التالي ليقوم بقسمة عدد كامل **b** إلى عدد كامل **a**.

```
int a, b, q;

System.out.println("Give Factors a and b:  ");
a = inputWindow.nextInt();
b = inputWindow.nextInt();
q = 0;

while (b>a)
{
   b=b-a; q=q+1;
}

System.out.println("quotient: "+q+" remainder: "+b);
```

ننفذ هذا البرنامج على البيانات التي ناقشناها أعلا (**a** يساوي 4 و **b** يساوي 27) فيسفر على النتيجة التالية، و هي النتيجة المنتظرة في هذا الحالة:

```
Give Factors a and b:
4 27
quotient: 6 remainder: 3
BUILD SUCCESSFUL (total time: 18 seconds)
```

بصفة عامة، نحرض القارئ المهتم أن ينفذ هذا البرنامج على بيانات تتمثل في عددين كاملين إجابيين، ليقتنع أن هذا البرنامج يحتسب القسمة الكاملة في كل حالة. لنقيم الدليل على ذلك، نتأمل في جسم المدار و نهتم بالعبارة التي يحتفظ بها هذا المدار خلال تنفيذه. بما أننا نطرح من في كل تكرار و نضيف 1 إلى فنحافظ دوما على العبارة التالية:

$$b+a*q.$$

علما أن قيمة هذه العبارة لا تتغير بعد كل تنفيذ لجسم المدار، نستنتج أنها تساوي قيمتها الإفتتاحية عند إنطلاق التنفيذ، فنكتب:

$$b+a*q=b0+a0*q0.$$

علما أن القيمة الإفتتاحية لمتغيرة **q** هي 0، نحول هذه المعادلة لشكل بسيط، كما يلي:

$$b+a*q=b0.$$

نحن نعلم (حسب تأويل تعليمة التكرار، الرسم 1.7) أن تنفيذ التعليمة ينتهي عندما يصبح شرط المدار غالطا، أي في هذه الحالة عندما تصبح قيمة **b** أقل من قيمة **a**. من المعادلتين

$$b<a$$
$$b+a*q=b0$$

نستنتج أن **q** يمثل نتيجة قسمة **b0** إلى **a** و أن **b** يمثل باقي هذه القسمة. نحرض القارئ المهتم أن يقوم بالتمرين رقم 3 لتوضيح تفصيل إضافي في هذا التحليل. تأكيدا لهذه النتيجة، نعود للبرنامج، فنطبع قيمات **a** و **b** و **q** كلما نفذنا جسم البرنامج، كما نطبع قيمة العبارة **b+a*q**، فتكون النتيجة كما يلي:

```
Give Factors a and b:
```

```
4 27
a= 4 b= 23 q= 1 b+a*q = 27
a= 4 b= 19 q= 2 b+a*q = 27
a= 4 b= 15 q= 3 b+a*q = 27
a= 4 b= 11 q= 4 b+a*q = 27
a= 4 b= 7  q= 5 b+a*q = 27
a= 4 b= 3  q= 6 b+a*q = 27
quotient: 6 remainder: 3
BUILD SUCCESSFUL (total time: 9 seconds)
```

5.1.7 عملية أكبر قاسم مشترك

نعتبر عددين كاملين إجابيين **a** و **b** و نهتم بعملية أكبر قاسم مشترك لهما. نمثل أكبر قاسم مشترك ل**a** و **b** برمز **gcd(a,b)** و نعلم أن هذه الوظيفة تخضع للقواعد التالية:

$$\gcd(a,b) = \gcd(b,a),$$
$$\gcd(a,a) = a,$$
$$a > b \Rightarrow \gcd(a,b) = \gcd(a-b,b).$$

بناء على هذه القواعد, نكتب البرنامج التالي:

```
int a, b;

System.out.println("Give Factors a and b:  ");
a = inputWindow.nextInt();
b = inputWindow.nextInt();

while (a!=b)
{
   if (a>b) {a=a-b;} else {b=b-a;}
}

System.out.println("gcd: "+a);
```

بناء على القواعد التي إستعرضناها أعلاه، يتضح أن جسم المدار يحتفض خلال تنفيذه بأكبر قاسم مشترك ل**a** و **b**. فنستنتج أن قيمة هذه الوظيفة في ختام تنفيذ المدر يساوي قيمتها في إفتتاح التنفيذ. فنكتب:

$$\mathbf{gcd(a,b)=gcd(a0,b0).}$$

نحن نعلم (حسب تأويل تعليمة التكرار، الرسم 1.7) أن تنفيذ التعليمة ينتهي عندما يصبح شرط المدار غالطا، أي في هذه الحالة عندما تصبح قيمتا **a** و **b** متساويتان. من المعادلتين

$$\mathbf{a=b}$$
$$\mathbf{gcd(a,b)=gcd(a0,b0)}$$

نستنتج

$$\mathbf{gcd(a,a)=gcd(a0,b0)}$$

فنستنتج، بناء على القواعد التي إستعرضناها أعلاه،

a=gcd(a0,b0).

أي أن القيمة النهائية لمتغيرة **a** تمثل أكبر قاسم مشترك للقيمتين الإفتتاحيتين للمتغيرتين **a** و **b.** توضيحا لهذا الإستنتاج المنطقي، نتأمل فيما يلي في تنفيذ هذا المدار على بيانات معينة، و نبين كيف أن أكبر قاسم مشترك لـ**a** و **b** يبقى على حاله خلال التنفيذ، بالرغم من أن **a** و **b** تتغيران.

a	b	gcd(a,b)
77	49	7
28	49	7
28	21	7
7	21	7
7	14	7
7	7	*7*

فعلا، عندما ننفذ البرنامج أعلاه على عددي 77 و 49، نجد النتيجة التالية:

```
Give Factors a and b:
49 77
gcd: 7
BUILD SUCCESSFUL (total time: 8 seconds)
```

6.1.7 إحتساب الأعداد الأولية

نقول عن عدد طبيعي كامل إنه أولي إذا و فقط إذا كان لا يقبل إلا قاسمين إثنين و هما 1 و العدد نفسه. نجد أن عدد 7 عدد أولي لأنه لا ينقسم إلا لـ1 و لـ7، بينما عدد 8 ليس عددا أوليا لأنه ينقسم إلى 2 و إلى 4 بالإضافة إلى 1 و 8. نعتبر أن 1 ليس عددا أوليا. نهتم في هذا الجزء بإحتساب و طبع الأعداد الأولية الموجودة بين 2 و عدد أقصى يضبطه المستخدم.

```
int rangeMax;                                                    //  1.
                                                                 //  2.
System.out.println("Give the range in which to look for primes:  ");
rangeMax = inputWindow.nextInt();                                //  4.
int primeTest = 2;                                               //  5.
boolean isPrime;  //  test for primeness                         //  6.
                                                                 //  7.
while (primeTest < rangeMax)                                     //  8.
{                                                                //  9.
   int j = 2;                                                    // 10.
   isPrime = true;                                               // 11.
   while (j <= (primeTest/2))                                    // 12.
   {                                                             // 13.
      isPrime = isPrime && (primeTest%j != 0);                   // 14.
      j++;                                                       // 15.
   }                                                             // 16.
   if (isPrime)                                                  // 17.
   {                                                             // 18.
      System.out.print(primeTest+", ");                          // 19.
   }                                                             // 20.
```

```
   primeTest ++;                                                    // 21.
}                                                                   // 22.
```

نعلق على هذا البرنامج فيما يلي:

- السطر الأول: تمثل هذه المتغيرة المقطع الذي نبحث فيه عن أعداد أولية.
- الأسطر 3 و 4: نطلب من المستخدم أن يضبط المقطع الذي نبحث فيه عن أعداد أولية.
- السطر 5: تمثل متغيرة **primeTest** العدد الذي نختبره لنبين إن كان أوليا أم لا.
- السطر 6: تمثل **isPrime** المتغيرة المنطقية التي نخزن فيها نتيجة إختبار العدد الحالي (**primeTest**) --- هل هو أولي أم لا؟
- السطر 8 و السطر 21: تتحول متغيرة **primeTest** من قيمة 2 إلى قيمة **rangeMax**.
- السطر 10: نستعمل متغيرة **j** لنختبر إن كانت **primeTest** تنقسم إليها أم لا.
- السطر 11: نفترض أن العدد الحالي أولي، حتي يأتي ما يخالف ذلك.
- السطر 12: نبحث عن أعداد ينقسم لها **primeTest** بين 2 و نصف **primeTest**.
- السطر 14: لكي تبقى **isPrime** صحيحة، يجب أن لا ينقسم **primeTest** لأي عدد بين 2 و نصف **primeTest**.

ننفذ هذا البرنامج بقيمة 100 لمتغيرة **rangeMax** فيسفر على النتيجة التالية:

```
run:
Give the range in which to look for primes:
100
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97,
BUILD SUCCESSFUL (total time: 6 seconds)
```

و هي فعلا الأعداد الأولية بين 2 و 100.

7.1.7 برنامج تفاعلي تكراري

في ختام هذا الجزء ننظر في برنامج تكراري تفاعلي يحتسب سعر بضاعة يشتريها حريف في مغازة، فتتمثل البيانات الواردة في مجموعة من البضائع نمثل كلا منها بسعرها و بالعدد الذي إشتراه الحريف. فيحتسب البرنامج السعر المضاعف بالنسبة لكل بضاعة ثم يحتسب السعر الجملي. علما أن المستخدم يعلن عن نهاية قائمة البضائع بإدخال 0 في عدد البضائع و في السعر الفردي. فنكتب البرنامج التالي:

```
Scanner inputWindow = new Scanner(System.in);                       //  1.
NumberFormat nf = NumberFormat.getNumberInstance();                 //  2.
NumberFormat cf = NumberFormat.getCurrencyInstance();               //  3.
                                                                    //  4.
int qty;  double unitPrice;  double itemPrice;                      //  5.
double totalPrice;                                                  //  6.
                                                                    //  7.
totalPrice = 0;                                                     //  8.
System.out.println("Give Qty, Unit Price: ");                       //  9.
qty = inputWindow.nextInt();                                        // 10.
unitPrice = inputWindow.nextDouble();                               // 11.
                                                                    // 12.
while (qty!=0)                                                      // 13.
{                                                                   // 14.
     itemPrice = qty*unitPrice;                                     // 15.
```

```
     totalPrice = totalPrice + itemPrice;                                  // 16.
     System.out.print(" qty: "+qty);                                       // 17.
     System.out.print(" Unit Price: "+cf.format(unitPrice));               // 18.
     System.out.print(" Item Price: "+cf.format(itemPrice));               // 19.
     System.out.println(" Running Total: "+cf.format(totalPrice));         // 20.
     System.out.println("Give Qty, Unit Price: ");                         // 21.
     qty = inputWindow.nextInt();                                          // 22.
     unitPrice = inputWindow.nextDouble();                                 // 23.
}                                                                          // 24.
System.out.println("End of order.  Order total: "+cf.format(totalPrice));// 25
```

نعلق على هذا البرنامج في ما يلي:

- الأسطر 1 إلى 3: تعيين النمذج المتعلقة بقراءة البيانات الواردة و طبع البيانات الصادرة.
- الأسطر 5 و 6: متغيرات للكمية و السعر الفردي و سعر البضاعة و السعر الجملي.
- الأسطر 9 إلى 11: نطلب و نتلقى الكمية و السعر الفردي.
- الأسطر 15 و 16: نحتسب سعر البضاعة الحالية و نضيفه للسعر الجملي.
- الأسطر 17 إلى 20: نطبع تفاصيل السطر الخاص بالبضاعة الحالية.
- الأسطر 21 إلى 23: نطلب و نتلقى الكمية و السعر الفردي.
- السطر 25: نختم الحساب و نطبع الجملة.

ننفذ هذا البرنامج على بيانات نزودها بصفة تفاعلية فورية، فتكون النتيجة على النحو التالي:

```
run:
Give Qty, Unit Price:
3 34.8
qty: 3 Unit Price: 34.8 د.ت. Item Price: 104.4 د.ت. Running Total: 104.4 د.ت.
Give Qty, Unit Price:
6 43.5
qty: 6 Unit Price: 43.5 د.ت. Item Price: 261 د.ت.   Running Total: 365.4 د.ت.
Give Qty, Unit Price:
3 26.5
qty: 3 Unit Price: 26.5 د.ت. Item Price: 79.5 د.ت.  Running Total: 444.9 د.ت.
Give Qty, Unit Price:
9 52.3
qty: 9 Unit Price: 52.3 د.ت. Item Price: 470.7 د.ت. Running Total: 915.6 د.ت.
Give Qty, Unit Price:
0 0.0
End of order.  Order total:  915.6 د.ت.
BUILD SUCCESSFUL (total time: 52 seconds)
```

2.7 التكرار ذات الشرط اللاحق

توفر لغة جافا تعليمة تكرار يكون شكلها على النحو التالي:

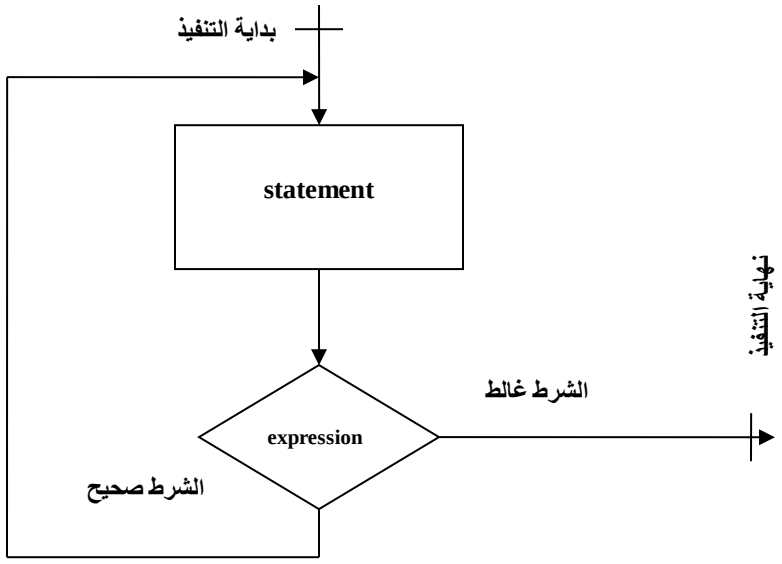
<doStament> ::= do <statement> while (<expression>)

نلقي على تعليمة التكرار إسم المدار. كما نلقي على **<expression>** إسم شرط المدار و نلقي على **<statement>** إسم جسم المدار.

تقوم جافا بتنفيذ هذه التعليمة على النحو التالي:

1. تنفذ جافا تعليمة <**statement**>. نشير لكل تنفيذ لتعليمة <**statement**> بأنه تكرار.
2. تختبر الشرط المتمثل في العبارة المنطقية (<**expression**>).
3. إذا فشلت في إحتساب القيمة المنطقية لهذه العبارة، فيفشل تنفيذ التعليمة.
4. إذا كانت قيمة هذه العبارة المنطقية سلبية، ينتهي تنفيذ التعليمة فورا بدون أي تغيير.
5. إذا كانت قيمة هذه العبارة المنطقية إجابية، تعود للسطر الأول لتنفيذ تعليمة <**statement**> من جديد.
6. إذا تواصلت جافا في إختبار الشرط (<**expression**>) و تنفيذ التعليمة <**statement**> بدون نهاية (أي أن الشرط كان دائما صحيحا) فنقول أن هذه التعليمة مدار لا منتهي، و نعتبر عندئذ أن تنفيذ هذا المدار فاشل و أنه لا يقبل نتيجة نهائية.

نمثل هذه التعليمة بالرسم التالي رقم 2.7.



الرسم رقم 2.7: تنفيذ التكرار ذات الشرط اللاحق

لا تختلف تعليمة التكرار ذات الشرط اللاحق بصفة جذرية عن تعليمة التكرار ذات الشرط السابق. أهم فرق بينهما يتمثل في أن تعليمة التكرار ذات الشرط اللاحق تنفذ جسم المدار مرة على الأقل، بينما تعليمة التكرار ذات الشرط السابق قد لا تنفذه بتاتا إذا كانت شرط المدار غالطا أول مرة نختبره. لنوضح إستعمال هذه التعليمة، ننظر في البرنامج الذي قدمناه في الجزء 7.1.7 و نلاحظ أن مجموعة التعليمات التي تطلب من المستخدم كمية البضاعة و سعرها يقع تنفيذها مرتين: مرة في الأسطر 9 إلى 11 و مرة في الأسطر 21 إلى 23. لماذا نكتبها مرتين؟ لأن شرط المدار يتطلب أن نكون طلبنا و قرأنا هذه البيانات قبل تنفيذ المدار (الأسطر 9 إلى 11) ثم بعد كل تكرار لاحق (الأسطر 21 إلى 23). بإستعمال تعليمة التكرار ذات الشرط اللاحق، نتمكن من كتابة هذه المجموعة مرة فقط، كما يبين البرنامج التالي:

```
Scanner inputWindow = new Scanner(System.in);                    //  1.
```

```
NumberFormat nf = NumberFormat.getNumberInstance();                    //  2.
NumberFormat cf = NumberFormat.getCurrencyInstance();                  //  3.
                                                                       //  4.
int qty;  double unitPrice;  double itemPrice;                         //  5.
double totalPrice;                                                     //  6.
                                                                       //  7.
totalPrice = 0;                                                        //  8.
                                                                       //  9.
do                                                                     // 10.
{                                                                      // 11.
    System.out.println("Give Qty, Unit Price: ");                      // 12.
    qty = inputWindow.nextInt();                                       // 13.
    unitPrice = inputWindow.nextDouble();                              // 14.
    if (qty!=0)                                                        // 15.
    {                                                                  // 16.
       itemPrice = qty*unitPrice;                                      // 17.
       totalPrice = totalPrice + itemPrice;                            // 18.
       System.out.print(" qty: "+qty);                                 // 19.
       System.out.print(" Unit Price: "+cf.format(unitPrice));         // 20.
       System.out.print(" Item Price: "+cf.format(itemPrice));         // 21.
       System.out.println(" Running Total: "+cf.format(totalPrice));   // 22.
    }                                                                  // 23.
}                                                                      // 24.
while (qty!=0);                                                        // 25.
System.out.println("End of order. Order total:  "+cf.format(totalPrice));// 26.
```

أهم فرق بين هذا البرنامج و البرنامج السابق هو أن مجموعة التعليمات التي تطلب كمية البضاعة و سعرها (الأسطر 12 إلى 14) وُضعت داخل جسم المدار، لتعوض مقامها السابق قبل المدار و في أسفل جسم المدار. ينجر عن هذا التحوير أننا الآن نتأكد من أن الكمية دون الصفر (السطر 15) قبل أن نقوم بالعمليات العادية (إحتساب سعر البضاعة الحالية و إضافته للسعر الجملي و طبع السطر الخاص بالبضاعة الحالية). إذا نفذنا هذا البرنامج على نفس البيانات التي قدمناها سابقا، نجد نفس النتيجة، كما يبين النص التالي.

```
run:
Give Qty, Unit Price:
3 34.8
 qty: 3 Unit Price: 34.8 د.ت. Item Price: 104.4 د.ت. Running Total: 104.4 د.ت.
Give Qty, Unit Price:
6 43.5
 qty: 6 Unit Price: 43.5 د.ت. Item Price: 261 د.ت. Running Total: 365.4 د.ت.
Give Qty, Unit Price:
3 26.5
 qty: 3 Unit Price: 26.5 د.ت. Item Price: 79.5 د.ت. Running Total: 444.9 د.ت.
Give Qty, Unit Price:
9 52.3
 qty: 9 Unit Price: 52.3 د.ت. Item Price: 470.7 د.ت. Running Total: 915.6 د.ت.
Give Qty, Unit Price:
0 0.0
End of order.  Order total:  915.6 د.ت.
BUILD SUCCESSFUL (total time: 54 seconds)
```

3.7 التكرار المعدد

في البرنامج السابق (أعلاه)، لا نعرف مسبقا كم من مرة سننفذ جسم المدار، فنجعل إنهاء المدار رهينة لشرط نختبره بعد كل تكرار. لكن هناك العديد من الحالات حيث نعرف مسبقا كم من مرة سننفذ جسم المدار. في هذه الحالات، نستعمل ما نسميه المدار المعدد، يكون شكله على النحو التالي:

<forStament> ::= for (<forInit>; <expression>; <forUpdate>) <statement>

<forInit> ::= <localVariableDeclaration>

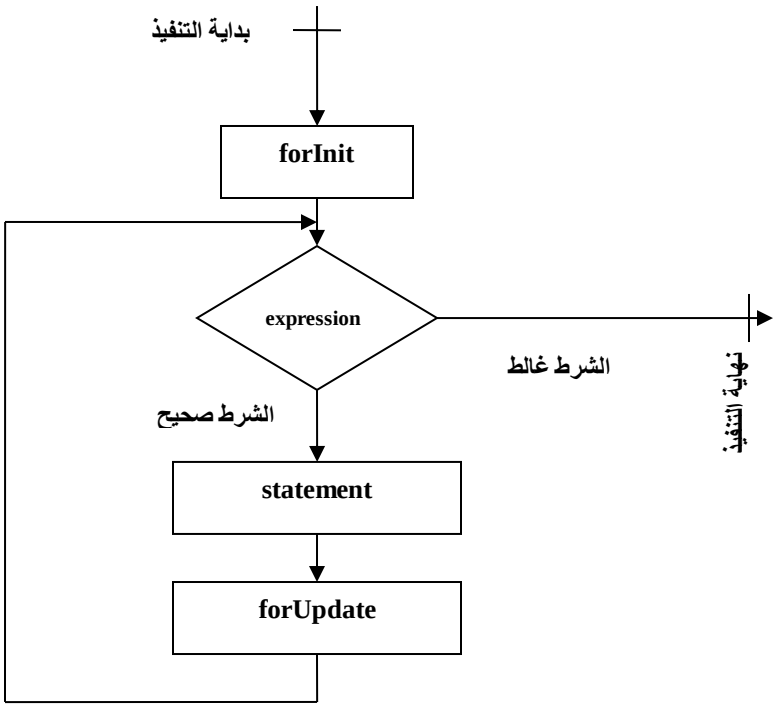
<forUpdate> ::= <statementExpression>

حيث تمثل **<localVariableDeclaration>** تصريح متغيرة محلية (عادة مصاحبة بإسناد قيمة إفتتاحية) نلقي عليها إسم متغيرة التكرار و تمثل **<statementExpression>** تعلمية إسناد تحيين متغيرة التكرار. نلقي على تعليمة التكرار من هذا الشكل إسم المدار المعدد. كما نلقي على **<expression>** إسم شرط المدار و نلقي على **<statement>** إسم جسم المدار.

تقوم جافا بتنفيذ هذه التعليمة على النحو التالي:

1. تنفذ جافا تعليمة **<forInit>**، التي تتمثل عادة في التصريح بمتغيرة التكرار و إسنادها قيمة إفتتاحية.
2. تختبر الشرط المتمثل في العبارة المنطقية **(<expression>).**
3. إذا فشلت في إحتساب القيمة المنطقية لهذه العبارة، فيفشل تنفيذ التعليمة.
4. إذا كانت قيمة هذه العبارة المنطقية سلبية، ينتهي تنفيذ التعليمة فورا بدون أي تغيير.
5. إذا كانت قيمة هذه العبارة المنطقية إجابية، تنفذ جافا تعليمة **<statement>** ثم تنفذ تعليمة **<forUpdate>** و تعود للسطر 2 اتختبر الشرط المتمثل في العبارة المنطقية **(<expression>).**
6. إذا تواصلت جافا في إختبار الشرط **(<expression>)** و تنفيذ التعليمة **<statement>** بدون نهاية (أي أن الشرط كان دائما صحيحا) فنقول أن هذه التعليمة مدار لا منتهي، و نعتبر عندئذ أن تنفيذ هذا المدار فاشل و أنه لا يقبل نتيجة نهائية.

نمثل هذه التعليمة بالرسم التالي رقم 3.7.



الرسم رقم 3.7: تنفيذ التكرار المعدد

رغبة في توضيح تعليمة المدار المعدد، ننظر من جديد في البرنامج الذي كتبناه سابقا (الجزء 6.1.7) فنعيد كتابته بواسطة هذه التعليمات، و نبرز أنها تجعل البرنامج أبسط بكثير. فنكتب:

```
int rangeMax;                                                    //  1.
                                                                 //  2.
System.out.println("Give the range in which to look for primes:  ");
rangeMax = inputWindow.nextInt();                                //  4.
boolean isPrime;  //  test for primeness                         //  5.
                                                                 //  6.
for (int primeTest=2; primeTest < rangeMax; primeTest++)         //  7.
{                                                                //  8.
   isPrime = true;                                               //  9.
   for (int j=2; j <= (primeTest/2);j++)                         // 10.
   {                                                             // 11.
      isPrime = isPrime && (primeTest%j != 0);                   // 12.
   }                                                             // 13.
   if (isPrime)                                                  // 14.
   {                                                             // 15.
      System.out.print(primeTest+", ");                          // 16.
   }                                                             // 17.
}                                                                // 18.
```

نحرض القارئ المهتم أن يتأمل في الفرق بين البرنامجين: البرنامج في الجزء 6.1.7 الذي يستعمل تعليمة التكرار ذات الشرط السابق من جهة؛ و البرنامج أعلاه من جهة أخرى، الذي يستعمل تعليمة التكرار المعدد. نلاحظ أن تعليمة

التكرار المعدد تؤدي إلى برنامج أبسط و أفصر، إذ أنه يدمج عمليات التصريح بالمتغيرة و الشرط على المتغيرة و تغيير قيمتها كلها في سطر واحد. ننفذ هذا البرنامج لنبحث عن أعداد أولية بين 2 و 150 فيسفر على النتيجة التالية.

```
run:
Give the range in which to look for primes:
150
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139,
149,
BUILD SUCCESSFUL (total time: 11 seconds)
```

4.7 تمارين

1 [س] فسروا لماذا تحافظ التعليمة التالية

```
{
   c=c+a; b=b-1;
}
```

على العبارة التالية: (**c+a*b**). أي أننا إذا قيمنا هذه العبارة قبل تنفيذ التعليمة و بعدها، نجد نفس النتيجة.

2 [س] فسروا لماذا تحافظ التعليمة التالية

```
{
   b=b-a; q=q+1;
}
```

على العبارة التالية: (**b+a*q**). أي أننا إذغ قيمنا هذه العبارة قبل تنفيذ التعليمة و بعدها، نجد نفس النتيجة.

3 [م] في الجزء 4.1.7، إستنتجنا أن **q** يمثل نتيجة قسمة **b0** إلى **a** و أن **b** يمثل باقي هذه القسمة من المعادلتين التاليتين:

$$b<a$$
$$b+a*q=b0$$

في الواقع، ينقصنا شرط لهذا الإستنتاج. ما هو هذا الشرط؟ هل هذا الشرط متوفر؟ لماذا؟

4 [س] المطلوب منكم تنفيذ المدار التالي على البيانات الواردة في الجدول التالي، مع تسجيل قيمات المتغيرات بعد كل تكرار. ثم نطلب منكم أن تحتسبوا العبارة الموجودة في العمود الأخير بالنسبة لكل سطر.

```
while (b!=0)
{
   c=c+a; b=b-1;
}
```

الجدول:

a	b	c	c+a*b
8	7	12	68
	6		
	5		
	4		
	3		

	2		
	1		
	0		

5 [س] المطلوب منكم تنفيذ المدار التالي على البيانات الواردة في الجدول التالي، مع تسجيل قيمات المتغيرات بعد كل تكرار. ثم نطلب منكم أن تحتسبوا العبارة الموجودة في العمود الأخير بالنسبة لكل سطر.

```
while (b>=a)
{
   b=b-a; q=q+1;
}
```

الجدول:

a	b	q	c+a*b
8	70	0	70

6 [م] المطلوب منكم تغيير البرنامج الوارد في الجزء 6.1.7 لكي يضبط المستخدم عدد الأعداد الأولية التي يريد طبعها. أي لا نقول مثلا "نريد الأعداد الأولية الموجودة بين 2 و 100" بل نقول "نريد قائمة ال100 عدد أولي الأولى".

7 [س] نعتبر مدارا ذات شرط سابق في الشكل التالي:

```
while t do B;
```

المطلوب منكم كتابة برنامج يقوم بنفس الوظيفة لكن لا يستعمل إلا مدارا ذات شرط لاحق.

8 [س] نعتبر مدارا ذات شرط لاحق في الشكل التالي:

```
do B while t;
```

المطلوب منكم كتابة برنامج يقوم بنفس الوظيفة لكن لا يستعمل إلا مدارا ذات شرط سابق.

9 [س] نعتبر مدارا معددا في الشكل التالي:

```
For (int i=1; i<N; i++) {B;}
```

المطلوب منكم كتابة برنامج يقوم بنفس الوظيفة لكن لا يستعمل إلا مدارا ذات شرط لاحق.

10 [س] نعتبر مدارا معددا في الشكل التالي:

```
For (int i=1; i<N; i++) {B;}
```

المطلوب منكم كتابة برنامج يقوم بنفس الوظيفة لكن لا يستعمل إلا مدارا ذات شرط سابق.

11 [س] المطلوب منكم إستعمال المدار المعدد لكتابة برنامج يطلب عددا كاملا إجابيا من المستخدم ثم يحتسب و يطبع جملة الأعداد الكاملة من 1 إلى هذا العدد.

12 [س] المطلوب منكم إستعمال المدار المعدد لكتابة برنامج يطلب عددا كاملا إجابيا من المستخدم ثم يحتسب و يطبع جملة مربعات الأعداد الكاملة من 1 إلى هذا العدد.

13 [م] حسب سلاسل تايلور المعروفة، يمكن تقريب اللوغاريثما الطبيعية بالصيغة التالية:

$$\ln(x) = \sum_{i \geq 1} (-1)^{i+1} \frac{(x-1)^i}{i}.$$

إستعملوا هذه المعادلة لكتابة برنامج يطلب من المستخدم عدد x و عدد الأطراف التي يريد أن نستعملها، فيحتسب عدد الأطراف المطلوبة ليقرب قيمة الوظيفة المعنية في x.

14 [م] حسب سلاسل تايلور المعروفة، يمكن تقريب وظيفة الأسّ بالصيغة التالية:

$$e^x = \sum_{i \geq 0} \frac{x^i}{i!}.$$

إستعملوا هذه المعادلة لكتابة برنامج يطلب من المستخدم عدد x و عدد الأطراف التي يريد أن نستعملها، فيحتسب عدد الأطراف المطلوبة ليقرب قيمة الوظيفة المعنية في x.

5.7 مصادر و مراجع

أهم مصدر للبيانات الواردة في هذا الفصل هو المقام الخاص بلغة جافا لدى شركة "سُنّ". للمزيد من الإرشادات، يمكن الرجوع لهذا المقام، أو لكتب البرمجة بجافا؛ نخص بالذكر كتاب [لويس و لفطوس، 2007].

الفصل الثامن
الجداول و المصفوفات

غالبا ما نجد نفسنا أمام مجموعة كبيرة من بيانات تختلف عن بعضها بعضا من حيث القيمة، لكن تتشاطر التأويل أو المعنى أو المعالجة. نظرا لعددها الكبير، يصعب علينا إسناد إسم مختلف لكل فرد منها؛ و نظرا لمعالجتها المشتركة نفظل أن نشير لهذه البيانات بواسطة أسم مشترك، مع إستعمال مؤشر عددي. أمثلة في مثل هذه الحالات: أعداد طلبة قسم ما في مادة ما؛ أعداد طالب ما في مجموعة مواد؛ أقصى درجات الحرارة لمجموعة من المدن في يوم ما؛ أقصى درجات الحرارة لمدينة ما في أيام شهر ما. أمثلة في عمليات قد نقوم بها على هذه البيانات: إحتساب معدل القسم في المادة المذكورة؛ إحتساب معدل طالب ما في المواد المذكورة؛ تعيين المدينة ذات أقصى درجة حرارة في اليوم المذكور؛ تعيين اليوم الذي طرأت فيه أقصى درجة حرارة في الشهر المذكور؛ إلخ. توفر لنا لغة جافا هيكل الجدول لإستيعاب مثل هذه البيانات، فندرس هذا الهيكل في هذا الفصل.

1.8 الجداول ذات البعد الواحد

1.1.8 التصريح و التأشير

يشمل التصريح بجدول في جافا البيانات التالية:

- أولا، إسم الجدول.
- ثانيا، نوع البيانات التي نعتزم خزنها في الجدول.
- ثالثا، حجم الجدول.

يقدم التصريح التالي جدولا من نوع الأعداد الحقيقية العادية (**float**) حجمه 30 و يلقي عليه إسم **temperatures**. علما أن تأشير هذا الجدول يبدأ في الصفر، بحيث أن آخر مؤشر لهذا الجدول تكون قيمته 1 دون الحجم (أي في هذه الحالة تتراوح المؤشرات بين 0 و 29).

```
float[] temperatures = new float [30];
```

تمكننا جافا من إسناد قيمة إفتتاحية للجدول في نطاق التصريح به، كما نفعل بالنسبة للمتغيرات العادية. عندئذ، نصرح بالقيمة الإفتتاحية للجدول بكتابة قائمة من القيمات بين قوسين، فيسنتج مصرف جافا حجم الجدول و قيمته. نكتب مثلا:

```
float[] studentGrades = {47, 89, 65, 78, 59, 70, 88};
```

فيسجل المصرف أننا صرحنا بجدول من نوع الأعداد الحقيقية إسمه و حجمه 7 يحتوي على البيانات المذكورة، حيث نقرأها من اليسار إلى اليمين، أي أن:

```
studentGrades[0]=47;
studentGrades[1]=89;
studentGrades[2]=65;
studentGrades[3]=78;
studentGrades[4]=59;
studentGrades[5]=70;
studentGrades[6]=88;
```

أما إذا أشّرنا هذا الجدول خارج ميدانه، أي بمؤشر أقل من 0 أو أكبر من 6، فيفشل تنفيذ هذا البرنامج. يوضح البرنامج التالي بعض العمليات التي نقوم بها على الجداول:

```
package arr;                                                        //  1.
/**                                                                 //  2.
 * @author Ali Mili and Ahmed Ferchichi                            //  3.
 */                                                                 //  4.
public class Main {                                                 //  5.
    /**                                                             //  6.
     * @param args the command line arguments                      //  7.
     */                                                             //  8.
    public static void main(String[] args) {                        //  9.
        // TODO code application logic here                         // 10.
        float[] temperatures = new float [30];                      // 11.
        float[] studentGrades = {47, 89, 65, 78, 59, 70, 88};       // 12.
                                                                    // 13.
        temperatures [0] = 25;                                      // 14.
        temperatures [1] = 23;                                      // 15.
        temperatures [2] = 27;                                      // 16.
                                                                    // 17.
        for (int i=0; i<10; i++)                                    // 18.
        {                                                           // 19.
            temperatures [3*i]= temperatures[0]+i;                  // 20.
            temperatures [3*i+1] = temperatures [1]+i;              // 21.
            temperatures [3*i+2] = temperatures [2]+i;              // 22.
        }                                                           // 23.
        for (int i=0; i<30;i++)                                     // 24.
        {                                                           // 25.
          System.out.println("day "+(i+1)+"temperature: "+temperatures[i]);
        }                                                           // 27.
        int numberOfGrades = studentGrades.length;                  // 28.
        float average = (studentGrades[0] +                         // 29.
                studentGrades[1]+                                   // 30.
                studentGrades[2]+                                   // 31.
                studentGrades[3]+                                   // 32.
                studentGrades[4]+                                   // 33.
                studentGrades[5]+                                   // 34.
                studentGrades[6])/numberOfGrades;                   // 35.
        System.out.println("Number of Grades: "+numberOfGrades);    // 36.
        System.out.println("Student Average: "+average);            // 37.
    }                                                               // 38.
}                                                                   // 39.
```

نعلق على هذا البرنامج في ما يلي:

- السطرين 11 و 12: التصريح بجدول درجات الحرارة و جدول الأعداد.
- الأسطر 14 إلى 16: إسناد درجات حرارة للثلاثة أيام الأولى.
- الأسطر 18 إلى 23: نفترض أن كل ثلاثة أيام، تكون درجة الحرارة على نفس المنوال، مع إضافة درجة.
- الأسطر 24 إلى 27: نستعرض درجات الحرارة لكامل الشهر. علما أن مؤشرات الجدول تتراوح بين 0 و 29 و أن الأشهر تتراوح بين 1 و 30، نضيف 1 للمؤشر لنتحصل على تاريخ اليوم في الشهر.
- السطر 28: نهتم بجدول الأعداد فنخزن طوله.
- الأسطر 29 إلى 35: نحتسب معدل أعداد الطالب.
- السطرين 36 و 37: نطبع عدد المواد التي أخذها الطالب، و معدله فيها.

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
day 1 temperature: 25.0
day 2 temperature: 23.0
day 3 temperature: 27.0
day 4 temperature: 26.0
day 5 temperature: 24.0
day 6 temperature: 28.0
day 7 temperature: 27.0
day 8 temperature: 25.0
day 9 temperature: 29.0
day 10 temperature: 28.0
day 11 temperature: 26.0
day 12 temperature: 30.0
day 13 temperature: 29.0
day 14 temperature: 27.0
day 15 temperature: 31.0
day 16 temperature: 30.0
day 17 temperature: 28.0
day 18 temperature: 32.0
day 19 temperature: 31.0
day 20 temperature: 29.0
day 21 temperature: 33.0
day 22 temperature: 32.0
day 23 temperature: 30.0
day 24 temperature: 34.0
day 25 temperature: 33.0
day 26 temperature: 31.0
day 27 temperature: 35.0
day 28 temperature: 34.0
day 29 temperature: 32.0
day 30 temperature: 36.0
Number of Grades: 7
Student Average: 70.85714
BUILD SUCCESSFUL (total time: 0 seconds)
```

نلاحظ أن درجات الحرارة تزداد ب1 كل ثلاثة أيام، حسب ما كتبنا في البرنامج. نستعرض الآن بعض العمليات الجارية على البرامج، لندرس كيف تقع برمجتها.

2.1.8 جمع جدول

نعتبر جدولا من نوع الأعداد الحقيقية و نهتم بجمع كل عناصر هذا الجدول (قصد إحتساب معدله، مثلا). فنكتب ما يلي:

```
float[] A = {47, 89, 65, 78, 59, 70, 88, 76, 82, 77, 92};
float sum, average;
sum = 0;
for (int i=0; i<A.length; i++)
{
   sum = sum+A[i];
}
average = sum/A.length;
```

```
System.out.println("Array Average: "+average);
```

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
Array Average: 74.818184
BUILD SUCCESSFUL (total time: 0 seconds)
```

نلاحض أن القيمة الإفتتاحية (0) لمتغيرة **sum** هي قيمة العنصر المحايد لعملية الجمع؛ أي أن جمع صفر لأي عدد يساوي العدد نفسه.

3.1.8 ضرب جدول

نعتبر جدولا من نوع الأعداد الحقيقية و نهتم بضرب كل عناصر هذا الجدول. فنكتب ما يلي:

```
float[] A = {47, 89, 65, 78, 59, 70, 88, 76, 82, 77, 92};
float product;
product = 1;
for (int i=0; i<A.length; i++)
{
   product = product*A[i];
}
System.out.println("Array Product: "+product);
```

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
Array Product: 3.4027855E20
BUILD SUCCESSFUL (total time: 0 seconds)
```

نلاحض أن القيمة الإفتتاحية (1) لمتغيرة **product** هي قيمة العنصر المحايد لعملية الضرب؛ أي أن ضرب واحد بأي عدد يساوي العدد نفسه.

4.1.8 إختبار وجود عنصر

نعتبر جدولا من نوع الأعداد الحقيقية فنهتم بالتفتيش عن عنصر في هذا الجدول، بحيث نريد فقط أن نعرف إن كان العنصر موجودا، و لا نهتم في موقعه بالذات، فنكتب ما يلي:

```
Scanner inputWindow = new Scanner(System.in);
float[] A = {47, 89, 65, 78, 59, 70, 88, 76, 82, 77, 92};
float x;
System.out.println("What Value are you Searching?");
x=inputWindow.nextFloat();
boolean found;
found=false;
for (int i=0; i<A.length; i++)
{
   found = found || (A[i]==x);
}
if (found)
{
  System.out.println("Found.");
```

```
}
else
{
   System.out.println("Not Found.");
}
```

يسفر تنفيذ هذا البرنامج بقيمة 70 على النتيجة التالية:

```
run:
What Value are you Searching?
70
Found.
BUILD SUCCESSFUL (total time: 4 seconds)
```

و يسفر تنفيذ هذا البرنامج بقيمة 23 على النتيجة التالية:

```
run:
What Value are you Searching?
23
Not Found.
BUILD SUCCESSFUL (total time: 3 seconds)
```

نلاحض أن القيمة الإفتتاحية (**false**) لمتغيرة **found** هي قيمة العنصر المحايد لعملية الخيار المنطقي (||)؛ أي أن خيار **false** بأي قيمة منطقية يساوي القيمة نفسها.

5.1.8 إختبار الترتيب

نعتبر جدولا من نوع الأعداد الحقيقية فنهتم بإختبار إن كان الجدول مرتبا أم لا. لهذا الغرض، نتأمل في كل عناصر الجدول فنقارن كل عنصر مع العنصر الموالي، فنحتسب العطف المنطقي لكل هذه النتائج، كما جاء في البرنامج التالي:

```
Scanner inputWindow = new Scanner(System.in);
final int arraySize = 8;
float[] A = new float [arraySize];
for (int i=0; i<arraySize; i++)
{
   System.out.println(" :إدخلوا العدد رقم "+(i+1));
   A[i] = inputWindow.nextFloat();
}
boolean sorted;
sorted = true;
for (int i=0; i<arraySize-1; i++)
{
   sorted = sorted && (A[i]<=A[i+1]);
}
if (sorted)
{
   System.out.println(".مرتب");
}
else
{
   System.out.println(".غير مرتب");
}
```

إذا أدخلنا سلسلة من الأعداد المرتبة، تكون النتيجة على النحو التالي:

```
run:
إدخلوا العدد رقم:: 1
12
إدخلوا العدد رقم:: 2
23
إدخلوا العدد رقم:: 3
34
إدخلوا العدد رقم:: 4
45
إدخلوا العدد رقم:: 5
56
إدخلوا العدد رقم:: 6
67
إدخلوا العدد رقم:: 7
78
إدخلوا العدد رقم:: 8
مرتب.
89
BUILD SUCCESSFUL (total time: 50 seconds)
```

إذا أدخلنا سلسلة من الأعداد غير المرتبة، تكون النتيجة على النحو التالي:

```
run:
إدخلوا العدد رقم:: 1
12
إدخلوا العدد رقم:: 2
89
إدخلوا العدد رقم:: 3
23
إدخلوا العدد رقم:: 4
78
إدخلوا العدد رقم:: 5
34
إدخلوا العدد رقم:: 6
78
إدخلوا العدد رقم:: 7
45
إدخلوا العدد رقم:: 8
54
غير مرتب.
BUILD SUCCESSFUL (total time: 50 seconds)
```

نلاحظ أن القيمة الإفتتاحية (**true**) لمتغيرة **sorted** هي قيمة العنصر المحايد لعملية العطف المنطقي (**&&**)؛ أي أن عطف **true** بأي قيمة منطقية يساوي القيمة نفسها.

6.1.8 إستخراج أصغر عدد

نهتم في هذا الجزء بإحتساب أدنى عنصر في الجدول، فنصرح بمتغيرة (**minA**) نخزن فيها أدنى قيمة، و نسند لها القيمة الإفتتاحية التي تمثل العنصر المحايد لعملية أدنى عددين، و هي العدد اللامنتهي (أي أكبر عدد حقيقي في جافا)؛ ثم نتأمل في عناصر الجدول على التوالي فنحتسب أدنى قيمة ضمن متغيرة **minA** و العنصر الحالي للجدول، فنجد البرنامج التالي:

```
final float infinity = 3.4E38f;
```

```
float[] A = {13,54,67,22,12,98,5,38,28,93,65,908,454,312};
float minA = infinity;
for (int i=0; i<A.length; i++)
{
  if (A[i]<minA){minA=A[i];}
}
System.out.println("أدنى قيمة في الجدول:"+minA);
```

الذي يسفر تنفيذه على النتيجة التالية:

```
run:
أدنى قيمة في الجدول:5.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

يمكن تمثيل التعليمة الموجودة في جسم المدار بالشكل التالي:

$$minA = minA \wedge A[i]$$

حيث يمثل رمز $\wedge$ عملية أدنى طرف، التي تنطبق على عنصرين فتستخرج أصغرهما. كما نلاحض في هذا الخصوص أن القيمة الإفتتاحية لهذه العملية هي أكبر عدد حقيقي في جافا (**infinity**)، وهو العنصر المحايد لهذه العملية: إذا إحتسبنا أصغر عددين و كان أحدهما **infinity**، فتساوي النتيجة العدد الآخر.

7.1.8 إستخراج أكبر عدد

إن البرنامج الذي يحتسب أكبر عدد في الجدول يشبه كثيرا البرنامج الذي يحتسب أصغرهم، فنكتبه في ما يلي بدون تعليق:

```
final float infinity = 3.4E38f;
float[] A = {13,54,67,22,12,98,5,38,28,93,65,908,454,312};
float maxA = -infinity;
for (int i=0; i<A.length; i++)
{
  if (A[i]>maxA){maxA=A[i];}
}
System.out.println("أقصى قيمة في الجدول:  "+maxA);
```

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
908.0  أقصى قيمة في الجدول:
BUILD SUCCESSFUL (total time: 1 second)
```

يمكن تمثيل التعليمة الموجودة في جسم المدار بالشكل التالي:

$$minA = minA \vee A[i]$$

حيث يمثل رمز ∨ عملية أقصى طرف، التي تنطبق على عنصرين فتستخرج أكبرهما. كما نلاحض في هذا الخصوص أن القيمة الإفتتاحية لهذه المتغيرة هي أصغر عدد حقيقي في جافا (**-infinity**)، وهو العنصر المحايد لهذه العملية: إذا إحتسبنا أكبر عددين و كان أحدهما **-infinity** ، فتساوي النتيجة العدد الآخر.

نلاحض أن كل البرامج التي كتبناها أعلاه (الجمع، الضرب، إختبار وجود عنصر، إختبار الترتيب، إحتساب أدنى عدد، إحتساب أقصى عدد) تمثل أمثلة خاصة من النموذج العام الآتي:

```
var = neutral;
for (int i=0; i<A.length; i++)
{
  var = var ⊗  A[i];
}
```

حيث تكون ⊗ عملية متجمعة و يكون **neutral** العنصر المحايد لها، أي

$$(a \otimes b) \otimes c = a \otimes (b \otimes c)$$
$$a \otimes neutral = a.$$

فيسفر تنفيذ هذا البرنامج على إحتساب العبارة التالية في متغيرة **var**:

$$A[0] \otimes A[1] \otimes A[2] \otimes \ldots \otimes A[A.length-1].$$

8.1.8 التفتيش عن عنصر

في هذا الجزء نهتم بالتفتيش عن عنصر في جدول، علما أننا لا نكتفي أن نختبر إن كان موجودا أم لا، بل نحتسب المؤشر الذي يوجد فيه العنصر المعني. فنكتب ما يلي:

```
Scanner inputWindow = new Scanner(System.in);
float[] A = {47, 89, 65, 78, 59, 70, 88, 76, 82, 77, 92};
float x;
System.out.println("What Value are you Searching?");
x=inputWindow.nextFloat();
int loc;
loc = -1;
for (int i=0; i<A.length; i++)
{
   if (A[i]==x) {loc=i;}
}
if (loc==-1)
{
    System.out.println("Element Not Found.");
}
else
{
    System.out.println("Element Found at Location: "+loc);
}
```

نسند لمتغيرة **loc** قيمة 1- ثم نفحص عناصر الجدول على التوالي نبحث عن قيمة **x**. كلما نجد قيمة **x** في الجدول، نخزن في متغيرة **loc** العنوان الحالي. إذا وجدنا أن قيمة **loc** لم تتغير بعد تنفيذ المدار (أي أنها تساوي 1-)، نستنتج أن **x** ليست موجودة في الجدول. و إذا وجدنا قيمة أخرى في **loc** نستنتج أنها أكبر مؤشر توجد فيه قيمة **x**.

9.1.8 إحتساب عدد الحروف في نص

نعتزم في هذا الجزء تحليل نص بالعربية لنحتسب كم من مرة يظهر كل حرف من حروف اللغة العربية (أ إلى ي). لهذا الغرض، نستعمل جدولا فيه عناصر من نوع الأعداد الكاملة، نسند عنصرا لكل حرف من حروف اللغة. فكلما نقرأ حرفا في النص المعني، نضيف 1 للعنصر الموافق لهذا الحرف. فنطرح السؤال: كيف نسند عناصر الجدول للحروف؟ لنجيب على هذا السؤال، نتأمل في تشفير الحروف العربية في جافا (حسب الملحق أ).

ص	ش	س	ز	ر	ذ	د	خ	ح	ج	ث	ت	ة	ب	ا	ئ	إ	ؤ	أ	آ	ء
1589	1588	1587	1586	1585	1584	1583	1582	1581	1580	1579	1578	1577	1576	1575	1574	1573	1572	1571	1570	1569
20	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

ي	ى	و	ه	ن	م	ل	ك	ق	ف	ـ	ڈ	ێ	ۍ	پ	ڭ	غ	ع	ظ	ط	ض
1610	1609	1608	1607	1606	1605	1604	1603	1602	1601	1600	1599	1598	1597	1596	1595	1594	1593	1592	1591	1590
41	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**

تبين هذه المائدة تسلسل الحروف العربية من ء إلى ي، كما تبين (في السطر الثاني) تشفير هذه الحروف. جدير بالذكر أن تحول الحروف إلى الأعداد اتي تمثل تشفيرها و تحول الأعداد إلى الحروف، كما يلي:

- نحول الأعداد إلى الحروف بواسطة و ظيفة (**char**).
- نحول الحروف إلى أعداد كلما إستعملنا الحرف كعدد (بإستعماله في صيغة عددية مثلا).

نعتزم إستعمال جدول من نوع الأعداد الكاملة نخزن فيه عدد المرات التي نجد فيها كل حرف. فيكون حجم هذا الجدول 42 (مؤشرات من 0 إلى 41). كيف نربط بين المؤشرات و الحروف؟

- من الحرف إلى المؤشر: إنطلاقا من حرف ما، نسميه c، نحتسب المؤشر المناسب بطرح تشفير حرف "ء" من تشفير حرف c. مثلا، نجد مؤشر حرف القاف بإحتساب تشفير حرف القاف (1602) إلا تشفير حرف الهمزة (1569)، فنجد 33.
- من المؤشر إلى الحرف: إنطلاقا من مؤشر (عدد بين 0 و 41)، نظيف له 1569، ثم نحوله لحرف. مثلا، إذا أخدنا مؤشر 25 و أضفنا له تشفير "ء" (1569) لوجدنا 1594 و هو تشفير حرف ال"غ".

بناء على هذه الإجراءات، نكتب البرنامج التالي:

```
Scanner inputWindow = new Scanner(System.in);                    //  1.
                                                                 //  2.
System.out.println("قدموا سطرا عربيا للتحليل");                  //  3.
String line = inputWindow.nextLine();                            //  4.
                                                                 //  5.
int [] letterCount = new int[42];                                //  6.
                                                                 //  7.
```

```
for (int i=0; i<42; i++)                                    //  8.
{                                                           //  9.
   letterCount[i]=0;                                        // 10.
}                                                           // 11.
for (int i=0; i<line.length(); i++)                         // 12.
{                                                           // 13.
   char ch = line.charAt(i);                                // 14.
   if ((ch>='ء')&&(ch<='ي'))                                // 15.
   {                                                        // 16.
      letterCount[ch-'ء']++;                                // 17.
   }                                                        // 18.
}                                                           // 19.
for (int i=0; i<42; i++)                                    // 20.
{                                                           // 21.
   char c = (char) (i+'ء');                                 // 22.
   System.out.print(c);                                     // 23.
   System.out.println(":  "+letterCount[i]);                // 24.
}                                                           // 25.
```

نعلق على هذا البرنامج في ما يلي:

- السطرين 3 و 4: نطلب السطر موضوح التحليل و نخزنه.
- السطر 6: نصرح بالجدول الذي نخزن فيه عدد ظهور كل حرف.
- الأسطر 8 إلى 11: نحتسب عدد ظهور كل حرف إنطلاقا من 0.
- السطر 12: نتأمل في كل حرف من حروف السطر.
- السطر 14: نستخرج الحرف الحالي و نخزنه في **ch**.
- الأسطر 15 إلى 18: إذا كان الحرف الجاري ضمن الحروف العربية الموجودة في الجدول، نضيف 1 لعنصر الجدول المناسب لهذا الحرف.
- السطر 20: نطبع محتوى الجدول.
- السطر 22: نستخرج في **c** الحرف المناسب للمؤشر الحالي **i**. لهذا الغرض، نتحصل على تشفير الحرف المعني، ثم نحول التشفير إلى حرف.
- السطر 23: نطبع الحرف المعني.
- السطر 24: نطبع عدد تكراره في النص.

إذا ننفذ هذا البرنامج كما هو، سيخصص سطرا لكل حرف. رغبة في إقتصاد السطور، و في تحسين التقديم، نقرر أن نقسم ال42 سطر إلى 7 أسطر في كل منها 6 عناصر، فنعوض المدار الأخير للبرنامج أعلاه بالنص التالي:

```
for (int i=0; i<7; i++)                                     // 20.
{                                                           // 21.
  for (int j=0; j<6; j++)                                   // 22.
  {                                                         // 23.
    int k = 6*i+j;                                          // 24.
    char c = (char) ((k)+'ء');                              // 25.
    System.out.print("\t"+c);                              // 26.
    System.out.print(":  "+letterCount[k]);                 // 27.
  }                                                         // 28.
  System.out.println();                                     // 29.
}                                                           // 30.
```

نعلق على هذا النص فيما يلي:

- السطر 20: نستعمل **i** كمؤشر السطور.
- السطر 22: نستعمل **j** كمؤشر الأعمدة.
- السطر 24: نستعمل **k** كمؤشر الحروف في الجدول.
- الأسطر 25 إلى 27: نكتب الستة حروف التابعة للسطر الحالي، مع عدد تكرارها في النص، كما فعلنا في البرنامج الأصلي، بدون تغيير السطر.
- السطر 29: نهاية السطر الجاري.

إذا ننفذ هذا البرنامج و نقدم له السطر الأول للفصل الحالي للتحليل، يسفر على النتيجة التالية:

```
run:
قدموا سطرا عربيا للتحليل
غالبا ما نجد نفسنا أمام مجموعة كبيرة من بيانات تختلف عن بعضها بعضا من حيث القيمة،
لكن تتشاطر التأويل أو
        0 :ئ         0 :إ         0 :ؤ         3 :أ         0 :آ         0 :ء
        2 :چ         1 :ث         6 :ت         3 :ة         5 :ب        12 :ا
        0 :ز         2 :ر         0 :ذ         1 :د         1 :خ         1 :ح
        0 :ظ         1 :ط         2 :ض         0 :ص         1 :ش         1 :س
        0 :?         0 :?         0 :?         0 :?         1 :غ         4 :ع
        0 :?:        6 :ل         2 :ك         1 :ق         2 :ف         0 :ـ
        5 :ي         0 :ى         3 :و         1 :ه         8 :ن         8 :م
BUILD SUCCESSFUL (total time: 7 seconds)
```

يمكن للقارئ المهتم أن يتأكد من سداد هذه النتيجة.

2.8 الجداول ذات بعدين و المصفوفات

1.2.8 تمثيل المصفوفات

تمكننا جافا من التصريح بجداول ذات بعدين إثنين، فيكون شكل التصريح مماثلا للتصريح بجدول ذات بعد واحد. مثلا إذا أردنا أن نصرح بجدول ذات بعدين يحتوي على أعداد كاملة و كان حجمه 10 على البعد الأول و 6 على البعد الثاني، فنكتب:

```
int twoD [][] = new int [10][6];
```

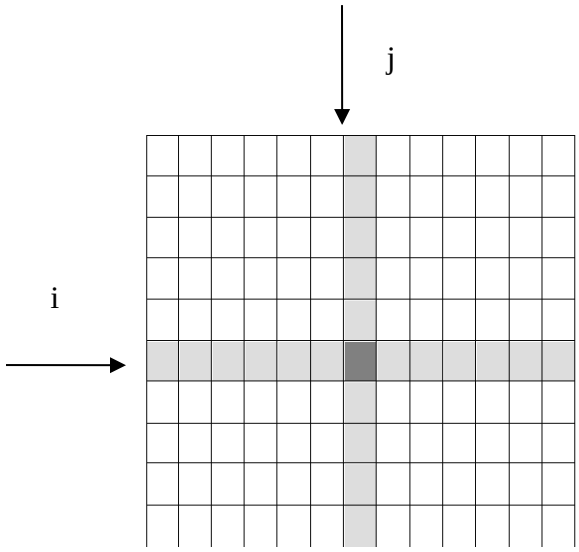
نتخيل كل جدول ذات بعدين كمصفوفة، و نتخيل أن البعد الأول يمثل أسطر المصفوفة و أن البعد الثاني يمثل أعمدتها. بالتالي، عندما نكتب عبارة مثل

```
twoD [i][j]
```

نتخيل أنها تمثل عنصر المصفوفة الموجود في تقاطع السطر رقم **i** مع العمود رقم **j**، كما يشير الرسم رقم 1.8. يمثل البرنامج التالي بعض النماذج في معالجة الجداول ذات البعدين.

```
int twoD [][]= new int [9][6];                                  //  1.
                                                                //  2.
for (int i=0; i<9; i++)                                         //  3.
{                                                               //  4.
   for (int j=0; j<6; j++)                                      //  5.
      twoD[i][j] = i*6+j;                                       //  6.
}                                                               //  7.
                                                                //  8.
```

```
System.out.println("Matrix Content:");                                    //  9.
for (int i=0; i<9; i++)                                                   // 10.
{                                                                         // 11.
   for (int j=0; j<6; j++)                                                // 12.
      System.out.print("\t"+twoD[i][j]);                                 // 13.
   System.out.println();                                                  // 14.
}                                                                         // 15.
                                                                          // 16.
System.out.println("Matrix Structure:");                                  // 17.
for (int i=0; i<9; i++)                                                   // 18.
{                                                                         // 19.
   for (int j=0; j<6; j++)                                                // 20.
      System.out.print("\t"+(1+twoD[i][j]/6)+":"+                       // 21.
                       (1+twoD[i][j]%6))                                 // 22.
   System.out.println();                                                  // 23.
}                                                                         // 24.
                                                                          // 25.
System.out.println("Inverse Matrix Structure:");                          // 26.
for (int j=0; j<6; j++)                                                   // 27.
{                                                                         // 28.
   for (int i=0; i<9; i++)                                                // 29.
      System.out.print("\t"+(1+twoD[i][j]/6)+":"+                       // 30.
                        (1+twoD[i][j]%6));                                // 31.
   System.out.println();                                                  // 32.
}                                                                         // 33.
```



الرسم رقم 1.8: تأويل لمؤشرات المصفوفة

نعلق على هذا البرنامج فيما يلي:

- السطر 1: التصريح بالجدول.
- الأسطر 3 إلى 7: خزن قيمات في خلايا الجدول.
- الأسطر 9 إلى 15: نطبع الجدول سطرا سطرا.
- الأسطر 17 إلى 24: حسب تنظيم الجدول، إذا طبعنا قسمة كل خلية ب6 و باقي كل خلية ب6، لوجدنا رقم السطر و رقم العمود التابعين للخلية.

- الأسطر 26 إلى 33: نطبع الجدول عمودا عمودا.

يسفر تنفيذ هذا البرنامج على النتيجة التالية (حيث غيرنا النص شيئا ما لنحسن التقديم):

```
run:
Matrix Content:
      0         1         2         3         4         5
      6         7         8         9         10        11
      12        13        14        15        16        17
      18        19        20        21        22        23
      24        25        26        27        28        29
      30        31        32        33        34        35
      36        37        38        39        40        41
      42        43        44        45        46        47
      48        49        50        51        52        53
Matrix Structure:
      1:1       1:2       1:3       1:4       1:5       1:6
      2:1       2:2       2:3       2:4       2:5       2:6
      3:1       3:2       3:3       3:4       3:5       3:6
      4:1       4:2       4:3       4:4       4:5       4:6
      5:1       5:2       5:3       5:4       5:5       5:6
      6:1       6:2       6:3       6:4       6:5       6:6
      7:1       7:2       7:3       7:4       7:5       7:6
      8:1       8:2       8:3       8:4       8:5       8:6
      9:1       9:2       9:3       9:4       9:5       9:6
Inverse Matrix Structure:
      1:1       2:1       3:1       4:1       5:1       6:1       7:1       8:1       9:1
      1:2       2:2       3:2       4:2       5:2       6:2       7:2       8:2       9:2
      1:3       2:3       3:3       4:3       5:3       6:3       7:3       8:3       9:3
      1:4       2:4       3:4       4:4       5:4       6:4       7:4       8:4       9:4
      1:5       2:5       3:5       4:5       5:5       6:5       7:5       8:5       9:5
      1:6       2:6       3:6       4:6       5:6       6:6       7:6       8:6       9:6
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.2.8 عمليات جبرية على المصفوفات

يمكننا مفهوم الجداول من القيام بالعديد من العمليات الجبرية على المصفوفات و على المتوجهات. نعين المتوجهات كمجموعة مرتبة من الأعداد الحقيقية؛ و نفرق شكليا بين المتوجهات الأفقية و المتوجهات العمودية. مثلا، إذا تأملنا في مصفوفة، نؤل كل سطر لها كمتوجهة أفقية و كل عمود لها كمتوجهة عمودية. نناقش على التوالي:

- تصريف متوجهين،
- تصريف متوجه بمصفوفة،
- تصريف مصفوفتين.

1.2.2.8 تصريف متوجهين

تعيين 1.8. التصريف العددي. نعتبر متوجهين A و B. نعتبر أنهما متساويان الحجم و نمثل حجمهما المشترك برمز N، كما نعتبر أن A متوجه عمودي و B متوجه أفقي. نلقي إسم التصريف العددي لمتوجه A بمتوجه B ، للصيغة التالية.

$$\sum_{i=1}^{N} A[i] \times B[i].$$

نمثل التصريف العددي لـA ب B بالعبارة التالية:

$$A \bullet B.$$

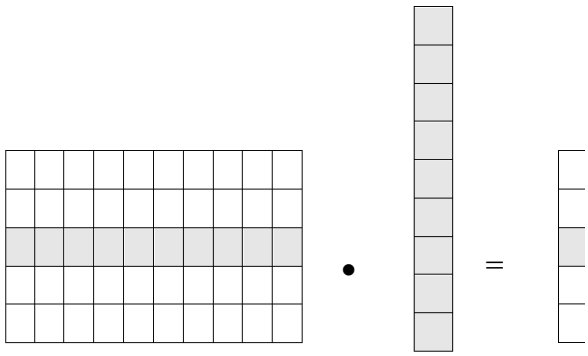
يحتسب البرنامج التالي التصريف العددي لمتوجهي A و B فيضع النتيجة في متغيرة c.

```
final int N=10;
float A [] = new float [N];
float B [] = new float [N];
float c;
c=0;
for (int i=0; i<N; i++) {c=c+A[i]*B[i];}
```

2.2.2.8 تصريف متوجه بمصفوفة

تعيين 2.8. تصريف متوجه بمصفوفة. نعتبر مصفوفة A عدد أسطرها M و عدد أعمدتها N و نعتبر متوجها عموديا V عدد عناصره N. نعين تصريف المصفوفة A بالمتوجه V بأنه المتوجه العمودي عدد عناصره M, حيث أن العنصر رقم i، أين i بين 1 و M ، يتمثل في التصريف العددي لسطر A رقم i مع المتوجه العمودي V.

يوضح الرسم رقم 2.8 عملية التصريف بين مصفوفة و متوجه.



الرسم رقم 2.8: تصريف متوجه بمصفوفة

يحتسب البرنامج التالي تصريف المتوجه العمودي V بالمصفوفة A فيضع النتيجة في المتوجه العمودي W.

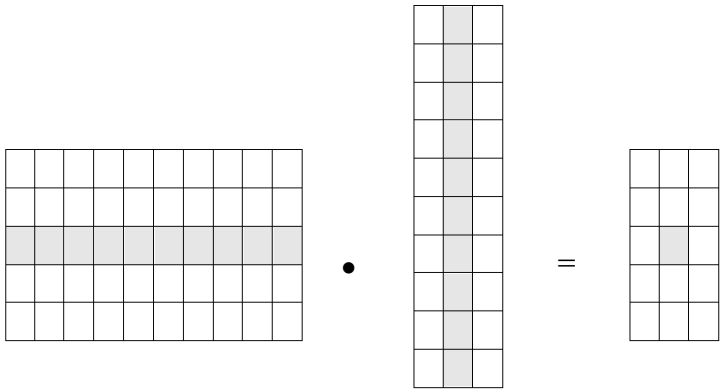
```
final int N=10;
final int M=5;

float A [][] = new float [M][N];
float V [] = new float [N];
float W [] = new float [M];

for (int i=0; i<M; i++)
{
    float val=0;
    for (int j=0; j<N; j++)
    {
        val = val + A[i][j]*V[j];
```

```
    }
    W[i]=val;
}
```

3.2.2.8 تصريف مصفوفة بمصفوفة

تعيين 3.8. تصريف مصفوفة بمصفوفة. نعتبر مصفوفة A عدد أسطرها M و عدد أعمدتها N و نعتبر مصفوفة B عدد أسطرها N و عدد أعمدتها L. نعين تصريف المصفوفة A بالمصفوفة B بأنه المصفوفة C, عدد أسطرها M و عدد أعمدتها L ، حيث أن عنصر المصفوفة في تقاطع السطر رقم i، أين i بين 1 و M ، و العمود رقم j، اين j بين 1 و L، يتمثل في التصريف العددي لسطر A رقم i مع عمود B رقم j.

يوضح الرسم رقم 3.8 عملية التصريف بين مصفوفتين.



الرسم رقم 3.8: تصريف مصفوفة بمصفوفة

يحتسب البرنامج التالي تصريف المصفوفة A بالمصفوفة B فيضع النتيجة في المصفوفة C.

```
final int N=10;                                                    //  1.
final int M=5;                                                     //  2.
final int L=3;                                                     //  3.
                                                                   //  4.
float A [][] = new float [M][N];                                   //  5.
float B [][] = new float [N][L];                                   //  6.
float C [][] = new float [M][L];                                   //  7.
                                                                   //  8.
for (int i=0; i<M; i++)                                            //  9.
{                                                                  // 10.
   for (int j=0; j<N; j++)                                         // 11.
        A[i][j] = i*6+j;                                           // 12.
}                                                                  // 13.
                                                                   // 14.
for (int i=0; i<N; i++)                                            // 15.
{                                                                  // 16.
   for (int j=0; j<L; j++)                                         // 17.
```

```
       B[i][j] = 1/(float)(1+i*6+j);                                    // 18.
}                                                                       // 19.
                                                                        // 20.
for (int i=0; i<M; i++)                                                 // 21.
{                                                                       // 22.
   for (int j=0; j<L; j++)                                              // 23.
   {                                                                    // 24.
      float val=0;                                                      // 25.
      for (int k=0; k<N; k++)                                           // 26.
      {                                                                 // 27.
          val = val + A[i][k]*B[k][j];                                  // 28.
      }                                                                 // 29.
      C[i][j]=val;                                                      // 30.
    }                                                                   // 31.
}                                                                       // 32.
System.out.println("A:");                                               // 33.
for (int i=0; i<M; i++)                                                 // 34.
{                                                                       // 35.
    for (int j=0; j<N; j++)                                             // 36.
    System.out.print("\t"+A[i][j]);                                    // 37.
    System.out.println();                                               // 38.
}                                                                       // 39.
System.out.println("B:");                                               // 40.
for (int i=0; i<N; i++)                                                 // 41.
{                                                                       // 42.
   for (int j=0; j<L; j++)                                              // 43.
   System.out.print("\t"+B[i][j]);                                     // 44.
   System.out.println();                                                // 45.
}                                                                       // 46.
System.out.println("C:");                                               // 47.
for (int i=0; i<M; i++)                                                 // 48.
{                                                                       // 49.
   for (int j=0; j<L; j++)                                              // 50.
   System.out.print("\t"+C[i][j]);                                     // 51.
   System.out.println();                                                // 52.
}                                                                       // 53.
```

نعلق على هذا البرنامج بإيجاز، في ما يلي:

- الأسطر 1 إلى 7: التصريح بالمصفوفات.
- الأسطر 9 إلى 19: إسناد المصفوفات العنية قيماتا إفتتاحية.
- الأسطر 21 إلى 32: إحتساب المصفوفة C كنتيجة تصريف المصفوفة A بالمصفوفة B.
- الأسطر 33 إلى 53: طبع الثلاثة مصفوفات.

يسفر تنفيذ هذا البرنامج على النتيجة التالية (مع تغيير النص لحسن التقديم).

```
run:
A:
 0.0     1.0     2.0     3.0     4.0     5.0     6.0     7.0     8.0     9.0
 6.0     7.0     8.0     9.0    10.0    11.0    12.0    13.0    14.0    15.0
12.0    13.0    14.0    15.0    16.0    17.0    18.0    19.0    20.0    21.0
18.0    19.0    20.0    21.0    22.0    23.0    24.0    25.0    26.0    27.0
24.0    25.0    26.0    27.0    28.0    29.0    30.0    31.0    32.0    33.0
B:
        1.0             0.5             0.33333334
        0.14285715      0.125           0.11111111
        0.07692308      0.071428575     0.06666667
        0.05263158      0.05            0.04761905
        0.04            0.03846154      0.037037037
        0.032258064     0.03125         0.030303031
```

```
        0.027027028       0.02631579       0.025641026

        0.023255814       0.022727273      0.022222223
        0.020408163       0.02             0.019607844
        0.018181818       0.017857144      0.01754386
C:
        1.4277428        1.3656532       1.3111241
        10.028997        6.783895        5.577636
        18.630257        12.202137       9.844146
        27.231514        17.62038        14.110657
        35.832764        23.038622       18.377167
BUILD SUCCESSFUL (total time: 0 seconds)
```

إن كل عمليات التصريف التي نظرنا فيها إلى الآن (بين متوجهين، أو بين متوجه و مصفوفة، أو بين مصفوفتين) أمثلة خاصة من التصريف بين مصفوفتين، علما أننا نؤوّل المصفوفة الأولى كمجموعة أسطر و المصفوفة الثانية كمجموعة أعمدة. فتكون النتيجة مصفوفة عدد أسطرها عدد أسطر الطرف اليساري و عدد أعمدتها عدد أعمدة الطرف الثاني. علما أن مصفوفة ذات سطر واحد تمثل متوجها أفقيا و أن مصفوفة ذات عمود واحد تمثل متوجها أفقيا؛ ثم أن مصفوفة ذات سطر واحد و عمود واحد تمثل عددا بسيطا.

3.8 جداول ذات ثلاثة أبعاد

تمكننا جافا من التصريح بجداول ذات ثلاثة أبعاد، و يكون هذا التصريح على نفس المنوال الذي درسناه إلى حد الآن. لذبين إستعمال مثل هذا الجدول، نأخذ مثالا بسيطا. علما أن هذا الفصل تم تحريره في شهر رمضان (سنة 2008 ميلادي، 1429 هجري)، نأخذ مثالا يتعلق بهذا الشهر. نهتم بإمساكية رمضان، و هي رزنامة تضبط بالنسبة لعدد من المدن، أوقات الإمساك و أوقات الإفطار. نضيف لها، للمزيد من تثرية المثال، أوقات الصلاة الخمسة بالنسبة لكل مدينة و بالنسبة لكل يوم. فنكتب ما يلي:

```
final int TUNIS = 0;                                                    //  1.
final int NABEUL = 1;                                                   //  2.
final int SOUSSE = 2;                                                   //  3.
final int SFAX = 3;                                                     //  4.
final int NBCITIES = 4;                                                 //  5.

final int IMSAK = 0;  final int SOBH = 0;                               //  6.
final int DHOHR = 1;                                                    //  7.
final int ASR = 2;                                                      //  8.
final int MAGHRIB = 3;  final int IFTAR = 3;                            //  9.
final int ISHAA = 4;                                                    // 10.
final int NBPRAYERS = 5;                                                // 11.

final int NBDAYS = 30;                                                  // 12.

int PrayerTimes [][][] = new int [NBCITIES][NBPRAYERS][NBDAYS];         // 13.

PrayerTimes [NABEUL][IMSAK][11]= 518;                                   // 14.
PrayerTimes [NABEUL][IFTAR][11]= 1932;                                  // 15.

ayerTimes [SFAX][IFTAR][26]= 1910;                                      // 16.
PrayerTimes [SFAX][IMSAK][27]= 535;                                     // 17.
```

نعلق على هذه التصريحات في ما يلي:

- الأسطر 1 إلى 5: بما أننا نعتزم إستعمال المدن كمؤشرات في الجدول، نسند لها أعداد كاملة من صفر إلى 3، و نخزن عدد المدن في المستقرة **NBCITIES**.

- الأسطر 6 إلى 11: بما أننا نعتزم إستعمال أوقات الصلاة كمؤشرات في الجدول، نسند لها أعداد كاملة من صفر إلى 4، و نخزن عدد أوقات الصلاة في المستقرة **NBPRAYERS**. بما أن الإمساك يصير في الصبح و الإفطار يصي في المغرب، فنعين مستقرات تشير للإمساك و الإفطار و نسند لها القيمات المناسبة. بالتالي، نشير لنفس الوقت كوقت إفطار أو وقت المغرب، و نشير لنفس الوقت كوقت إمساك أو وقت صبح.
- السطر 12: نصرح بمستقرة تخزن عدد الأيام في الشهر (على الأكثر).
- السطر 13: نصرح بجدول ذات ثلاثة أبعاد، نستعمل أولهم لتمثيل المدن و ثانيهم لتمثيل أوقات الصلاة و الثالث لتمثيل أيام الشهر.
- السطرين 14 و 15: نسند أوقات الإمساك و الإفطار لمدينة نابل لليوم الحادي عشر.
- السطر 16: نسند وقت الإفطار لمدينة صفاقس في اليوم السادس و العشرين من رمضان.
- السطر 17: نسند وقت الإمساك لمدينة صفاقس في اليوم السابع و العشرين من رمضان.

يمكننا إستعمال هذا الجدول للقيام بالعديد من التحاليل، مثلا:

- إذا أردنا أن نطبع أوقات الصلاة بالنسبة لمدينة معينة (سوسة، مثلا) خلال شهر رمضان (بإعتبار أنه يدوم ثلاثين يوما)، فنكتب:

```
for (int day=0; day<NBDAYS; day++)
{
   for (int prayer = SOBH; prayer<NBPRAYERS; prayer++)
   {
      System.out.print(PrayerTimes[SOUSSE][prayer][day]);
   }
   System.out.println();
}
```

- إدا أردنا أن نطبع أوقات الإفطار بالنسبة لكل المدن و كل أيام رمضان، فنكتب:

```
for (int day=0; day<NBDAYS; day++)
{
    for (int city=TUNIS; city<NBCITIES; city++)
    {
       System.out.print(PrayerTimes[city][IFTAR][day]);
    }
    System.out.println();
}
```

- إذا أردنا أن نطبع أوقات الصلاة بالنسبة لكل المدن يوم السابع و العشرين من رمضان، فنكتب:

```
for (int city=TUNIS; city<NBCITIES; city++)
{
    for (int prayer=SOBH; prayer<NBPRAYERS; prayer++)
    {
       System.out.print(PrayerTimes[city][prayer][27]);
    }
    System.out.println();
}
```

4.8 تمارين

1 [م/س] تأملوا في البرنامج الوارد في الجزء 4.1.8 و غيروه بحيث يتوقف فحص الجدول ما إن نجد قيمة **x** في الجدول. ملاحضة: يجب تعويض المدار المعدد بمدار ذات شرط سابق أو مدار ذات شرط لاحق.

2 [م/س] تأملوا في البرنامج الوارد في الجزء 5.1.8 و غيروه بحيث يتوقف فحص الجدول ما إن نجد عنصرين متتاليين ليسا في الترتيب المطلوب. ملاحضة: يجب تعويض المدار المعدد بمدار ذات شرط سابق أو مدار ذات شرط لاحق.

3 [م] أكتبوا برنامجا يفتش عن عنصر في جدول فيحتسب عدد المرات التي يظهر فيها في الجدول.

4 [م] أكتبوا برنامجا يفتش عن عنصر في جدول فيحتسب أكبر مؤشر يوجد فيه العنصر.

5 [م] أكتبوا برنامجا يفتش عن عنصر في جدول فيحتسب أصغر مؤشر يوجد فيه العنصر.

6 [س] في البرنامج الذي قدمناه في الجزء 9.1.8، صففنا النتيجة إلى سبعة أسطر ذات ستة أعمدة بواسطة مدارين معددين. يمكن التحصل على نفس النتيجة بمدار واحد، إذا قررنا أن نرجع للسطر كلما وجدنا أن مؤشر الجدول ينقسم ل6. المطلوب منكم كتابة برنامج يشخص هذا الحل.

7 [م] المطلوب منكم أن تحللوا هذا البرنامج و أن تبينوا ما هي نتيجة تنفيذه.

```
For (int i=0; i<9; i++)
{
   for (int j=0; j<6; j++)
      System.out.println(i*6+j);
}
```

8 [م] المطلوب منكم أن تتأملوا في المعادلة التالية `(k=i*6+j)` حيث أن `j` يتراوح بين الصفر و 5، و أن تبينوا ما هي قيمة العبارتين التاليتين

k/6
k%6.

9 [م] إستعملوا هيكل البيانات الذي قدمناه في الجزء 3.8 لكتابة البرامج التالية، بعد إتمام جميع البيانات التابعة لجدول أوقات الصلاة.

- برنامج يطبع مدة الصيام لمدينة سوسة بالنسبة لكل يوم من أيام رمضان.
- برنامج يحتسب اليوم ذات أطول فترة صيام بالنسبة لكل مدينة.
- برنامج يطبع جميع أوقات الصلاة يالنسبة لمدينة صفاقس في اليوم الصابع و العشرين من رمضان.
- برنامج يطبع أوقات الإمساك بالنسبة لك المدن و كل الأيام.

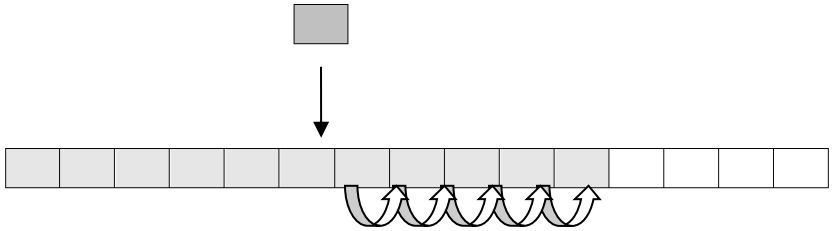
الفصل التاسع
هياكل البيانات

في الفصول السابقة درسنا أنواع بيانات بدائية (مثل الأعداد و الحروف و المنطقيات، إلخ)، ثم درسنا نوعا من البيانات المهيكلة، و هو هيكل الجداول. من خصائص الجداول يجدر بالذكر أنها ثابتة، أي أن حجمها يُضبط عند التصريح بها و لا يتغير بصفة حركية في طور التنفيذ. نهتم في هذا الفصل بهياكل بيانات يتغير حجمها أو شكلها بصفة حركية حسب الحاجة في طور التنفيذ.

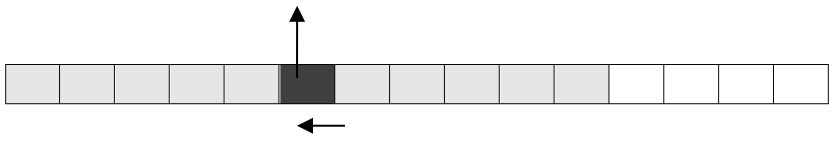
1.9 الهياكل الخطية

نفترض أننا نهتم بقائمة من العناصر، كقائمة تلاميذ في مؤسسة تعليمية أو قائمة موظفين في إدارة أو قائمة بضائع في مخزن، إلخ. و نفترض أن هذه العناصر معرّفة بواسطة مفاتيح مرتبة، كرقم التلميذ أو رقم الموظف أو رقم البضاعة. قد يفكر القارئ أن هيكل الجداول يمكننا من خزن هذه العناصر و معالجتها عند الحاجة. لكن هناك عدد من الأسباب تجعل الجداول غير لائقة لهذا الغرض:

- عادة، لا نعلم مسبقا عدد العناصر المعنية. إذا صرحنا بجدول ذات حجم كبير، قد لا نستعمل منه إلا القليل. و إذا صرحنا بجدول ذات حجم صغير، قد نحتاج لأكبر منه في وقت ما خلال التنفيذ.
- حتى لو إفترضنا أن عدد العناصر معروف و لا يتغير كثيرا، فتبقى مشكلة عمليات الإضافة و النزع في الهيكل، التي نطرحها كما يلي: إذا كانت عناصر الجدول مرتبة و أردنا أن نضيف عنصرا آخر مع المحافضة على الترتيب، فقد نضطر لتحويل عدد كبير من العناصر؛ و كذلك بالنسبة لعمليات النزع (الرجوع للرسم رقم 1.9). إذا كان التطبيق يتطلب عددا كبيرا من عمليات الإضافة و النزع، فيكون هذا الحل أكثر تكاليفا.



الإضافة



النزع

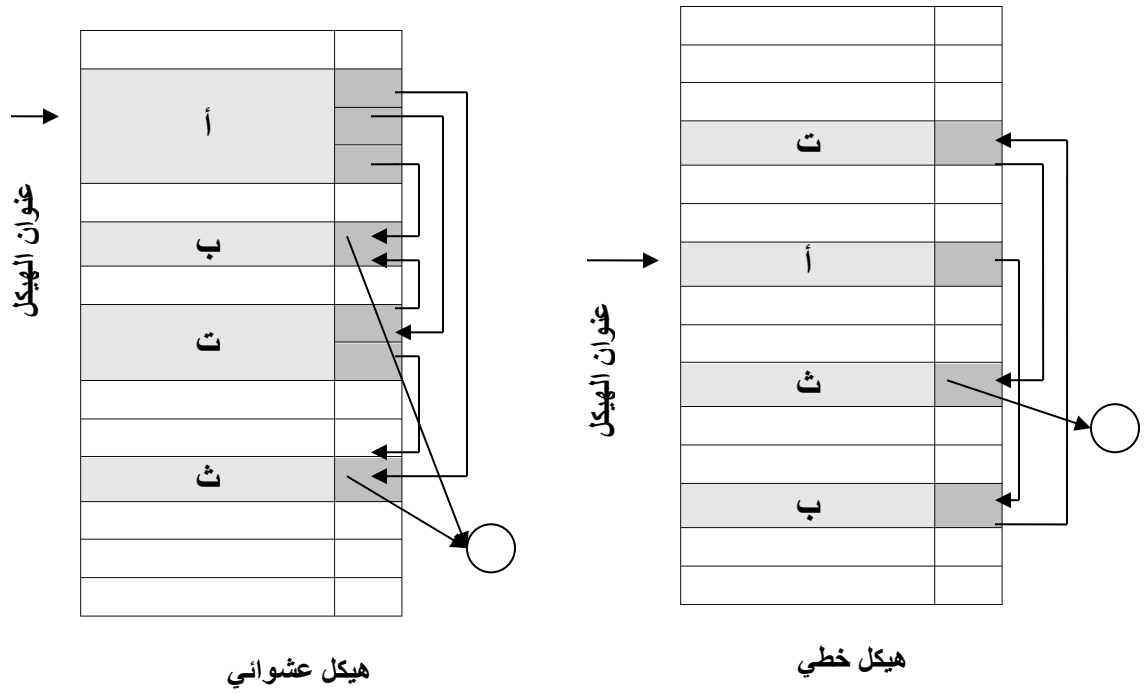
لهذه الأسباب، نهتم بهياكل بيانات دون الجدول، نشترط أن تلبي المتطلبات التالية:

- لا تتطلب تصريحا مسبقا بعدد العناصر،

- يتغير حجمها حسب متطلبات البرنامج عند التنفيذ،
- تسترجع ذاكرة الحاسوب الفضاء الذي نفرغ من إستعماله.

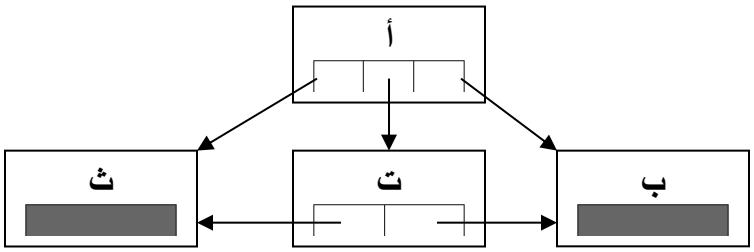
لهذا الغرض، توفر لغة جافا مفهوم الموجّه, و هو عنوان رمزي لهيكل بيانات. لندرك هذا المفهوم، نتخيل أن ذاكرة الحاسوب تتمثل في جدول من المواقع نخزن فيها بيانات، فيمكن لنا أن نمثل هياكل بيانات في أي شكل شئنا، ما دمنا نتمكن من ربط عناصر ببعضها بواسطة عناوينها؛ تأملوا في الرسم رقم 2.9.

- في الرسم الموجود على اليمين، نبين كيف يمكن تمثيل قائمة خطية متكونة من عناصر أ، ب، ت، ث. لو إستعملنا جدولا لخزن نفس البيانات، لوضعنا العناصر أ، ب، ت و ث في مواقع متتالية في الجدول، مما يجعلهم في مواق متتالية في ذاكرة الحاسوب. أما في التشكيلة المبنية على الموجّهات، فموقع العناصر و الترتيب بينهم يختلفان عن بعضهما: يقع خزن العناصر أينما وجدنا فضاء في الذاكرة؛ و يقع الترتيب بين العناصر بواسطة الموجهات. إنطلاقا من عنوان الهيكل، نجد عنصر أ؛ نتابع الموجه المنطلق من هذا العنصر فنجد أنه يشير لعنصر ب؛ نتابع الموجه المنطلق من عنصر ب فنجد أنه يشير لعنصر ت؛ نتابع الموجه المنطلق من عنصر ت فنجد أنه يشير لعنصر ث؛ نتابع الموجه المنطلق من عنصر ث فنجد أنه يشير لنهاية القائمة.



الرسم رقم 2.9: هياكل بيانات مبنية على الموجهات

- أما عن الرسم الموجود على اليسار، فهو يمثل مخططا عشوائي الشكل، يتمثل في عقد و علاقات بين العقد، نمثلها بواسطة الموجهات. إنطلاقا من عنوان الهيكل، نجد عنصر أ و نتفرع إلى ثلاثة فروع؛ يتوجه الفرع الأول لعنصر ب و الفرع الثاني لعنصر ت و الفرع الثالث لعنصر ث؛ يتفرع عنصر ت فيتوجه أول فرع لعنصر ب و الفرع الثاني لعنصر ث؛ أم عن عنصري ب و ث، فهما يشيران لنهاية الهيكل. نجد في الرسم رقم 3.9 (حيث نمثل باللون الأسود الموجهات الفارغة) تمثيلا رسوميا للهيكل الذي نمثله على يسار الرسم رقم 2.9.



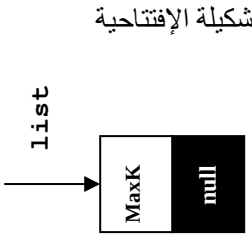
الرسم رقم 3.9: الهيكل العشوائي

نخصص بقية هذا الجزء لبرمجة الهيكل الخطي في لغة جافا، فدرس على التوالي كيف نبرمج التشكيلة الإفتتاحية، ثم كيف نقوم بعملية إضافة ثم كيف نقوم بعملية نزع، ثم نجرب البرنامج الناتج عن هذه العمليات لنوضح تنفيذه.

1.1.9 التصريح و التشكيلة الإفتتاحية

لكي نتمكن من بناء قائمة متسلسلة بواسطة الموجهات، يجب أن نصرح أولا بهيكل بيانات يمثل عقدة فردية من التسلسل الخطي. تتكون هذه العقدة من ثلاثة حقول: حقل يحمل المعرف الوحيد لكل عقدة؛ حقل يحمل موجها للعقدة الموالية في التسلسل؛ و حقلا يحمل بيانات إضافية تتعلق بالعقدة (مثلا، إذا كانت كل عقدة تمثل طالبا في جامعة ما، فنخزن في هذا الحقل بيانات عن الطالب مثل إسمه و عنوانه و أعداده إلى آخره). في نطاق هذه الدراسة، نتجاهل الحقل الثالث.

عملا بهذا الإختيار، نصرح بهيكل ذات حقلين، و نتخذ حقلا من نوع الأعداد الكاملة كمعرف وحيد. أما عن التسلسل، فنشير له بأول عنصر له، لأننا نتمكن من الوصول لكل العناصر إنطلاقا من العنصر الأول. فنطرح إذا مشكلة: ما هي قيمة المفتاح / المعرف الوحيد التي نسندها للعقدة الإفتتاحية، علما أن العقد مرتبة صاعدا حسب المعرف الوحيد. لأسباب ستتضح في الأجزاء الموالية، نقترح أن يكون مفتاح العنصر الإفتتاحي أكبر قيمة ممكنة للمعرف الوحيد. بناء على هذه القرارات، نكتب النص التالي في لغة جافا، فنعلق عليه بإيجاز فيما يلي:



- الأسطر 5 إلى 9: التصريح بالعقدة.
- السطر 17: التصريح بمستقرة تمثل أكبر معرف وحيد.
- السطر 18: التصريح بمتغيرة إسمها `list` نخزن فيها موجها يشير للعقدة الإفتتاحية.
- السطر 19: نسند لمفتاح هذه العقدة قيمة أكبر معرف وحيد؛ علما أن هذه العقدة تبقى دائما آخر عقدة في التسلسل (الآن هي أيضا أول عقدة لأن التسلسل لا يشمل إلا عقدة واحدة).
- السطر 20: نسند قيمة `null` للموجه العقدة الموالية لنبين أن لا توجد عقدة موالية، أي هذه آخر عقدة في التسلسل.

```
package ds;                                                    //  1.
/**                                                            //  2.
 * @author Ali Mili and Ahmed Ferchichi                       //  3.
 */                                                            //  4.
class node                                                     //  5.
{                                                              //  6.
    int key;                                                   //  7.
    node link;                                                 //  8.
}                                                              //  9.
                                                               // 10.
```

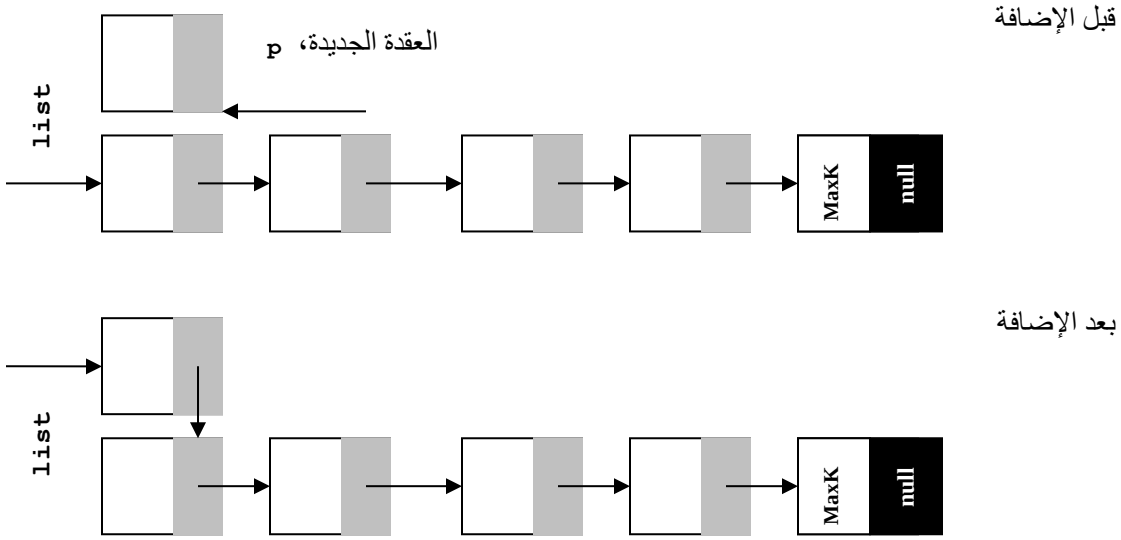
```
public class Main {                                                    // 11.
    /**                                                                // 12.
     * @param args the command line arguments                         // 13.
     */                                                               // 14.
    public static void main(String[] args) {                          // 15.
        // TODO code application logic here                           // 16.
        final int MAXKEY = 2147483647;                                // 17.
        node list = new node();                                       // 18.
        list.key = MAXKEY;                                            // 19.
        list.link = null;                                             // 20.
    }                                                                 // 21.
}                                                                     // 22.
```

2.1.9 إضافة عنصر

نفرق بين نوعين من عمليات الإضافة:

- الإضافة في أول التسلسل،
- الإضافة في وسط التسلسل.

يتمثل الفرق بينهما في أن الإضافة في أول التسلسل تغير قيمة `list` بينما الإضافة في آخر التسلسل تغير روابط بين عقد متتالية في وسط التسلسل. يمثل الرسم رقم 4.9 عملية الإضافة عندما يكون العنصر الجديد في أول التسلسل فيوضح على التوالي تشكيلة التسلسل قبل عملية الإضافة و بعدها. تقع إضافة عنصر في أول التسلسل كلما كان مفتاح العنصر الجديد أصغر من مفتاح أول عنصر في التسلسل. تطرء هذه الحالة بالخصوص عندما نضيف أول عنصر بعد عملية الإفتتاح

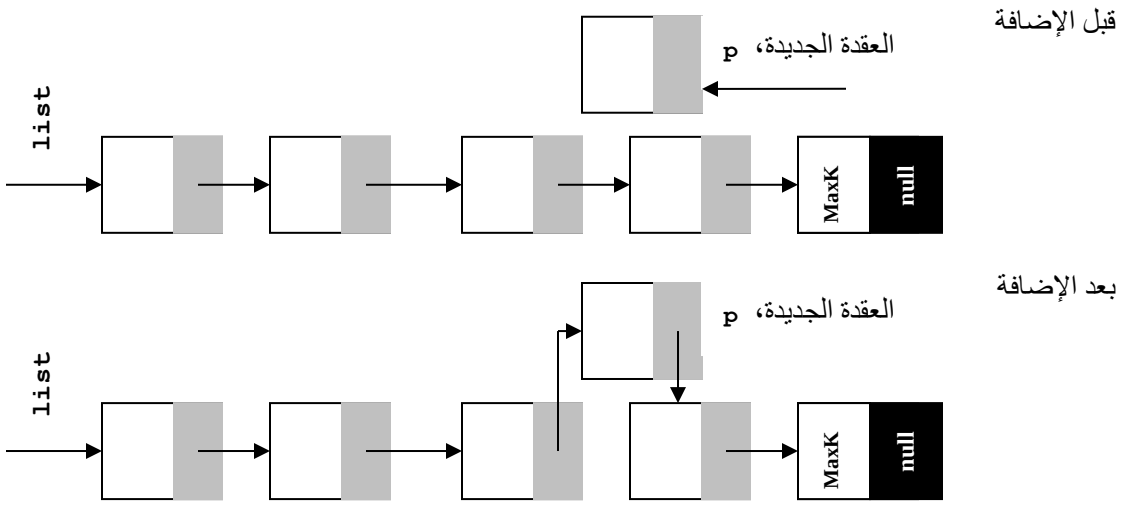


الرسم رقم 4.9: إضافة عنصر في أول التسلسل

في لغة جافا، تتم هذه العملية بربط العنصر الجديد برأس التسلسل و تحويل رأس التسلسل ليشير للعنصر الجديد (الذي يصبح أول عنصر في التسلسل الجديد). نكتبها كما يلي، حيث يشير `p` لمؤشر العنصر الجديد.

```
p.link = list;
list = p;
```

أما عن الإضافة في وسط التسلسل، فهي تتمثل في إيجاد العنصرين المجاورين للعنصر الجديد و تحديث الروابط من العنصر السابق إلى العنصر الجديد و من العنصر الجديد، إلى العنصر اللاحق. نبين في الرسم رقم 6.9 تفاصيل هذه العملية، فنمثل تشكيلة التسلسل قبل عملية الإضافة و بعدها.



الرسم رقم 5.9: إضافة عنصر في وسط التسلسل

بالطبع، أصعب خطوة في هذا الأسلوب هي تعيين المكان الذي تقع فيه إضافة العنصر الجديد. يتعين هذا المكان بعنوان عنصرين إثنين: العنصر الذي يسبق العنصر الجديد في ترتيب المفاتيح، و العنصر الذي يليه. لذا، نحتاج لموجهين إثنين في عملية التفتيش عن مقام العنصر الجديد. نكتب نص هذه العملية في جافا كما يلي، حيث يشير p لمؤشر العنصر الجديد.

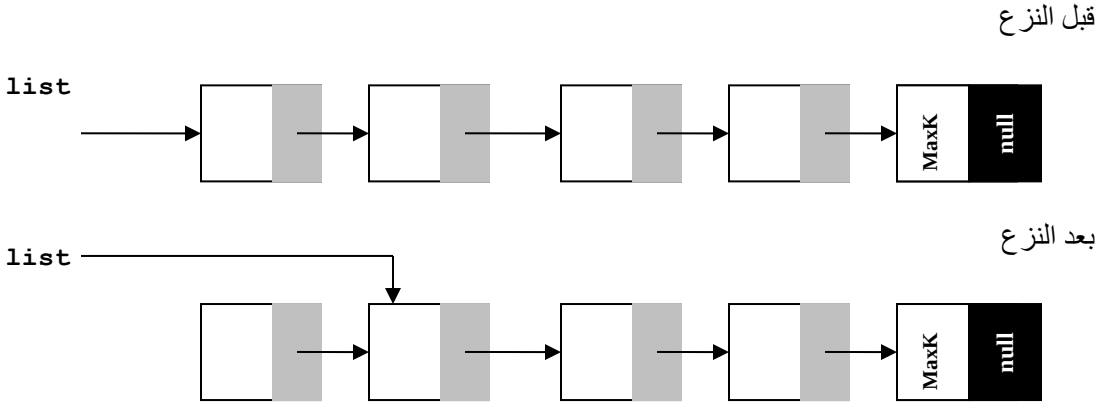
```
node front, back;  //   موجهان يشيران للعنصر السابق و العنصر اللاحق للعنصر الجديد
front = list.link; //              يشير الموجه الأمامي للعنصر الثاني من التسلسل
back = list;       //                يشير الموجه الخلفي للعنصر الأول من التسلسل

while (p.key > front.key)
{
   back = front;          //                    نقدم المؤشر الخلفي عبر التسلسل.
   front = front.link;    //                    نقدم المؤشر الأمامي عبر التسلسل.
}
back.link = p;      //              إسناد الروابط حسب ما جاء في الرسم رقم 5.9
p.link = front;     //              إسناد الروابط حسب ما جاء في الرسم رقم 5.9
```

إن وجود العنصر ذات أقصى مفتاح في آخر التسلسل يمكننا من التأكد أن شرط المدار قد يصبح غالطا في وقت ما، على الأقل عندما يصل موجه front لعنوان العنصر الأخير. نقول عن هذا العنصر الأخير (الذى لا يخزن بيانات هامة بل يمكننا من تبسيط خوارزمية البحث) أنه يلعب دور حارس.

3.1.9 نزع عنصر

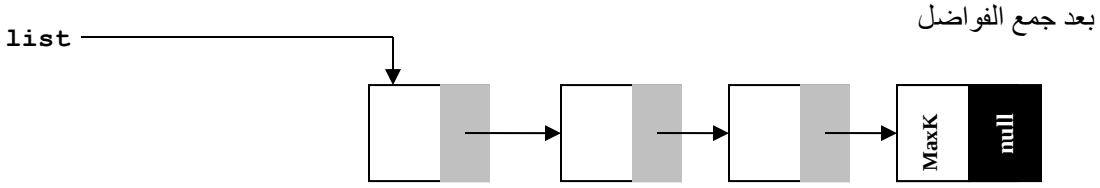
تتمثل عملية نزع عنصر في التفتيش عن هذا العنصر في التسلسل حسب مفتاحه و طرحه من التسلسل. كما فعلنا في عملية الإضافة، نفرق في هذه العملية بين النزع في وسط التسلسل و النزع في أوله، لأن تنفيذ هذه العملية يختلف جذريا

حسب موقع العنصر موضوع النزع. نعتبر أنه لا يقع نزع العنصر ذات أقصى مفتاح، **MAXKEY**. يمثل الرسم 6.9 عملية النزع عندما تنطبق على أول عنصر في التسلسل، فيبين تشكيلة التسلسل قبل عملية النزع و بعدها.



الرسم رقم 6.9: نزع عنصر في أول التسلسل

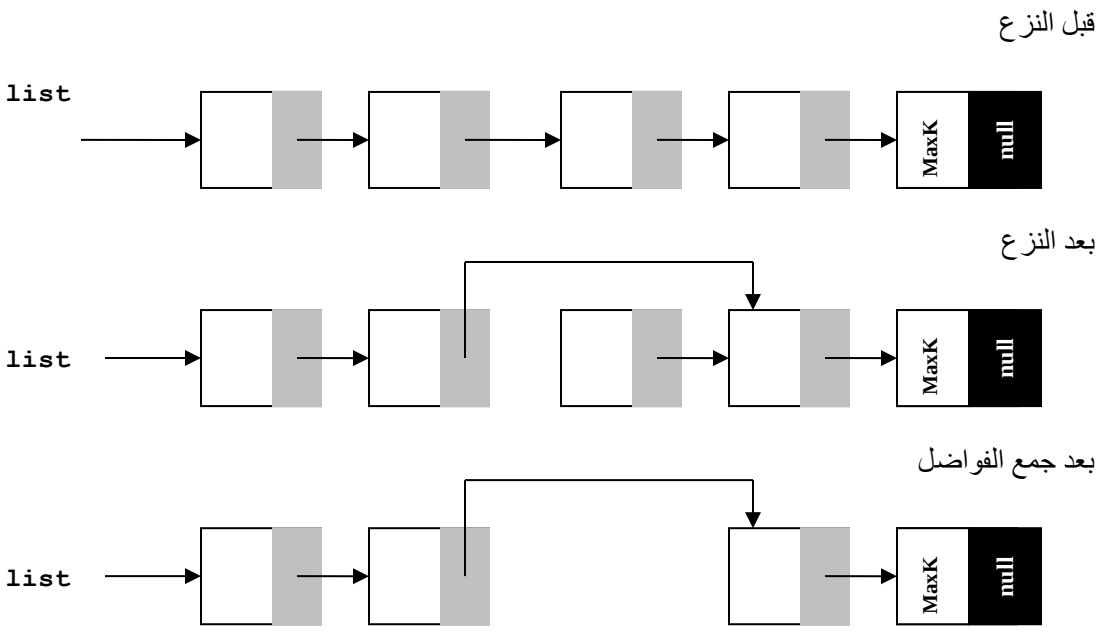
في هذه الحالة تتمثل عملية النزع في مجرد أن نحول موجه **list** من العنصر الأول إلى العنصر الثاني من التسلسل. قد يتسائل المرء إذا: ماذا يحصل عندئذ للعنصر الأول؟ إن تنفيذ لغة جافا متزامن مع برنامج تصرف حركي لذاكرة الحاسوب يُلقى عليه إسم جمع الفواضل. يتفقد هذا البرنامج جميع عناصر الذاكرة، بحثا عن عناصر لا يشير لها أي موجه، فيجد أن العنصر الأول لا يشير له أي موجه، فيسترجعه لذاكرة الحاسوب، فيصبح التسلسل على الشكل التالي:



أما إذا كان العنصر موضوع النزع في وسط التسلسل عوضا عن أوله، فتتمثل عملية النزع في ربط العنصر السابق بالعنصر اللاحق، كما يبين الرسم رقم 7.9. أما عن برمجة هذه العملية في جافا، فأصعب طور فيها هو البحث عن العنصر موضوع النزع، و ضرورة التعرف عن العنصر الشابق له، مع ضرورة معالجة الحالات حيث يكون العنصر غير موجود. نكتب البرنامج التالي للقيام بهذه العملية، حيث تمثل المتغيرة **key** مفتاح العنصر الذي نريد نزعه من التسلسل.

```
int key;
node cur, pred;

cur = list.link;
pred = list;
while (cur.key < key)
{
   pred = cur;
   cur = cur.link;
}
if (cur.key == key)
   {pred.link = cur.link;}
else
```

```
{System.out.println(key+" not found.");}
```



الرسم رقم 7.9: نزع عنصر في أول التسلسل

4.1.9 مثال توضيحي

توضيحا لما سبق، نقدم فيما يلي برنامجا بسيطا يصرح بهيكل خطي و يسنده قيمة إفتتاحية كما ذكرنا في الجزء 1.1.9، ثم ينطلق في مدار يطلب من المستخدم أن يقوم بعمليات إضافة و نزع حسب طلب المستخدم. يتمثل تدخل المستخدم في عددين، يمثل أولهما نوعية العملية التي نطلب القيام بها (1 للإضافة و 2 للنزع) و يمثل ثانيهما مفتاح العنصر الذي تنطبق عليه العملية. إذا قدم المستخدم 0 كرمز للعملية (مع تقديم عدد آخر يقوم مقام المفتاح)، فينتهي تنفيذ البرنامج مع طبع العناصر الموجودة بالتسلسل. نكتب هذا البرنامج كما يلي:

```
Scanner inputWindow = new Scanner(System.in);                    //  1.
                                                                 //  2.
final int MAXKEY = 2147483647;                                   //  3.
node list = new node();                                          //  4.
list.key = MAXKEY;                                               //  5.
list.link = null;                                                //  6.
int operator;                                                    //  7.
int operand;                                                     //  8.
System.out.println("Enter Operation and Operand:");              //  9.
operator = inputWindow.nextInt();                                // 10.
operand = inputWindow.nextInt();                                 // 11.
                                                                 // 12.
while (operator != 0)                                            // 13.
{                                                                // 14.
   switch (operator)                                             // 15.
   {                                                             // 16.
     case 1:                                                     // 17.
        node p = new node();                                     // 18.
        p.key = operand;                                         // 19.
```

```
        if (p.key < list.key)                                          // 20.
        {                                                              // 21.
            p.link = list;                                             // 22.
            list = p;                                                  // 23.
        }                                                              // 24.
        else                                                           // 25.
        {                                                              // 26.
            node front, back;                                          // 27.
            front = list.link;                                         // 28.
            back = list;                                               // 29.
            while (p.key > front.key)                                  // 30.
            {                                                          // 31.
               back = front;                                           // 32.
               front = front.link;                                     // 33.
            }                                                          // 34.
            back.link = p;                                             // 35.
            p.link = front;                                            // 36.
        }                                                              // 37.
        break;                                                         // 38.
     case 2:                                                           // 39.
        if (operand < list.key)                                        // 40.
        {                                                              // 41.
           System.out.println(operand+" not Found.");                  // 42.
        }                                                              // 43.
        else if (operand == list.key)                                  // 44.
        {                                                              // 45.
           list = list.link;                                           // 46.
        }                                                              // 47.
        else                                                           // 48.
        {                                                              // 49.
            node cur, pred;                                            // 50.
            cur = list.link;                                           // 51.
            pred = list;                                               // 52.
            while (cur.key < operand)                                  // 53.
            {                                                          // 54.
               pred = cur;                                             // 55.
               cur = cur.link;                                         // 56.
            }                                                          // 57.
            if (cur.key == operand)                                    // 58.
               {pred.link = cur.link;}                                 // 59.
            else                                                       // 60.
               {System.out.println(operand+" not found.");}            // 61.
            break;                                                     // 62.
         }                                                             // 63.
     }                                                                 // 64.
     System.out.println("Enter Operation and Operand:");               // 65.
     operator =  inputWindow.nextInt();                                // 66.
     operand = inputWindow.nextInt();                                  // 67.
  }                                                                    // 68.
  node c = list;                                                       // 69.
  while (c.key != MAXKEY)                                              // 70.
  {                                                                    // 71.
     System.out.print(c.key+"  ");                                     // 72.
     c = c.link;                                                       // 73.
  }                                                                    // 74.
  System.out.println();                                                // 75.
}                                                                      // 76.
```

نعلق على هذا البرنامج في ما يلي:

- السطر 1: التصريح بنافذة توريد البيانات.

- الأسطر 3 إلى 6: إفتتاح التسلسل، كما فسرنا في الجزء 1.1.9.
- الأسطر 7 إلى 11: إجراءات توريد البيانات من المستخدم.
- السطر 13: نواصل ما دام المستخدم لم يقدم صفرا كرقم العملية.
- السطر 15: نوجه التنفيذ حسب قيمة رقم العملية.
- السطر 17: حالة إضافة.
- السطران 18 و 19: تكوين عقدة جديدة و خزن المفتاح المطلوب فيها.
- السطر 20: إذا كان هذا الشرط صحيحا، نجد حالة إضافة في أول التسلسل.
- السطران 22 و 23: الإضافة في أول السطر.
- الأسطر 27 إلى 36: الإضافة في وسط التسلسل، حسب ما جاء في الجزء 2.1.9.
- السطر 39: حالة نزع.
- الأسطر 40 إلى 43: إذا كان مفتاح العنصر المطلوب نزعه أقل من أول مفتاح في التسلسل، فلا يوجد عنصر بهذا المفتاح في التسلسل.
- الأسطر 44 إلى 47: إذا كان المفتاح موضوع النزع يساوي مفتاح رأس التسلسل، فنجد حالة النزع من أول التسلسل، حسب ما إقترحنا في الجزء 3.1.9.
- الأسطر 50 إلى 61: حالة النزع في وسط التسلسل، حسب ما جاء في الجزر رقم 3.1.9.
- الأسطر 65 إلى 67: الإنتقال إلى العملية الموالية.
- الأسطر 69 إلى 75: نطبع مفاتيح عناصر التسلسل.

يسفر تنفيذ هذا البرنامج على النتيجة التالية، حسب البيانات التي نقدمها:

```
run:
Enter Operation and Operand:
1 8
Enter Operation and Operand:
1 5
Enter Operation and Operand:
1 9
Enter Operation and Operand:
2 4
4 not found.
Enter Operation and Operand:
1 11
Enter Operation and Operand:
1 12
Enter Operation and Operand:
2 8
Enter Operation and Operand:
0 0
5  9  11  12
BUILD SUCCESSFUL (total time: 1 minute 49 seconds)
```

فعلا، إذا أضفنا للتسلسل عناصر 8، 5، 9، 11، 12 و نزعنا منه عناصر 8 و 4 (الذي لا يغير التسلسل نظرا لغيابه منه عندما قمنا بعملية النزع) لوجدنا تسلسلا متكونا (حسب الترتيب الصاعد) من: 5، 9، 11، 12؛ و هو ما تنص عليه نتيجة البرنامج.

2.9 الأشجار الثنائية

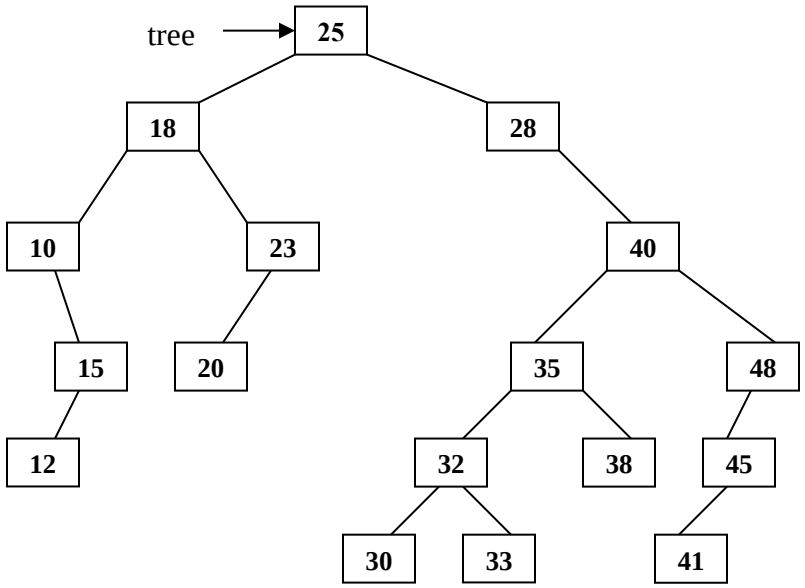
يمكننا مفهوم الموجهات من تكوين هياكل لاخطية، نذكر منها بالخصوص في هذا الجزء هيكل الأشجار الثنائية. إن علم الحاسوب حافل بتطبيقات عديدة و متنوعة لمثل هذه الهياكل، نذكر منها مثلا الأشجار النحوية و أشجار العبارات

و أشجار الترتيب و غيرها. نناقش في هذا الجزء أشجار الترتيب فنستعملها على سبيل التوضيح لنرتب أعدادا كاملة نقدمها في ترتيب عشوائي.

تتمثل شجرة ترتيب في شجرة ثنائية تحمل عقدها أعدادا حيث أن في كل عقدة،

- يكون العدد الموجود في العقدة أكبر من كل الأعداد الموجودة في الشجرة الجزئية على يسار العقدة.
- يكون العدد الموجود في العقدة أصغر من كل الأعداد الموجودة في الشجرة الجزئية على يمين العقدة.

نقدم في الرسم رقم 8.9 مثال شجرة ترتيب.

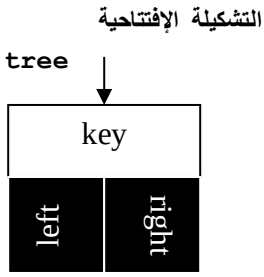


الرسم رقم 8.9: شجرة ترتيب

يمكن بناء مثل هذه الشجرة إنطلاقا من سلسلة أعداد عشوائية بالصفة التالية: يوضع أول عدد كجذر الشجرة؛ ثم كلما نتطلع على عدد جديد نطوف به خلال الشجرة إنطلاقا من الجذر، فننزل على اليسار إذا كان العدد الجديد أصغر من عدد العقدة الحالية و ننزل على اليمين إذا كان العدد الجديد أكبر من عدد العقدة الحالية. نواصل النزول إلى أن نصل إلى ورقة من أوراق الشجرة؛ فنربط الورقة بالعدد الجديد يمينا أو شمالا حسب إن كان العدد الجديد أكبر أو أصغر من عدد الورقة. ندرس في الأجزاء التالية كيف يمكن التصريح بعقدة هذه الشجرة في لغة جافا، ثم ندرس كيف يمكن إضافة الأعداد للشجرة مع الحفاظ على ترتيبها، ثم كيف يمكن زيارة الشجرة لنستعرض الأعداد في ترتيبها الصاعد.

1.2.9 التصريح و الإفتتاح

بالنسبة لهيكل الشجرة الثنائية، نصرح بعقدة ذات ثلاثة حقول، حقل يحمل مفتاح العنصر و حقلين من نوع الموجهات يشيران للشجرة الجزئية على اليمين و الشجرة الجزئية على اليسار. أما عن التشكيلة الإفتتاحية فلا يمكن أن نكونها إلا إذا تحصلنا على العدد الأول في سلسلة الأعداد التي نريد وضعها بالشجرة. لهذا الغرض، نصرح بناسخة داخلية من نوع سلسلة الحروف و نسند لها سطرا يحتوي على جميع الأعداد، فنقرأ أولها و نضعه في حقل المفتاح مع وضع القيمة في الحقلين الموجهين للأشجار الجزئية، فنكتب النص التالي في جافا:




```
Scanner inputWindow = new Scanner(System.in);                    //  1.
String dataString = new String();                                //  2.
System.out.println("الرجاء تقديم سلسلة الأعداد:");                 //
3.
dataString = inputWindow.nextLine();                             //  4.
Scanner numberSequence = new Scanner(dataString);                //  5.
                                                                 //  6.
int curKey;                                                      //  7.
                                                                 //  8.
curKey = numberSequence.nextInt();                               //  9.
node tree = new node();                                          // 10.
tree.key = curKey;                                               // 11.
tree.left = null;                                                // 12.
tree.right = null;                                               // 13.
```

نعلق على هذا النص في ما يلي:

- السطر 1: التصرح بالناسخة الخارجية، **inputWindow**.
- الأسطر 2 إلى 4: نصرح بسلسلة حروف نضع فيها سطرا من البيانات الواردة.
- السطر 5: نصرح بناسخة تمكننا من قراءة السطر **dataString** كسلسلة أعداد (عوضا عن سلسلة حروف).
- السطر 9: قراءة أول عدد في سلسلة الأعداد.
- الأسطر 10 إلى 13: تكوين عقدة جديدة و خزن العدد فيها مع إسناد القيمات الإفتتاحية للموجهين اليميني و اليساري.

2.2.9 بناء شجرة الترتيب

نقوم ببناء شجرة الترتيب بواسطة موجهين: موجه **p** يتقدم مسار التفتيش في الشجرة و موجه **parent** يخزن دائما العقدة التي تسبق عقدة **p**. نحتاج لموجه **parent** لأننا نستعمله لربط العقدة الجديدة.

```
while (numberSequence.hasNextInt())                              //  1.
{                                                                //  2.
  curKey = numberSequence.nextInt();                             //  3.
  node p, parent;                                                //  4.
  parent = tree;                                                 //  5.
  if (curKey<parent.key)                                         //  6.
     {p=parent.left;}                                            //  7.
  else                                                           //  8.
     {p=parent.right;}                                           //  9.
  while (p!=null)                                                // 10.
```

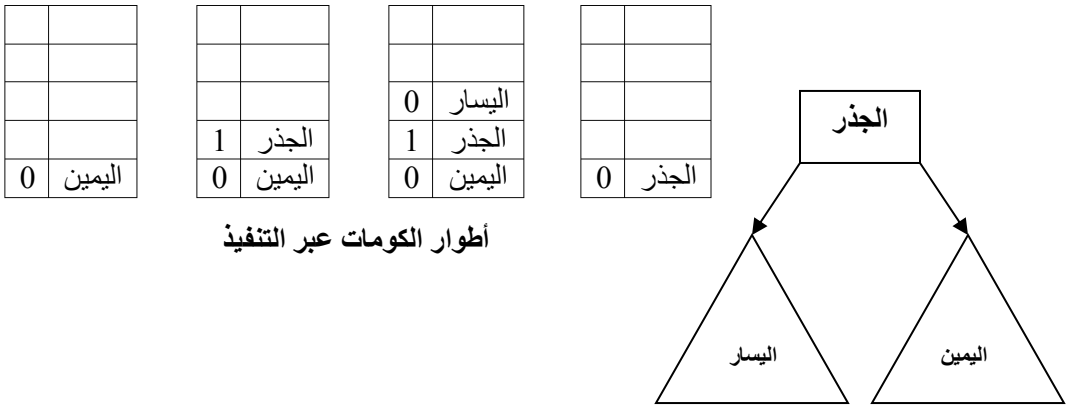
```
  {                                                    // 11.
      parent = p;                                      // 12.
      if (curKey<parent.key)                           // 13.
         {p=parent.left;}                              // 14.
      else                                             // 15.
         {p=parent.right;}                             // 16.
    }                                                  // 17.
    node newp = new node();                            // 18.
    newp.key=curKey;                                   // 19.
    if (curKey<parent.key)                             // 20.
    {                                                  // 21.
       parent.left=newp;                               // 22.
    }                                                  // 23.
    else                                               // 24.
    {                                                  // 25.
       parent.right=newp;                              // 26.
    }                                                  // 27.
}                                                      // 28.
```

نعلق على هذا البرنامج في ما يلي:

- السطر 1: نعالج الأعداد على التوالي.
- السطر 3: قراءة العدد الجاري.
- الأسطر 4 إلى 9: التصريح بموجهتي التفتيش و إسنادهما القيمات الإفتتاحية.
- السطر 10: الطواف في الشجرة بحثا على موقع المفتاح الجديد.
- الأسطر 12 إلى 16: تحديث الموجهين، **p** و **parent**.
- السطر 17: عند الخروج من المدار، يكون الموجه **parent** يشير للموجه الذي نربط به العقدة الجديدة.
- السطران 18 و 19: تكوين عقدة جديدة تحتوي على المفتاح الجديد.
- الأسطر 20 إلى 27: ربط العقدة **parent** بالعقدة الجديدة على اليمين أو على اليسار حسب مقارنة المفتاح الجديد بمفتاح عقدة **parent**.

3.2.9 الطواف على الشجرة

في هذا الجزء نعتزم بالطواف في الشجرة بكيفية تسمح لنا أن نزور العقد في ترتيب صاعد. علما أن الشجرة مرتبة بحيث أن كل عقدة مفتاحها أكبر من مفاتيح الشجرة الجزئية الموجودة على اليسار و أصغر من مفاتيح الشجرة الجزئية الموجودة على اليمين، يمكن زيارة العقد في الترتيب الصاعد إذا زرنا الشجرة من اليسار إلى اليمين. لهذا الغرض، نستعمل هيكل الكومة، و هو هيكل بيانات يخزن بيانات و يستخرجها في الترتيب المعاكس لترتيب خزنها. يبين الرسم رقم 9.9 تنفيذ هذه الفكرة حيث نضع شجرة يرافقها رمز 0 (لنبين أننا لم نزرها بعدُ)؛ ثم يقع نزعها من الكومة لنتفقد إن كان لها أشجار جزئية. إذا كان لها أشجار جزئية، فنضع على الكومة الشجرة اليمنى (لنزورها في الآخر) يرافقها رمز 0 (انبين أننا لم نشرع في معالجتها)؛ ثم نضع جذر الشجرة يرافقها رمز 1 (لنبين أننا عندما نصل هذا الجذر سيحين وقت طباعتها)؛ ثم نضع على الكومة الشجرة اليسرى (لنزورها قبل الجذر و قبل الشجرة اليمنى) يرافقها رمز 0 (لنبين أننا لم نشرع في معالجتها).



أطوار الكومات عبر التنفيذ

الرسم رقم 9.9: توضيح خوارزمية الطواف

نكتب البرنامج التالي:

```
Stack treeStack = new Stack();                                    //  1.
Stack intStack = new Stack();                                     //  2.
                                                                  //  3.
treeStack.push(tree);                                             //  4.
intStack.push(0);                                                 //  5.
while (!treeStack.empty())                                        //  6.
{                                                                 //  7.
   node t = (node) treeStack.peek();                              //  8.
   treeStack.pop();                                               //  9.
   int code = (Integer) intStack.peek();                          // 10.
   intStack.pop();                                                // 11.
   if (code == 0)                                                 // 12.
   {                                                              // 13.
      if (t.right!=null)                                          // 14.
      {                                                           // 15.
         treeStack.push(t.right);                                 // 16.
         intStack.push(0);                                        // 17.
      }                                                           // 18.
      treeStack.push(t);                                          // 19.
      intStack.push(1);                                           // 20.
      if (t.left!=null)                                           // 21.
      {                                                           // 22.
         treeStack.push(t.left);                                  // 23.
         intStack.push(0);                                        // 24.
      }                                                           // 25.
   }                                                              // 26.
   else                                                           // 27.
   {                                                              // 28.
      System.out.print(t.key+ " ");                               // 29.
   }                                                              // 30.
}                                                                 // 31.
```

نعلق على هذا البرنامج في ما يلي:

- السطران 1 و 2: التصريح بكومة الأشجار و كومة الرموز العددية.
- السطر 4: خزن الشجرة على الكومة.
- السطر 5: لم نعالج هذه الشجرة.

- السطر 6: تخزن الكومة دوما مجموعة الأشجار التي لم نعالجها بعد.
- الأسطر 8 إلى 11: نزع الشجرة و الرمز في قمة الكومة.
- السطر 12: أول زيارة لهذه الشجرة.
- الأسطر 14 إلى 18: خزن الشجرة اليمنى، إذا كانت موجودة.
- السطران 19 و 20: خزن الجذر، مرفوقا برمز يدل على أن هذا الجذر قمنا بزيارته سابقا، بحيث نكتفي بطباعته عندما نلاقيه في المرة المقبلة (عندما ننتهي من معالجة الشجرة الجزئية اليسارية).
- السطر 29: هذه المرة الثانية حيث نلاقي هذه العقدة، لأن يرافقها الرمز عدد 1. يدل هذا أن الشجرة الجزئية على يسار هذه العقدة تمت معالجتها، فحان وقت طبع هذه العقدة.

ننفذ هذا البرنامج على مجموعة الأعداد الموجودة في الشجرة التي نمثلها في الرسم رقم 8.9، فيسفر التنفيذ على النتيجة التالية.

```
run:
الرجاء تقديم سلسلة الأعداد:
25 18 28 10 23 40 15 12 20 35 48 32 38 45 30 33 41
10 12 15 18 20 23 25 28 30 32 33 35 38 40 41 45 48
BUILD SUCCESSFUL (total time: 6 seconds)
```

يمثل السطر الأول مجموعة الأعداد كما قدمناها في ترتيب عشوائي للبرنامج، و يمثل السطر الثاني نفس المجموعة، بعد ترتيبها صاعدا.

3.9 المخططات

توفر لنا الموجهات في جافا إمكانيات واسعة في تمثيل مخططات ذات أشكال مختلفة، ننظر في البعض منها، في الأجزاء التالية.

1.3.9 مخططات ذات الدرجات المحدودة

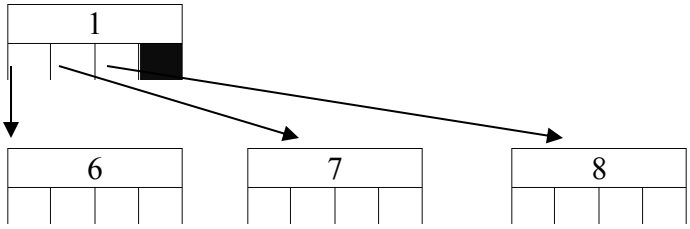
إذا أردنا أن نمثل مخططا يلبي الشروط التالية:

- درجته محدودة و منخفضة،
- درجات العقد تقريبا متساوية (لا تتغير كثيرا بين العقدة و الأخرى)،

فقد يليق أن نمثل المخطط بواسطة عقد تحمل بيانات العقدة بالإضافة لعدد من الموجهات يساوي درجة المخطط. إذا إعتبرنا مخططا درجته 4 مثلا، نمثل عقده كما يلي:

```
class node
{
   int key;
   node link1, link2, link3, link4;
}
```

إذا كانت العقدة رقم 1 مرتبطة بالعقد رقم 6 و 7 و 8 مثلا، نمثل هذه العلقات كما يلي (حيث نمثل قيمة `null` بالمنطقة السوداء):



2.3.9 مخططات ذات الدرجات العشوائية

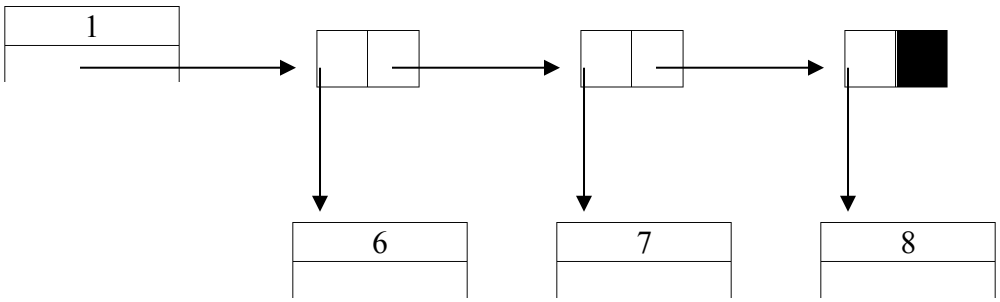
إذا أردنا أن نمثل مخططا يلبي الشروط التالية:

- درجته لا محدودة،
- درجات العقد تتغير كثيرا بين العقدة و الأخرى،

فقد يليق أن نمثل المخطط بواسطة عقد تحمل بيانات العقدة بالإضافة لموجه يشير لهيكل تسلسلي يخزن قائمة العقد التي ترتبط بها العقدة الحالية. إذا أردنا أن نمثل مخططا من هذا النوع في لغة جافا، فنكتب:

```
class adrlist
{
    node nodeadr;
    adrlist next;
}
class node
{
    int key;
    adrlist adrs;
}
```

إذا كانت العقدة رقم 1 مرتبطة بالعقد رقم 6 و 7 و 8 مثلا، نمثل هذه العلقات كما يلي (حيث نمثل قيمة `null` بالمنطقة السوداء):



3.3.9 مخططات صغيرة و كثيفة

إذا أردنا أن نمثل مخططا يلبي الشروط التالية:

- عدد عقده محدد و صغير،
- عدد سهامه مرتفع بالنسبة لعدد عقده،

فقد يليق أن نمثل المخطط بواسطة مصفوفة منطقية. إذا أردنا أن نمثل مخططا من هذا النوع في لغة جافا، و كان عدد العقد في هذا المخطط 10 مثلا، فنكتب:

```
final int N=10;
boolean [][] graph = new boolean[N][N];
```

إذا كانت العقدة رقم 3 مرتبطة بالعقد رقم 6 و 7 و 8 مثلا، نمثل هذه العلقات كما يلي (حيث نمثل قيمة **true** بالمنطقة السوداء):

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

4.9 تمارين

1 [م] يمكن صنع تسلسلا مماثلا لما درسناه في الجزء 1.9 بوايطة عقد تحتوي لا على موجه واحد (للعنصر الموالي) بل على موجهين: موجه يشير للعنصر الموالي في التسلسل و موجه يشير للعنصر السابق. المطلوب منكم كتابة برنامج في حافا مثل البرنامج الذي قدمناه في الجزء 4.1.9، يستعمل هذا الهيكل الجديد للعقد. المطلوب منكم أيضا نقاش الفروق بين الحلّين.

2 [م/ص] يمكن برمجة الهيكل الخطي الذي ناقشناه في الجزء 1.9 بواسطة جدولين، جدول يحمل البيانات الخاصة بالعقد (نكتفي بخزن مفتاح كل عنصر) و جدول يحمل عناوين العنصر الموالي. المطلوب منكم برمجة عمليات الإفتتاح و الإضافة و النزع؛ مع برمجة التصرف في فضاء الذاكرة (أي التصرف في المواقع الجاهزة للإستعمال).

3 [م/ص] نفس السؤال كالتمرين السابق، مع التصرف في فضاء الذاكرة بواسطة التنفيذ الدوري لبرنامج جمع الفواضل.

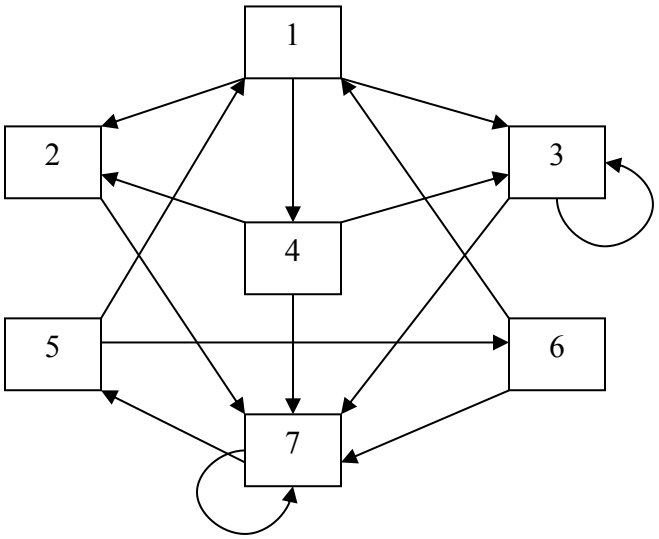
4 [س] المطلوب منكم إقامة شجرة ترتيب من سلسلة الأعداد التالية:

20, 12, 32, 8, 15, 25, 34, 24, 27, 26, 6, 2, 7, 13.

5 [س] كيف يكون شكل شجرة ترتيب إذا بنيناها من سلسلة أعداد مرتبة صاعدا؟ أو مرتبة نازلا؟ كيف يجب أن يكون ترتيب الأعداد الواردة إذا أردنا أن يكون الفرق بين مسافات الأوراق من الجذر أقل ما يمكن.

6 [م/ص] في الجزء رقم 2.9 بنينا شجرة الترتيب ثم زرنا عناصرها لكن لم نطلع على تفاصيل الشجرة، أي لا نعرف تشكيلتها. المطلوب منكم أن تكتبوا برنامجا يطبع هذه الشجرة و أن تتأكدوا من خلال ذالك أن الشجرة التي يبنيها برنامجنا هي فعلا صحيحة.

7 [م/ص] المطلوب منكم تمثيل المخطط التالي حسب الأساليب التي ناقشناها في الجزء رقم 3.9. كما نطلب منكم أن تكتبوا برنامجا في جافا يبني هذا المخطط.



5.9 مصادر و مراجع

للمزيد من البيانات حول المخططات و الأشجار، يمكن الرجوع إلى [الميلي و العابد، 2006].

الفصل العاشر

الملفات

جل البرامج التي درسناها لحد الآن تعالج بيانات نوفرها بصفة فورية عند التنفيذ. لكن هناك العديد من التطبيقات حيث نريد أن نعالج بيانات قارة يقع خزنها في ملفات توجد قبل تنفيذ البرنامج و تبقى بعده. في هذا الفصل نهتم بمعالجة الملفات في لغة جافا.

1.10 البيانات الواردة و الصادرة

تمكننا لغة جافا من التصريح بملفات، بواسطة نوع البيانات **File**، كما نصرح بأي متغيرة من أي نوع. فنكتب مثلا:

```
File myFile;
```

لكن هذا التصريح، و لئن ضبط نوع البيانات التابع لمتغيرة **myFile** ، فهم يترك تفصيلين إثنين في حالة غموض، و هما:

- هل نستعمل هذا الملف لقراءة بيانات أم لكتابتها؟
- أي ملف خارجي نربط بالملف الذي صرحنا به داخل البرنامج؟ علما أن إذا كانا نستعمل الملف لقراءة البيانات، يجب أن يكون هذا الملف موجودا في فضاء برنامج جافا عند التنفيذ (و أنه يحتوي على البيانات المعنية). أيضا، إذا كنا نستعمل هذا الملف لكتابة البيانات، سيتكون هذا الملف عند التنفيذ و يقع خزنه في فضاء برنامج جافا. إذا كان هناك ملف يحمل نفس الإسم في فضاء البرنامج، يقع فسخه و تعويضه بالملف الجديد، فعلى المبرمج أن يحتاط.

تمكننا جافا من ربط إسم الملف الداخلي بإسم ملف خارجي بواسطة تصريح شكله كما يلي:

```
myFile = new File ("extFileName.ext");
```

أما عن تعيين هذا الملف كملف قابل للقراء أو للكتابة، فنقوم بذلك بمقتضى ربط هذا الملف إما بناسخ (للكتابة) أو بصادر (للقراءة).

- الناسخ. إذا أردنا أن نسعمل الملف للقراءة، فنربطه بناسخ، مما يمكننا من تطبيق كل عمليات الناسخ عليه، مثل قراءة الأعداد و قراءة الكلمات و قراءة الأسطر، إلى غير ذلك. نربط الملف بناسخ بواسطة تصريح شكله كما يلي:

```
myScanner = new Scanner(myFile);
```

فيفسح لنا هذا التصريح المجال لتطبيق جميع عمليات الناسخ ل **myScanner** ، بما فيها مثلا

```
myScanner.hasNext(), myScanner.nextInt(), myScanner.nextLine(),
```

إلخ.

- الصادر. إذا أردنا أن نستعمل الملف للكتابة، فنربطه بصادر، مما يمكننا من تطبيق كل عمليات الصادر عليه، مثل الكتابة في الملف أو غلقه. نربط الملف بناسخ بواسطة تصريح شكله كما يلي:

```
myStream = new FileOutputStream(myFile);
```

فيفسح لنا هذا التصريح المجال لتطبيق جميع عمليات الناسخ ل **myStream** ، بما فيها مثلا

```
myStream.Write(), myStream.flush(), myStream.close(),
```

إلخ.

كتوضيح لهذه المفاهيم، نقدم البرنامج التالي، فنعلق عليه ثم نتأمل في نتيجة تنفيذه.

```
package file;                                                    //  1.
/**                                                              //  2.
 * @author Ali Mili and Ahmed Ferchichi                         //  3.
 */                                                              //  4.
                                                                 //  5.
import java.util.Scanner;                                        //  6.
import java.io.*;                                                //  7.
                                                                 //  8.
public class Main {                                              //  9.
                                                                 // 10.
    /**                                                          // 11.
     * @param args the command line arguments                   // 12.
     */                                                         // 13.
    public static void main(String[] args)                       // 14.
       throws IOException {                                      // 15.
        // TODO code application logic here                      // 16.
                                                                 // 17.
        File masterFile, storeFile, transFile;                   // 18.
        Scanner transScanner, storeScanner;                      // 19.
        FileOutputStream masterStream;                           // 20.
                                                                 // 21.
        storeFile = new File ("store.txt");                      // 22.
        transFile = new File ("trans.txt");                      // 23.
        masterFile = new File ("master.txt");                    // 24.
                                                                 // 25.
        storeScanner = new Scanner (storeFile);                  // 26.
        transScanner = new Scanner (transFile);                  // 27.
        masterStream = new FileOutputStream(masterFile);         // 28.
                                                                 // 29.
        String storeString = storeScanner.nextLine();            // 30.
        String transString = transScanner.nextLine();            // 31.
                                                                 // 32.
        masterStream.write(storeString.getBytes());              // 33.
        masterStream.write(transString.getBytes());              // 34.
                                                                 // 35.
        masterStream.flush();                                    // 36.
        masterStream.close();                                    // 37.
                                                                 // 38.
        System.out.println("done.");                             // 39.
    }                                                            // 40.
}                                                                // 42.
```

نعلق على هذا البرنامج في ما يلي:

- السطران 6 و 7: التصريح بأننا نحتاج لوسائل الإصدار و التوريد.

- السطر 15: يتعلق هذا السطر بمعالجة الحالات الإستثنائية، و هو خارج عن نطاق نقاشاتنا في هذا الفصل و هذا الكتاب. نكتفي بأن نذكر أن هذا السطر يعين أسلوب معالج الحالات الإستثنائية الذي يجب تنفيذه في حالة فشل غملية قراءة من ملف أو كتابة في ملف.
- السطر 18: التصريح بثلاثة ملفات.
- السطر 19: التصريح بناسخين.
- السطر 20: التصريح بصادر.
- الأسطر 22 إلى 24: ربط الأسماء الداخلية للملفات بملفات خارجية معرّفة بإسمها في نظام التشغيل.
- السطران 26 و 27: ربط ملفين بناسختين، تهيئة للقراءة منهما.
- السطر 28: ربط ملف بصادر، تهيئة للكتابة فيه.
- السطران 30 و 31: قراءة السطر الأول من الناسختين.
- السطران 33 و 34: كتابة السكرين في الصادرة.
- السطر 36: عندما ننفذ عملية كتابة في ملف، تبلغ جافا هذه البيانات لنظام التشغيل، فيضعها نظام التشغيل في سجلات وقتية قبل أن يحولها نهائيا للملف المقصود. تتأكد هذه التعليمة أن ينظف نظام التشغيل سجلاته فيبعث بكل البيانات للملف المقصود.
- السطر 37: غلق الملف المقصود لكي نتمن من فتحه بعد التنفيذ.
- السطر 39: نعلن عن نهاية التنفيذ.

ننفذ هذا البرنامج على الملفين التاليين:

store.txt
هذا أول سطر من ملف الخزن و هذا السطر الثاني و هذا السطر الثالث إلى آخره

trans.txt
هذا أول سطر من ملف المعاملات و هذا السطر الثاني ثم الثالث إلى غير ذلك.

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
done.
BUILD SUCCESSFUL (total time: 1 second)
```

و نجد في ملف **master.txt** البيانات التالية:

master.txt
هذا أول سطر من ملف الخزنهذا أول سطر من ملف المعاملات

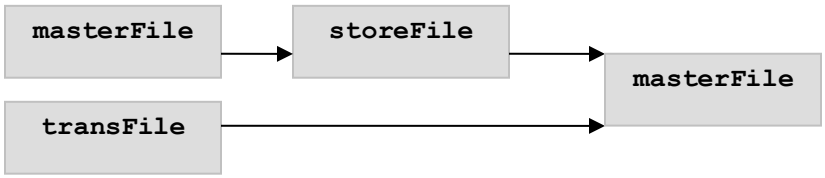
و هو ما كنا ننتظر منه.

في البرنامج السابق، قسمنا بين قسمين من الملفات: الملفات التي تختص في القراءة، فأسندنا لها ناسخات؛ و الملفات التي تختص في الكتابة، فأسندنا لها صادرات. لكن هناك تطبيقات حيث نحتاج في نفس البرنامج للقراءة من ملف، و الكتابة فيه. مثلا، لدينا ملفان:

- ملف رئيسي يحمل بيانات قارة (طويلة المدى)، نلقي عليه إسم **masterFile**.

- ملف ثانوي يحتوي على بيانات وقتية تتعلق بمعاملات نريد تسجيلها لدى الملف الرئيسي، نلقي عليه إسم **transFile**.

لكي نقوم بهذه العملية، يجب أن نحول محتوى الملف الرئيسي إلى ملف مساعد وقتي نلقي عليه إسم **storeFile** ، ثم نعالج البيانات في الملف المساعد و ملف المعاملات فنضع النتيجة في الملف الرئيسي. تأمل في الرسم رقم 1.10.



الرسم رقم 1.10: القراءة و الكتابة في الملف الرئيسي.

تسمح لنا لغة جافا بالقراءة من و الكتابة لنفس الملف، على أن نسند لهذا الملف ناسخا نستعمله للقراءة و صادرا نستعمله للكتابة، كما يبين البرنامج التالي:

```
File masterFile, storeFile, transFile;                    //  1.
Scanner transScanner, masterScanner, storeScanner;        //  2.
FileOutputStream masterStream, storeStream;               //  3.
                                                          //  4.
storeFile = new File ("store.txt");                       //  5.
transFile = new File ("trans.txt");                       //  6.
masterFile = new File ("master.txt");                     //  7.
                                                          //  8.
//  from master to store                                  //  9.
masterScanner = new Scanner(masterFile);                  // 10.
storeStream = new FileOutputStream(storeFile);            // 11.
String masterLine = masterScanner.nextLine();             // 12.
storeStream.write(masterLine.getBytes());                 // 13.
storeStream.flush();                                      // 14.
storeStream.close();                                      // 15.
                                                          // 16.
//  from store and trans to master                        // 17.
storeScanner = new Scanner (storeFile);                   // 18.
transScanner = new Scanner (transFile);                   // 19.
masterStream = new FileOutputStream(masterFile);          // 20.
                                                          // 21.
String storeString = storeScanner.next();                 // 22.
String transString = transScanner.nextLine();             // 23.
masterStream.write(storeString.getBytes());               // 24.
masterStream.write(transString.getBytes());               // 25.
masterStream.flush();                                     // 26.
masterStream.close();                                     // 27.
                                                          // 28.
//  signal the end                                        // 29.
System.out.println("done.");                              // 30.
```

نعلق على هذا البرنامج في ما يلي:

- السطر الأول: نصرح بكل الملفات التي نحتاج لها.
- السطر الثاني: نصرح بناسخات لكل الملفات، لأننا سنقرأ من كل الملفات.

- السطر الثالث: نصرح بصادرات للملف الرئيسي و الملف المساعد (ملف الخزن) لأننا سنكتب فيهما.
- الأسطر 5 إلى 7: نربط الأسماء الداخلية للملفات (كما عُرفت داخل البرنامج) بأسمائها الخارجية (كما عُرفت لدى نظام التشغيل).
- السطر 10: نعين ناسخ الملف الرئيسي.
- السطر 11: نعين صادر ملف الخزن.
- السطر 12: نقرأ سطرا من الملف الرئيسي، بواسطة ناسخه.
- السطر 13: نكتب هذا السطر في ملف الخزن، بواسطة صادره.
- السطر 14: نبعث كلما كُتب في صادر الخزن لملف الخزن.
- السطر 15: نغلق ملف الخزن.
- الأسطر 18 إلى 20: إسناد ناسخات للملفين حيث سنقرأ (ملف الخزن و ملف المعاملات) و إسناد صادرة للملف حيث سنكتب (الملف الرئيسي).
- السطر 22: نقرأ كلمة من ملف الخزن.
- السطر 23: نقرأ سطرا من ملف المعاملات.
- السطران 24 و 25: نكتب الكلمة و السطر في الملف الرئيسي.
- السطر 26: نبعث كلما كُتب في صادر الخزن للملف الرئيسي.
- السطر 27: نغلق الملف الرئيسي.
- السطر 30: نعلن عن نهاية التنفيذ.

ننفذ هذا البرنامج على الملفات التالية:

store.txt
هذا أول سطر من ملف الخزن و هذا السطر الثاني و هذا السطر الثالث إلى آخره

trans.txt
ملف المعاملات: هذا أول سطر في ملف المعاملات و هذا السطر الثاني في نفس الملف و هذا السطر الثالث، إلخ....

master.txt
الملفالرئيسي: هذا أول سطر من الملف الرئيسي و هذا السطر الثاني ثم الثالث إلى غير ذلك.

يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
done.
BUILD SUCCESSFUL (total time: 1 second)
```

و تصبح تشكيلة الملفات على النحو التالي:

store.txt

الملفالرئيسي: هذا أول سطر في الملف الرئيسي

trans.txt
ملف المعاملات: هذا أول سطر في ملف المعاملات و هذا السطر الثاني في نفس الملف و هذا السطر الثالث، إلخ....

master.txt
الملفالرئيسي:ملف المعاملات: هذا أول سطر في ملف المعاملات

2.10 الملفات المهيكلة

إن الملفات التي نهيأها للكتابة بواسطة الصادرات لا تليق لتمثيل النصوص، بل تليق أساسا لتمثيل ملفات لامهيكلة مثل ملفات صور أو ملفات أصوات أو غيرها، أي أنها ملفات مجعولة للقراءة من طرف الحاسوب عوضا عن قراءة من طرف أشخاص. هذا ما يفسر مثلا أن الصادرات لا توفر تعليمات للرجوع إلى السطر. في هذا الجزء نستعرض بعض الوسائل للقراءة من الملفات و الكتابة فيها، مع دراسة المفاهيم النظرية وراء هذه الوسائل.

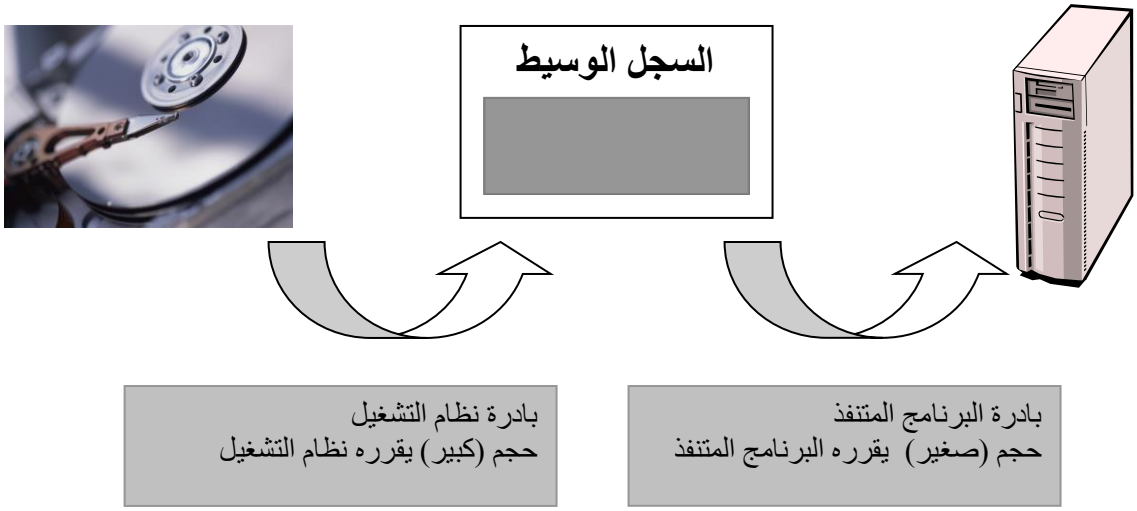
1.2.10 القارءات و الكاتبات

تمكننا لغة جافا من القراءة من ملفات بواسطة وظيفتين إثنتين، و هما

- قارءة الملف، معروفة في جافا تحت إسم **FileReader**.
- وسيطة القراءة، معروفة في لغة جافا تحت إسم **BufferedReader**. لماذا نحتاج لوسيطة القراءة؟ إن جل الملفات التي نستعملها في التطبيقات الجارية يقع خزنها في القرص الصلب للحاسوب، و لا يمكن معالجتها إلا في الذاكرة المركزية للحاسوب. فعندما نقرأ حرفا أو كلمة أو عددا أو سطرا من الملف يقع جلبه من القرص الصلب. لكن نظام التشغيل يجتنب تجنيد القرص الصلب و الذاكرة المركزية و وسائل نقل البيانات و وحدة المعالجة المركزية لنقل عنصر صغير من البيانات (كحرف أو عدد أو كلمة أو حتى سطر). بل يفضل نظام التشغيل أن ينقل حجما كبيرا من البيانات من القرص الصلب إلى منطقة في الذاكرة المركزية، نلقي عليها إسم السجل الوسيط. فبقع نقل البيانات من القرص الصلب إلى متغيرات البرنامج في طورين إثنين:
  - في الطور الأول، و ببادرة من نظام التشغيل، يقع نقل البيانات من القرص الصلب إلى السجل الوسيط، في حجم يقرره نظام التشغيل.
  - في الطور الثاني، و ببادرة من البرنامج المتنفذ، يقع نقل البيانات من السجل الوسيط إلى متغيرات البرنامج، في حجم يقرره البرنامج المتنفذ (الحرف الواحد أو الكلمة الواحدة أو الغدد الواحد أو السطر الواحد، إلخ).

أنظر إلى الرسم رقم 2.10. يقع التصريح على وسيطة القراءة عن طريق قارءة الملف، و يقع التصريح على قارءة الملف بربطها مع إسم ملف يعرفه نظام التشغيل، فنكتب مثلا:

```
FileReader masterFile;
BufferedReader masterBuffer;
masterFile = new FileReader("master.txt");
masterBuffer = new BufferedReader(masterFile);
```



الرسم رقم 2.10: وسيطة القراءة

بصفة مماثلة، تمكننا لغة جافا من الكتابة في ملفات بواسطة وظيفتين إثنتين، و هما

- كاتبة الملف، معروفة في جافا تحت إسم **`FileWriter`**.
- وسيطة الكتابة، معروفة في لغة جافا تحت إسم **`BufferedWriter`**. لها وظيفة مماثلة لوسيطة القراءة، مع تحويل الإتجاه، كما يبين الرسم رقم 3.10.

مع العلم أن عندما نصرح أن التصريح بكاتبة الملف يصحبه إختيار أسلوب لإستعمال الملف على النحو التالي: إذا أردنا أن نكتب في ملف موجود، فيمكن أن نكتب فيه بالإضافة لمحتواه الحالي أو عوضا عن محتواه الحالي. أما إذا كان الملف غير موجود، فالسؤال لا يُطرح. بالتالي، فإن التصريح بكاتبة ملف يكون شكله على النحو التالي:

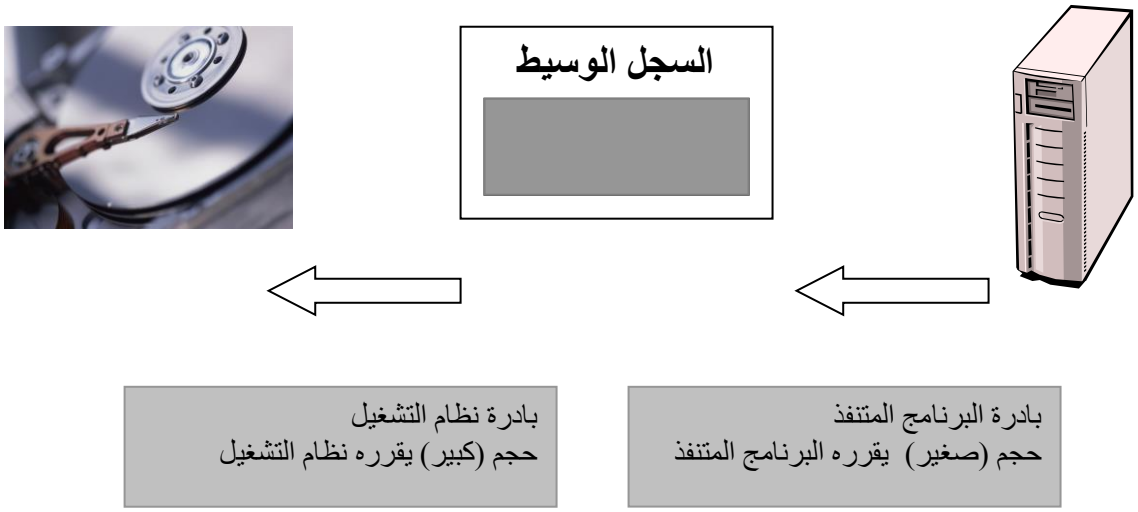
```
javaFileName = new FileWriter ("OsFileName.ext", mode);
```

حيث أن

- **`javaFileName`**: تمثل الإسم الداخلي للملف.
- **`OsFileName.ext`**: تمثل إسم الملف عند نظام التشغيل.
- **`mode`**: تمثل إختيار المبرمج إن كان يريد أن يكتب في هذا الملف عوضا عن محتواه الحالي أو إضافة له، حسب ما يبين الجدول التالي. علما أن إذا لم نبين هذا العامل، يعتبر مصرف جافا أننا إخترنا التعويض ضمنيا.

	إن كان الملف موجودا أم لا	
	موجود	لاموجود

القيمة المنطقية ل mode			
غلط	يقع تعويض المحتوى الحالي للملف بالبينات الجديدة.	ثقع كتابة البيانات الجددية في الملف الجديد.
صحيح	تقع إضافة البيانات الجديدة للملف الموجود.	لا ينطبق.



الرسم رقم 3.10: وسيطة الكتابة

2.2.10 الصادرات و كاتبات الطبع

إن وظائف قارءة الملف و كاتبة الملف تنطبق أساسا على ملفات نصوص. أي أنها تختص في كتابة و قراءة سلاسل الحروف. أما إذا أردنا أن نكتب أنواع بيانات دون الحوف و سلاسل الحروف و الأسطر، فنلتجئ إلى كاتبات الطبع، التي نربطها بصادرات و نستعملها لطبع أنواع بيانات دون الحروف و السلاسل. نصرح بكاتبات الطبع كمل يلي:

```
FileOutputStream masterStream = new FileOutputStream("master.txt");
PrintWriter masterPrinter = new PrintWriter(masterStream);
```

حيث

- يصرح السطر الأول بصادرة إسمها **masterStream** يرلطها بالملف الخارجي إسمه **master.txt** يهيئه للكتابة فيه.
- يصرح السطر الثاني بكاتبة طبع (**PrintWriter**) يلقي عليها إسم **masterPrinter** تمكننا من طبع بيانات في الصادرة المذكورة أعلاه. يتيح لنا هذا التصريح تنفيذ عمليات طبع تنطبق على شتى أنواع البيانات، منها الأعداد الكاملة و الأعداد الحقيقية و غيرها.

3.2.10 تطبيق: تحيين حسابات بنكية

في هذا الجزء نعتزم كتابة برنامج يتلقى ملفين:

- ملفا فيه حسابات بنكية نمثلها برقم الحساب و الرصيد الموجود فيه، نلقي عليه إسم **master.txt** .
- ملفا فيه عمليات بنكية منها الإداع و السحب نمثلها برقم حساب و رمز يدل على نوع العملية (سحب أو إداع) و قيمة العملية؛ نلقي على هذا الملف إسم **trans.txt**. نفترض أن كلا الملفان مرتبان حسب أرقام الحساب و أن ملف المعاملات لا يحتوي على أكثر من عملية واحدة بالنسبة لكل رقم حساب.

نطلب من هذا البرنامج أن يسجل في الملف الرئيسي نتيجة تطبق العمليات الموجودة في ملف المعاملات. يقع تنفيذ هذا البرنامج في طورين إثنين، كما يشير الرسم رقم 1.10:

- في الطور الأول نحول محتوى الملف الرئيسي إلى ملف وقتي نلقي عليه إسم **store.txt** أي ملف الخزن.
- في الطور الثاني نطلع على المعاملات الموجودة في ملف المعاملات، فنطبقها على الحسابات المعنية في ملف الخزن و نحول البيانات النهائية إلى الملف الرئيسي.

يتمثل الطور الثاني في مدارين إثنين:

- في المدار الأول نستعرض المعاملات الموجودة في ملف المعاملات على التوالي. فنهتم بالتفتيش عن الحساب المناسب لهذه المعاملة في ملف الحسابات (وهو ملف الخزن، بصفة وقتية).
- في المدار الثاني، الموجود في داخل المدار الأول، نفتش على الحساب الذي تنطبق عليه العملية الحالية. قبل أن نصل للحساب المعني، قد نمر بحسابات لا تنطبق عليها عمليات، فننسخها كما هي من ملف الخزن إلى الملف الرئيسي.

عند ختام المدار الأول (الخارجي)، يحتوي الملف الرئيسي على جملة الحسابات البنكية حيث أن الحسابات التي تنطبق عليها معاملات وقع تغييرها على ضوء المعاملات، و الحسابات التي لا تخصها معاملات وقع نقلها كما هي من ملف الخزن إلى الملف الرئيسي. نقدم البرنامج التالي:

```
package file;                                                    //  1.
/**                                                              //  2.
 * @author Ali Mili and Ahmed Ferchichi                         //  3.
 */                                                              //  4.
                                                                 //  5.
import java.util.Scanner;                                        //  6.
import java.io.*;                                                //  7.
                                                                 //  8.
public class Main {                                              //  9.
                                                                 // 10.
    /**                                                          // 11.
     * @param args the command line arguments                   // 12.
     */                                                         // 13.
    public static void main(String[] args)                       // 14.
       throws IOException {                                      // 15.
        // TODO code application logic here                      // 16.
                                                                 // 17.
        FileReader masterFile;                                   // 18.
        BufferedReader masterBuffer;                             // 19.
                                                                 // 20.
```

```
masterFile = new FileReader("master.txt");                          // 21.
masterBuffer = new BufferedReader(masterFile);                      // 22.
                                                                    // 23.
FileWriter storeFile;                                               // 24.
BufferedWriter storeBuffer;                                         // 25.
                                                                    // 26.
storeFile = new FileWriter("store.txt");                            // 27.
storeBuffer = new BufferedWriter(storeFile);                        // 28.
                                                                    // 29.
String masterLine = new String (masterBuffer.readLine());          // 30.
while (masterLine != null)                                          // 31.
{                                                                   // 32.
    storeBuffer.write(masterLine);                                  // 33.
    storeBuffer.newLine();                                          // 34.
    masterLine = masterBuffer.readLine();                           // 35.
}                                                                   // 36.
storeBuffer.flush();                                                // 37.
storeBuffer.close();                                                // 38.
                                                                    // 39.
FileReader storeFileReader;                                         // 40.
BufferedReader storeBufferReader;                                   // 41.
                                                                    // 42.
storeFileReader = new FileReader("store.txt");                      // 43.
storeBufferReader = new BufferedReader(storeFileReader);            // 44.
                                                                    // 45.
FileReader transFileReader;                                         // 46.
BufferedReader transBufferReader;                                   // 47.
                                                                    // 48.
transFileReader = new FileReader("trans.txt");                      // 49.
transBufferReader = new BufferedReader(transFileReader);            // 50.
                                                                    // 51.
FileOutputStream masterStream = new FileOutputStream("master.txt");
PrintWriter masterPrinter = new PrintWriter(masterStream);          // 53.
                                                                    // 54.
String transLine = new String (transBufferReader.readLine());      // 55.
                                                                    // 56.
while (transLine != null)                                           // 57.
{                                                                   // 58.
    Scanner transScanner = new Scanner(transLine);                  // 59.
    int acctNumber = transScanner.nextInt();                        // 60.
    String operationCode = transScanner.next();                     // 61.
    float operationValue = transScanner.nextFloat();                // 62.
                                                                    // 63.
    String storeLine;                                               // 64.
    storeLine = storeBufferReader.readLine();                       // 65.
    Scanner storeScanner = new Scanner (storeLine);                 // 66.
    int curAcctNumber = storeScanner.nextInt();                     // 67.
    float acctBalance = storeScanner.nextFloat();                   // 68.
                                                                    // 69.
    while (curAcctNumber<acctNumber)                                // 70.
    {                                                               // 71.
        masterPrinter.println(curAcctNumber+"\t"+acctBalance);      // 72.
        storeLine = storeBufferReader.readLine();                   // 73.
        storeScanner = new Scanner(storeLine);                      // 74.
                                                                    // 75.
        curAcctNumber = storeScanner.nextInt();                     // 76.
        acctBalance = storeScanner.nextFloat();                     // 77.
    }                                                               // 78.
    // now curAcctNumber = acctNumber                               // 79.
                                                                    // 80.
    if (operationCode.equals("W"))                                  // 81.
    {                                                               // 82.
        acctBalance = acctBalance - operationValue;                 // 83.
```

```
            }                                                                   // 84.
            Else                                                                // 85.
            {                                                                   // 86.
                acctBalance = acctBalance + operationValue;                     // 87.
            }                                                                   // 88.
            masterPrinter.println(curAcctNumber+"\t"+acctBalance);              // 89.
            transLine = transBufferReader.readLine();                           // 90.
        }                                                                       // 91.
        masterPrinter.close();                                                  // 92.
        //  signal the end                                                      // 93.
        System.out.println("done.");                                            // 94.
    }                                                                           // 95.
}                                                                               // 96
```

نعلق على هذا البرنامج في ما يلي:

- السطران 6 و 7: التصريح بالباقات اللازمة لمعالجة الملفات.
- الأسطر 18 إلى 22: التصريح بقارءة الملف و وسيطة القراءة التابعتين للملف الرئيسي. ربطهما بالملف الرئيسي كما هو معروف من طرف نظام التشغيل.
- الأسطر 24 إلى 28: التصريح بكتابة الملف و وسيطة الكتابة التابعتين لملف الخزن. ربطهما بالملف الرئيسي كما هو معروف من طرف نظام التشغيل.
- الأسطر 30 إلى 36: تحويل محتوى الملف الرئيسي إلى ملف الخزن، سطرا سطرا. نعرف أننا وصلنا إلى نهاية الملف عندما تسترجع عملية `readLine()` سطرا فارغا.
- السطر 37: تسفر هذه العملية على تفريغ السجل الوسيط إلى القرص الصلب.
- السطر 38: غلق الملف لكي نتمكن من فتحه للكتابة مستقبلا.
- الأسطر إلى 40 إلى 44: هنا يبدأ الطور الثاني، فنهيئ ملف الخزن للقراءة بواسطة قارءة ملف و وسيطة قراءة.
- الأسطر 46 إلى 50: نهيئ ملف المعاملات للكتابة بواسطة كاتبة ملف و وسيطة الكتابة.
- السطران 52 و 53: نهيئ الملف الرئيسي للكتابة بواسطة صادر و كاتبة الطبع.
- السطر 55: نقرأ ملف المعاملات سطرا سطرا.
- السطر 57: نعرف أننا وصلنا إلى نهاية ملف المعاملات عندما تسترجع عملية `readLine()` سطرا فارغا.
- السطر 59: نستعمل ناسخة `transScanner` لكي نفكك عناصر المعاملة الحالية.
- الأسطر 60 إلى 62: نستخرج من السطر الحالي عناصر المعاملة، و هي رقم الحساب و نوع العملية و قيمة العملية.
- الأسطر 64 إلى 66: نقرأ سطرا من ملف الخزن (يمثل حسابا بنكيا) و نصرح بناسخة لهذا السطر تمكننا من تفكيكه.
- السطران 67 و 68: نستخرج رقم الحساب الجاري في ملف الخزن و رصيده.
- الأسطر 70 إلى 78: ما دمنا لم نجد الحساب موضوع المعاملة الحالية، نواصل قراءة البيانات من ملف الخزن و نقلها للملف الرئيسي. نفترض أن كل المعاملات الموجودة في ملف المعاملات تخص حسابات موجودة في ملف الخزن.
- السطر 79: عند نهاية المدار، نفترض أننا وجدنا الحساب المعني بالعملية الحالية.
- الأسطر 81 إلى 88: نحتسب الرصيد الجديد للحساب المعني حسب الرصيد القديم و نوعية العملية و قيمة العملية.
- السطر 89: كتابة رقم الحساب الجاري و رصيده الجديد في الملف الرئيسي.
- السطر 90: التأمل في المعاملة الموالية في ملف المعاملات.
- السطر 92: غلق الملف الرئيسي.
- السطر 94: الإعلان على نهاية التنفيذ.

عندما ننفذ هذا البرنامج على البيانات التالية

store.txt

trans.txt		
231321	D	50.00
547654	W	260.00
897657	W	270.00
987879	D	350.00

master.txt	
154654	300.00
231321	450.00
342543	280.00
432657	657.00
547654	760.00
675432	540.00
700879	563.00
897657	570.00
987879	650.00

يسفر على النتائج التالية

```
run:
done.
BUILD SUCCESSFUL (total time: 1 second)
```

و تصبح الملفات المعنية على النحو التالي:

store.txt	
154654	300.00
231321	450.00
342543	280.00
432657	657.00
547654	760.00
675432	540.00
700879	563.00
897657	570.00
987879	650.00

trans.txt		
231321	D	50.00
547654	W	260.00

897657	W	270.00
987879	D	350.00

master.txt	
154654	300.0
231321	500.0
342543	280.0
432657	657.0
547654	500.0
675432	540.0
700879	563.0
897657	300.0
987879	1000.0

كما نلاحظ، فإن الملف الرئيسي الجديد يعكس فعلا المعاملات التي يحتوي عليها ملف المعاملات.

3.10 تمارين

1 [م] غيروا البرنامج الذي قدمناه في الجزء 2.10 لكي يتأكد من أن المعاملات الموجودة في ملف المعاملات تخص حسابات موجودة في الملف الرئيسي. إن كان عدد الحساب الموجود في ملف المعاملات غير موجود في الملف الرئيسي، فيثير البرنامج خطأَ.

2 [س] غيروا البرنامج الذي قدمناه في الجزء 2.10 لكي يتأكد من أن عمليات السحب ممكنة، أي أن قيمة السحب أقل من رصيد الحساب. إن كانت قيمة السحب تفوق رصيد الحساب، فيثير البرنامج خطأَ.

3 [س] غيروا البرنامج الذي قدمناه في الجزء 2.10 لكي يتأكد من أن العمليات الموجودة في ملف المعاملات تختصر على العمليتين المعروفتين (الأيداع و السحب). إن كان رمز العملية دون الرمزين المختارين لغرض الإيداع و السحب، فيثير البرنامج خطأَ.

4 [م/ص] بخصوص البرنامج الذي قدمناه في الجزء 2.10، نفترض أن لدينا ملفين إثنين للمعاملات، صادرين مثلا من فرعين للبنك. هناك عدد من الأساليب لمعالجة هذا الوضع:

أ. تطبيق البرنامج بإستعمال الملفين على التوالي.
ب. إدماج ملفي المعاملات لملف موحد، ثم تطبيق البرنامج.
ت. فتح الملفين بصفة متزامنة و تطبيق البرنامج بتحليل ملفين عوضا عن ملف واحد.

المطلوب منكم تطبيق الحل رقم (ب) أعلاه.

5 [م/ص] بخصوص البرنامج الذي قدمناه في الجزء 2.10، نفترض أن لدينا ملفين إثنين للمعاملات، صادرين مثلا من فرعين للبنك. هناك عدد من الأساليب لمعالجة هذا الوضع:

أ. تطبيق البرنامج بإستعمال الملفين على التوالي.
ب. إدماج ملفي المعاملات لملف موحد، ثم تطبيق البرنامج.
ت. فتح الملفين بصفة متزامنة و تطبيق البرنامج بتحليل ملفين عوضا عن ملف واحد.

المطلوب منكم تطبيق الحل رقم (ت) أعلاه.

6 [م] غيروا البرنامج الذي قدمناه في الجزء 2.10 لكي يتمكن من معالجة عدد من العمليات الخاصة بنفس الحساب. أي أن ملف المعاملات قد يحتوي على عدد من العمليات على نفس الحساب، فتكون متتالية.

الباب الرابع

وسائل التجرد في جافا

تتكون برامج جافا من مكونين إثنين، و هما البيانات (التي تمثل الكائنات التي تهمنا) و التعليمات (التي تمثل علميات التحليل و التحويل التي نقوم بها على هذه الكائنات). بقدر ما تتعقد البيانات و التعليمات، فيتعقد تحرير البرامج و التأكد من سدادها. فيأتي التجرد كأهم وسيلة للتحكم في تعقيد البرامج و السيطرة عليه. نفرث بين نوعين من التجرد:

- تجرد التعليمات، الذي يتمثل في أننا نعوض هيكلا معقدا من التعليمات بإسم مجرد يدل على مفعول هذه العمليات و يمثل تطبيقها.

- تجرد البيانات، الذي يتمثل في أننا نعوض هياكل بيانات معقدة بإسم مجرد يدل على تأويل هذه البيانات و بعمليات مطابقة لهذا التأويل.

في هذا الجزء ننظر على التوالي في تجرد التعليمات و تجرد البيانات. ففي الفصل الحادي عشر ننظر في الوظائف، التي تمثل تجرد التعليمات، و في الفصل الموالي نهتم بنوع خاص من الوظائف، و هو الوظائف الإستقرائية. هذا، و

نهتم في الفصل الثالث عشر بإستعمال الكائنات في لغة جافا، التي تمثل تجرد البيانات، ثم نهتم في الفصل الموالي ببرمجة الكائنات.

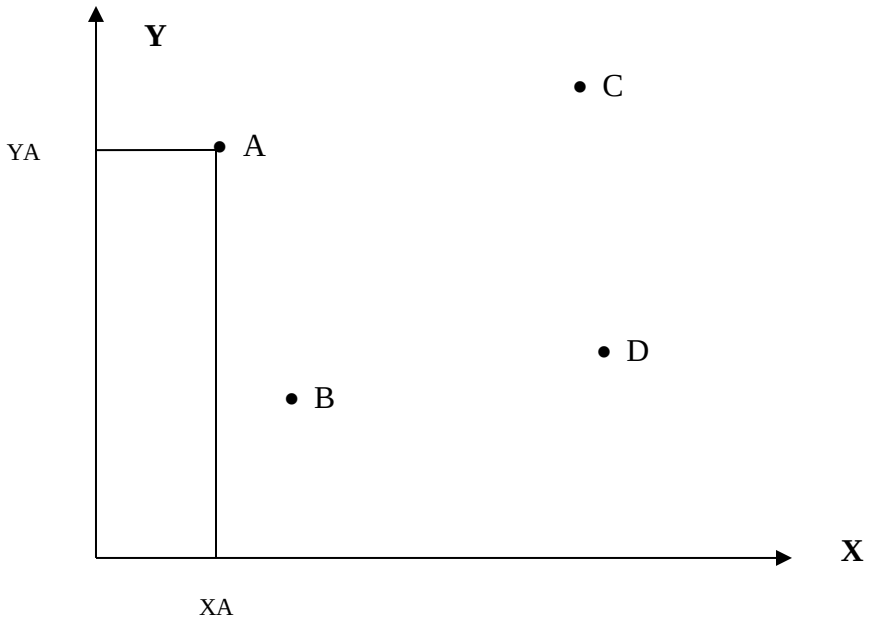
الفصل الحادي عشر

تجرد التعليمات، الوضائف

في لغة جافا، نفرق بين نوعين إثنين من الوضائف: الوضائف التي تسترجع نتائج من نوع بيانات ما (كنتيجة منطقية أو نتيجة عددية أو غيرها) و تقوم أساس بعملية تحليل بيانات؛ الوضائف التي لا تسترجع نتائجا في حد ذاتها بل تقوم بعمليات تحويل البيانات. سننظر فيها على التوالي، في الأجزاء التالية، بواسطة أمثلة تطبيقية.

1.11 وضيفة منطقية: تقييم و مقارنة المسافات

نعتبر عددا من النقط على سطح عادي، و نهتم بأن نتفقد إن كانت هذه النقط تمثل مثلثا مستطيلا، مثلثا ذات جانبين متساويين أو مثلثا ذات ثلاثة جوانب متساوية. نمثل كل نقطة بإحداثياتها في السطح، حسب المحورين الأفقي (X) و العمودي (Y). علما أننا نعتبر أربعة نقط، فهذا يفرض علينا النظر في أربعة مثلثات.



لنتمكن من تحليل مثلث قصد التأكد من صيغته (كمثلث مستطيل أو مثلث ذات جانبين متساويين أو مثلث ذات ثلاثة جوانب متساوية)، نقوم بتقييم مسافاته بين كل نقطتين، فنعين المثلث المستطيل بالخبر المنطقي التالي:

$$d(A,B)^2 + d(B,C)^2 = d(A,C)^2 \vee$$
$$d(A,C)^2 + d(C,B)^2 = d(A,B)^2 \vee$$
$$d(B,A)^2 + d(A,C)^2 = d(B,C)^2.$$

و نعين المثلث ذات الجانبين المتساويين بالخبر المنطقي التالي:

$$d(A,B) = d(A,C) \vee d(B,A) = d(B,C) \vee d(C,A) = d(C,B).$$

و نعين المثلث ذات الجانبين المتساويين بالخبر المنطقي التالي:

$$d(A,B) = d(A,C) = d(B,C).$$

حيث نقيم المسافة بين نقطتين بواسطة الإحداثيات التابعة للنقطتين حسب المعادلة التالية:

$$d(A,B) = \sqrt{(XA - XB)^2 + (YA - YB)^2}.$$

أول وضيفة نقوم بإنجازها هي وضيفة تقيم المسفة بين نقطتين في السطح. لهذا الغرض، نعين السطح كقسم من الكائنات، و نزود هذا القسم بوضيفة المسافة، حسب ما جاء في الأسطر 20 إلى 23. في هذه الأسطر، نقوم بتعيين وضيفة المسافة، فنصرح بأن هذه الوضيفة لها أربعة معلمات، و هي إحداثيات النقطتين المعنيتين بتقييم المسافة. ففي عنوان هذه الوضيفة، أي

```
float dist (float X1, float Y1, float X2, float Y2)
```

نصرح بإسم الوضيفة و بنوع البيانات التي تحسبها هذه الوضيفة، كما نستعرض قائمة المعلمات التي تستعملها هذه الوضيفة، مع التصريح بنوع بياناتها. في حالة هذه الوضيفة، نستعمل أربعة معلمات من نوع الأعداد الحقيقية و نستخرج المسافة بين النقطتين في شكل عدد حقيقي. أما جسم هذه الوضيفة، و هو

```
{
   return (float)Math.sqrt((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2));
}
```

فهو ينص على أن المسافة من نوع الأعداد الحقيقية، و أنها تساوي الجذر الثاني لجملة مربع الفرق بين إحداثيات X و مربع الفرق بين إحداثيات Y.

```
1.  /*
2.   * Main.java
3.   *
4.   * Created on October 17, 2007, 2:03 PM
5.   *
6.   * To change this template, choose Tools | Template Manager
7.   * and open the template in the editor.
8.   */
9.
10.  package mylatest;
11.
12.  /**
13.  *
14.  * @author mili
15.  */
16.
17.  class plane {
18.
19.
20.  float dist (float X1, float Y1, float X2, float Y2)
21.     {
22.        return (float)Math.sqrt((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2));
23.     }
24.  }
25.
26.  public class Main {
27.     /**
28.      * @param args the command line arguments
29.      */
30.     public static void main(String[] args) {
31.         // TODO code application logic here
32.         float XA, YA;  // point A
33.         float XB, YB;  // point B
34.         XA=2; YA=7; XB=9; YB=6;
35.         plane p1=new plane();
36.         System.out.println(p1.dist(XA,YA,XB,YB));
37.
38.     }
39.  }
```

بينما تمثل الأسطر 20 إلى 23 تعيين وضيفة المسافة، فإن سطر 36 يمثل نداء هذه الوضيفة، حيث نطبقها على النقطتين A و B حسب إحداثياتها الأفقية (X) و العمودية (Y). نفرق في هذا الخصوص بين المعلمات الشكلية، التي تظهر في تعيين الوضيفة، في هذه الحالة

X1, Y1, X2, Y2

و المعلمات الحقيقية، التي تضهر في نداء الوضيفة، في هذه الحالة

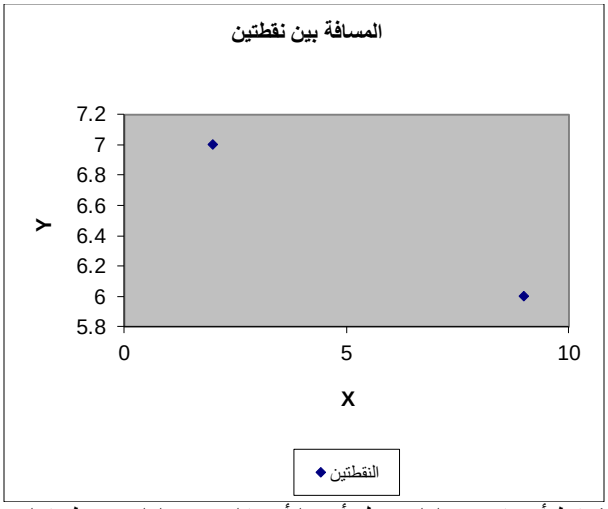
XA, YA, XB, YB.

عند نداء الوضيفة، يقع تعويض المعلمات الشكلية بالمعلمات الحقيقية في جسم الوضيفة، لكي تتنفذ الوضيفة على البيانات الحقيقية. إذا قمنا بنداء الوضيفة عدة مرات، فيقع تنفيذها على عدة بيانات حقيقية.

في هذا البرنامج، نصرح بمتغيرات تمثل نقطتين، A و B (في السطرين 32 و 33)، ثم نسند قيمات لهذه المتغيرات لتعيين النقط التي تهمنا (في السطر 34) ثم نطلب أن يقع طبع المسافة بين النقطتين (سطر 36)، فتكون النتيجة على النحو التالي:

```
init:
deps-jar:
compile:
run:
7.071068
BUILD SUCCESSFUL (total time: 0 seconds)
```

فيجد الحاسوب أن المسافة بين النقطتين هي 7.071068.



إن تحليل المثلثات يتطلب، لا فقط أن نقيم مسافات، بل أيضا أن نقارن مسافات. تطبيقيا، نحتاج لوضيفة تمكننا أن نعرف إن كانت مسافتان متساويتان أم لا. فنقترح وضيفة يكون عنوانها على النحو التالي،

```
boolean equaldist (float dist1, float dist2)
```

تستعمل معلمتين تمثلان المسافتين موضوع المقارنة، و تنتج قيمة منطقية حسب إن كانت المسافتان متساويتان أم لا. فنهتم بجسم هذه الوضيفة. علما أن المسافات أعداد حقيقية لا يمكن تمثيلها بدقة تامة في الحاسوب، فغالبا ما نحسب هذه الأعداد بالتقريب. لذا، لا يمكن التأكد من مساواة عددين حقيقيين بالضبط، بل بالتقريب: فنعتبر أن عددين متساويين إذا كان الفرق بينهما ضئيلا، حسب معيار معيين للدقة، مثلا

```
double epsilon = 0.000001;
```

بناء على إختيار هذا المعيار، نعتبر أن عددين متساويين كلما كان الفرق بينهما لا يتجاوز هذه القيمة، فنجد الوضيفة التالية:

```
boolean equaldist (float dist1, float dist2)
    {
        double epsilon = 0.000001;
        return (boolean)(Math.abs(dist1-dist2)<epsilon);
    }
```

نلاحظ أن هذه الوضيفة تقارن القيمة المطلقة للفرق بين المسافتين، لأننا لا نعرف أي مسافة أكبر من الأخرى. تطبيقا لهذه الوضيفة، نعتزم أن نقارن المسافة من A إلى B مع المسافة من B إلى A، فنكتب التعليمات التالية:

```
if (pl.equaldist(pl.dist(XA,YA,XB,YB),pl.dist(XB,YB,XA,YA)))
        {System.out.println("same distance");}
        else
        {System.out.println("different distance");}
```

فيسفر تنفيذ هذا البرنامج على نص:

```
same distance
```

أي أن الحاسوب إعتبر المسافتين متساويتين. أما إذا إعتزمنا على مقارنة المسافة بين نقتطي A و B مع المسافة بين نقطتي A و A، التي تساوي الصفر، فنتوقع أن تختلف النتيجة. فعلا، عندما نكتب

```
if (pl.equaldist(pl.dist(XA,YA,XB,YB),pl.dist(XA,YA,XA,YA)))
        {System.out.println("same distance");}
        else
        {System.out.println("different distance");}
```

فيسفر تنفيذ هذا البرنامج على نص:

```
different distance.
```

2.11 وضائف عملية: تحليل مثلثات

إن الوضائف التي درسناها لحد الآن تتسم بأن مفعولها يتمثل في حساب قيمة ما، من نوع عددي (مسافة) أو منطقي (مساوات بين عددين). فننظر الآن في وضائف لا ترجع قيمة ما، بل تقوم بعمليات تتمثل في طبع نتائج أو في تغيير حالة البرنامج (قيمة متغيرات) أو غيرها. يقال عن النوع الأول من الوضائف أنها وضائف صافية بينما يقال عن النوع الثاني من الوضائف (موضوع هذا الجزء) أنها وضائف ذات مفعول جانبي.

نعتبر مجموعة من النقط على سطح ما، و ننظر إلى ثلاثة نقط منها فنهتم بتحليلها لنعين نوع المثلث المبني على هذه النقط، علما أن لنا إمكانية تقييم المسافات بين النقط و إمكانية مقارنة المسافات. لهذا الغرض، نعين وضيفة يكون عنوانها على النحو التالي:

```
void triangularanalysis
(float XA, float YA, float XB, float YB, float XC, float YC)
```

تستعمل هذه الوضيفة ستة معلمات، لتمثل الثلاثة النقط المعنية، مستعملة إحداثيتين بالنسبة لكل نقطة. نحرر هذه الوضيفة حسب نقاش مواصفات المثلثات، فأول ما نختبر هو إن كان المثلث مستطيلا؛ إن لم يكن مستطيلا، نختبر إن كان ذات ثلاث جوانب متساوية؛ إن لم يكن، نتفقد إن كان ذات جانبين متساويين؛ و إلا، فنصرح أنه لا يتسم بأي صفة تهمنا. نقدم جسم هذه الوضيفة فيما يلي: نلاحض أننا وضعنا المسافات الثلاثة بين النقط في ثلاثة متغيرات، لنوضح نص البرنامج.

```
{
```

```
        float dAB = dist(XA,YA,XB,YB);
        float dBC = dist(XB,YB,XC,YC);
        float dAC = dist(XA,YA,XC,YC);

        if ((equaldist(dAB*dAB+dAC*dAC,dBC*dBC) ||
            equaldist(dAB*dAB+dBC*dBC,dAC*dAC) ||
            equaldist(dAC*dAC+dBC*dBC,dAB*dAB))  )
        {
            System.out.println("rectangular triangle");
        }
        else if ((equaldist(dAB,dAC) && equaldist(dAB,dBC)))
        {
            System.out.println("equilateral");
        }
        else if ((equaldist(dAB,dAC) ||
                  equaldist(dAB,dBC) ||
                  equaldist(dAB,dBC)) )
        {
            System.out.println("isoceles");
        }
        else
        {
            System.out.println("no property");
        }
    }
```

نلاحظ أن بفضل إستعمال وضائف المسافة و المساوات، أصبح نص البرنامج في غاية من البساطة، إذ أنه يعكس تحليلنا بصفة مباشرة. ننفذ هذا البرنامج على النقط التالية:

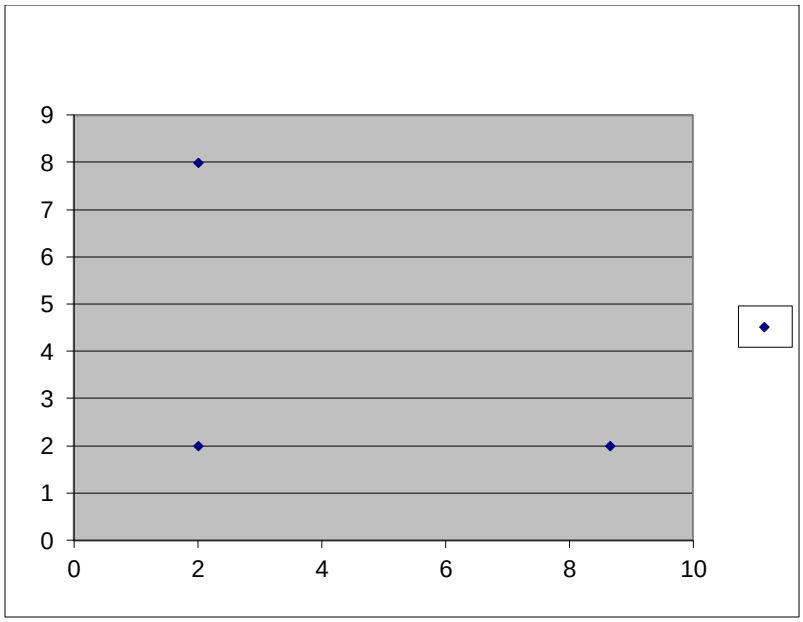
النقط	الإحداثية الأفقية	الإحداثية العمودية
A	2	2
B	8.6543787	2
C	2	7.987678

كما نرى في الرسم التالي، فإن هذه النقط تمثل مثلثا مستطيلا. ننفذ هذا البرنامج، فيصرح أن المثلث مستطيلا، حسب توقعنا. فنعتزم بتغيير المثلث قليلا، حيث نحول نقطة B إلى اليمين قليلا و نحول نقطة C إلى الفوق قليلا، حيث تصبح إحداثيات النقط على النحو التالي:

```
XA=2; YA=2; XB=8.6543678F; YB=2; XC=2; YC=7.9876678F;
```

فمبدئيا يبقي المثلث مستطيلا، كما كان، لأن زاوية A لم تتغير في هذه العملية، لكن البرنامج يجد أن المثلث ليس مستطيلا، إذ يصرح.

```
no property.
```



فنعتزم أن نغير معيار المساوات، فعوضا أن يكون المعيار

```
double epsilon = 0.000001;
```

نجعله

```
double epsilon = 0.00001;
```

فيصرح البرنامج أن المثلث مستطيلا. كيف نفسر أن تغيير النقط تغييرا ضئيلا في إتجاه لا يغير طبيعة المثلث المستطيل يؤدي إلى تقرير مختلف، حيث يجد أن المثلث ليس مستطيلا؛ ثم عندما نغير معيار المساواة بدون أن نغير المثلث، يجد البرنامج من جديد أن المثلث مستطيلا.

نلاحظ أن وضيفة تحليل الثلثات تختبر صفات المثلثات بصفة تسلسلية، بدون إعتبار أن نفس المثلث قد يتسم بأكثر من صفة؛ مثلا، يمكن لنفس المثلث أن يكون مستطيلا و ذات جانبين متساويين. ننظم صفات المثلثات في المائدة التالية:

		وجود زاوية مستقيمة	
		لا	نعم
مساواة بين الجوانب	جوانب مختلفة	مثلث عادي	مثلث مستقيم عادي
	جانبان متساويان	مثلث عادي ذات جانبين متساويين	مثلث مستقيم متناسق
	جوانب متساوية	مثلث ذات ثلاثة جوانب متساوية	

بحيث أن هناك خمسة حالات يجب أن نفرق بينها (تناسب الخمسة مواقع المشار إليها في المائدة أعلاه). لتنفيذ برنامج يفرز بين هذه الحالات، نهتم بوظائف منطقية تختبر وجود زاوية مستقيمة و وظائئف تختبر وجود جوانب متساوية. إنطلاقا من هذه الوظائف، نألف الوظيفة التي تحلل المثلثات كما يلي:

```
1.  void triangularanalysis (float XA, float YA, float XB, float YB, float XC, float YC)
```

```
2.      {
3.          if (rectangular(XA,YA,XB,YB,XC,YC))
4.          {
5.              if (isoceles(XA,YA,XB,YB,XC,YC))
6.              {
7.                  System.out.println("isoceles rectangular");
8.              }
9.              else
10.             {
11.                 System.out.println("plain rectangular");
12.             }
13.         }
14.         else
15.         {
16.             if (equilateral(XA,YA,XB,YB,XC,YC))
17.             {
18.                 System.out.println("equilateral");
19.             }
20.             else
21.             {
22.                 if (isoceles(XA,YA,XB,YB,XC,YC))
23.                 {
24.                     System.out.println("non-rectangular isoceles");
25.                 }
26.                 else
27.                 {
28.                     System.out.println("plain triangle");
29.                 }
20.             }
21.         }
22.     }
23. }
```

نلاحظ أن هذه الوظيفة تطابق هيكل المائدة سطرا سطرا: فتعكس الأسطر 3 إلى 13 العمود الأيمن للمائدة، الذي يخص المثلثات المستطيلة، بحيث تختبر المثلث بالنسبة لوجود زاوية مستقيمة، و في هذه الحالة تسختبر الثلث يالنسبة لوجود جانبين متساويين. أما الأسطر 15 إلى 21، فهي تعكس العمود اليساري للمائدة أعلاه، بحيث نعرف أن المثلث غير مستطيلا، فنختبر جوانبه لنبين إن كانت كلها متساوية، ام كان جانبان متياويان، أم كانت الجوانب الثلاثة تختلف عن بعضها بعضا. إن وظيفة تحليل المثلثات، أعلاه، مبنية على ثلاثة وظائف، تختبر على التوالي إن كان المثلث مستطيلا، و إن كان ذات جانبين متساويين، و إن كان ذات ثلاثة جوانب متساوية. نألف هذه الوظائف كما يلي:

```
boolean rectangular (float XA, float YA, float XB, float YB, float XC, float YC)
{
    float dAB = dist(XA,YA,XB,YB);
    float dBC = dist(XB,YB,XC,YC);
    float dAC = dist(XA,YA,XC,YC);

    return ((equaldist(dAB*dAB+dAC*dAC,dBC*dBC) ||
        equaldist(dAB*dAB+dBC*dBC,dAC*dAC) ||
        equaldist(dAC*dAC+dBC*dBC,dAB*dAB)));
}

boolean isoceles (float XA, float YA, float XB, float YB, float XC, float YC)
{
    float dAB = dist(XA,YA,XB,YB);
    float dBC = dist(XB,YB,XC,YC);
    float dAC = dist(XA,YA,XC,YC);

    return ((equaldist(dAB,dAC) ||
              equaldist(dAB,dBC) ||
              equaldist(dAB,dBC)));
}

boolean equilateral (float XA, float YA, float XB, float YB, float XC, float YC)
{
```

```
        float dAB = dist(XA,YA,XB,YB);
        float dBC = dist(XB,YB,XC,YC);
        float dAC = dist(XA,YA,XC,YC);

        return ((equaldist(dAB,dAC) && equaldist(dAB,dBC)));
    }
```

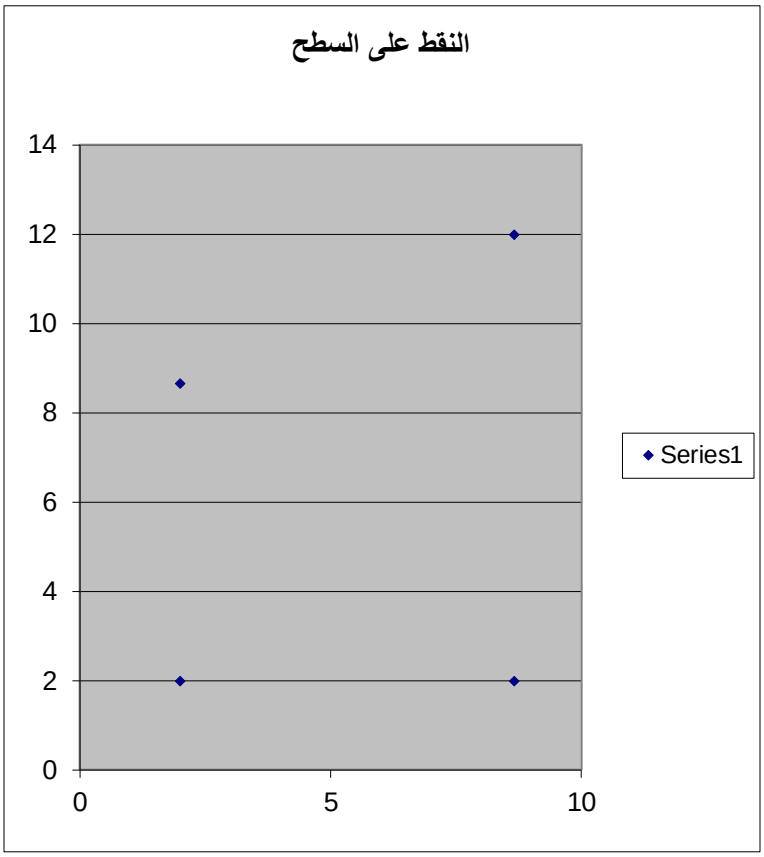
إن هذه الوظائف تستغنى عن التفسير، إذ أنها تعكس الصفات المعنية بصفة مباشرة. فننكب على تطبيق هذا البرنامج، بصفة تبرز فضل الوظائف في البرمجة. مثلا، نضع أربعة نقط على السطح، فنستعمل وظيفة التحليل على كل مثلث نكونه من ثلاثة نقط. لهذا الغرض، نكتب البرنامج الرئيسي كما يلي:

```
1.  public static void main(String[] args) {
2.        // TODO code application logic here
3.        float XA, YA;  // point A
4.        float XB, YB;  // point B
5.        float XC, YC;  // point C
6.        float XD, YD;  // point D
7.        XA=2;YA=2;XB=8.6543678F;YB=2; XC=2;YC=8.6543678F;XD=8.6543678F; YD=12;
8.        plane pl=new plane();
9.        //  exclude D
10.       System.out.println("Triangle:  A, B, C");
11.       pl.triangularanalysis(XA,YA,XB,YB,XC,YC);
12.
13.       //  exclude A
14.       System.out.println("Triangle:  D, B, C");
15.       pl.triangularanalysis(XD,YD,XB,YB,XC,YC);
16.
17.       //  exclude B
18.       System.out.println("Triangle:  A, D, C");
19.       pl.triangularanalysis(XD,YD,XA,YA,XC,YC);
20.
21.       //  exclude C
22.       System.out.println("Triangle:  A, B, D");
23.       pl.triangularanalysis(XD,YD,XA,YA,XB,YB);
24.       }
```

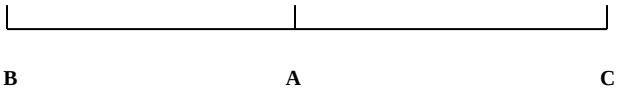
في الأسطر 3 إلى 6 نصرح بالأربعة نقط، حيث نمثل كل نقطة بالعددين الذين يمثلان الإحداثيتين التابعتين لهذه النقطة. في السطر السابع، نضع هذه النقط على السطح. طبعا، هناك أربعة مثلثات مختلفة يمكن بنائها على أربعة نقط، حيث أن كل مثلث يمكن تعيينه بواسطة نقطة يقع حذفها. ففي كل مجموعة سطرين من شكل 10 و 11، أو 14 و 15، أو 18 و 19، أو 22 و 23، نعين مثلثا يمكن بنائه بواسطة النقط الأربعة، ثم ننادي برنامج تحليل المثلثات على هذا المثلث. فيسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
Triangle:  A, B, C

isoceles rectangular

Triangle:  D, B, C

plain triangle

Triangle:  A, D, C

plain triangle

Triangle:  A, B, D

plain rectangular
```

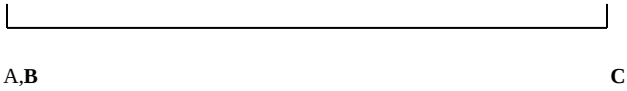
نتأمل فيما يلي في الرسم الذي يمثل النقط المعنية، لنمكن القارئ من التأكد من سداد التحليل:



نلاحض أن هذا البرنامج لا يتأكد من أن كانت الثلاثة نقط تعين مثلثا أم لا: مثلا، إذا كانت النقط الثلاثة في الشكل التالي:



لوجد برنامجنا أن هذا مثلثا ذات جانبين متساويين. ثم إذا كانت النقط الثلاثة في الشكل التالي:



لوجد برنامجنا أن هذا المثلث ذات جانبين متساويين، و أنه مستطيلا. ثم إذا كانت النقط الثلاثة متساوية، فيجد برنامجنا أن المثلث ذات ثلاثة جوانب متساوية و أنه مستطيلا.

يمكن إجتناب كل هذه الحالات بإختبار بسيط يتأكد أن الثلاثة نقط ليست على نفس السطر. علما أن لنا إمكانية قيس المسافات و مقارنة المسافات، نبين أن ثلاثة نقط على نفس السطر إذا كانت المسافة بين نقطتين منها تساوي جملة المسافتين بين هاتين النقتطين و النقطة الثالثة، فنكتب:

```
boolean inline (float XA, float YA, float XB, float YB, float XC, float YC)
    {
        float dAB = dist(XA,YA,XB,YB);
```

```
        float dBC = dist(XB,YB,XC,YC);
        float dAC = dist(XA,YA,XC,YC);

        return ((equaldist(dAB,dAC+dBC))||
                (equaldist(dAC,dAB+dBC))||
                (equaldist(dBC,dAC+dAB)));
    }
```

ثم نغير وظيفة تحليل المثلثات بحيث تختبر المثلث قصد التأكد من أن نقطه ليست على نفس السطر، فنكتبها كما يلي:

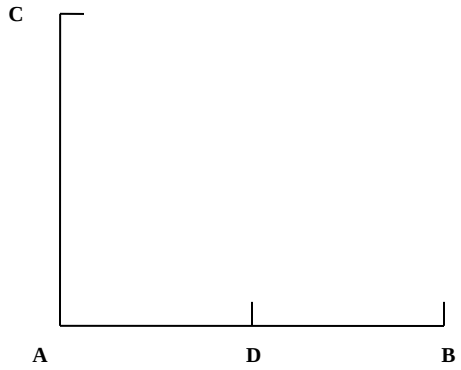
```
void triangularanalysis (float XA,float YA,float XB,float YB,float XC,float YC)
    {
        if (inline (XA,YA,XB,YB,XC,YC))
        {
            System.out.println("not a triangle");
        }
        else
        {
            if (rectangular(XA,YA,XB,YB,XC,YC))

            {
                if (isoceles(XA,YA,XB,YB,XC,YC))
                {
                System.out.println("isoceles rectangular");
                }
                else
                {
                    System.out.println("plain rectangular");
                }
            }
            else
            {
                if (equilateral(XA,YA,XB,YB,XC,YC))
                {
                    System.out.println("equilateral");
                }
                else
                {
                    if (isoceles(XA,YA,XB,YB,XC,YC))
                    {
                        System.out.println("non-rectangular isoceles");
                    }
                    else
                    {
                        System.out.println("plain triangle");
                    }
                }
            }
        }
    }
}
```

فنختبر هذه الوظيفة في حالات تكون فيها النقط على نفس السطر أو حتى متساوية تماما، حيث نكتب:

```
    XA=2; YA=2; XB=8.6543678F; YB=2; XC=2; YC=8.6543678F; XD=4.3271839F; YD=2;
    plane pl=new plane();

    System.out.println("Triangle:  A, B, C");
    pl.triangularanalysis(XA,YA,XB,YB,XC,YC);

    System.out.println("Triangle:  A, C, D");
    pl.triangularanalysis(XA,YA,XD,YD,XC,YC);
```

```
System.out.println("Triangle:  A, D, B");
pl.triangularanalysis(XD,YD,XA,YA,XB,YB);

System.out.println("Triangle:  A, A, D");
pl.triangularanalysis(XD,YD,XA,YA,XA,YA);
```

حسب السطر الأول من هذا البرنامج، تتوزع هذه النقط على السطح حسب الشكل التالي:



فيسفر تنفيذ البرنامج على النتيجة التالية،

```
Triangle:  A, B, C

isoceles rectangular

Triangle:  A, C, D

plain rectangular

Triangle:  A, D, B

not a triangle

Triangle:  A, A, D

not a triangle
```

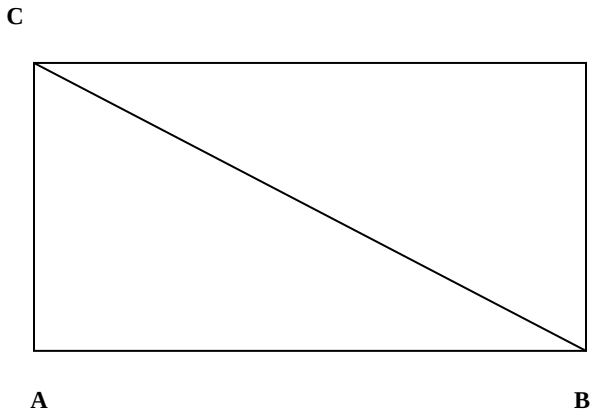
و هي فعلا مطابقة للواقع.

3.11 وضائف مزدوجة: تحليل مثلثات و تقييم مساحتهم

كتطوير للبرنامج الذي يحلل المثلثات، نهتم ببرنامج يقيم مساحة المثلث إذا ما كان المثلث مستطيلا أو ذات ثلاثة جوانب متساوية. ننظر في هذه المشكلة حسب نوع المثلث، فيما يلي:

1.3.11 المثلث المستطيل

نعرف أن مساحة مثلث مستطيل تتمثل في نصف المستطيل الذي يعيناه الجانبان المجاوران للزاوية المستقيمة، كما يبين الرسم التالي:

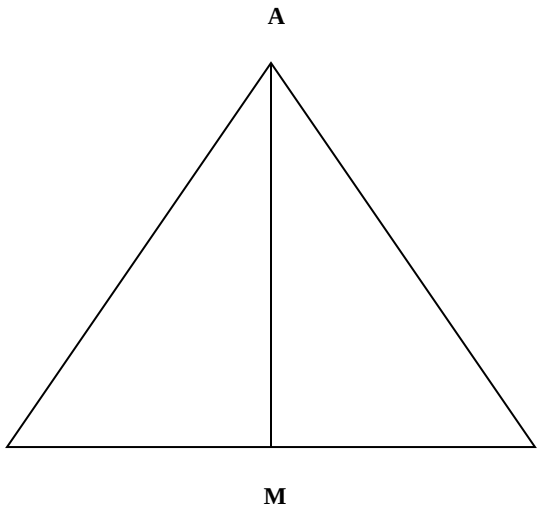


بحيث أن تطبيقيا، يكفي أن نعين وتر المثلث (في هذا الرسم، الجانب الذي يربط بين B و C) فنظرب طول الجانبين الآخرين، و نقسم على إثنين. فنكتب:

```
float rectsurface (float XA, float YA, float XB, float YB, float XC, float YC)
    {
        float dAB = dist(XA,YA,XB,YB);
        float dBC = dist(XB,YB,XC,YC);
        float dAC = dist(XA,YA,XC,YC);

        if (dAB>=dBC && dAB>=dAC)
        {// AB is hypothenuse
            return ((dBC*dAC)/2);
        }
        else if (dBC>=dAC && dBC>=dAB)
        {
            return ((dAC*dAB)/2);
        }
        else
        {
            return ((dAB*dBC)/2);
        }
    }
```

2.3.11 المثلث ذات الجوانب المتساوية

ننظر في شكل مثلث ذات جوانب متساوية، حسب الرسم التالي:



يبين هذا الرسم أن مساحة هذا المثلث تتمثل في ضرب طول القطر (AM) في نصف الجانب النظير (علما أن كل الجوانب متساوية الطول، لا فرق بينها). لكي نتمكن من تقييم المساحة، يجب أن نقيم طول القطر؛ إذا مثلنا هذا الطول برمز d و مثلنا طول جانب المثلث برمز j، فنكتب المعادلة التالية:

$$d^2 + (\frac{d}{2})^2 = j^2.$$

فنحل هذه المعادلة لإستخراج d كما يلي:

$$5d^2 = 4j^2,$$
$$d = \frac{4}{\sqrt{5}}\, j.$$

بالتالي، نجد أن مساحة هذا المثلث تساوي:

$$\frac{j}{2} \times d = \frac{j}{2} \times \frac{2}{\sqrt{5}} \times j = \frac{j^2}{\sqrt{5}} \approx 0.4472136 \times j^2.$$

فنكتب الوظيفة التالية:

```
float equisurface (float XA, float YA, float XB, float YB, float XC, float YC)
    {
        float dAB = dist(XA,YA,XB,YB);

        return ((0.4472136F*dAB*dAB));
```

3.3.11 إدماج حساب المساحة

نستعمل برنامجي المساحة لتقييم مساحة الثلث كلما تبين أنه مستطيلا أو ذات جوانب كتساوية، فنكب الوظيفة الجديدة التالية:

```
void triangularanalysis (float XA, float YA, float XB, float YB, float XC,
float YC)
    {
        if (inline (XA,YA,XB,YB,XC,YC))
        {
            System.out.println("not a triangle");
        }
        else
        {
            if (rectangular(XA,YA,XB,YB,XC,YC))

            {
                System.out.println("surface: ");
                System.out.println(rectsurface(XA,YA,XB,YB,XC,YC));

                if (isoceles(XA,YA,XB,YB,XC,YC))
                {
                System.out.println("isoceles rectangular");
                }
                else
                {
                    System.out.println("plain rectangular");
                };
            }
            else
            {
                if (equilateral(XA,YA,XB,YB,XC,YC))
                {
                    System.out.println("equilateral");
                    System.out.println("surface: ");
                    System.out.println(equisurface(XA,YA,XB,YB,XC,YC));
                }
                else
                {
                    if (isoceles(XA,YA,XB,YB,XC,YC))
                    {
                        System.out.println("non-rectangular isoceles");
                    }
                    else
                    {
                        System.out.println("plain triangle");
                    }
                }
            }
        }
    }
}
```

و يسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:

Triangle:  A, B, C

surface:

22.140303
```

```
isoceles rectangular

Triangle:  A, C, D

surface:

7.7429676

plain rectangular

Triangle:  A, D, B

not a triangle

Triangle:  A, A, D

not a triangle
```

مكنتنا وظائف جافا من نمذجة المشكلة بصفة واضحة و بسيطة، كما مكنتنا من تحليل المثلثات بصفة دقيقة و منتظمة. نشجع القارئ على إستعمال الوظائف لهيكلة البرامج، كما إجتهدنا في هذه الأمثلة.

4.11 تمارين

1 [م] ناقشنا في الجزء الثاني أن تنفيذ وضيفة تحليل المثلثات

```
void triangularanalysis
(float XA, float YA, float XB, float YB, float XC, float YC)
```

على النقط التالية

```
XA=2; YA=2; XB=8.6543678F; YB=2; XC=2; YC=7.9876678F;
```

مستعملين معيار المساوات

```
double epsilon = 0.000001;
```

يؤدي إلى نتيجة أن المثلث لا يلبي أي صفة، بالرغم من أن المثلث مستطيلا. كيف تفسرون هذا؟

2 [م] ناقشنا في الجزء الثاني أن تنفيذ وضيفة تحليل المثلثات

```
void triangularanalysis
(float XA, float YA, float XB, float YB, float XC, float YC)
```

على النقط التالية

```
XA=2; YA=2; XB=8.6543678F; YB=2; XC=2; YC=7.9876678F;
```

مستعملين معيار المساوات

```
double epsilon = 0.00001;
```

يؤدي إلى نتيجة أن المثلث مستطيلا. كيف تفسرون أن تغيير معيار المساوات يجعل البرنامج يتفطن لصفته بينما لم يتفطن بها بإستعمال معيار

```
double epsilon = 0.000001;
```

3 [م/ص] في الجزء 2.11، إستعملنا وظيفة تحليل المثلثات لنختبر مثلثات يمكن بنائها على أربعة نقط في السطح. المطلوب منكم القيام بنفس العملية بالنسبة لخمسة نقط. المطلوب منكم أيضا تمثيل النقط على رسم و التأكد عمليا أن تحليل البرنامج سديد. كم هناك من مثلث؟

4 [ص] نفس التمرين بالنسبة لستة نقط، علما أن إختيار و تعيين المثلثات يقع بصفة آلية، نظرا لعدد المثلثات. كم هناك من مثلث؟

[5] في الجزء رقم **2.3.11** بيننا كيف نقوم بتقييم مساحة مثلث ذات ثلاثة جوانب متساوية. في هذا التمرين نطلب منكم تقييم مساحة مثلث ذات جانبين متساويين. ثم المطلوب منكم كتابة وظيفة تحسب هذه المساحة. ثم المطلوب منكم إدماج هذه الوظيفة في برنامج تحليل المثلثات و تنفيذه على النقط التي درسناها في الجزء **3.3.11.**

4.11. مراجع

نظرا لأن لغة جافا لغة موجهة للكائنات، فعادة لا نجد معالجة دقيقة لموضوع الوظائف، لأن هذا الموضوع عادة منسوبا للغات الخوارزمية.

الفصل الثاني عشر

تجرد التعليمات، البرمجة الإستقرائية

نهتم في هذا الفصل بنوع خاص من الوضائف، نلقي عليه إسم الوضائف الإستقرائية. و نلقي على أسلوب البرمجة التابع لهذه الوضائف إسم البرمجة الإستقرائية. كما يدل النعت، فإن هذا الأسلوب مبني على مبدء الإستقراء المنطقي. بصفة عامة، يتمثل هذا المبدأ في أننا نقيم وضيفة أو نحلل وضعا أو نحل مشكلة في حالة معقدة بناء على حل المشكلة في حالة أبسط. إن كل الأساليب الإستقرائية مبنية ضمنيا على علاقة ترتيب ترتب العناصر المعنية من بسيط إلى معقد، أو من صغير إلى كبير، أو من قريب إلى بعيد، أو من فرعي إلى شامل، أو من خطي إلى متعدد الأبعاد، إلخ. سندرس البرمجة الإستقرائية في هذا الفصل، فنستعرض على التوالي عددا من أصناف هذا الأسلوب من البرمجة.

1.12 الإستقراء الخطي

1.1.12 وظيفة العوملة

في عدد من الحالات، يمكن تعيين وظيفة عددية (تنطبق على الأعداد الطبيعية) بواسطة إسلوب إستقرائي يتمثل في معادلة بين قيمة الوظيفة في عدد ما و قيمتها في العدد الموالي. مثلا، نعين وظيفة العوملة (التي نمثلها برمز f) كما يلي:

$$f(0) = 1,$$
$$n > 0 \Rightarrow f(n) = n \times f(n-1).$$

فطبقا لهذه المعادلة، نكتب الوظيفة التالية في لغة جافا:

```
int f (int n)
    {
        if (n==0) {return 1;}
        else
        {return n*f(n-1);}
    }
```

أما إضا طبقنا هذه الوظيفة حسب ما جاء في النص التالي،

```
int n; n=5;
        recfunction rf = new recfunction();

        System.out.println("factorial of "+n+":  "+rf.f(n));
```

فنجد النتيجة التالية.

```
factorial of 5:  120
```

إذا تأملنا في تعيين هذه الوظيفة، وجدنا أنها تنطبق على النحو التالي إذا كان عدد n يساوي 5 (مثلا):

الخطوة رقم:	نتيجة إختبار الشرط: (n==0)	الصيغة الناتجة	القيمة النتجة

1	غالط	5*f(4)	
2	غالط	5*4*f(3)	
3	غالط	5*4*3*f(2)	
4	غالط	5*4*3*2*f(1)	
5	غالط	5*4*3*2*1*f(0)	
6	صحيح	5*4*3*2*1*(1)	1
7		5*4*3*2*(1*1)	1
8		5*4*3*(2*1*1)	2
9		5*4*(3*2*1*1)	6
10		5*(4*3*2*1*1)	24
11		(5*4*3*2*1*1)	120

2.1.12 وظيفة المربع

كمثال توضيحي ثان، نختار وظيفة لا نعتبرها عادة كوظيفة إستقرائية، فنكتبها في صيغة إستقرائية، ثم ننفذها في لغة جافا بواسطة وظيفة إستقرائية. إنطلاقا من المعادلة التالية،

$$(n+1)^2 = n^2 + 2\times n + 1,$$

نستنتج التعيين التالي لوظيفة f التي تمثل مربع عدد طبيعي:

$$f(0) = 0,$$
$$n > 0 \Rightarrow f(n) = f(n-1) + 2\times n - 1.$$

فبناء على هذه المعادلة نكتب الوظيفة التالية في لغة جافا:

```
int sqr (int n)
    {
        if (n==0) {return 0;}
        else
        {return sqr(n-1)+2*n-1;}
    }
```

فإذا نفذنا هذه الوظيفة حسب النص التالي:

```
int n; n=5;
recfunction rf = new recfunction();
System.out.println("square of "+n+":  "+rf.sqr(n));
```

لوجدنا النتيجة التالية:

```
square of 5:  25
```

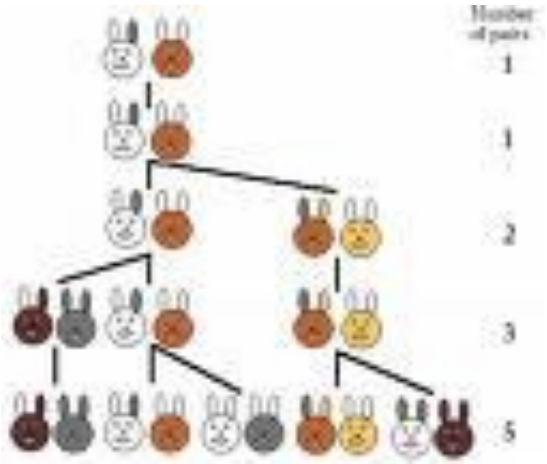
نبين فيما يلي كيف يقوم البرنامج بإيجاد هذه النتيجة:

$$
\begin{aligned}
&sqr(5)\\
&= sqr(4) + 2\times 5 - 1 = sqr(4) + 9\\
&= sqr(3) + 2\times 4 - 1 + 9 = sqr(3) + 16\\
&= sqr(2) + 2\times 3 - 1 + 16 = sqr(2) + 21\\
&= sqr(1) + 2\times 2 - 1 + 21 = sqr(1) + 24\\
&= sqr(0) + 2\times 1 - 1 + 24 = sqr(0) + 25\\
&= 0 + 25 = 25.
\end{aligned}
$$

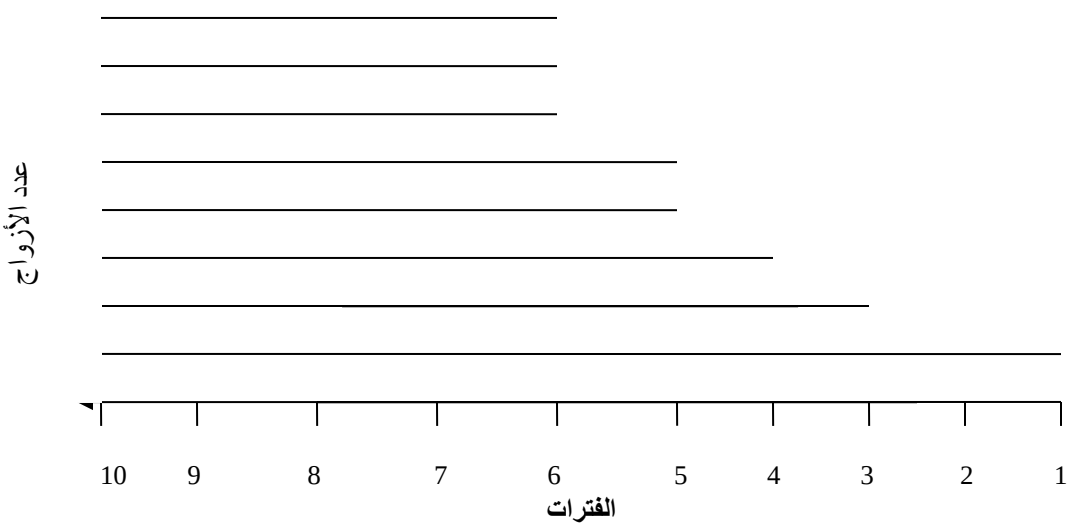
3.1.12 أعداد فيبوناشي

ولد العالم الإيطالي ليوناردو بيسانو سنة 1175 بمدينة بيسا الإطالية و عاش بالجزائر منذ صغر السن حيث كان والده يمارس التجارة فتعلم الحساب من مصادر عربية في الميدان. كان والده يسمى جيارمو بوناشي، فأطلق عليه إسم فيليو بوناشي (أي إبن بوناشي)، فإختصر إسمه ب: فيبوناشي. تعرض فيبوناشي سنة 1202 لمشكلة تتمثل في التكهن بعدد الأرانب الناتجة عن مجموعة إفتتاحية معينة، فنمذج المشكلة كما يلي:

- في الحالة الإفتتاحية، لنا أرنبان، ذكر و أنثى.
- عمر الأرنبين في الحالة الإفتتاحية صفر.
- هذه الأرانب تلد و لا تموت.
- كل زوج أرانب (ذكر و أنثى ذوي عمر بالغ) يلد مرة كل فترة (شهر، أو سنة، أو نصف سنة،...).
- تبلغ الأرانب عمر الولادة بعد مرور فترتين من ولادتهم.
- كلما يلد زوج من الأرانب، فيلد زوجا يتكون من ذكر و أنثى.



فنهتم بعدد الأرانب بعد مرور عدد معين من الفترات. مثلا، في السنة الأولى و الثانية، يبقى عدد الأرانب على حاله، أي إثنين. في الفترة الثالثة، يصبح عدد الأرانب أربعة، لأن الزوج الذي بلغ من العمر فترتين أنجب زوجا يتكون من ذكر و أنثى. في الفترة الرابعة، يصبح عدد الأرانب ستة، لأن الأرنبان البالغان ضمن الأربعة في الفترة السابقة قد أنجبا زوجا من الأرانب. في الفترة الخامسة، يساوي عدد الأرانب عددهم في الفترة الرابعة بالإضافة لعدد الأرانب البالغة في الفترة الخامسة. ضمن الستة أرانب الموجودة في الفترة الخامسة، هناك أربعة يبلغ سنها فترتين على الأقل، تنجب أربعة أرانب صغار، فنجد أن عدد الأرانب في الفترة السادسة هو 6 و 4 أي 10.



بصفة عامة، يكون عدد أزواج الأرانب في سنة ما يساوي: عددهم في السنة السابقة، نظيف له عدد الأزواج التي ولدت في تلك السنة. كم من زوج أرانب ولدت في سنة ما؟ حسب ما ناقشنا سابقا، كل زوج أرانب يكون بالغ عمر الولادة ينجب أرنبين، ذكرا و أنثى. فيصبح السؤال آنذاك: ما هو عدد أزواج الأرانب البالغة في سنة ما؟ فالإجابة على هذا السؤال هي: عدد الأرانب ذات سن إثنين أو أكثر؛ أي هو عدد أزواج الأرانب التي كانت موجودة في النسة قبل السابقة. بحيث نجد أن عدد أزواج الأرانب في سنة ما يساوي عددهم في السنة السابقة نجمعه بعددهم في السنة قبل السابقة، فنعين الوظيفة التالية:

$$\left\{\begin{array}{l} F(1)=1, \\ F(2)=1, \\ n\geq 3 \Rightarrow F(n)=F(n-1)+F(n-2). \end{array}\right\}$$

التي نشير لها بإسم وظيفة فيبوناشي، و نصرح بأن عدد الأرانب في الفترة رقم n يساوي مرتين قيمة وظيفة فيبوناشي في هذا الرقم (إذ أن وظيفة فيبوناشي تمثل عدد الأزواج).

فبناء على هذه المعادلة نكتب الوظيفة التالية في لغة جافا:

```
int fibonacci (int n)
    {
        if (n<=2) {return 1;}
        else
        {
            return (fibonacci(n-1)+fibonacci(n-2));
        }
    }
```

فإذا نفذنا هذه الوظيفة حسب النص التالي:

```
   int n; n=9;
     recfunction rf = new recfunction();
     for (int i=1; i<=n; i++)
     {
      System.out.println(i+" "+rf.fibonacci(i)+" "+2*rf.fibonacci(i));
     }
```

لوجدنا النتيجة التالية:

```
run:
```

```
1 1 2
2 1 2
3 2 4
4 3 6
5 5 10
6 8 16
7 13 26
8 21 42
9 34 68
BUILD SUCCESSFUL (total time: 0 seconds)
```

نضع هذه النتائج في الجدول التالي:

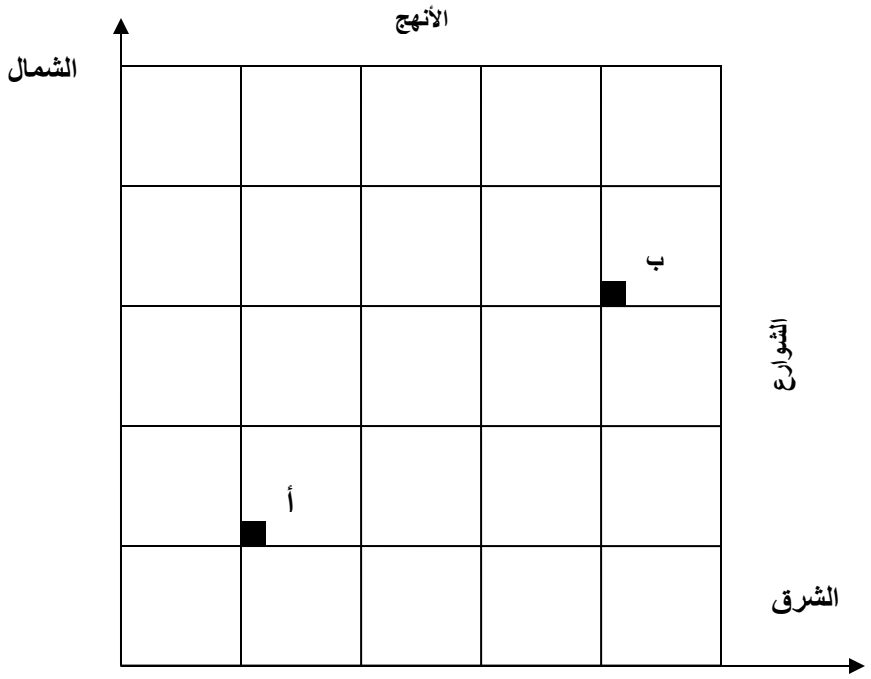
عدد الفترات	عدد الأزواج	عدد الأرانب
1	1	2
2	1	2
3	2	4
4	3	6
5	5	10
6	8	16
7	13	26
8	21	42
9	34	68

2.12 الإستقراء المتعدد الأبعاد

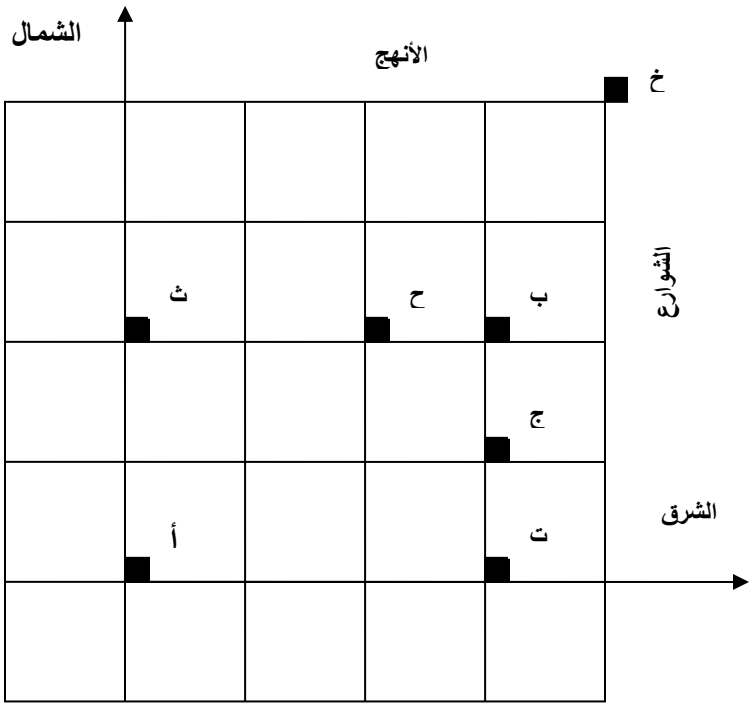
بينما درسنا في الفصل السابق الإستقراء على بعد واحد، يتمثل في عدد طبيعي، ففي هذا الجزء نهتم بدراسة الإتقراء بالنسبة لأبعاد متعددة، نبدأها بمثال ذات بعدين.

1.2.12 الإستقراء على بعدين

نعتبر مدينة تتكون من شوارع تتجه من الشرق إلى الغرب و أنهج تتجه بين الشمال و الجنوب و نعتبر نقتطين في هذه المدينة تبعد على بعضها بعضا عددا من الأنهج و عددا من الشوارع. و نهتم بإحتساب عدد المسارات بين هاتين النقطتين، علما أننا لا نتراجع في إتجاهنا: مثلا، إذا كانت نقطة ب في شمال شرق نقطة أ، فنسافر من أ إلى ب متوجهين نحو الشمال و الشرق فقط.



نحرض القارئ المهتم أن يحتسب عدد المسارات من أ إلى ب في هذا الرسم، فيقارنه بما يجد البرنامج الذي نكتبه أسفله. مبدئيا، إذا أردنا أن نحتسب عدد المسارات بين نقطتين، يجب أن نتطلع على إحداثيات النقتطين، لكن هذا غير ضروري. فرغبة في تبسيط المشكلة، نضع وسط السطح في نقطة من النقطتين؛ بالإضافة، علما أننا سنسعمل أسلوبا إستقرائيا، نوجه أبعاد السطح بحيث تكون إحداثيات النقطة الثانية إجابية.



بالتالي، تكون إحداثيات نقطة أ هي (0،0)، بينما تكون إحداثيا نقطة ب هي (3،2). هذا، و تتعين المشكلة بصفة حتمية بمجرد تعيين إحداثيات ب. أخيرا، علما أن النقتطان أ و ب تلعبان دورين متناظرين، فيمكن أن نفكر فيها كإحتساب عدد المسارات من ب إلى أ. فتعتبر الحالات التالية:

- إذا كانت نقطة ب على المحور الأفقي (ت) أو على المحور العمودي (ث)، فيكون عدد المسارات من ب إلى أ واحدا: إذ عندئذ لا يبقى لنا إلا أن نسير مع المحور المذكور حتى نصل إلى نقطة أ.

- أما إذا كانت نقطة ب بعيدة عن كلا المحورين، فيمكن الإنتقال من ب إلى أ بصفتين إثنين: إما أن تكون أول خطوة لنا في إتجاه الجنوب (نقطة ج)، أو تكون أول خطوة لنا في إتجاه الغرب (نقطة ح). فبالتالي يساوي عدد المسارات من ب إلى أ جملة عدد المسارات من ج و عدد الممسارات من ح. بناء على هذا التحليل، نكتب الوظيفة التالية:

```
int TwoD (int x, int y)
    {
        if ((x==0)||(y==0)) {return 1;}
        else // x!=0 && y!=0
        {
            {return (TwoD(x-1,y)+TwoD(x,y-1));}
        }
    }
```

إذا نفذنا هذه الوظيفة حسب البرنامج التالي:

```
recfunction rf = new recfunction();
int x, y;  x=3; y=2;
System.out.println("Number of paths from "+x+" "+y+": "+rf.TwoD(x,y));
```

لوجدنا النتيجة التاية:

```
run:
Number of paths from 3 2: 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

أي أن عدد المسارات بين نقطة ب (في الرسم أعلاه) و نقطة أ هو 10، كما يمكن للقارئ أن يتأكد. أما إذا أردنا أن نحتسب عدد المسارات بين نقطة أ و نقطة خ (الموجودة في أقصى الرسم) فننفذ وظيفة المسارات حسب البرنامج التالي (حيث تمثل (4،4) إحداثيات النقطة خ).

```
recfunction rf = new recfunction();
int x, y;  x=4; y=4;
System.out.println("Number of paths from "+x+" "+y+": "+rf.TwoD(x,y));
```

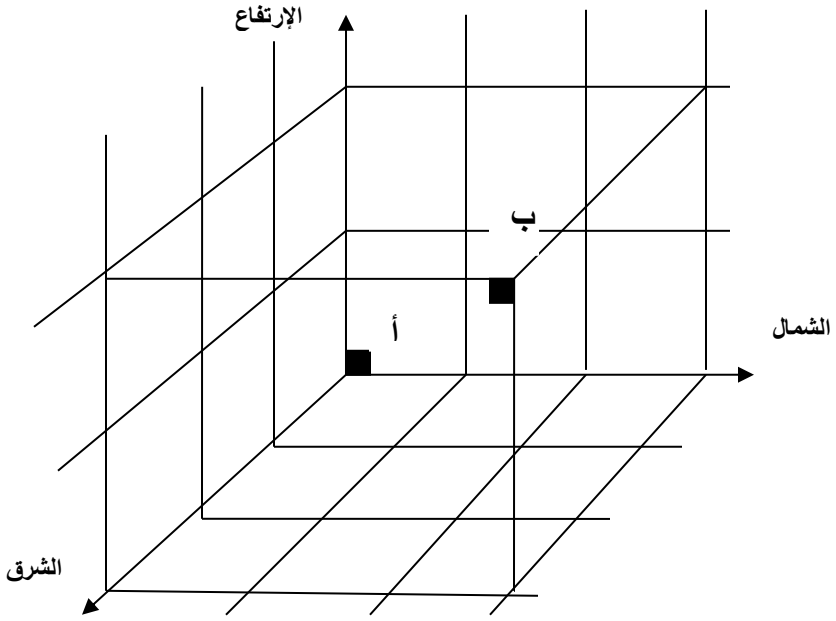
فنجد النتيجة التالية:

```
run:
Number of paths from 4 4: 70
```

أي أن عدد المسارات بين نقطة خ و نقطة أ في السطح يساوي 70.

2.2.12 الإستقراء على ثلاثة أبعاد

أما الآن فنهتم بإحتساب عدد المسارات، لا في سطح، بل في فضاء ذات ثلاثة أبعاد، حسب الرسم التالي، بين نقطة ب و نقطة أ. كما فعلنا سابقا، نتخذ أ كمركز للفضاء و نوجه محاور الفضاء بحيث تكون إحداثيات ب كلها لاسلبية.



نهتم بوظيفة عدد المسارات في الفضاء ذات ثلاثة أبعاد و نجد أن هذه الوظيفة تلبي المعادلات التالية:

- إذا كانت نقطة ب موجودة على سطح من الثلاثة أسطح المعينة بالمحاور الثلاثة، فيكون عدد المسارات بين ب و أ يساوي ما تجده وظيفة البعدين، كما درسناها سابقا، فنكتب:

$$\left\{\begin{array}{l} x = 0 \Rightarrow ThreeD(x, y, z) = TwoD(y, z), \\ y = 0 \Rightarrow ThreeD(x, y, z) = TwoD(x, z), \\ z = 0 \Rightarrow ThreeD(x, y, z) = TwoD(x, y). \end{array}\right\}$$

- إذا لم تكن نقطة ب موجودة على أي سطح من الثلاثة أسطح المعينة بالمحاور الثلاثة، فهناك ثلاثة إمكانيات للسفر من أ إلى ب: إما بالإتجاه خطوة للغرب، أو بالإتجاه خطوة للجنوب، أو بالنزول درجة من الإرتفاع الحالي. فيكون عدد المسارات من ب إلى أ جملة عدد المسارات بين الثلاثة نقط المجاورة لنقطة ب و نقطة أ. بالتالي، نكتب ما يلي:

$$x > 0 \wedge y > 0 \wedge z > 0 \Rightarrow$$
$$ThreeD(x, y, z) = ThreeD(x-1, y, z) + ThreeD(x, y-1, z) + ThreeD(x, y, z-1).$$

بناء على هذا التحليل، نكتب الوظيفة التالية في لغة جافا:

```
int ThreeD (int x, int y, int z)
    {
        if (x==0) {return (TwoD(y,z));}
        else if (y==0) {return (TwoD(x,z));}
        else if (z==0) {return (TwoD(x,y));}
        else
        {
            return (ThreeD(x-1,y,z)+ThreeD(x,y-1,z)+ThreeD(x,y,z-1));
        }
    }
```

و إذا نفذنا هذه الوظيفة حسب البرنامج التالي (حيث أن (2،3،3) تمثل إحداثيات نقطة ب):

```
int x, y, z;  x=3; y=3; z=2;
        System.out.println("Number of paths from "+x+" "+y+" "+z+": "+rf.ThreeD(x,y,z));
```

لوجدنا النتيجة التالية، و هي تمثل عدد المسارات بين ب و أ في الفضاء ذات ثلاثة أبعاد:

```
run:
Number of paths from 3 3 2: 560
```

أما إذا وجهنا إهتمامنا لنقطة توجد على بعد 4 شمالا و 4 شرقا و على إرتفاع 4، فنجد النتيجة التالية:

```
run:
Number of paths from 4 4 4: 34650
```

أي أن عدد المسارات يتجاوز الأربعة و ثلاثين ألف.

3.12 الإستقراء على هياكل البيانات

بينما نفكر عادة في الإستقراء كأسلوب ينطبق على الأعداد الطبيعية، فهم في الواقع ينطبق على نوع أوسع من البيانات، خاصة البيانات الخطية. ننظر فيما يلي في بعض التطبيقات للبرمجة الإستقرائية، كما نمارسها بالنسبة لهياكل بيانات.

1.3.12 عمليات على الجداول الخطية

نعتبر جدولا ذات بعد واحد، حجمه N، و نهتم مثلا بإحتساب جملة عناصره. لهذا الغرض، نعين وظيفة عامة تحتسب جملة عناصر الجدول بين العنصر الأول و العنصر رقم i، و نحلل ميزاتها:

$$\left\{\begin{array}{l}\sum_{k=0}^{-1} a[k] = 0, \\ i \geq 0 \Rightarrow \sum_{k=0}^{i} a[k] = \sum_{k=0}^{i-1} a[k] + a[i].\end{array}\right\}$$

فنستنتج من هذه المعادلة الوظيفة الإستقرائية التالية في لغة جافا:

```
float inductivearraysum (float [] a, int i)
    {
```

```
    if (i<0) {return 0;}
    else {return inductivearraysum(a,i-1)+a[i];}
}
```

نستعمل هذه الوظيفة لنعين وظيفة تحتسب جملة جدول خطي حجمه N، كما يلي:

$$arraysum(a) = inductivearraysum(a, N-1).$$

رغبة في تنفيذ هذه الوظيفة، نكتب البنامج التالي، الذي يضع قيمات معينة في الجدول ثم يجمعها.

```
int N;  N=7000;
float [] a = new float [N];
for (int i=0; i<N; i++)
{
    if ((i%2)==0) {a[i]=(float)4.0/(2*i+1);}
    else          {a[i]=(float)-4.0/(2*i+1);}
}
System.out.println("array sum:  "+rf.inductivearraysum(a,N-1));
```

تفسيرا للصيغات الموجودة، نذكر المعادلة التالية

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + ... + \frac{(-1)^i}{2i+1} + ...$$

فنجد أن البرنامج يضع قيمات هذا التسلسل في المواقع المتتالية للجدول، مع ضربها ب4. بالتالي، ننتضر أن تنفيذ هذا البرنامج يؤدي إلى كتابة قيمة تقارب بي. فيسفر تنفيذه في الواقع على النتيجة التالية:

```
run:
array sum:  3.1414566
```

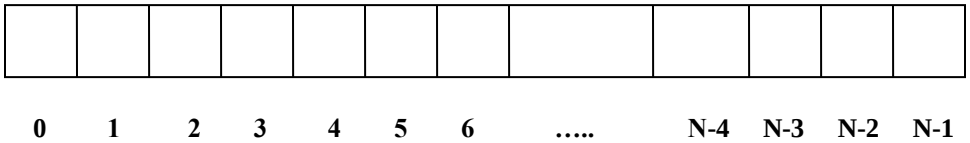
كما نرى، فإن هذه السلسلة لا تتطور نحو قيمة بي إلا بسرعة بطيئة جدا. إذ أن بعد إضافة سبعة آلاف طرفا نجد قيمة تطابق قيمة بي إلى غاية ثلاثة أعداد بعد الفاصل (3.141). يقدم التمرين الخامس حلا بديلا يمكننا من تقييم بي بواسطة تسلسل يتطور نحو بي بأكثر سرعة.

نواصل دراسة برامج إستقرائية تعالج هياكل بينات خطية، فننظر في وظيفة تحتسب أكبر عنصر في جدول خطي. نعتبر أن لدينا وظيفة تحتسب أكبر عنصر من عنصرين إثنتن، فنكتب عنوانها كما يلي:

```
float max2 (float x, float y).
```

و نعتزم تعيين وظيفة تحتسب أكبر عنصر في جدول خطي. كما فعلنا في المثال السابق، نعين وظيفة إستقرائية تحتسب أكبر عنصر في الجدول بين بداية الجدول (موقع 0) و مشير i (حيث أن i يمثل المشير الجاري في الجدول)، نكتب عنوانها كما يلي:

```
float inductivemax(float [] a, int i)
```



عكسا لما فعلنا في المثال السابق، نبدأ بتحليل الحالة العامة، حيث يكون حجم الجدول دون الصفر. فنكتب المعادلة التالية:

$$inductive\max(a,i) = \max 2(a[i], inductive\max(a,i-1)).$$

أي إذا كانا نعرف أكبر عنصر بين بداية الجدول (موقع 0) و العنصر رقم (i-1) فنستنتج أكبر عنصر بين بداية الجدول و موقع N بإحتساب أكبر العنصرين التاليين: العنصر الموجود في عنوان i و أكبر عنصر بين موقع 0 و (i-1). أما عن قاعدة الإستقراء، فنكتبها كما يلي:

$$i < 0 \Rightarrow inductive\max(a,i) = -\infty.$$

ليقتنع القارئ من سداد هذه المعادلة، نحرضه على تطبيق خطوة الإستقراء لجدول يتكون فقط من العنصر الأول (في عنوان 0)، فنجد

$$\begin{aligned} & inductive\max(a,0) \\ & = \max 2(a[0], inductive\max(a,-1)) \\ & = \max 2(a[0], -\infty) \\ & = a[0]. \end{aligned}$$

فعلا، إذا كان الجدول يختصر في عنصر واحد، فيكون هذا العنصر أكبر عنصر للجدول. نستعمل الوظيفة الإستقرائية

$$inductivemax(a,i)$$

بناء على هذا التحليل، نؤلف الوظائف التالية في لغة جافا:

```
float mininfinity= -3.4E+38F;

float max2 (float x, float y)
{
    if (x>y) {return x;}
    else {return y;}
}

float inductivemax (float [] a, int i)
{
    if (i<0) {return mininfinity;}
    else {return max2(a[i],inductivemax(a,i-1));}
}
```

لنعين الوظيفة التي تحتسب أكبر عنصر في جدول حجمه N حسب المعادلة التالية:

$$max(a)=inductivemax(a,N-1).$$

فنكتب البرنامج التالي، الذي يحتسب و يطبع أكبر عنصر في الجدول:

```
    recfunction rf = new recfunction();
    int N=700;
    float [] a = new float [N];
    for (int i=0; i<N; i++)
    {
        if ((i%2)==0) {a[i]=(float)4.0/(2*i+1);}
```

```
        else          {a[i]=(float)-4.0/(2*i+1);}
    }
    System.out.println("array max:  "+rf.inductivemax(a,N-1));
```

فيسفر تنفيذ هذا البرنامج على النتيجة التالية:

```
run:
array max:  4.0
```

فعلا، فإن أكبر عنصر في الجدول هو العنصر الأول، الموجود في عنوان 0.

نهتم الآن بمثال ثالث يتمثل في وظيفة تحلل جدولا خطيا لتختبر إن كان مرتبا أم لا. فعكسا للوظائف التي درسناها إلى حد الآن، فإن هذه الوظيفة لا تحتسب قيمة عددية، بل تحتسب قيمة منطقية. لهذا الغرض، نقدم وظيفة إستقرائية تختبر إن كان الجدول مرتبا بين عنوان 0 (بداية الجدول) و عنوان عشوائي i بين 0 و (N-1)، فنلقي عليها الإسم التالي:

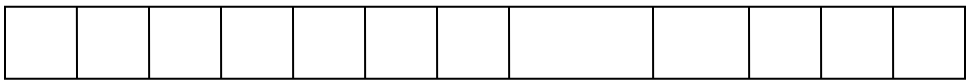
```
boolean inductivesort (float [] a, int i)
```

فنكتب المعادلة الإستقرائية التالية:

$$i < 1 \Rightarrow inductivesort(a,i) \equiv true,$$
$$i \geq 1 \Rightarrow inductivesort(a,i) \equiv a[i-1] \leq a[i] \wedge inductivesort(a,i-1).$$

نؤول هذه المعادلة كما يلي:

- قاعدة الإستقراء: إذا كان الجدول فارغا، أو كان يتكون من عنصر واحد، فنعتبر أنه مرتبا.
- درجة الإستقراء: إذا كان الجدول يتكون من عنصرين على الأقل، فيكون مرتبا إلى عنوان i إذا و فقط إذا كان مرتبا إلى عنوان (i-1) و كان العنصر الموجود في عنوان (i-1) لا يتجاوز العنصر الموجود في عنوان i.



0 1 2 3 4 5 6 ….. N-4 N-3 N-2 N-1

بناء على هذا التحليل، نكتب الوظيفة التالية في لغة جافا:

```
boolean inductivesort (float [] a, int i)
{
   if (i<1) {return true;}
   else {return ((a[i-1]<=a[i]) && inductivesort(a,i-1));}
}
```

فنكتب البرنامج التالي، الذي يختبر إن كان الجدول مرتبا،

```
recfunction rf = new recfunction();
int N=70;
float [] a = new float [N];
for (int i=0; i<N; i++)
{
    if ((i%2)==0) {a[i]=(float)4.0/(2*i+1);}
    else          {a[i]=(float)-4.0/(2*i+1);}
```

```
        }
        System.out.println("array sorted:  "+rf.inductivesort(a,N-1));
```

فيسفر هذا البرنامج على النتيجة التالية:

```
run:
array sorted:  false
```

أما إذا غيرنا الجدول حيث نضع فيه تسلسلا مرتبا،

```
        recfunction rf = new recfunction();
        int N=70;
        float [] a = new float [N];
        for (int i=0; i<N; i++)
        {
            a[i]=2*i;
        }
        System.out.println("array sorted:  "+rf.inductivesort(a,N-1));
```

فتكون النتيجة كما يلي،

```
run:
array sorted:  true
```

و هو ما ننتظر في هذه الحالة.

1.3.12 عمليات على الأشجار

نعتبر أن القارئ متطلع على أهم مفاهيم الأشجار، خاصة منها الأشجار الثنائية، و خاصة من هذه الأشجار الثنائية القانونية. بحيث لا نناقش هذه المفاهيم من جديد هنا، بل نكتفي بإستعمالها مباشرة، لنبين مفاهيم البرمجة الإستقرائية على هذا النوع من الأشجار. نذكر بإيجاز أن الأشجار الثنائية القانونية هي أشجار تكزن درجتها تساوي صفرا (إذا كانت أوراقا) أم إثنين (إذا كانت عقودا داخلية). على باب المثال، نقدم شجرة في الرسم رقم 2.12، تمثل التعبير العددي التالي:

$$(((5-3)+9)*(8/4)).$$

نكون هذه الشجرة في لغة جافا بواسطة البرنامج التالي:

```
void BinaryTree ()
    {
      Node r111 = new Node('5');
      r111.left = null;  r111.right = null;
      Node r112 = new Node ('3');
      r112.left = null; r112.right = null;
      Node r11 = new Node ('-');
      r11.left = r111;  r11.right = r112;

      Node r12 = new Node ('9');
      r12.left = null; r12.right = null;

      Node r1 = new Node ('+');
      r1.left = r11; r1.right = r12;

      Node r21 = new Node ('8');
      r21.left = null; r21.right = null;

      Node r22 = new Node ('4');
      r22.left = null; r22.right = null;

      Node r2 = new Node ('/');
      r2.left = r21; r2.right = r22;
```

```
  root = new Node('*');
  root.left = r1;  root.right = r2;

}
```

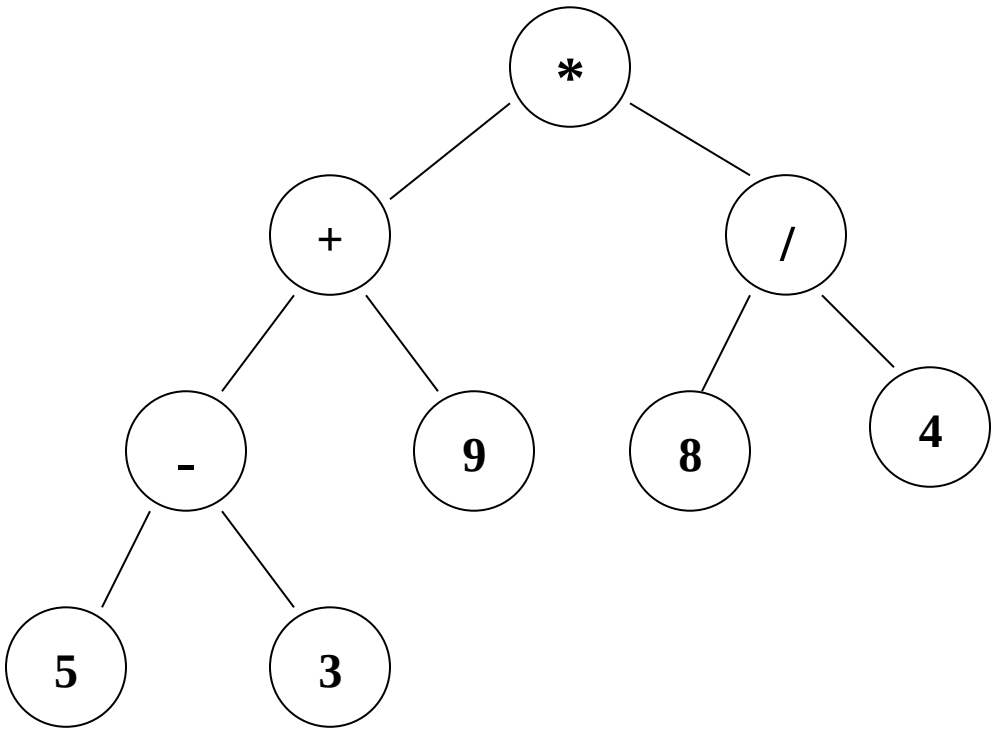
نعتزم أولا كتابة وظيفة إستقرائية تحتسب عدد أوراق الشجرة (و هو عدد عوامل التعبير العددي). فإذا تأملنا في شجرة ثنائية قانونية، لوجدنا المعادلات التالية:

قاعدة الإستقراء: إذا كانت الشجرة تتكون من مجرد ورقة، فيكون عدد الأوراق فيها 1.

درجة الإستقراء: إذا لم تكن ورقة، فتكون لها حتما شجرتان جزئيتان (لأن الشجرة ثنائية قانونية)، فنجد أن عدد الأوراق في الشجرة الأم يمثل جملة الأوراق في الشجرتين (حسب الرسم رقم 3.12).

فنكتب الوظيفة الإستقرائية التالية:

```
int nbleaves (Node n)
{
    if (leaf(n)) {return 1;}
    else {return (nbleaves(n.left)+nbleaves(n.right));}
}
```



الرسم رقم 2.12: شجرة تعبير عددي

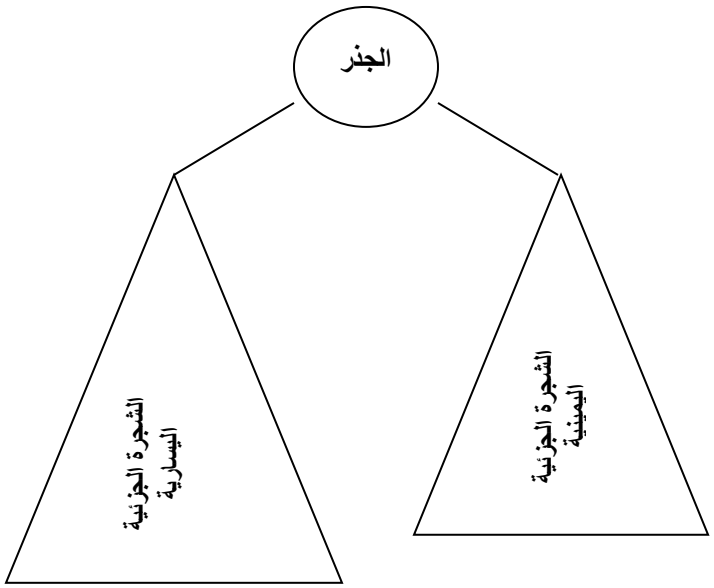
حيث نمثل الوظيفة المنطقية التي تتفقد إن عقدة ورقة أم لا، كما يلي:

```
boolean leaf (Node n)
```

```
{
     return ((n.left == null) && (n.right == null));
}
```

و نستعمل الوظيفة الإستقرائية لإحتساب عدد أوراق الشجرة كاملة كما يلي:

```
int nbleaves ()
{
    return nbleaves(root);
}
```



الرسم رقم 3.12: الإستقراء على الأشجار الثنائية القانونية

إذا نعتزم الآن كتابة وظيفة إستقرائية تحتسب عدد عقود الشجرة سواء كانت أوراقا أم عقودا داخلية (و هو عدد عوامل و عمليات التعبير العددي). فإذا تأملنا في شجرة ثنائية قانونية (الرسم رقم 3.12)، لوجدنا المعادلات التالية:

قاعدة الإستقراء: إذا كانت الشجرة تتكون من مجرد ورقة، فيكون عدد العقود فيها 1.

درجة الإستقراء: إذا لم تكن ورقة، فتكون لها حتما شجرتان جزئيتان (لأن الشجرة ثنائية قانونية)، فنجد أن عدد العقود في الشجرة الأم يمثل جملة العقود في الشجرتين، نظيف لها 1 (يمثل الجذر)؛ حسب الرسم رقم 3.12.

فنكتب الوظيفة الإستقرائية التالية:

```
int nbnodes (Node n)
{
    if (leaf(n)) {return 1;}
    else {return 1+nbnodes(n.left)+nbnodes(n.right);}
}
```

و نستعملها لإحتساب عدد أوراق الشجرة كاملة كما يلي:

```
int nbnodes ()
    {
        return nbnodes(root);
    }
```

ختاما، إذا نعتزم كتابة وظيفة إستقرائية تحتسب إرتفاع الشجرة (و هو يمثل عمق التعبير العددي). فإذا تأملنا في شجرة ثنائية قانونية (الرسم رقم 3.12)، لوجدنا المعادلات التالية:

قاعدة الإستقراء: إذا كانت الشجرة تتكون من مجرد ورقة، فيكون إرتفاعها صفرا.

درجة الإستقراء: إذا لم تكن ورقة، فتكون لها حتما شجرتان جزئيتان (لأن الشجرة ثنائية قانونية)، فنجد أن إرتفاع الشجرة الأم يمثل 1، و إرتفاع أرفع الشجرتين الجزئيتين، حسب الرسم رقم 3.12.

فنكتب الوظيفة الإستقرائية التالية:

```
int depth (Node n)
{
    if (leaf(n)) {return 0;}
    else {return 1+ max (depth(n.left),depth(n.right));}
}
```

حيث نعين الوظيفة التي تحتسب أكبر عددين كما يلي:

```
int max (int x, int y)
{
    if (x>y) {return x;}
    else {return y;}
}
```

و نستعمل الوظيفة الإستقرائية لإحتساب إرتفاع الشجرة كاملة كما يلي:

```
int depth ()
{
    return depth(root);
}
```

نختبر هذه الوظائف بتنفيذها على الشجرة المرسومة في الرسم رقم 2.12، حسب النص التالي،

```
BinaryTree bintree = new BinaryTree ();
bintree.BinaryTree();

System.out.println("number of nodes:  "+bintree.nbnodes());
System.out.println("number of leaves:  "+bintree.nbleaves());
System.out.println("tree depth:  "+bintree.depth());
```

فيسفر تنفيذ هذا البرنامج على النتائج التالية،

```
run:
number of nodes:  9
number of leaves:  5
tree depth:  3
```

و هي كلها صحيحة.

4.12 الإستقراء كأسلوب حل مشاكل

في جميع الأمثلة التي نظرنا فيها إلى الآن، إستعملنا الإستقراء كوسيلة لإحتساب وظيفة معقدة. في هذا الجزء نستعمل الإستقراء كوسيلة لجدولة عمليات. لهذا الغرض، نعتبر مشكلة أبراج هانوي، التي تُطرح كما يلي:

أبراج هانوي:

1. لدينا ثلاثة أبراج (محاور عمودية)، برج الإنطلاق، برج المرمى، و برج العبور (حسب الرسم 1.12).
2. لنا عدد من الأقراص مختلفة الحجم توضع على هذه الأبراج.
3. في الحالة الإفتتاحية توجد كل الأقراص على برج الإنطلاق؛ المطلوب منا تحويل هذه الأقراص إلى برج المرمى مع مراعاة القواعد التالية:
   1.3 نحول الأقراص واحدا بعد الواحد.
   2.3 لا نحول الأقراص إلا من برج إلى آخر (لا نخزنها في مكان آخر).
   3.3 لا نضع قرصا إلا على أقراص أكبر منه.

قام بتعيين هذه المشكلة العالم الفرنسي إدوار لوكاس سنة 1883 ميلادي، بناء (حسب ما يضن المؤرخون) على أساطير هندية قديمة. يبين الرسم رقم 1.12 شكل هذه الأبراج و الأقراص في الحالة الإفتتاحية، و الحالة الختامية و حالة وسطى. في الحالة الوسطى مثلا، نكون حولنا أكبر قرص من برج الإنطلاق إلى برج المرمى و وضعنا بقية الأقراص في برج العبور. نلاحظ أن هذا الوضع يشبه الوضع الإفتتاحي، في أننا نجد مجموعة من الأقراص المتراكمة على بعضها، يجب تحويلها من برج العبور إلى برج المرمى، مستعملين برج الإنطلاق كبرج عبور. فالفرق بين الحالة الإفتتاحية و الحالة الجارية / الوسطى هو أن عدد الأقراص التي يجب تحويلها لبرج المرمى في الحالة الجارية أقل من عدد الأقراص التي يجب تحويلها لبرج المرمى في الحالة الإفتتاحية. بناء على هذا النقاش، نقدم الوظيفة التالية

$$TowersOfHanoi(From, To, Through, n),$$

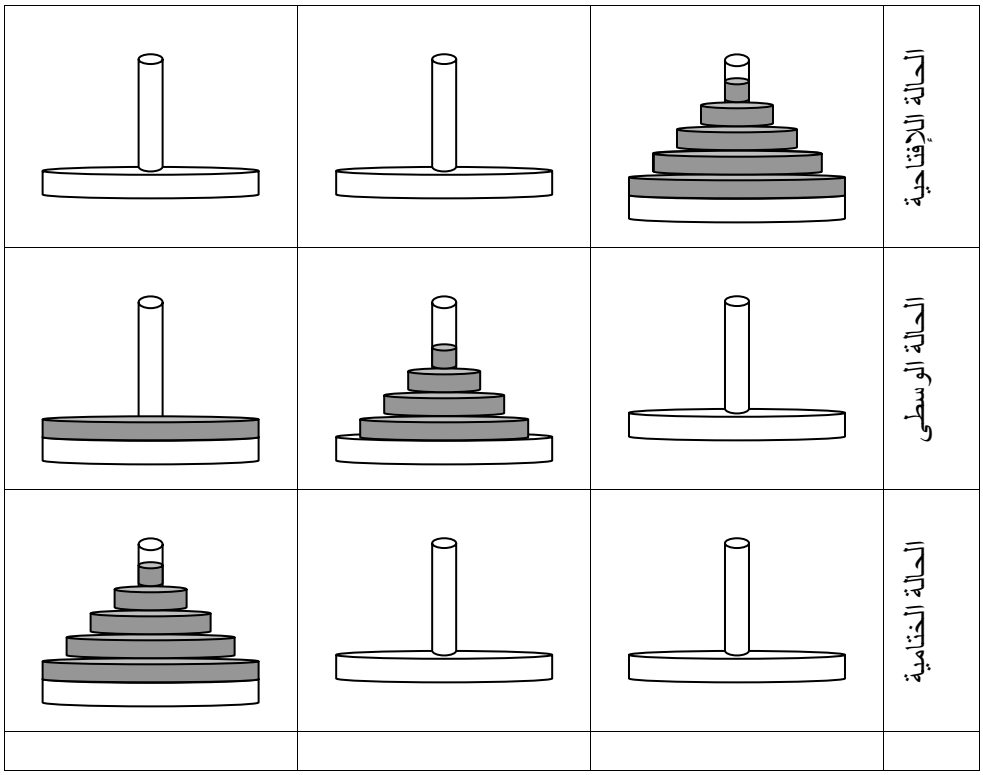
حيث تمثل الرموز

$$From, To, Through$$

أبراج الإنطلاق و المرمى و العبور، و يمثل رمز n عدد الأقراص التي نعتزم تحويلها. نستعمل هذه الوظيفة لنكتب المعادلة التالية:

$$\begin{aligned} &TowersOfHanoi(From, To, Through, n) \equiv \\ &TowersOfHanoi(From, Through, To, n-1); \\ &Move(From, To, n); \\ &TowersOfHanoi(Through, To, From, n-1). \end{aligned}$$

	برج الإنطلاق	برج العبور	برج المرمى



الرسم رقم 1.12: أبراج هانوي

نؤول هذه المعادلة كما يلي: إذا أردنا أن نحول عددا n من الأقراص من برج الإنطلاق إلى برج المرمى مستعملين برج العبور، فيمكن تحويل الأقراص العليا (بعدد (n-1)) من برج الإنطلاق إلى برج العبور، مستعملين برج المرمى؛ ثم تحويل أسفل (و أكبر) قرص من برج الإنطلاق إلى برج المرمى؛ ثم تحويل مجموعة الأقراص الموجودة في برج العبور إلى برج المرمى مستعملين برج الإنطلاق.
قد يتسائل القارئ: ما هي قاعدة الإستقراء؟ نجيب هذا السؤال بالملاحظة أن إذا كان عدد الأقراص صفرا، فلا نغير شيئا. بالتالي، حسب معادلة درجة الإستقراء، إذا كان عدد الأقراص يساوي 1، فتختصر الوظيفة على الشكل التالي:

$$\begin{aligned}&TowersOfHanoi(From,To,Through,1) \equiv\\ &TowersOfHanoi(From,Through,To,0);\\ &Move(From,To,1);\\ &TowersOfHanoi(Through,To,From,0) =\\ &Move(From,To,1).\end{aligned}$$

أي تختصر على تحويل القرص الوحيد من برج الإنطلاق إلى برج المرمى. بناء على هذا التحليل، نكتب الوظيفة التالية في لغة جافا:

```
void move (char from, char to)
{
    System.out.println("move top disk from "+from+" to "+to);
}

void towersofhanoi (char from, char to, char through, int n)
{
```

```
        if (n>0)
        {
            towersofhanoi(from,through,to,n-1);
            move(from,to);
            towersofhanoi(through,to,from,n-1);
        }
    }
```

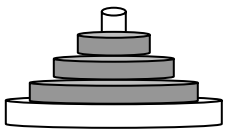
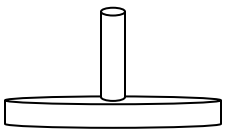
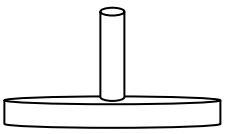
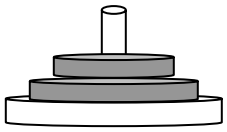
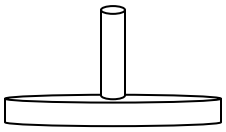
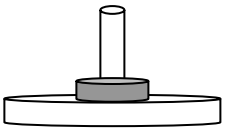
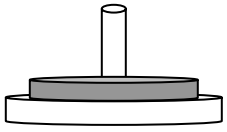
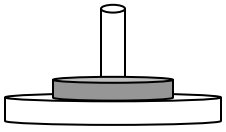
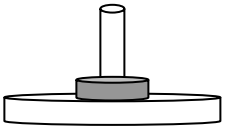
ننفذ هذه الوظيفة بواسطة البرنامج التالي (حيث ننطلق من A إلى C مستعملين B للعبور):

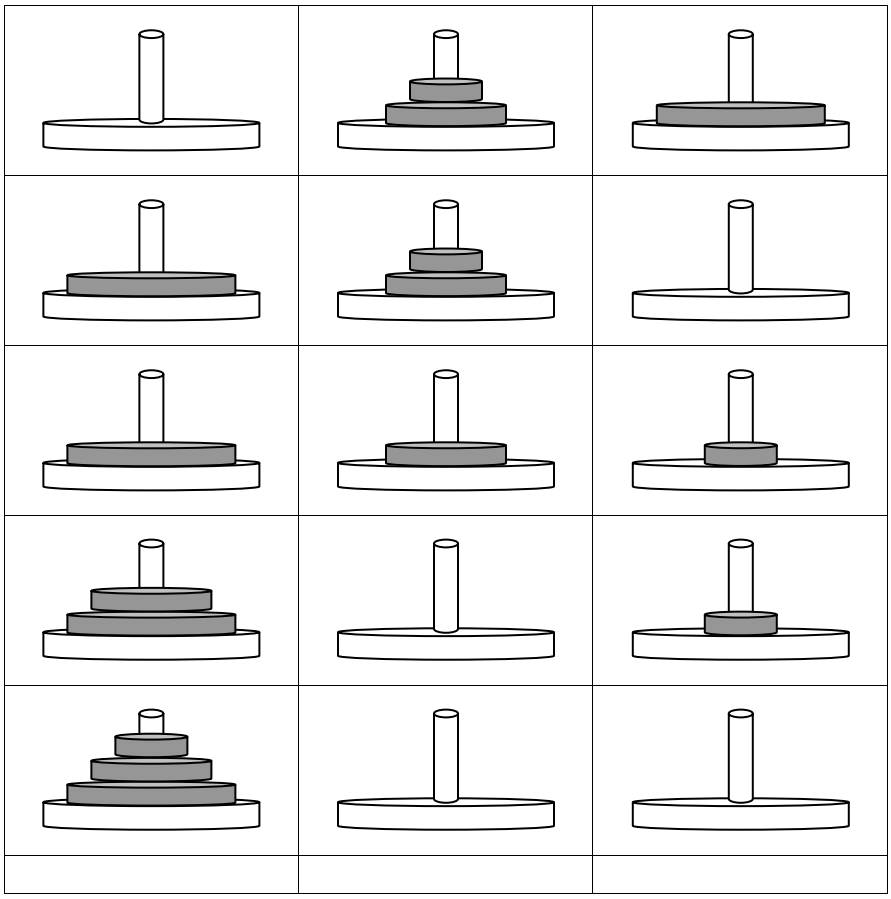
```
    recfunction rf = new recfunction();
    rf.towersofhanoi('A','C','B',3);
```

فيسفر التنفيذ على النتيجة التالية:

```
run:
move top disk from A to C
move top disk from A to B
move top disk from C to B
move top disk from A to C
move top disk from B to A
move top disk from B to C
move top disk from A to C
```

نبين هذا التنفيذ كما يلي:

برج الإنطلاق	برج العبور	برج المرمى
A	B	C
		
		
		



يمكن تغيير عدد الأقراص بقدر ما نشاء، فينص هذا البرنامج البسيط على كيفية تحويلها. لمن شاء أن يجرب، نكتب في ما يلي تسلسل العمليات بالنسبة لخمسة أقراص.

```
move top disk from A to C
move top disk from A to B
move top disk from C to B
move top disk from A to C
move top disk from B to A
move top disk from B to C
move top disk from A to C
move top disk from A to B
move top disk from C to B
move top disk from C to A
move top disk from B to A
move top disk from C to B
move top disk from A to C
move top disk from A to B
move top disk from C to B
move top disk from A to C
move top disk from B to A
move top disk from B to C
move top disk from A to C
move top disk from B to A
move top disk from C to B
move top disk from A to C
move top disk from B to A
move top disk from B to C
move top disk from A to C
```

5.12 تمارين

1 [س] نعتبر أننا عيننا وظيفة العوملة (التي نمثلها برمز g) كما يلي:

$$g(1)=1,$$
$$n>1 \Rightarrow g(n)=n\times g(n-1).$$

فما هو الفرق بين وظيفة العوملة f كما عيّناها في الجزء 1.12 و وظيفة g كما نعينها هنا؟

2 [م] نعتبر المعادلة التالية حيث يمثل رمز n عددا طبيعيا:

$$(n+1)^3=n^3+3n^2+3n+1.$$

المطلوب منكم متابعة الأسلوب الذي إتخذناه في الجزء 1.12، لكتابة برنامج إستقرائي في لغة جافا يحتسب وظيفة المقدرة الثالثة لعدد طبيعي ما.

3 [م] إعتبر فيبوناشي أن الأرانب لا تموت أبدا، و هذا ما يجعل أعداد فيبوناشي تزداد بالكيفية التي يصفها. المطلوب منكم أن تغيروا المعادلات التي تصف تطور عدد الأرانب علما أنها تعيش عشرة فترات فقط. كما نطلب منكم كتابة وظيفة في جافا تحصي عدد الأرانب حسب عدد الفترات منذ إنطلاق عملية التربية.

4 [ص] إعتبر فيبوناشي أن الأرانب لا تموت أبدا، و هذا ما يجعل أعداد فيبوناشي تزداد بالكيفية التي يصفها. المطلوب منكم أن تغيروا المعادلات التي تصف تطور عدد الأرانب علما أنها تعيش عشرة فترات فقط و أنها لا تنجب بعد سن الثمانية فترات. كما نطلب منكم كتابة وظيفة في جافا تحصي عدد الأرانب عسب عدد الفترات منذ إنطلاق عملية التربية.

5 [م] نعتبر المعادلة التالية، التي توفر تسلسلا يتطور نحو قيمة بي بسرعة أكبر مما رأينا في الجزء 1.3.12.

$$\frac{\pi}{2}=\frac{2\times 2}{1\times 3}\times\frac{4\times 4}{3\times 5}\times\frac{6\times 6}{5\times 7}\times\frac{8\times 8}{7\times 9}\ldots$$

المطلوب منكم تصميم و تأليف وظيفة إستقرائية في جافا تحتسب ضرب قيمات جدول خطي. ثم المطلوب منكم إستعمال هذه الوضيفة لإحتساب قيمة تقريبية لعدد بي.

6 [س] المطلوب منكم تطبيق المنهج الذي قدمناه في الجزء 1.3.12 لتأليف وظيفة في جافا تحتسب أصغر عنصر في جدول ما. المطلوب تطبيق هذه الوظيفة على أمثلة تطبيقية.

7 [م] المطلوب منكم تغيير الوظيفة الإسقرائية التي قدمناها لإحتساب جملة عناصر جدول لتكون قاعدة الإستقراء في نهاية الجدول عوضا عن بدايته.

8 [م] المطلوب منكم تغيير الوظيفة الإسقرائية التي قدمناها لإحتساب أكبر عنصر في جدول لتكون قاعدة الإستقراء في نهاية الجدول عوضا عن بدايته.

9 [م] المطلوب منكم تغيير الوظيفة الإسقرائية التي قدمناها لإختبار إن كان الجدول مرتبا لتكون قاعدة الإستقراء في نهاية الجدول عوضا عن بدايته.

10 [س] المطلوب منكم تغيير الوظيفة الإسقرائية التي قدمناها لإختبار إن كان الجدول مرتبا بالتتيب النازل عوضا عن الترتيب الصاعد.

11 [م] المطلوب منكم أن تجربوا حل مشكلة أبراج هانوي بخمسة أقراص. ثم نطلب منكم تنفيذ البرنامج الذي درسناه في الفصل 4.12 بخمسة أقراص، و المقارنة بين حلكم و حل البرنامج.

12 [م] المطلوب منكم كتابة وظيفة إستقرائية تحتسب عدد العقد في شجرة ثنائية (لا تكون بالضرورة قانونية)؛ ثم نطلب منكم تنفيذ هذه الوظيفة على شجرة ثنائية غير قانونية.

13 [م] المطلوب منكم كتابة وظيفة إستقرائية تحتسب عدد الأوراق في شجرة ثنائية (لا تكون بالضرورة قانونية)؛ ثم نطلب منكم تنفيذ هذه الوظيفة على شجرة ثنائية غير قانونية.

14 [م] المطلوب منكم كتابة وظيفة إستقرائية تحتسب عمق شجرة ثنائية (لا تكون بالضرورة قانونية)؛ ثم نطلب منكم تنفيذ هذه الوظيفة على شجرة ثنائية غير قانونية.

15 [م] المطلوب منكم كتابة وظيفة إستقرائية تحتسب عدد السهام في شجرة ثنائية قانونية؛ ثم نطلب منكم تنفيذ هذه الوظيفة على الشجرة التي إستعملناها في الجزء 2.3.12.

16 [م] المطلوب منكم كتابة وظيفة إستقرائية تحتسب عدد السهام في شجرة ثنائية (لا تكون بالضرورة قانونية)؛ ثم نطلب منكم تنفيذ هذه الوظيفة على شجرة ثنائية غير قانونية.

17 [م] المطلوب منكم كتابة وظيفة إستقرائية تحتسب قيمة التعبير العددي الذي تمثله الشجرة المرسومة في الجزء 2.3.12.

6.12 مصادر و مراجع

يناقش كتاب [الميلي و العابد، 2006] مبدأ الإستقراء كمبدئ إستنتاج منطقي، فقد يستنفع القارئ بالرجوع لهذا الكتاب كأساس لهذا الفصل. جل الكتب التي تهتم بالبرمجة تخصص فصلا للبرمجة الإستقرائية، فعلى القارئ المهتم الرجوع لأي كتاب برمجة للمزيد من المعلومات حول هذا الموضوع. كما يجد القارئ في هذا الكتاب تفاصيل إضافية حول الأشجار و ميزاتها.

يجد القارئ المهتم جميع البرامج التي إستعملناها في هذا الفصل في العنوان التالي لدى موقع شركة النشر:

http://www.phillips-publishing.com/jav/rec.java

الفصل الثالث عشر
تجرد البيانات: تعيين الكائنات

يتمثل التجرد، بصفة عامة، في أن نمثل كائنات معقدة بصفة محكمة، تبرز بيانات هامة فتخفي تفاصيل ثانوية غير هامة. في الفصلين السابقين، درسنا وسائل تجرد العمليات في لغة جافا، فننكب في هذا الفصل و الفصل الموالي على تجرد البيانات.

1.13 الكائنات و الفئات

1.1.13 مفاهيم نظرية

إن البيانات التي تعرضنا إليها إلى حد الآن في برامج جافا تمثل كائنات خاضعة، نوعا ما، إذ أنها تخضع لعمليات تقوم بها البرامج، مع أنها لا تتمكن من أي مبادرة تلقائية. ممّا يجهلعا غير لائقة لتمثيل كائنات واقعية بصفة كاملة و دقيقة.

- مثلا، إذا أردنا أن نمثل تلاميذا في جامعة، فنمثل بالنسبة لكل تلميذ إسمه و رقم تسجيله في الجامعة و البرنامج الدراسي الذي ينخرط فيه و قائمة الدروس التي تابعها و قائمة الأعداد التي تحصل عليها، و قائمة الدروس التي يتابعها في السداسي الحالي، إلخ. أما عن كفائاته، فنمثل الإنخراط في الجامعة و التسجيل في درس و الإنسحاب من درس و مطلب التخرج من الجامعة، إلخ.
- أيضا، إذا أردنا أن نمثل متعالجي مستشفى، فنمثل بالنسبة لكل متعالج إسمه و رقم تسجيله في المستشفى و تاريخ ولادته و ماضيه الصحي و أهم أمراضه و أهم حساسياته و أهم الأدوية التي تعرض لها سابقا و أهم العمليات الجراحية التي تعرض لها سابقا و نوعية دمه، إلخ. أما عن كفائاته، فنمثل الإنخراط في المستشفى و القيام بعيادات خارجية و الإقامة بالمستشفى و التعرض لعمليات جراحية، إلخ.

كل هذا يتطلب هياكل بيانات تمكننا من تمثيل كائنات معقدة، مع تمثيل صفاتها و كفائاتها و علاقاتها بكائنات أخرى، إلخ. سعيا لتلبية هذه المتطلبات، فتتزود لغات البرمجية العصرية (بما فيها جافا) بإمكانيات لغوية و تنفيذية لتأييد ما نسميه البرمجة الموجهة للكائنات. إن هذا النوع من البرمجة يتطلب دراسة عميقة و مدققة، تتجاوز نطاق هذا الكتاب؛ نكتفي في هذا الكتاب بطرق بعض أوجهها. في هذا الفصل, ندرس مفهوم الكائنات و الفئات.

> تتمثل الكائنة في هيكل يعكس صفات و كفائات و علاقات ذات أهمية في ميدان تطبيقي.

كيف نعين كائنة في برامج جافا؟ نقوم بذلك عن طريق مفهوم الفئة، الذي نناقشه فيما يلي. إذا أردنا أن نكتب برنامجا يتحدث عن تلاميذ جامعة، مثلا، فيجب أن نعين فئة تمثل مفهوم تلميذ جامعي في ميدان التطبيق الذي نهتم به. يتمثل تعيين الفئة في ضبط التفاصيل التالية:

- الصفات التي تتميز بها كائنات هذه الفئة؛ مثلا، كل تلاميذ الجامعة متحصلون على شهادة التعليم الثانوي؛ و كلهم قدموا مطلبا للإنخراط بهذه الجامعة و تم قبولهم؛ و كلهم قاموا بالإنخراط في الجامعة المعنية بعد أن تم

فبولهم؛ و كلهم يتابعون، أو يتهيئون لمتابعة، دروسا في الجامعة؛ و كل فرد منهم له رقم تسجيل في الجامعة؛ و كلهم يدفع معاليم الدراسة؛ إلخ.

- الكفائات التي تتسم بها كائنات هذه الفئة؛ مثلا، كل تلميذ يتمكن من الإنخراط في الجامعة؛ و في التسجيل في درس ما في فصل ما؛ و في الإنسحاب من درس سجل فيه حسب شروط مضبوطة؛ و في التخرج من الجامعة حسب شروط مضبوطة؛ و في الإنسحاب من الجامعة بدون شهادة؛ و في تغيير برنامج دراسته حسب شروط مضبوطة؛ إلخ.
- العلاقات التي تربط بين هذه الفئة و فئات أخرى: مثلا، تلميذ جامعي هو نوع من فئة التلاميذ، بجانب تلاميذ التعليم الثانوي و تلاميذ التعليم الإبتدائي (الأساسي)؛ أيضا، فئة التلاميذ الجامعيين تنقسم بدورها إلى عدة أقسام، بينها تلاميذ المرحلة الأولى، و تلاميذ المرحلة الثانية و تلاميذ المرحلة الثالثة من التعليم الجامعي؛ أيضا، فئة التلاميذ الجامعيين نوع من فئة المجموعة الجامعية التي تشمل أيضا الأساتذة الجامعيين و الإطار الإداري للجامعة و فئة عملة الجامعة، إلخ. من السهل جدا أن نقتنع أن كل هذه الفئات تتسم بصفات مختلفة و كفائات مختلفة و علاقات مختلفة مع بعضها بعضا. تأملوا في الرسم رقم 1.13.

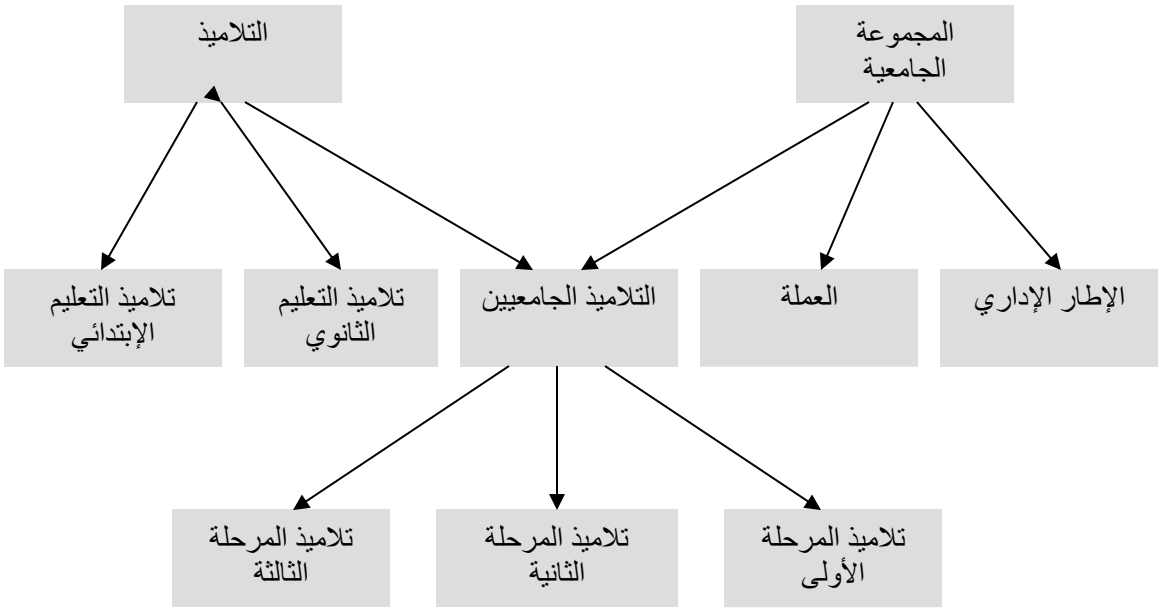
تتمثل الفئة في جملة من الصفات و الكفائات و العلاقات
ذات أهمية في ميدان تطبيقي.

عن الفرق بين الفئة و المجموعة. هناك فرق ضئيل لكن هام بين مفهوم الفئة (كما نناقشه في هذا الفصل) و مفهوم المجموعة (كما نناقشه في نظرية المجموعات الرياضية). نعتبر قسما يتكون من عدد من التلاميذ، و نعتبر الجملتين التاليتين:

هذا القسم نجيب
هذا القسم عديد.

هاتان الجملتان متشابهتان تماما من حيث التحليل النحوي، لكنها تختلفان من حيث التأويل. ففي الجملة الأولى يرجع النعت على كل فرد من أفراد القسم (أي أن كل تلميذ نجيب)؛ بينما في الجملة الثانية ينطبق النعت لا على كل فرد من افراد القسم، بل على القسم كمجموعة. فالفرق راجع إلى أن في الجملة الأولى تدل كلمة القسم على الفئة بينما تدل نفس الكلمة في الجملة الثانية على المجموعة. بصفة عامة، يمكن تشخيص هذا الفرق بأن

- المجموعة تدل على مجموعة عناصر معينة، بينما
- الفئة تدل على صفات مشتركة تتميز بها العناصر (أو على شروط تميز هذه العناصر).



الرسم رقم 1.13 علاقات بين الفئات

إنطلاقا من فئة ما, نستخرج كائنة بواسطة عملية نلقي عليها إسم التجسيد. فيكون مفعول هذه العملية هو خلق كائنة تتسم بكل الأوصاف و تتمتع بكل الكفاءات و تشارك في كل العلاقات المنسوبة للفئة المعنية.

إذا طبقنا عملية التجسيد على فئة ما, نستخرج كائنة تتسم بالصفات و الكفائات و العلاقات المنسوبة للفئة.

2.1.13 الكائنات و الفئات في جافا

تمكننا لغة جافا من تعيين فئات و من إستخراج كائنات منها بواسطة عملية التجسيد. في هذا الفصل، لا ندرس تعيين الفئات (إذ أن هذا موضوع الفصل الموالي)، بل نكتفي بإستعمال فئات موجودة في نطاق اللغة أو في نطاق الوسائل البرمجية المرافقة لها. لكي نستعمل فئة ما في برنامجنا، يجب أن نطلع على خصائصها بصفة دقيقة و شاملة. نمثل التفاصيل المتعلقة بخصائص الفئة في هيكل نلقي عليه إسم مواصفة الفئة. تتمثل مواصفة الفئة فيما يلي:

- مجموعة أساليب تنطبق على كائنات هذه الفئة، منها أسلوب إفتتاحي يقع تطبيقه ضمن عملية التجسيد.
- وصف مدقق لمفعول كل أسلوب.

تتمثل مواصفة فئة في قائمة أساليب الفئة مع وصف مدقق لمفعول كل أسلوب.

إذا كانت لدينا فئة سميناها **`SomeClass`** و أردنا أن نستخرج منها كائنة سميناها **`SomeObject`** فنكتب في جافا:

```
SomeObject = new SomeClass;
```

يكون مفعول هذه العملية إستخراج كائنة من الفئة المعنية مع تنفيذ الأسلوب الإفتتاحي المتعلق بالفئة. بعد هذا التصريح، يمكن تطبيق أي أسلوب ضمن أساليب الفئة على الكائنة المعنية، بواسطة عملية النقطة: إذا أردنا أن نطبق أسلوب **Method** على كائنة **SomeObject**، فنكتب:

```
SomeObject.Method();
```

3.1.13 الفئات و الباقات في جافا

نظرا لأن الفئات تقدم خدمات مختصة جدا، فإن لغة جافا لا توفر جميع الفئات ضمن اللغة نفسها. لأن لغة جافا تنطبق على عدد واسع من ميادين التطبيق، فإذا أردنا أن نوفر الفئات المتعلقة بكل هذه الميادين لأصبحت اللغة كبيرة جدا و غير ناجعة. بالتالي، نفرق بين لغة جافا بالذات، التي تبقى صغيرة الحجم و ناجعة التنفيذ، و مجموعة الفئات التي نظيفها حسب الطلب. تنقسم الفئات العادية إلى باقات، حيث أن كل باقة تشمل عددا من الفئات، فتؤيد ميدانا تطبيقيا معينا. بالتاي يتحتم علينا في كل برنامج جافا أن نصرح، عند الحاجة، بالباقات التي نحتاج لها في نطاق هذا البرنامج. نقوم بهذا التصريح بواسطة تعليمة **import** يتبعها إسم الباقة؛ فيكون مفعول هذه التعليمة أن يضع المصرف على ذمة المبرمج مجموعة الفئات اموجودة ضمن الباقة المعنية.

على باب المثال، نقدم فيما يلي مجموعة من الباقات، فنبين بالنسبة لكل باقة إسمها و خدماتها و أمثلة من الميادين التطبيقية التي تستعمل هذه الباقة.

الباقة	الخدمات / الفئات	الميادين التطبيقية
java.util	تحتوي هذه الباقة على فئات تمثل هياكل بيانات عامة الإستعمال مثل الكومات و الصفوف و المجموعات و غيرها.	البرمجة الخوارزمية العادية.
java.io	تحتوي هذه الباقة على فئات تتصرف في عمليات إصدار و توريد البيانات.	البرامج التي تتطلب معاملة فورية مع المستخدم.
java.lang	تحتوي هذه الباقة على الفئات المتعلقة بتنفيذ البرنامج و مراقبته، بما فيها الفئات التي تعالج أخطاء التنفيذ و بعض الفئات العامة. نظرا لأهميتها، فإن هذه الباقة يقع توريدها ضمنيا في كل برنامج.	كل البرامج/ كل التطبيقات.
java.math	تحتوي هذه الباقة على فئات تقوم بعمليات حسابية، بأي دقة يطلبها المستخدم.	التطبيقات الهندسية و تطبيقات الرياضيات.
java.sql	تحتوي هذه الباقة على فئات تختص في عمليات على قواعد البيانات.	تطبيقات التصرف التي تتطلب قواعد بيانات.

java.awt	تحتوي هذه الباقة على فئات تختص في الرسم و في إنجاز واحهات رسومية.	تطبيقات تتطلب واجهات رسومية مع المستخدم.
java.swing	تمدد هذه الباقة إمكانيات و مقدرات الباقة التالية.	تطبيقات تتطلب واجهات رسومية مع المستخدم.
java.security	تحتوي هذه الباقة على فئات تختص في تنفيذ إجراءات أمنية في البرنامج، مثل مراقبة المستخدمين و صيانة الوارد و غير ذلك.	تطبيقات تتطلب إجراءات أمنية.

2.13 مثال: فئة سلسة الحروف

توفر لنا لغة جافا فئة تحت إسم **String** تمثل سلسلة من الحروف و الأرقام و الرموز. تندرج هذه الفئة في نطاق باقة **java.lang** التي يقع توريدها ضمنيا لكل برنامج جافا؛ بحيث نتمكن من إستعمال هذه الفئة بدون أن نستورد الباقة المعنية بواسطة تعليمة **import**. قبل أن نستعمل هذه الفئة، نطلع على مواصفاتها التي تتكون، كما ذكرنا أعلاه، من مجموعة أساليب مع تفسير يوضح مفعول كل أسلوب. يجد القارئ في المقام التالي مواصفة دقيقة و شاملة لهذه الفئة في المقام التالي للعنكبوتية:

http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html

في نطاق هذا الفصل، نكتفي بمواصفة جزئية تشمل بعض الأساليب، فنعرضها في المائدة رقم 1.13. بالنسبة لكل أسلوب، نبين عنوان الأسلوب في لغة جافا و إسمه باللغة العربية ثم وصفا باللغة العربية.

إفتتاح مجرد	**String()**
يقع تنفيذ هذا الأسلوب كلما قام المبرمج بعملية تجسيد، فتتنفذ على الكائنة العمنية لتسندها قيمة إفتتاحية فارغة.	
إفتتاح بسلسلة	**String(char [] value)**
يقع تنفيذ هذا الأسلوب كلما قام المبرمج بعملية تجسيد، فتتنفذ على الكائنة العمنية لتسندها قيمة إفتتاحية تساوي سلسلة الحروف الموجودة في متغيرة **value**.	
إفتتاح بكائنة	**String(String original)**

يقع تنفيذ هذا الأسلوب كلما قام المبرمج بعملية تجسيد، فتتنفذ على الكائنة العمنية لتسندها قيمة إفتتاحية تساوي سلسلة الحروف التي تمثلها كائنة **original**.

إستخراج حرف	**char CharAt (int index)**

عندما نطبق هذا الأسلوب على كائنة ما، نسترجع الحرف الموجود ضمن سلسلة الكائنة في المقام الذي يشير له مؤشر**index** .

مقارنة قاموصية	**int compareTo (String aString)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع نتيجة المقارنة القاموصية لسلسلة هذه الكائنة مع سلسلة كائنة **aString.** تكون النتيجة عددا سلبيا إذا كانت سلسلة الكائنة تسبق سلسلة **aString** في الترتيب القاموصي؛ و تكون عددا إجابيا إذا كانت سلسلة الكائنة تلي سلسلة **aString** في الترتيب القاموصي؛ و تكون صفرا إذا كانت سلسلة الكائنة تساوي سلسلة **aString**.

التسلسل	**String concat (String str)**

إذا طبقنا هذا الأسلوب على كائنة ما، تصبح سلسلة هذه الكائنة تسلسل السلسلة السابقة للكائنة مع سلسلة **str**.

إسناد الكائنة قيمة سلسلة	**String copyValueOf (char [] data)**

إذا طبقنا هذا الأسلوب على كائنة ما، تصبح سلسلة هذه الكائنة سلسلة الحروف الموجودة في **data**.

مقارنة آخر السلسلة	**boolean endsWith (String suffix)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع القيمة المنطقية **true** أو القيمة المنطقية **false** حسب إن كانت سلسلة الكائنة تنتهي بسلسلة **suffix** أم لا .

مقارنة أول السلسلة	**boolean startsWith (String prefix)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع القيمة المنطقية **true** أو القيمة المنطقية **false** حسب إن كانت سلسلة الكائنة تنبتدئ بسلسلة **prefix** أم لا .

مقارنة السلسلة	**boolean equals (String aString)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع القيمة المنطقية **true** أو القيمة المنطقية **false** حسب إن كانت سلسلة الكائنة تساوي سلسلة **aString** أم لا .

نزع الفضاءات	**String trim ()**

إذا طبقنا هذا الأسلوب على كائنة ما، تنزع الفضاءات الفارغة في بداية السلسلة و نهايتها.

إستخراج موقع حرف	**int indexOf (char ch)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع أول موقع لحرف **ch** في سلسلة الكائنة. إذا لم يكن الحرف موجودا في اسلسلة، نسترجع قيمة خاصة.

إستخراج موقع حرف بعد مؤشر	**int indexOf (char ch, int fromIndex)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع أول موقع لحرف **ch** في سلسلة الكائنة بعد مؤشر **fromIndex**. إذا لم يكن الحرف موجودا في السلسلة، نسترجع قيمة خاصة.

إستخراج موقع سلسلة	**int indexOf (String str)**

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع أول موقع لسلسلة **str** في سلسلة الكائنة. إذا لم يكن الحرف موجودا في اسلسلة، نسترجع قيمة خاصة.

إستخراج موقع سلسلة بعد مؤشر	`int indexOf (String str, int fromIndex)`

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع أول موقع لسلسلة **str** في سلسلة الكائنة بعد مؤشر **fromIndex**. إذا لم يكن الحرف موجودا في اسلسلة، نسترجع قيمة خاصة.

طول السلسلة	`int length ()`

إذا طبقنا هذا الأسلوب على كائنة ما، نسترجع طول هذه السلسلة، أي عدد حروفها.

تعويض حرف بحرف	`String replace (char oldchar, char newchar)`

إذا طبقنا هذا الأسلوب على كائنة ما، تتغير هذه السلسلة حيث يظهر حرف **newchar** في كل مقام ظهر فيه **oldchar** في السلسلة الأصلية للكائنة.

إستخراج سلسلة جزئية	`String substring (int beginIndex)`

إذا طبقنا هذا الأسلوب على كائنة ما، نستخرج سلسلة جزئية منها تبدأ في مؤشر **beginIndex** و تنتهي مع نهاية سلسلة الكائنة.

إستخراج سلسلة جزئية محددة	`String substring (int beginIndex, int endIndex)`

إذا طبقنا هذا الأسلوب على كائنة ما، نستخرج سلسلة جزئية منها تبدأ في مؤشر **beginIndex** و تنتهي في مؤشر **endIndex**.

المائدة رقم 1.13: مواصفة سلسلة الحروف، فئة String

نهتم فيما يلي بكتابة برنامج توضيحي يبين إستعمال فئة السلسلة، فنذكر بعض الأساليب في هذا البرنامج لنوضح مفعولها على الكائنة المعنية.

```
1.  /*
2.  * Main.java
3.  *
4.  * Created on 26 ص 06:40 ,2008 ,يوليو
5.  *
6.  * To change this template, choose Tools | Template Manager
7.  * and open the template in the editor.
8.  */
9.
10. package javaapplication1;
11.
12. /**
13. *
14. * @author Ali Mili
15. */
16. public class Main {
17.
18.  /** Creates a new instance of Main */
19.    public Main() {
20.   }
21.
22.   /**
23.   * @param args the command line arguments
24.   */
```

```
25.     public static void main(String[] args) {
26.      // TODO code application logic
27.       String string1, string2;
28.       String product1, product2, product3;
29.       int leng;
30.       boolean samestr;
31.       string1 = new String ("Java is fun.");
32.       string2 = new String ("C++ too.");
33.       product1 = string1.concat(string2);
34.       product2 = string2.concat(string1);
35.       if (product1.equals(product2))
36.        {System.out.println("concat may be commutative");}
37.        else
38.        {System.out.println("concat is not commutative");}
39.        System.out.println("product 1:" + product1);
40.        System.out.println("product 2:" + product2);
41.        product3 = product1.concat(product2);
42.        System.out.println(product3);
43.        System.out.println("overall length: "+product3.length());
44.        System.out.println("substring: "+product3.substring(12,28));
45.        }
46.    }
```

يمثل السطران 27 و 28 عمليات تصريح: ففي السطر 27 نصرح بأن إسمي **string1** و **string2** يشيران إلى كائنتين من فئة سلسلة و في السطر 28 نصرح بأن أسماء **product1, product2, product3** تشير إلى أسماء كائنات من فئة سلسلة أيضا. هذه البيانات تهم أساسا مصرف جافا، الذي يتعرف خلالها عن هذه الأسماء، فيراقب إستعمال هذه الأسما ليتأكد أن هذا الإستعمال مطابق للفئة المعنية. أما السطران رقم 29 و 30، فهما يصرحان بمتغيرتين من نوع الأعداد الكاملة و القيمات المنطقية.

السطر 31 يقوم بعملية تجسيد، حيث أنه يحجز فضاء في ذاكرة الحاسوب لإستعاب كائنة **string1** و يطبق عليها عملية الإفتتاح بسلسلة. لماذا يطبق هذه العملية بالذات؟ إن كل عملية تجسيد يرافقه ضمنيا تنفيذ عملية إفتتاح ضمن عمليات الإفتتاح التابعة للفئة. كيف يتعرف المصرف على أساليب الإفتتاح؟ هي كل الأساليب التي تحمل نفس الإسم كإسم الفئة (في مثالنا هذا، كل الأساليب التي تسمى **String**). في المائدة رقم 1.13 قدمنا ثلاثة نسخ لهذا الأسلوب؛ إن فئة **String** أساليب إفتتاحية أخرى، يمكن الإطلاع عليها في مقام العنكبوتية المذكور في بداية هذا الجزء، لكننا نكتفي بهؤلاء الثلاثة في نقاشنا هذا. فنطرح السؤال، كيف يقرر المصرف أي أسلوب إفتتاحي ينطبق في كل حالة؟ يفرز المصرف بين نسخ الأساليب الإفتتاحية بواسطة نموذج المعلمات المذكورة في عملية التجسيد. في هذا السطر، سطر 31، نذكر عملية التجسيد بمعلمة واحدة، من نوع سلسلة حروف، فيقع إختيار الأسلوب الثاني في مائدة 1.13. فبالتالي يقع إسناد سلسلة

Java is fun.

لكائنة **string1**. لنفس السبب، يقع في السطر 32 تجسيد **الكائنة string2** ككائنة من فئة **String** مع إسنادها سلسلة

C++ too.

في سطر 33 يقع إسناد كائنة **product1** تسلسل السلسلتين **string1** و **string2**، و ذالك بذكر عملية التسلسل على كائنة **string1** مع إستعمال **string2** كمعلمة. في سطر 34 يقع إسناد كائنة **product2** تسلسل السلسلتين **string2** و **string1**، و ذالك بذكر عملية التسلسل على كائنة **string2** مع إستعمال **string1** كمعلمة. في السطر 35 نختبر إن كانتا هتان السلسلتان متساويتان؛ نظرا لأنهما مختلفتان، يتم تنفيذ السطر رقم 38، الذي يطبع النص التالي:

concat is not commutative

في سطري 39 و 40 نقوم بطبع السلسلتين **product1** و **product2** و في السطر 41 ننفذ أسلوب التسلسل على كائنة **product1** مستعملين كائنة **product2** كمعلمة، و نسند النتيجة لكائنة **product3** التي نطبعها في السطر الموالي، فيقع طبع النص التالي:

```
Java is fun.C++ too.C++ too.Java is fun.
```

في السطر 43 نحتسب و نطبع طول هذه السلسلة، حيث نجد و نطبع 40، ثم في السطر 44 نستخرج و نطبع السلسلة الجزئية لهذه السلسلة بين مؤشري 12 و 28, حيث نطبع

```
C++ too.C++ too.
```

ننفذ هذا البرنامج في جافا فيسفر على النتيجة التالية، و هو مطابقة لما ناقشنا أعلاه:

```
run:
concat is not commutative
product 1:Java is fun.C++ too.
product 2:C++ too.Java is fun.
Java is fun.C++ too.C++ too.Java is fun.
overall length: 40
substring: C++ too.C++ too.
BUILD SUCCESSFUL (total time: 11 seconds)
```

3.13 الفئات العامة و فئات التغليف

هناك عدد من الفئات العامة ذات الأهمية في نطاق واسع من التطبيقات، نريد أن نوفرها للمبرمج في نطاق لغة جافا. أول عقبة نتعرض لها في هذا المجهود هي توفير الفئات المطلوبة بأشكال مختلفة (حسب طلب المبرمج) مع الإقتصاد في مجهود البرمجة. مثلا، غالبا ما نحتاج، في العديد من ميادين التطبيق، لهيكل بيانات يخزن البيانات و ينتجها في ترتيب معاكس، أي أنه ينتج آخر ما خزن، ثم ما جاء مباشرة قبل آخر ما خزن، ثم ما جاء قبل ما جاء قبل آخر ما خزن، إلخ. نلقي على هذا الهيكل إسم الكومة، و ندرس في هذا الجزء، بواسطة مثال بسيط، كيفية إستعمالها.

1.3.13 فئات التغليف

توفر لنا لغة جافا فئة من نوع الكومة, نقدم مواصفتها في المائدة رقم 2.13. رغبة في توسيع نطاق تطبيق هذه الكومة، نريد أن نجعل هذه الكومة قابلة لخزن أي نوع من أنواع البيانات. مثلا، نريد أن نتمكن من خزن أعداد كاملة، أو خزن حروف، أو خزن سلاسل حروف، أو خزن تسجيلات مختصة، إلخ. كيف توفر لغة جافا فئة الكومة مع تمكين المبرمج من خزن أي نوع من البيانات؟ تقوم بذلك بتوفير كومة تخزن بيانات من نوع فئة عامّة تلقي عليها إسم ...**Object**. علما أن هذه الفئة أعمّ من أي فئة قد نصرح بها في لغة جافا، فإن كومة الفئات من نوع **Object** هي أعمّ نوع من الكومات التي نستعملها في برنامج جافا. بالتالي، فإن كومة الأعداد الكاملة و كومة الحروف و كومة سلاسل الحروف، إلخ، كل هذه الكومات تُعتبر حالات خاصة من كومة فئات من نوع **Object**.

إختبار الفراغ	`boolean empty();`
تتفقد هذه العملية المنطقية إن كانت الكومة فارغة أم لا.	
التأمل	`Object peek();`

تتأمل هذه العملية في قمة الكومة و تستخرجها بدون أن تغيير الكومة أو قهمتها. إذا كانت الكومة فارغة، تصرح بخطإ.	
النزع	`Object pop();`
تنزع هذه العملية قمة الكومة. إذا كانت الكومة فارغة، تصرح بخطإ.	
التكويم	`Object push (Object item);`
إيداع كائنة `item` على قمة الكومة.	

المائدة رقم 2.13: مواصفة الكومة في جافا

تفرق لغة جافا بين أنواع البيانات العادية، مثل الأعداد الكاملة و الأعداد الحقيقية و الحروف و القيمات المنطقية من جهة، و الفئات مثل الفئة العامة `Object` و فئة السلسلة `String` و فئة الكومة `Stack` من جهة أخرى. بالتالي، لا يمكن لنا أن نطبق عملية التكويم لعدد كامل، مثلا، لأن العدد الكامل نوع بيانات و ليس فئة. لذا، إذا أردنا أن نكون كومة من الأعداد الطبيعية مثلا، يجب أن نحول الأعداد الطبيعية من نوع بيانات إلى فئة؛ نلقي على هذه العملية إسم التغليف و نلقي على الفئة التي نتحصل عليها إثر هذه العملية إسم فئة التغليف. بالنسبة لأنواع البيانات الجارية، توفر لغة جافا فئات تغليف حسب القائمة التالية:

نوع البيانات	فئة التغليف
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
char	Character
boolean	Boolean
void	Void

نتعرض في الجزء المقبل لمثال برنامج يستعمل فئات التغليف، فيخصص فئة الكومة لخزن حروف.

2.3.13 تطبيق: توازن الأقواس في العبارات

نتأمل في عبارات حسابية تشمل أعدادا و رموزا تمثل أعدادا كما تشمل أيضا عمليات عددية مثل الجمع و الضرب و الطرح و غير ذلك و تشمل أقواسا من ثلاثة أنواع: الأقواس العادية ("(" و ")")؛ و الأقواس المستقيمة ("[" و "]")؛ و الأقواس المنعرجة ("{" و "}"). نهتم في هذا الجزء ببرنامج يحلل عبارة عددية قصد التأكد من أن الأقواس فيها متوازنة، أي أن كل قوس مفتوح يقابله قوس مغلق على نفس المستوى. نقدم فيما يلي أمثلة في العبارات المتوازنة:

([{a}+(c-d)]*(b-2)+{{3}-(((d)))}/{{[4]-a}-b}
[{[{a}]+b}-((a+2)*(({(0)}))/(3)]+[21]

كما نقدم فيما يلي أمثلة لعبارات ذات أقواس غير متوازنة

[(a])

[(a]
())
[a)
{b{.

كيف نقوم بهذا التحليل؟ نقدم الخوارزمية التالية لهذا الغرض. نصرح بكومة حروف ضمنة فئة الكومات، مما يسند لها القيمة الإفتتاحية الفارغة. ثم نهتم بالعبارة فنقرءها حرفا حرفا من اليسار إلى اليمين، مع القيام بالإجراءات التالية:

- كلما وجدنا حرفا دون الأقواس، نتجاهله.
- كلما وجدنا قوسا مفتوحا (أي "(" أو "[" أو "{") نقوم بتكويمه على كومة الحروف.
- كلما وجدنا قوسا مغلقا (أي ")" أو "]" أو "}") فنتأمل في قمة الكومة لنتأكد أنه يطابق القوس الموجود حاليا أمامنا، ثم ننزع قمة الكومة. إذا كانت الكومة فارغة عندما تعرضنا للقوس المغلق، فإن العبارة حتما غير متوازنة، إذ أن هذا القوس المغلق لا يقابله قوسا مفتوحا.

كما تحدثنا في الجزء السابق، فإن جافا توفر كومة فئات، و لا توفر كومة بيانات عادية مثل الحروف. بالتالي، إذا أردنا أن نستعمل كومة حروف، فيجب أن نحول الحروف إلى فئات بواسطة فئات التغليف. لهذا الغرض، نستعمل فئة **Character**. يجد القارئ المواصفة الكاملة لهذه الفئة في الموقع التالي:

http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Character.html.

بالنسبة لتطبيقنا هذا، نكتفي بذكر الأسلوب التالي لهذه الفئة، الذي يستخرج قيمة الحرف الذي تمثله الكائنة، و هو:

```
char charValue();
```

نقدم فيما يلي برنامجا يحلل المعبارات الحسابية قصد ضبط إن كانت أقواسها متوازنة أم لا. يطبق هذا البرنامج الخوارزمية البسيطة التي درسناها أعلاه.

```
1.   /*
2.    * Main.java
3.    *
4.    * Created on 30 م 12:24 ,2008 ,يـولـيـو
5.    *
6.    * To change this template, choose Tools | Template Manager
7.    * and open the template in the editor.
8.    */
9.
10.  package balancedexpression;
11.
12.  /**
13.   *
14.   * @author Ali Mili and Ahmed Ferchichi
15.   */
16.
17.      import java.util.*;
18.
19.      class parsclass
20.      {
21.
22.          boolean openbracket (char ch)
23.          {
24.              return((ch=='(') || (ch=='[') || (ch == '{'));
25.          }
26.
27.
28.          boolean closebracket (char ch)
```

```
29.          {
30.              return((ch==')') || (ch==']') || (ch == '}'));
31.          }
32.
33.          boolean matchpar (char ch1, char ch2)
34.          {
35.              return ((ch1=='(') && (ch2==')') ||
36.                      (ch1=='[') && (ch2==']') ||
37.                      (ch1=='{') && (ch2=='}') );
38.          }
39.
40.
41.      }
42.
43.  public class Main {
44.
45.      /** Creates a new instance of Main */
46.      public Main() {
47.      }
48.
49.      /**
50.       * @param args the command line arguments
51.       */
52.
53.      protected void finalize() throws Throwable {
54.      }
55.      public static void main(String[] args) {
 56.          // TODO code application logic here
 57.
 58.          Stack stackObject;
 59.          String stringObject;
 60.          parsclass pars;
 61.
 62.          stringObject = new String ("{(a+b)+[(c-d)]}*(a+(b+(c()(()))))-d)");
 63.          stackObject = new Stack();
 64.          pars = new parsclass();
 65.
 66.          char ch;
 67.          boolean balanced;
 68.
 69.          balanced = true;
 70.
 71.
 72.          while (stringObject.length() >0)
 73.          {
 74.              ch = stringObject.charAt(0);
 75.              stringObject = stringObject.substring(1);
 76.
 77.              if (pars.openbracket(ch))
 78.              {
 79.                 Character Ch;
 80.                 Ch = new Character (ch);
 81.                 stackObject.push(Ch);
 82.              }
 83.              else
 84.              {if (pars.closebracket(ch))
 85.               {
 86.                   if (!(stackObject.empty()))
 87.                   {
 88.                   Character Ch1 = (Character) stackObject.peek();
 89.                   char ch1 = Ch1.charValue();
 90.                   balanced = (balanced && (pars.matchpar(ch1,ch)));
 91.                   stackObject.pop();
```

```
92.                     }
93.                     else
94.                     {balanced = false;}
95.                 }
96.
97.                 }
98.             };
99.
100.            if (balanced) {System.out.println("balanced");}
101.            else {System.out.println("not balanced");}
102.            }
103.    }
104.
```

نقدم فيما يلي تعليقات تخص مختلف أجزاء هذا البرنامج.

- السطر 17: يجب التصريح بهذه الباقة لكي نتمكن من إستعمال كائنة الكومة.
- الأسطر 19 إلى 41: نعين في هذا الجزء الأساليب التي تضبط إن كان حرف قوسا مفتوحا أم لا، إن كان حرف قوسا مغلقا أم لا، و إن كانا قوسان متقابلان أم لا.
- الأسطر 58 إلى 59: نصرح بأهم متغيرات البرنامج، أي العبارة العددية التي نحللها، و الكومة التي نستعملها في هذا التحليل.
- السطر 62: نضع في متغيرة السلسلة العبارة العددية التي نريد تحليلها.
- السطر 63: نسند للكومة القيمة الإفتتاحية الفارغة، حسب ما جاء في الخوارزمية.
- السطر 69: نستعمل هذه المتغيرة المنطقية لخزن نتيجة التحليل، فنسند لها القيمة المنطقية الإجابية حتى نرى ما يخالف ذلك.
- الأسطر 72 إلى 75: نتأمل في رموز العبارة رمزا رمزا، ففكل دورة نستخرج الرمز الأول للعبارة ثم ننزعه من العبارة.
- الأسطر 77 إلى 82: إذا كان الرمز الحالي قوسا مفتوحا فنضعه على الكومة (إلى أن نلاقي القوس المغلق المقابل له).
- الأسطر 86 إلى 92: إذا كان الرمز الحالي قوسا مغلقا و كانت الكومة غير فارغة، فنتأمل في الرمز الحالي و قمة الكومة لنضبط إن كانا هذان الرمزان متقابلان، فنسجل هذه النتيجة في المتغيرة المنطقية و ننزع قمة الكومة.
- السطر 88: بما أن الكومة لا تحتوي إلا على كائنات من نوع `Object`، فنطبق على نتيجتها عملية (`Character`) لتحويا النتيجة من كائنة تنتمي لفئة `Object` إلى كائنة تنتمي لفئة `Character`.
- السطر 89: من كائنة تنتمي لفئة `Character` نستخرج قيمة نوعها `char`.
- السطر 94: إذا وجدنا قوسا مغلقا بينما كانت الكومة فارغة، فهذا يعني حتما أن هذا القوس المغلق لم يسبق له قوسا مفتوحا مقابلا. بالتالي فإن العبارة غير متوازنة.
- الأسطر 100 و 101: نصرح بالنتيجة حسب قيمة المتغيرة المنطقية.

إذا نفذنا هذا البرنامج على السلسلة الموجودة في سطر 62، و هي

```
{(a+b)+[(c-d)]}*(a+(b+(c()(()))))-d)
```

لوجدنا النتيجة التالية،

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\Ali Mili\queuesim\build\classes
compile:
run:
balanced
BUILD SUCCESSFUL (total time: 0 seconds)
```

مما يدل على أن هذه العبارة متوازنة. و إذا غيرنا هذه العبارة على النحو التالي،

```
{(a+b)+[(c-d])}*(a+(b+(c()(()))))-d)
```

لوجدنا النتيجة التالية،

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\Ali Mili\queuesim\build\classes
compile:
run:
not balanced
BUILD SUCCESSFUL (total time: 3 seconds)
```

مما يدل على أن هذه العبارة غير متوازنة.

4.13 المكونات و الحاويات

نهتم في هذا الجزء بنوع خاص من الفئات، توفرها لغة جافا لمعالجة الرسوم على الحاسوب، و خاصة الواجهات الرسومية بين الحاسوب و المستخدم. توفر لغة جافا هذه الوسائل البرمجية من خلال باقتي **java.awt** و **javax.swing**.

1.4.13 الإطارات و اللوحات و العلامات

تشمل باقات البمجة الرسومية في لغة جافا ثلاثة فئات أساسية في برمجة الواجهات الرسومية، و هي:

- فئة الإطار (**JFrame**) التي تمكننا من رسم نافذة عامة للمناقشة مع المستخدم، مع إسنادها إسما من إختيارنا. يمكننا التصرف في هذه النافذة كما نفعل مع أي نافذة في شاشة الحاسوب، حيث يمكننا تحويلها على الشاشة و تغيير حجمها؛ كما يمكننا أن هذا الإطار فيه ثلاثة علامات في الركن الأعلى على اليمين، بما فيها علامة لغلق النافذة و علامة لتوسيع النافذة على كامل الشاشة و علامة لخزن النافذة في طبق الشاشة.
- فئة اللوحة (**JPanel**) التي نستعملها لتعمير و تنظيم الإطارات. تتسم كل لوحة بعدد من الأوصاف، مثل حجمها و لونها و محتواها، إلخ. قد تحتوي لوحة على نصوص، أو على صور أو على رسوم هندسية أو على ... لوحات أخرى، إلخ.
- فئة العلامة (**JLabel**) التي نستعملها لإستيعاب عناصر رسومية أساسية مثل النصوص و الصور و الأشكال الهندسية و الرسوم، إلخ.

نقول عن الإطارات و اللوحات أنها حاويات (لأنها قد تستوعب عناصر رسومية أخرى) و نقول عن اللوحات و العلامات أنها مكونات لأنها قد تظهر في حاويات. إن المواصفة الكاملة لهذه الفئات تتجاوز نطاق كتابنا، فنوجه القارئ المهتم إلى المواقع الرسمية لدى شركة "سُن" حيث يجد القارئ مواصفات دقيقة و كاملة.

http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JFrame.html
http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JPanel.html
http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JLabel.html

في هذا الفصل، نكتفي بتقديم و شرح بعض الأساليب التابعة لهذه الفئات، تمهيدا لإستعمالها في برامج لاحقة.

JFrame (String title);	**الإفتتاح بعنوان**

يمثل هذا الأسلوب الإفتتاحي عملية التجسيد مصحوبة بإسناد عنوان للإطار المعني.

عملية الغلق الضمني	`void setDefaultCloseOperation(int operation);`

يضبط هذا الأسلوب كيف يقع غلق الإطار المعني في حالة غياب أساليب أخرى (ناتجة مثلا عن المعاملة مع المستخدم).

إستخراج فضاء المحتوى	`void getContentPane();`

يمكننا هذا الأسلوب من إستخراج فضاء المحتوى التابع للإطار المعني. إذا أردنا أن نظيف لوحة (نسميها **mypanel**) للإطار المعني (نسميه **myframe**) ، نكتب ما يلي:

```
myframe.getContenPane().add(mypanel);
```

عملية التركيب	`void pack();`

يقوم هذا الأسلوب بتركيب اللوحات الموجودة ضمن الإطار المعني، بإعتبار الحجم المفضل لكل لوحة (الرجوع للمائدة الموالية).

عملية الكشف	`void setVisible(Boolean b);`

تكشف هذه العملية الإطار المعني أو تخفيه حسب القيمة المنطقية للمعلمة.

المائدة رقم 3.13: مواصفة جزئية للإطار في جافا

الإفتتاح	`JPanel ();`

يمثل هذا الأسلوب الإفتتاحي عملية تجسيد لوحة.

عملية ضبط الحجم	`void setPreferredSize(Dimension preferredSize);`

يضبط هذا الأسلوب الحجم المفضل للوحة المعنية. في نطاق هذا الأسلوب، يقع تعيين هذا الحجم بواسطة عددين يمثلان عرض اللوحة و إرتفاعها.

عملية ضبط اللون	`void setBackground (Color bg);`

يضبط هذا الأسلوب لون اللوحة المعنية بواسطة معلمة **bg**.

إضافة المكونة c للوحة المعنية	`Component add (Component c);`

يضيف هذا الأسلوب المكونة المعنية في نهاية اللوحة المعنية.

المائدة رقم 4.13: مواصفة جزئية للوحة في جافا

توضيحا لما جاء في المائدتين 3.13 و 4.13، نقدم فيما يلي برنامجا يرسم إطارا جاريا، فيضع فيه لوحة فارغة نضبط حجمها و لونها.

```
/*                                                                          //1.
 * Main.java                                                                //2.
 *                                                                          //3.
 * Created on 01 ص 10:00 ,2008 ,أغسطس                                      //4.
 *                                                                          //5.
 * To change this template, choose Tools | Template Manager                 //6.
 * and open the template in the editor.                                     //7.
 */                                                                         //8.
                                                                            //9.
package blancframe;                                                         //10.
                                                                            //11.
/**                                                                         //12.
 *                                                                          //13.
 * @author Ali Mili and Ahmed Ferchichi                                     //14.
 */                                                                         //15.
                                                                            //16.
import java.awt.*;                                                          //17.
import javax.swing.*;                                                       //18.
                                                                            //19.
public class Main {                                                         //20.
                                                                            //21.
    /** Creates a new instance of Main */                                   //22.
    public Main() {                                                         //23.
    }                                                                       //24.
                                                                            //25.
    /**                                                                     //26.
     * @param args the command line arguments                               //27.
     */                                                                     //28.
    public static void main(String[] args) {                                //29.
        // TODO code application logic here                                 //30.
                                                                            //31.
        JFrame frame = new JFrame ("إطار تجريبي فارغ");                      //32.
                                                                            //33.
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);               //34.
                                                                            //35.
        JPanel tile = new JPanel();                                         //36.
                                                                            //37.
        tile.setBackground(Color.magenta);                                  //38.
        tile.setPreferredSize(new Dimension(300,80));                       //39.
                                                                            //40.
        frame.getContentPane().add(tile);                                   //41.
        frame.pack();                                                       //42.
        frame.setVisible(true);                                             //43.
    }                                                                       //44.
                                                                            //46.
}                                                                           //47.
```

نضيف بعض التعليقات فيما يلي:

- السطران 17 و 18: نستورد الباقتان المتعلقتان بالوظائف الرسومية.

- السطر 32: التصريح بالإطار الجديد و تجسيده مع إسناده عنوان "إطار تجريبي فارغ".
- السطر 34: ضبط شرط غلق الإطار.
- السطر 36: التصريح بلوحة جديدة، و تجسيدها.
- السطران 38 و 39: ضبط لون اللوحة و حجمها.
- السطر 41: إضافة اللوحة للإطار.
- السطر 42: تركيب الإطار.
- السطر 43: إبراز الإطار على الشاشة.

إذا نفذنا هذا البرنامج، أسفر على رسم الإطار التالي على الشاشة:

2.4.13 تطبيق: إطار لقاموص عربي أنجليزي

في الجزء السابق، رأينا كيف نصرح بإطار و نضع فيه لوحة. في هذا الجزء، نناقش كيف نستعمل مفهوم العلامة لنضع بيانات في اللوحة التي نضعها في الإطار. نتعرف على فئة العلامة بواسطة المواصفة التالية.

الإفتتاح	`JLabel (String text);`
يمثل هذا الأسلوب الإفتتاحي عملية تجسيد علامة مع إسنادها عنوانا يظهر فيها.	
إستخراج العنوان	`String getText();`
يستخرج هذا الأسلوب عنوان العلامة المعنية.	
ضبط نص العلامة	`void setText (String text);`
يضبط هذا الأسلوب نص العلامة..	

المائدة رقم 5.13: مواصفة جزئية للعلامة في جافا

بواسطة مفهوم العلامة، نألف البرنامج التالي الذي يرسم قاموصا بسيطا عربي أنجليزي.

```
/*                                                    //1.
```

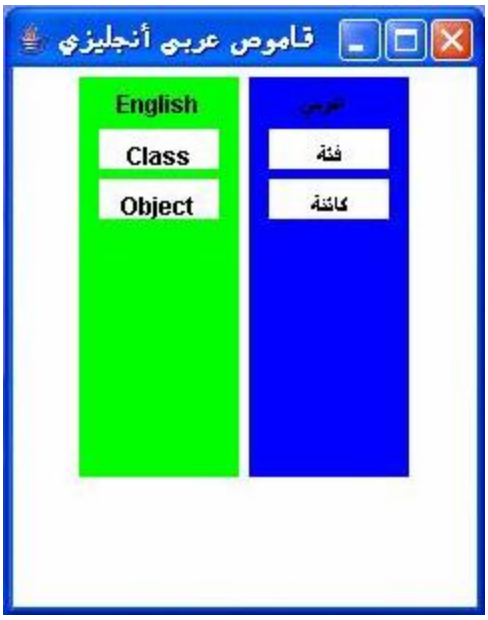
```
 * Main.java                                                                    //2.
 *                                                                             //3.
 * Created on 31 م 12:38 ,2008 ,يوليو                                          //4.
 *                                                                             //5.
 * To change this template, choose Tools | Template Manager                    //6.
 * and open the template in the editor.                                        //7.
 */                                                                            //8.
                                                                               //9.
package graphicsillustration;                                                  //10.
                                                                               //11.
/**                                                                            //12.
 *                                                                             //13.
 * @author Ali Mili and Ahmed Ferchichi                                        //14.
 */                                                                            //15.
                                                                               //16.
import java.awt.*;                                                             //17.
import javax.swing.*;                                                          //18.
                                                                               //19.
public class Main {                                                            //20.
                                                                               //21.
    /** Creates a new instance of Main */                                      //22.
    public Main() {                                                            //23.
    }                                                                          //24.
                                                                               //25.
    /**                                                                        //26.
     * @param args the command line arguments                                  //27.
     */                                                                        //28.
    public static void main(String[] args) {                                  //29.
        // TODO code application logic here                                    //30.
                                                                               //31.
        JFrame frame = new JFrame ("قاموس عربي أنجليزي");
//32.
                                                                               //33.
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  //34.
                                                                               //35.
        JPanel englishCol = new JPanel ();                                     //36.
                                                                               //37.
        JPanel arabicCol = new JPanel ();                                      //38.
                                                                               //39.
        englishCol.setPreferredSize(new Dimension(80,200));                   //40.
        englishCol.setBackground(Color.green);                                 //41.
        JLabel englishTerm = new JLabel ("English");                           //42.
        englishCol.add(englishTerm);                                           //43.
        JPanel englishEntry = new JPanel ();                                   //44.
        englishEntry.setPreferredSize (new Dimension (60,20));                 //45.
        englishEntry.setBackground(Color.white);                               //46.
        englishTerm = new JLabel ("Class");                                    //47.
        englishEntry.add(englishTerm);                                         //48.
        englishCol.add(englishEntry);                                          //49.
        JPanel englishEntry1 = new JPanel ();                                  //50.
        englishEntry1.setPreferredSize (new Dimension (60,20));                //51.
        englishEntry1.setBackground(Color.white);                              //52.
        englishTerm = new JLabel ("Object");                                   //53.
        englishEntry1.add(englishTerm);                                        //54.
        englishCol.add(englishEntry1);                                         //55.
                                                                               //56.
        arabicCol.setPreferredSize(new Dimension(80,200));                     //57.
        arabicCol.setBackground(Color.blue);                                   //58.
        JLabel arabicTerm = new JLabel ("  عربي");                             //59.
        arabicCol.add(arabicTerm);                                             //60.
        JPanel arabicEntry = new JPanel ();                                    //61.
        arabicEntry.setPreferredSize (new Dimension (60,20));                  //62.
        arabicEntry.setBackground(Color.white);                                //63.
```

```
        arabicTerm = new JLabel ("فـئـة");                                  //64.
        arabicEntry.add(arabicTerm);                                        //65.
        arabicCol.add(arabicEntry);                                         //66.
        JPanel arabicEntry1 = new JPanel ();                                //67.
        arabicEntry1.setPreferredSize (new Dimension (60,20));              //68.
        arabicEntry1.setBackground(Color.white);                            //69.
        arabicTerm = new JLabel ("كـائـنـة");                               //70.
        arabicEntry1.add(arabicTerm);                                       //71.
        arabicCol.add(arabicEntry1);                                        //72.
                                                                            //73.
        JPanel globalPanel = new JPanel();                                  //74.
        globalPanel.setBackground (Color.white);                            //75.
        globalPanel.add(englishCol);                                        //76.
        globalPanel.add(arabicCol);                                         //77.
                                                                            //78.
        frame.getContentPane().add(globalPanel);                            //79.
        frame.pack();                                                       //80.
        frame.setVisible(true);                                             //81.
                                                                            //82.
                                                                            //83.
    }                                                                       //84.
                                                                            //85.
}                                                                           //86.
```

نضيف بعض التعليقات فيما يلي:

- السطران 17 و 18: نستورد الباقتان المتعلقتان بالوظائف الرسومية.
- السطر 32: التصريح بالإطار الجديد و تجسيده مع إعطائه إسم "قاموص عربي أنجليزي".
- السطر 34: ضبط شرط غلق الإطار.
- الأسطر 36 إلى 38: التصريح بالعمودين العربي و الأنجليزي كلوحتين.
- الأسطر 40 إلى 55: معالجة العمود الأنجليزي.
- السطر 40: ضبط حجم العمود الأنجليزي.
- السطر 41: ضبط لون العمود الأنجليزي.
- السطر 42: التصريح بعلامة تستوعب عنوان العمود الأنجليزي و إسنادها محتواها.
- السطر 43: إضافة العنوان المذكور أعلاه للعمود الأنجليزي.
- السطر 44: التصريح بلوحة جديدة نخصصها لإستيعاب لفظة أنجليزية.
- السطرين 45 و 46: ضبط حجم اللوحة و لونها.
- السطر 47: التصريح بعلامة تحمل لفظة أنجليزية.
- السطر 48: إضافة هذه اللفظة للوحة المخصصة لها.
- السطر 49: إضافة اللوحة التي تحمل اللفظة الأنجليزية للعمود الأنجليزي.
- الأسطر 50 إلى 55: نفس العملية التي قمنا بها من سطر 44 إلى سطر 49، بالنسبة للفظة أنجليزية ثانية.
- الأسطر57 إلى 72: نفس العملية التي قمنا بها من سطر 40 إلى سطر 55، بالنسبة للعمود العربي.
- السطر 74: التصريح بلوحة شاملة.
- السطر 75: ضبط لون اللوحة.
- السطرين 76 و 77: إضافة لوحتي العمود العربي و العمود الأنجليزي للوحة الشاملة.
- السطر 79: إضافة اللوحة الشاملة للإطار.
- السطر 80: تركيب الإطار.
- السطر 81: إبراز الإطار على الشاشة.

إذا نفذنا هذا البرنامج، أسفر على رسم الإطار التالي على الشاشة:



5.13 تمارين

1 [م] في الجزء 1.13 ناقشنا علاقات بين فئات مختلفة بواسطة مثال فئة التلاميذ. أقيموا قائمة لهذه الفئات المختلفة و إبرزوا فروقا في الصفات و الكفائات المتعلقة بهذه الفئات.

2 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إفتتاح بسلسلة.

3 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إفتتاح بكائنة.

4 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إستخراج حرف.

5 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان مقارنة قاموصية.

6 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إسناد الكائنة قيمة سلسلة.

7 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان مقارنة آخر السلسلة.

8 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان مقارنة أول السلسلة.

9 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان مقارنة السلسلة.

10 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان نزع الفضاءات.

11 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إستخراج موقع حرف.

12 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إستخراج موقع حرف بعد مؤشر.

13 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إستخراج موقع سلسلة.

14 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إستخراج موقع سلسلة بعد مؤشر.

15 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان تعويض حرف بحرف.

16 [م] غيروا (بإضافة تعليمات) البرنامج الموجود في الجزء 2.13 لتبرزوا مفعول الأسلوب الذي قدمناه في مائدة 1.13 تحت عنوان إستخراج سلسلة جزئية.

17 [م] تأملوا في البرنامج التالي. كيف يختلف عن البرنامج الذي ناقشناه في الجزء 2.13؟ هل تختلف نتيجته عن نتيجة البرنامج الأصلي؟ لماذا أو لما لا؟

```
/*
 * Main.java
 *
 * Created on 26 ص 06:40 ,2008 ,يوليو
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package javaapplication1;

/**
 *
 * @author Ali Mili
 */
public class Main {

    /** Creates a new instance of Main */
    public Main() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic
        String string1, string2;
        String product1, product2, product3;
        int leng;
        boolean samestr;
        string1 = new String ("Java is fun.");
```

```
        string2 = new String ("C++ too.");
        product1 = new String(string1.concat(string2));
        product2 = new String(string2.concat(string1));
        if (product1.equals(product2))
        {System.out.println("concat may be commutative");}
        else
        {System.out.println("concat is not commutative");}
        System.out.println("product 1:" + product1);
        System.out.println("product 2:" + product2);
        product3 = new String(product1.concat(product2));
        System.out.println(product3);
        System.out.println("overall length: "+product3.length());
        System.out.println("substring: "+product3.substring(12,28));
    }
}
```

18 [س] إقترح صديقكم برنامجا للمشكلة التي طرحناها في الجزء 2.3.13 يكتفي بتحليل العبارة العددية ليحتسب عدد الأقواس المفتوحة (بالنسبة لكل نوع) و عدد الأقواس المغلقة (بالنسبة لكل نوع) في العبارة و يتأمل إن كان هذان العددان متساويان بالنسبة لكل نوع من الأقواس (العادية، المستقيمة، المنعرجة). هل حله سديد؟ لماذا أو لما لا؟

19 [س] غيروا البرنامج الذي قدمناه في الجزء 2.3.13 ليبين

- إن كان عدد الأقواس المفتوحة يفوق عدد الأقواس المغلقة.
- إن كان عدد الأقواس المغلقة يفوق عدد الأقواس المفتوحة.

بدون إستعمال متغيرات عددية، بل بتحليل السلسلة و الكومة.

20 [س] تأملوا في البرنامج الذي قدمناه في الجزء 2.4.13 و غيروه بحيث يشمل القاموص الكلمات التالية مع ترجمتها: فئات التغليف؛ التجسيد؛ المكونات؛ الحاويات.

6.13 مصادر و مراجع

بالنسبة للتفاصيل اللغوية الخاصة بلغة جافا، نحرض القارئ أن يرجع للمرجع الأصلي لهذه اللغة، و هو موقع العنكبوتية الذي تخصصه شركة "سن" للغة جافا في العنوان التالي:

http://java.sun.com/

أما بالنسبة لمبادئ البرمجة الموجهة للكائنات التي طرقناها في هذا الفصل، يمكن الرجوع لكتاب لويس و لفطوس، [لويس و لفطوس، 2007].

الفصل الرابع عشر

تجرد البيانات، البرمجة الموجهة للكائنات

بينما درسنا في الفصل السابق كيف نؤلف برامج في جافا بواسطة فئات موجودة، فنهتم في هذا الفصل بمسألة متكاملة، و هي كيف نؤلف فئات في جافا، و كيف نربطها ببرامج تستعملها. يدور هذا الفصل حول مثال برنامج نتأمل فيه بدقة، يشمل عددا من الفئات؛ فندرس كيفية تأليفها بالإضافة إلى كيفية إستعمالها. إخترنا لهذا الغرض برنامج محاكاة، أي برنامجا يمثل ظاهرة ما، فيبين تنفيذ هذه الظاهرة بواسطة قياسات تعكس أوجه هامة من هذه الظاهرة. نستعمل برامج المحاكات عادة قصد القيام بتجارب على الظاهرة الرمزية كلما كان القيام بتجارب على الظاهرة الحقيقية عيسرا أو مرتفع التكاليف. نستهل هذا الفصل بمناقشة موجزة لهذه المشكلة (في الجزء الأول) ثم نقدم مواصفة الفئات التي نحتاج لها في إنجاز برنامج المحاكات (في الجزء الثاني) ثم نناقش تصميم و تأليف و تنفيذ / تجربة برنامج يستعمل هذه الفئات للقيام بالمحاكاة (في الجزء الثالث) ثم نوجه إهتمامنا لتصميم و تأليف الفئات نفسها (في الجزء الرابع). في ختام هذا الفصل، نناقش بعض المسائل العامة المتعلقة بالبرمجة الموجهة للكائنات (في الجزء الخامس).

1.14 برنامج محاكاة: حركة الحرفاء في مركز خدمات

نعتبر مركز خدمات يتمثل في عدد من محطات الخدمات و عدد من صفوف الحرفاء و نهتم بتحليل حركة الحرفاء عبر هذا النظام. يمثل هذا النموذج العديد من الحالات الواقعية، مثل:

- مجموعة من محطات الدفع في مغازة و حرفاء المغازة.
- مجموعة من محطات شرطة الجدود و مسافرين واردين.
- مجموعة من محطات تسجيل الأمتعة في مطار و مسافرين راحلين.
- مجموعة من محطات التزود بالوقود و سيارات تحتاج الوقود.
- مجموعة من شبابيك الخدمات في إدارة و مواطنين يطالبون بهذه الخدمات،
- مجموعة من شبابيك الخدمات في بنك و حرفاء البنك.
- مجموعة من موزعي النقود في بنك و حرفاء البنك.
- إلخ...

تأمل في الرسم رقم 1.14. إذا كنا مسؤلين على تنظيم مثل هذه المحطات، فقد نريد أن نجيب على أسئلة عديدة، مثل:

- هل نضع صفا واحدا يزود كل المحطات أم نخصص صفا لكل محطة؟
- إذا خصصنا صفا لكل محطة، كيف نوزع الحرفاء الواردين على المحطات؟ هل يكون ذلك بصفة عشوائية؟ حسب طول الصفوف؟ حسب سرعة محطات الخدمات؟
- كيف نرتب الحرفاء في كل صف؟ هل نرتبهم حسب وقت الوصول أم حسب حجم الخدمات المطلوبة، مثلا؟
- هل نخصص محطات للحرفاء الذين يحتاجون خدمات سريعة أم لا نفرق بين الحرفاء؟
- هل نخصص محطات الخدمات حسب نوع الخدمة أم نجعل كل المحطات تقدم كل الخدمات؟
- إذا كانت بعض المحطات أسرع من الأخرى في تقديم الخدمات، هل نوجه لها أكثر حرفاء؟ أكثر بكم؟
- إلخ....

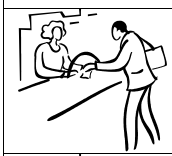
لا يمكن لنا أن نقدم إجابات لهذه التساؤلات إلا إذا ضبطنا المعيار الذي نختاره ضمن المعايير العديدة، مثل:

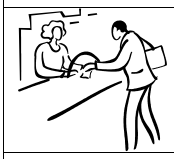
- نريد أن يكون معدل إنتظار الحرفاء (الوقت الذي يمضي بين وصولهم لمركز الخدمات و وصولهم لمحطة خدمات) أقل ما يمكن.
- نريد أن يكون معدل إقامة الحرفاء (الوقت الذي يمضي بين وصولهم لمركز الخدمات و خروجهم منه) أقل ما يمكن.
- نريد أن يكون أقصى إنتظار الحرفاء (الوقت الذي يمضي بين وصولهم لمركز الخدمات و وصولهم لمحطة خدمات) أقل ما يمكن.
- نريد أن يكون أقصى إقامة الحرفاء (الوقت الذي يمضي بين وصولهم لمركز الخدمات و خروجهم منه) أقل ما يمكن.
- نريد أن يكون عدد الحرفاء الذين يتلقون خدمة في فترة زمنية ما أكثر ما يمكن.
- نريد أن تكون نسبة إنشغال المحطات (نسبة الزمن الذي تقضيه المحطات في خدمة حريف ما) أكثر ما يمكن.
- نريد أن تكون صفوف الإنتظار أقصر ما يمكن، في المعدل.
- نريد أن يكون النظام عادلا، أي مثلا: من قدم قبل غيره تحصل على محطة خدمات قبل غيره.

إن الإجابة على كل هذه الأسئلة و التوصل إلى أسلوب ملائم لتوجيه الحرفاء، يمكن أن نهتدي إليه بواسطة نوعين من المناهج:

- المناهج التحليلية، التي تنمذج حركة الحرفاء و تحللها،
- المناهج التجريبية، التي تنفذ برامج محاكاة لحركة الحرفاء، فتلتقط قياسات عنها.

نهتم في هذا الفصل بتطبيق منهج تجريبي لتحليل حركة الحرفاء في مركز خدمات، فنستعمل برنامج محاكاة نكتبه في لغة جافا. هذا موضوع الجزء المقبل.

الحرفاء الواردون	صفوف الإنتظار	محطات الخدمات
		
		

	...	...
		

الرسم رقم 1.14: تنظيم مركز خدمات

في نطاق هذا الجزء، سعيا للبساطة، نكتفي بنمذجة مركز خدمات يتكون من محطة خدمات واحدة و صف إنتظار واحد. فننفذ تجربة تنطلق عند فتح مركز الخدمات في الصباح (مثلا: الساعة التاسعة صباحا) و تنتهي عند غلقه في المساء (مثلا: الساعة الخامسة بعد الظهر). هذا, كما نكتفي بتحليل وجه واحد من هذه التجربة، و هو عدد الحرفاء الموجودين في صف الإنتظار عندم تنتهي التجربة. إذا كانت محطة الخدمات تخدم الحرفاء بسرعة كافية و كان الحرفاء لا يأتون إلا نادرا، فقد يكون الصف فارغا أو يكاد يكون فارغا. أما إذا كان الحرفاء يأتون بكثرة و يطلبون خدمات تستغرق وقتا طويلا و لم تكن محطة الخدمات ذات سرعة كافية، فقد نجدا صفا طويلا عندما يحين وقت الغلق.

2.14 فئات البرنامج

قبل أن نبادر بتحليل المشكلة و تأليف برنامج جافا، نناقش ما هي الفئات التي نحتاج لها في نطاق هذا البرنامج، و نكتب مواصفاتها.

الحرفاء الواردون	صف الإنتظار	محطة الخدمات
		

الرسم رقم 2.14: مركز خدمات ذات محطة واحدة

حسب الرسم رقم 2.14، يتضح أننا نحتاج لفئة لكل من الحرفاء، و صف الإنتظار و محطة الخدمات. نخصص الثلاثة أجزاء التالية لهذه الفئات الثلاثة.

1.2.14 فئة تدفق الحرفاء

نعين فئة نلقي عليها إسم

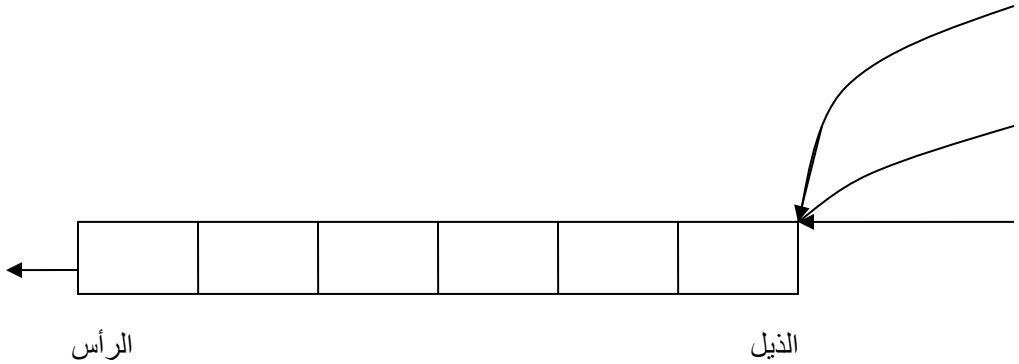
customerflow

تتكلف بتنظيم و تعديل تدفق الحرفاء على مركز الخدمات. نقدم مواصفة هذه الفئة في المائدة التالية.

مواصفة فئة تدفق الحرفاء **customerflow**	
أسلوب الإفتتاح	**`customerflow (int arrivals)`**
يقع ذكر هذا الأسلوب ضمنيا كلما قمنا بتجسيد هذه الفئة، فيكوّن هذا الأسلوب كائنة من هذه الفئة و يضبط معدل الزمن المقتضي بين ظهور حريفين متتاليين. إذا وضعنا قيمة 10 في متغيرة **`arrivals`** فيظهر حريف جديد كل عشة دقائق تقريبا. إن ظهور الحرفاء يقع بصفة عشوائية، لكن هذه المتغيرة تضبط معدل الزمن المقتضي بين حريفين متتاليين.	
قدوم حريف جديد	**`boolean newcustomer()`**
هذا أسلوب منطقي كلما ناديناه يسترجع قيمة منطقية تشير إلى ظهور أو عدم ظهور حريف جديد. إذا كانت قيمة **`arrivals`** عشرة مثلا، فكلما نادينا هذا الأسلوب عشرة مرات، كانت إحداهم صحيحة و التسعة الأخرى غالطة، في المعدل.	
إنتاج حريف جديد	**`int generatecustomer ()`**
هذا أسلوب يسترجع قيمة عددية نستعملها للتعريف بالحرفاء.	

2.2.14 فئة صف الإنتظار

تتكلف هذه الفئة بقبول الحرفاء و وضعهم في مقامهم ضمن صف الإنتظار. يقع ترصيف الحرفاء حسب وقت وصولهم فقط، بحيث أنهم يغادراو صف الإنتظار في نفس الترتيب الذي وردوا فيه عليه. بصفة مجردة، ننمذج صف الإنتظار كما يدل الرسم رقم 3.14، حيث يرد الحرفاء على الصف من ذيله و يغادرونه من رأسه.



الرسم رقم 3.14: صف الإنتظار

نقدم مواصفة هذه الفئة في المائدة التالية.

مواصفة فئة صف الإنتظار **intqueue**	

أسلوب الإفتتاح	intqueue (int buffersize)
يقع ذكر هذا الأسلوب ضمنيا كلما قمنا بتجسيد هذه الفئة، فيكوّن هذا الأسلوب كائنة من هذه الفئة و يضبط أقصى حجم للصف، أي أقصى عدد حرفاء قد يتواجدون في الصف في نفس الوقت.	
إيداع حريف في ذيل الصف	**void enqueue (int item)**
يقوم هذا الأسلوب بإيداع حريف **(item)** في ذيل الصف. نفترض أن هذا الأسلوب لا يقع ندائه إلا إذا كان الصف يتسع لحريف إضافي.	
نزع حريف من رأس الصف	**int dequeue ()**
يقوم هذا الأسلوب بنزع الحريف الموجود حاليا في رأس الصف. نفترض أن هذا الأسلوب لا يقع ندائه إلا إذا كان الصف غير فارغ.	
حجم الصف	**int size()**
عدد الحرفاء المتواجدين في الصف حاليا.	
هل الصف فارغ؟	**boolean empty ()**
يسترجع هذا الأسلوب قيمة منطقية صحيحة أو غالطة حسب إن كان الصف فارغا أم لا.	

3.2.14 فئة محطة الخدمات

نستعمل فئة محطة الخدمات لنمثل تصرف محطة الخدمات، فتقوم هذه الفئة بعمليات مثل إسناد حريف لمحطة، تقديم خدمات لحريف حسب سرعة المحطة و متطلبات الحريف، إنهاء خدمة الحريف، إلخ. نسند لكل محطة خدمات عددا يمثل سرعة الأداء لهذا المحطة. إذا كان حريف يتطلب 60 وحدة من الخدمة و كانت سرعة المحطة 1 فتستغرق خدمة هذا الحريف على هذه المحطة 60 دقيقة؛ و إذا كانت سرعة المحطة 2 فتستغرق خدمة الحريف 30 دقيقة؛ و إذا كانت سرعة المحطة 3 فتستغرق خدمة الحريف 20 دقيقة، إلخ. نقدم مواصفة هذه الفئة في المائدة التالية.

مواصفة فئة محطة الخدمات **serviceStation**	
أسلوب الإفتتاح	**serviceStation** **(int stationRate, int avgneed)**
يقع ذكر هذا الأسلوب ضمنيا كلما قمنا بتجسيد هذه الفئة، فيكوّن هذا الأسلوب كائنة من هذه الفئة و يضبط متغيرتين ذات إهتمام بالنسبة لهذه الفئة: الأولى هي سرعة هذه المحطة، و الثانية هي معدل متطلبات الحرفاء. كان من المنطقي أن نسند المتغيرة الثانية لفئة تدفق الحرفاء، لكن لم نتمكن من ذلك، لأسباب ستتضح في الجزء الموالي.	
إسناد حريف للمحطة	**void loadCustomer (int custID)**
يقوم هذا الأسلوب بإسناد الحريف **custID** لمحطة الخدمة. يتمثل ذلك في إحتساب متطلبات الحريف المعني بصفة عشوائية و خزن قيمة هذه المتطلبات لنتحتسب كم تبقى المحطة منشغلة مع هذا الحريف.	
تقديم الخدمة	**void serve (int time)**
يقوم هذا الأسلوب بتقديم خدمة للحريف الموجود حاليا على محطة الخدمة. يتمثل هذا في إحتساب ما بقي من متطلبات الحريف حسب سرعة المحطة.	

هل المحطة مستعدة؟	boolean available ()
يسترجع هذا الأسلوب قيمة منطقية صحيحة أو غالطة حسب إن كانت المحطة مستعدة أم لا. .	

3.14 خوارزمية المحاكاة

إنطلاقا من الثلاثة فئات التي قدمناها في الجزء السابق، نقوم بعملية محاكة حركة الحرفاء في مركز الخدمات كما يلي:

- نعتبر متغيرة نلقي عليها إسم **time** نستعملها لتمثيل مرور الزمن بحساب الدقائق. ننفذ التجربة بين الساعة التاسعة صباحا (540 بحساب الدقائق) و الساعة الخامسة بعد الظهر (1020 بحساب الدقائق).
- نجسد كائنة من فئة تدفق الحرفاء نختبرها كل دقيقة لنتفقد إن ظهر حريف جديد أم لا.
- نجسد كائنة من فئة صف الإنتظار نخزن فيها الحرفاء عندما يظهرون و نجلب منها الحرفا عندما يحين وقت خدماتهم.
- نجسد كائنة من فئة محطة الخدمات نستعملها لنمثل تقديم الخدمات للحرفاء.
- نسجل أهم الأحداث في حركة كل حريف، و هي وقت ظهوره في باب المركز و وقت إسناده لمحطة الخدمات و وقت مغادرته مركز الخدمات.
- نسجل عدد الحرفاء في صف الإنتظار على الساعة الخامسة بعد الظهر، عند غلق أبواب المركز.

نقدم فيما يلي البرنامج الذي يشخص هذه الخوارزمية في لغة جافا:

```
public static void main(String[] args) {                    // 1.
        // TODO code application logic here                 // 2.
                                                            // 3.
        // work variables                                   // 4.
                                                            // 5.
        int customerid;                                     // 6.
        int hours, minutes;                                 // 7.
                                                            // 8.
        // global variables                                 // 9.
                                                            // 10.
        // int time;                                        // 11.
        int opening;                                        // 12.
        int closing;                                        // 13.
                                                            // 14.
        customerflow cflow;                                 // 15.
        intqueue intq;                                      // 16.
        serviceStation servestation;                        // 17.
                                                            // 18.
        //  initializations                                 // 19.
                                                            // 20.
        opening = 540;                                      // 21.
        closing = 1020;                                     // 22.
        cflow = new customerflow(3);                        // 23.
        intq = new intqueue (60);                           // 24.
        servestation = new serviceStation(3,11);            // 25.
                                                            // 26.
        // main loop                                        // 27.
                                                            // 28.
        for (int time =opening; time<(closing); time++)     // 29.
        {                                                   // 30.
```

```
            hours = time / 60;                                          // 31.
            minutes = time % 60;                                        // 32.
            if (cflow.newcustomer())                                    // 33.
            {                                                           // 34.
                customerid = cflow.generatecustomer();                  // 35.
                intq.enqueue(customerid);                               // 36.
                System.out.println("customer "+customerid+" arrived at time "+
                        hours+":"+minutes );                            // 38.
            }                                                           // 39.
            servestation.serve(time);                                   // 40.
            if (!(intq.empty()) && servestation.available())            // 41.
            {                                                           // 42.
                customerid = intq.dequeue();                            // 43.
                servestation.loadCustomer(customerid);                  // 44.
                System.out.println("customer "+customerid+" served at time "+
                        hours+":"+minutes);                             // 46.
            }                                                           // 47.
        }                                                               // 48.
        System.out.println("Queue Length at Closing: "+intq.size());    // 49.
        }                                                               // 50.
    }                                                                   // 51.
```

نعلق على أهم ما جاء في هذا البرنامج:

- الأسطر 6 إ 13: التصريح بهوية الحريف، و بأوقات العمل.
- الأسطر 15 إلى 17: التصريح بالثلاثة كائنات التي تمثل تدفق الحرفاء، صف الإنتظار، و محطة الخدمات.
- الأسطر 21 إلى 25: إسناد قيمات إفتتاحية لأهم متغيرات البرنامج,
  - أوقات العمل، من التاسعة إلى الخامسة.
  - تدفق الحرفاء بمعدل حريف كل ثلاثة دقائق.
  - التصريح بصف إنتظار حمولته 60 حريف.
  - التصريح بمحطة خدمات سرعتها 3 و معدل حاجيات الحرفاء 11.
- سطر 29: تمثل كل دورة من هذا المدار مرور دقيقية واحدة.
- السطرين 31 و 32: تحويل الوقت من الدقائق غلى الساعات و الدقائق.
- الأسطر 33 إلى 39: إذا طرأ حريف جديد، نسند له رقم تعريف ثم نضعه في صف الإنتظار.
- السطر 40: أداء الخدمة حسب ما ناقشنا أعلاه.
- الأسطر 41 إلى 47: إذا كانت محطة الخدمات مستعدة و كان صف الحرفاء غير فارغ، نخرج حريفا من الصف و نسند له محطة الخدمات.
- السطر 49: نفحص صف الإنتظار عندما يغلق المركز بابه.

إذا نفذنا هذا البرنامج حسب القيمات الحالية (من متطلبات الحرفاء و سرعة محطة الخدمات و كثافة ظهور الحرفاء، إلخ)، نجد النتيجة التالية (حيث لا نكشف إلا عن الدقائق الأخيرة للتجربة):

```
customer 115 completed service at time 16:53
customer 116 served at time 16:53
customer 168 arrived at time 16:56
customer 116 completed service at time 16:56
customer 117 served at time 16:56
customer 117 completed service at time 16:57
customer 118 served at time 16:57
customer 169 arrived at time 16:58
customer 170 arrived at time 16:59
Queue Length at Closing: 52
BUILD SUCCESSFUL (total time: 6 seconds)
```

أي أن عند غلق الأبوا على الساعة الخامسة، بقي 52 حريفا في الصف، و هو عدد هائل. ماذا يصير لو سرعنا في أداء الخدمات؟ نغير سرعة الخدمات من 3 إلى 5، فنجد:

```
customer 151 completed service at time 16:54
customer 152 arrived at time 16:55
customer 152 served at time 16:55
customer 153 arrived at time 16:56
customer 152 completed service at time 16:56
customer 153 served at time 16:56
customer 154 arrived at time 16:59
Queue Length at Closing: 1
BUILD SUCCESSFUL (total time: 1 second)
```

أي أن طول صف الإنتظار أصبح 1 عوضا عن 52. نلاحظ أن الحريف رقم 153 وصل على الساعة الرابعة و 56 دقيقة، فتحصل على محطة الخدمات حالا لأنها أصبحت مستعدة في نفس الوقت. ففي الساعة الرابعة و 59 دقيقة وصل الحريف رقم 154 بينما الحريف رقم 153 لم يتم من إستعمال محطة الخدمات، فبقي الحريف 154 في صف الحرفاء، و كان بمفرده هناك. في التجربة الثالثة، نخفض في سرعة محطة الخدمات فتصبح 2، و نوسع في الوقت بين حريفين متتاليين ليصبح 5، و نغير في معدل متطلبات الحرفاء ليصبح 10؛ فنجد:

```
customer 88 completed service at time 16:51
customer 89 served at time 16:51
customer 98 arrived at time 16:52
customer 99 arrived at time 16:55
customer 100 arrived at time 16:57
customer 89 completed service at time 16:57
customer 90 served at time 16:57
Queue Length at Closing: 10
BUILD SUCCESSFUL (total time: 2 seconds)
```

في هذه الحالة نجد أن عدد الحرفاء عند نهاية فترة التجربة يساوي 10. إذ أن على الساعة الرابعة و 57 دقيقة ظهر الحريف رقم 100 و إكتمل الحريف رقم 89 خدمته و وقع إسناد محطة الخدمات للحريف رقم 90، فبقي في الصف كل من الحرفاء ذوي رقم 91 إلى 100، أي عشرة حرفاء.

4.14 تصميم و تأليف الفئات

في الجزء الماضي إفترضنا وجود الفئات المناسبة (تدفق الحرفاء و صف الإنتظار و محطة الخدمات) و إفترضنا أن هذه الفئات تتصرف كما تنص عليه المواصفات التي قدمناها في الجزء 2.14، فألفنا برنامج المحاكاة بناء على هذه الإفتراضات و نفذناه العديد من المرات. في هذا الجزء، نوجه إهتمامنا لمسألة تأليف الفئات، فنستهل نقاشنا بدراسة مفاهيم عامة عن الفئات، ثم نوضح هذه المفاهيم بواسطة برنامج المحاكاة.

1.4.14 تركيب الفئات في جافا

بصفة عامة، نستعمل الفئات في جافا لنمثل كائنات ذات أهمية في محيطنا، فنمثلها بصفة تعكس صفات هذه الكائنات و كفاءاتها. توفر لنا لغة جافا هيكل الفئة، تحت إسم **class**، حسب النحو التالي،

```
class <classname>
   {
       <variableDeclarations>
       <methodDeclarations>
   }
```

حيث نتمكن في نطاق التصريح بالمتغيرات (**<variableDeclarations>**) من تمثيل الأوجه الهامة للكائنات، و نتمكن في نطاق التصريح بالأساليب (**<methodDeclarations>**) من تمثيل العمليات التي قد نقوم بها على هذه الكائنات. يتغير تمثيل الكائنة حسب التطبيق الذي نخططه، حيث أن نفس الكائنة الواقعة تتمثل بصفة أو بأخرى حسب

التطبيق. مثلا، نهتم بسفرة جوية و نمثلها بخمسة فئات مختلفة حسب وجهة نظرنا: كوكالة أسفار، أو كإدارة الطيران المدني، أو كشركة الطيران التي تملك الطائرة، أو كشركة تزويد الوقود، أو كشركة تزويد الأكلات على متن الطائرة.

وجهة النظر / التطبيق	البيانات	الأساليب
وكالة أسفار	تاريخ الرحلة ساعة الإنطلاق من، إلى عدد البقاع الفارغة في كل درجة أسعار كل درجة	حجز بقاع إلغاء الحجز إشتراء تذاكر ...
إدارة الطيران المدني	تاريخ الرحلة و زمانها من، إلى المسار الجوي للرحلة أبراج المراقبة التي تمر بها الطائرة محطات الرادار التي تتابع الطائرة طراز الطائرة رقم ذيل الطائرة	إسناد مسار الطائرة الإطلاع على مسار الطائرة الإطلاع على موقع الطائرة
شركة الطيران	تاريخ الرحلة و زمانها من، إلى طاقم الطائرة قائمة الركاب	إسناد طاقم للطائرة إحتساب تكاليف السفرة إحتساب دخل السفرة
شركة تزويد الوقود	تاريخ الرحلة و زمانها من، إلى طراز الطائرة كمية الوقود الموجودة قبل التزويد عدد الركاب وزن الشحن	إحتساب الكمية اللازمة من الوقود إصدار فاتورة لشركة الطيران
شركة تزويد الأكل	تاريخ الرحلة و زمانها من، إلى عدد الركاب حسب الدرجة عدد أفراد الطاقم	إحتساب عدد الأكلات لكل درجة إصدار فاتورة لشركة الطيران

نفرق بين نوعين من التصريحات في فئات جافا: التصريحات الخفية، تسبقها كلمة (**private**) فتدل أن هذه المتغيرة أو الأسلوب لا يمكن التوصل إليه من خارج الفئة؛ و التصريحات المكشوفة (أو العمومية)، تسبقها كلمة (**public**) فتدل أن هذه المتغيرة أو الأسلوب يمكن التوصل إليه من خارج الفئة. تبين المائدة التالية أهمية هذا الفرق.

	الخفية	المكشوفة
المتغيرات	يمكننا إخفاء المتغيرات من حماية حالة الكائنة، بحيث لا يمكن تغيير حالة الكائنات إلا عن طريق الأساليب.	إن مناهج البرمجة تنهي بكل شدة عن إستعمال متغيرات مكشوفة في الفئات، بل تحرض إلى إستعمال الأساليب للكشف عن المتغيرات.

الأساليب	إن الأساليب المكشوفة تمكن مستعملي الكائنة من الإطلاع على حالة الكائنة و من تغيير حالة الكائنة بصفة منتظمة و مراقبة.	نستعمل الأساليب الخفية قصد القيام بعمليات لا تهم البرامج الأخرى، بل تؤيد الأساليب المكشوفة.

هذا، و نفرق بين نوعين من الأساليب المكشوفة، و هي:

- اساليب الإطلاع (أو أساليب الكشف) التي تكشف عن حالة الكائنة لكن لا تغيرها. يشمل هذا النوع أساليب مثل: أسلوب الحجم و أسلوب الفراغ في فئة صف الإنتظار (الجزء 2.2.14).
- أساليب التغيير، التي تغير حالة الكائنة لكن لا تكشف عنها. يشمل هذا النوع أساليب مثل: كل الأساليب الإفتتاحية؛ أسلوب الإيداع في الصف في كائنة صف الإنتظار (الجزء 2.2.14).

نشير إلى أن بعض الاساليب لا تخضع لهذا التقسيم البسيط: مثلا، أسلوب النزع من صف الإنتظار يغير حالة الصف و يكشف عنها في نفس الوقت. يمكن تعويضه بأسلوبين، أسلوب يكشف عن رأس الصف و لا يغيره، و أسلوب ينزع رأس الصف بدون أن يكشفه.

2.4.14 تأليف فئة تدفق الحرفاء

إن مواصفة الفئة تمثل عقدا بين مستخدم الفئة و مؤلفها. فبينما تأملنا في مواصفة الفئة عندما إستعملناها، لنطلع على الخدمات التي تقدمها لنا، نتأمل فيها من جديد لنقرر كيف نوفر هذا الخدمات بواسطة لغة جافا:

مواصفة فئة تدفق الحرفاء **customerflow**	
أسلوب الإفتتاح	**`customerflow (int arrivals)`**
يقع ذكر هذا الأسلوب ضمنيا كلما قمنا بتجسيد هذه الفئة، فيكوّن هذا الأسلوب كائنة من هذه الفئة و يضبط معدل الزمن المقتضي بين ظهور حريفين متتاليين.	
قدوم حريف جديد	**`boolean newcustomer()`**
هذا أسلوب منطقي كلما ناديناه يسترجع قيمة منطقية تشير إلى ظهور أو عدم ظهور حريف جديد	
إنتاج حريف جديد	**`int generatecustomer ()`**
هذا أسلوب يسترجع قيمة عددية نستعملها للتعريف بالحرفاء.	

لكي نوفر الخدمات المطلوبة، نقترح تعيين متغيرتين إثنتين:

- متغيرة تضبط معدل الزمن المنقضي بين ظهور حريفين متتاليين. يقع إسنادها قيمة من طرف الأسلوب الإفتتاحي، بواسطة معلمة **`arrivals`**. نلقي على هذه المتغيرة إسم **`arrivalrate`**.
- متغيرة نستعملها لإسناد عدد لكل حريف جديد، قصد التعريف عنه. يقع إسنادها قيمة إفتتاحية من طرف الأسلوب الإفتتاحي، و يقع تحيينها كلما نفذنا أسلوب إنتاج حريف جديد. نلقي على هذه المتغيرة إسم **`customerID`**.

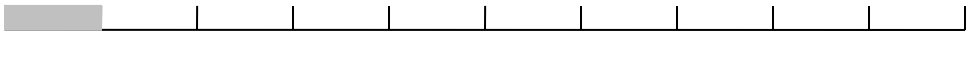
فنجد الفئة التالية:

```
class customerflow                                          // 1.
{                                                           // 2.
    private int customerID;      // customer identification  // 3.
```

```
    private int arrivalrate;      // a new customer arrives every        // 4.
                                  // ''arrival rate'' minutes on avg     // 5.
                                                                         // 6.
    Random generator;                                                    // 7.
                                                                         // 8.
    public customerflow (int arrivals)                                   // 9.
    {                                                                    // 10.
        customerID = 1;                                                  // 11.
        arrivalrate = arrivals;                                          // 12.
        generator = new Random();                                        // 13.
    }                                                                    // 14.
                                                                         // 15.
     public boolean newcustomer ()                                       // 16.
     {                                                                   // 17.
         float draw = generator.nextFloat();                             // 18.
         return (draw < 1/((float)arrivalrate));                         // 19.
     }                                                                   // 20.
                                                                         // 21.
     public int generatecustomer ()                                      // 22.
     {                                                                   // 23.
         customerID++;                                                   // 24.
         return customerID;                                              // 25.
     }                                                                   // 26.
                                                                         // 27.
}                                                                        // 28.
```

نقدم فيما يلي بعض الملاحضات حول هذه الفئة:

- الأسطر 2 و 3: نصرح بمتغيران الفئة كمتغيرات مخفية.
- الأسطر 9 إلى 14: إسناد القيمات الإفتتاحية لمتغيرات الفئة، مع تجسيد مولد الأعداد العشوائية.
- السطر 18: نضع في متغيرة **draw** قيمة عشوائية بين 0.0 (ضمنا) و 0.1 (لاضمنا). نعلن عن قدوم حريف جديد إذا كان هذا العدد موجودا بين الصفر و قيمة 1 منقسم على **arrivalrate**. لماذا؟ إذا كانت قيمة **arrivalrate** تساوي 10 مثلا، فنحن نريد أن يتم قدوم حريف جديد مرة كل عشرة دقائق تقريبا. و هذا ما يصير إذا إخترنا عددا عشوائيا بين 0.0 و 1.0 إختبرنا إن وقع هذا العدد بين 0.0 و 0.1. تأملوا في الرسم أسفله: إذا إخترنا عددا عشوائيا بين 0.0 و 1.0 ففي المعدل يكون هذا العدد في المنطقة بين 0.0 و 0.1 في مرة على عشرة مرات.



- الأسطر 22 إلى 26: عملية بسيطة تتمثل في إسناد عدد للحريف الحالي و تهيئة متغيرة **customerID** للحريف القادم.

يقتنع القارئ بدون شك أن أسلوب الإفتتاح من نوع أساليب التغيير بينما الأسلوبان الآخران من نوع اساليب الإطلاع.

3.4.14 تأليف فئة صف الإنتظار

نستعرض بإيجاز مواصفة صف الإنتظار كما قدمناها في الجزء 2.2.14.

مواصفة فئة صف الإنتظار **intqueue**

أسلوب الإفتتاح	intqueue (int buffersize)
يقع ذكر هذا الأسلوب ضمنيا كلما قمنا بتجسيد هذه الفئة، فيكوّن هذا الأسلوب كائنة من هذه الفئة و يضبط أقصى حجم للصف، أي أقصى عدد حرفاء قد يتواجدون في الصف في نفس الوقت.	
إيداع حريف في ذيل الصف	**void enqueue (int item)**
يقوم هذا الأسلوب بإيداع حريف **(item)** في ذيل الصف. نفترض أن هذا الأسلوب لا يقع ندائه إلا إذا كان الصف يتسع لحريف إضافي.	
نزع حريف من رأس الصف	**int dequeue ()**
يقوم هذا الأسلوب بنزع الحريف الموجود حاليا في رأس الصف. نفترض أن هذا الأسلوب لا يقع ندائه إلا إذا كان الصف غير فارغ.	
حجم الصف	**int size()**
عدد الحرفاء المتواجدين في الصف حاليا.	
هل الصف فارغ؟	**boolean empty ()**
يسترجع هذا الأسلوب قيمة منطقية صحيحة أو غالطة حسب إن كان الصف فارغا أم لا.	

مبدئيا، يمكن إنجاز كل هذه الأساليب بواسطة جدول طويل نشير له بمؤشرين إثنين: مؤشر يشير إلى ذيل الصف، حيث تقع إضافة الحرفاء الواردين، و مؤشر يشير إلى رأس الصف، حيث يقع نزع الحرفاء المغادرين، حسب ما جاء في الرسم التالي:

فضاء نستعمله لاحقا						الذيل	الفضاء المستعمل حاليا						الرأس	فضاء وقع إستعماله سابقا و لن نستعمله مجددا						
…														…	5	4	3	2	1	0

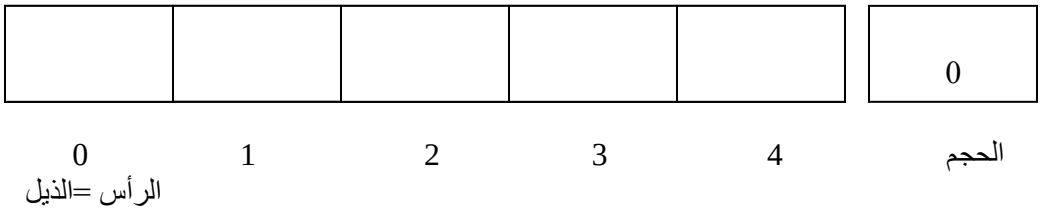
لو كان لدينا جدولا طويلا (لا نهاية له؟) لتمكننا من إنجاز كل الأساليب الموجودة في مواصفة الصف، حيث

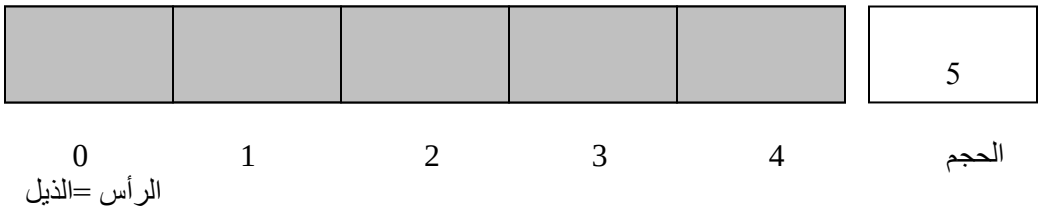
- ننجز عملية الإيداع بوضع العنصر الوارد في الخلية التي يشير لها مؤشر الذيل، ثم تحويل هذا المؤشر.
- ننجز عملية النزع بإستخراج العنصر الذي يشير له مؤشر الرأس، ثم تحيل هذا المؤشر.
- نحتسب حجم الصف حسب الفرق بين المؤشرين.
- نحتسب إن كان الصف فارغا بتقييم حجمه.

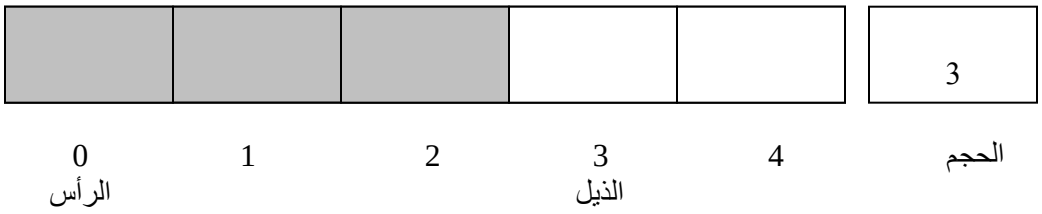
لكن هذا الحل غير مرضي لأنه يتسبب في تبذير مفرط لفضاء الذاكرة، فكل خلية من خلايا الجدول لا يقع إستعمالها إلا مرة واحدة على الأكثر. إذا دامت فترة التجربة مدة طويلة و ورد على الصف ألف حريف، فنحتاج لجدول حجمه ألف على الأقل، حتى لو لم تتجاوز حجم الصف العشرة حرفاء مثلا. نريد حلا يمكننا من إختيار حجم الجدول حسب عدد العناصر المتواجدين في الصف في نفس الوقت، لا حسب عدد العناصر الذين يمرون بالصف في أي وقت من الأوقات. لهذا الغرض، نختار جدولا ذات حجم محدود، فنجعل المؤشران (الذيل و الرأس) يدوران حول الجدول عوضا عن أن يتحولان بصفة خطية مسترسلة كما هو الحال في الرسم أعلاه. نمثل هذا الحل في الرسم الموالي، حيث يكون حجم الجدول 5.

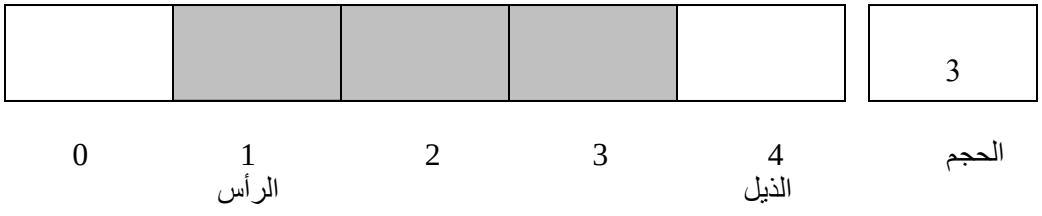
- نستعمل مؤشر الذيل لنشير لعنوان الخلية التي نضع فيها العنصر الوارد المقبل.
- نستعمل مؤشر الرأس لنشير لعنوان أول عنصر موجود في الصف (إذا لم يكن الصف فارغا) سيغادر الصف عندما ننفذ عملية النزع.
- كيف نعرف إذا إن كان الصف فارغا أم لا؟ بواسطة متغيرة إضافية نخزن فيها حجم الصف و نحينه بعد كل عملية.

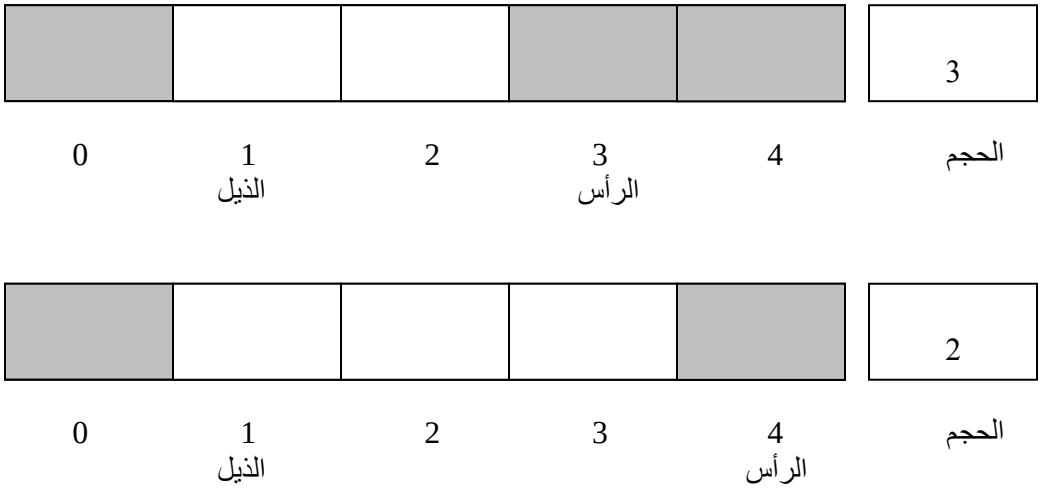
من نتائج هذا النظام الجديد أن مؤشر الرأس يمكن أن يكون أكبر من مؤشر الذيل (عكسا لما جاء في الحل السابق). نبين في الرسم رقم 4.14 عددا من الحالات، من بينها حالات يكون فيها مؤشر الرأس أكبر من مؤشر الذيل. في هذا الرسم، نبين خلايا الجدول التي تحمل عناصر الصف باللون الرمادي.











الرسم رقم 4.14: مختلف حلات الصف

حسب هذا النقاش، يتضح أن حالة الصف تتمثل في الجدول و مؤشري الجدول و حجم الجدول. نقدم فيما يلي نص الفئة التي ألفناها في جافا لتلبية مواصفة الصف.

```
class intqueue                                                  // 1.
{                                                               // 2.
    private int maxqsize;               // buffer size          // 3.
                                                                // 4.
    private int [] carrier;             // holds queue elements // 5.
                                                                // 6.
    private int head, tail;             // head, tail of queue  // 7.
                                                                // 8.
    private int qsize;                  // nbr of elts in queue // 9.
                                                                // 10.
    public intqueue (int buffersize)                            // 11.
    {                                                           // 12.
        head = 0;  tail = 0;                                    // 13.
        qsize = 0;                                              // 14.
        maxqsize = buffersize;                                  // 15.
        carrier = new int [maxqsize];                           // 16.
    }                                                           // 17.
                                                                // 18.
    public void enqueue (int item)                              // 19.
    {                                                           // 20.
        // assumption:  always called when not full             // 21.
        carrier[tail] = item;                                   // 22.
        tail = (tail + 1) % maxqsize;                           // 23.
        qsize++;                                                // 24.
    }                                                           // 25.
                                                                // 26.
    public int dequeue ()                                       // 27.
    {                                                           // 28.
        // assumption: always called when not empty             // 29.
        int val = carrier[head];                                // 30.
        head = (head + 1) % maxqsize;                           // 31.
```

```
        qsize--;                                        // 32.
        return val;                                     // 33.
    }                                                   // 34.
                                                        // 35.
    public int size()                                   // 36.
    {                                                   // 37.
        return qsize;                                   // 38.
    }                                                   // 39.
                                                        // 40.
    public boolean empty ()                             // 41.
    {                                                   // 42.
        return (qsize == 0);                            // 43.
    }                                                   // 44.
                                                        // 45.
    public boolean full()                               // 46.
    {                                                   // 47.
        return (qsize == maxqsize);                     // 48.
    }                                                   // 49.
}  // intqueue                                          // 50.
```

نقدم بعض الملاحضات بخصوص هذه الفئة:

- السطر الثالث: نخزن في **maxqsize** حجم الجدول حسب التنظيم الجديد. يجب أن يتجاوز هذا الحجم أكبر عدد من العناصر التي قد تتواجد في الصف في نفس الوقت.
- الأسطر 11 إلى 17: نسند القيمة الإفتتاحية لكل متغيرات الفئة. عندما يمدنا المستخدم بقيمة **maxqsize** حسب تقديره، عندئذ نتكمن من حجز الفضاء في الذاكرة لجدول الصف.
- السطر 23: إن عملية **tail = (tail + 1) % maxqsize;** تجعل مؤشر الذيل يدور حول الجدول كما بيننا في الرسم 4.14.
- الأسطر 46 إلى 49: لم نستعمل هذا الأسلوب في برنامج المحاكاة. كان من واجبنا أن نتأكد من أن الصف غير ملئان قبل كل عملية إيداع. لم نفعل ذلك، بل إخترنا حجما كبيرا للجدول، لتجنب أن يمتلئ الصف.

4.4.14 تأليف فئة محطة الخدمات

نستعرض بإيجاز مواصفة محطة الخدمات كما قدمناها في الجزء 3.2.14.

مواصفة فئة محطة الخدمات **serviceStation**	
أسلوب الإفتتاح	`serviceStation (int stationRate, int avgneed)`
يقع ذكر هذا الأسلوب ضمنيا كلما قمنا بتجسيد هذه الفئة، فيكوّن هذا الأسلوب كائنة من هذه الفئة و يضبط متغيرتين ذات إهتمام بالنسبة لهذه الفئة: الأولى هي سرعة هذه المحطة، و الثانية هي معدل متطلبات الحرفاء.	
إسناد حريف للمحطة	`void loadCustomer (int custID)`
يقوم هذا الأسلوب بإسناد الحريف **custID** لمحطة الخدمة.	
تقديم الخدمة	`void serve (int time)`
يقوم هذا الأسلوب بتقديم خدمة للحريف الموجود حاليا على محطة الخدمة.	

هل المحطة مستعدة؟	boolean available ()
يسترجع هذا الأسلوب قيمة منطقية صحيحة أو غالطة حسب إن كانت المحطة مستعدة أم لا.	

قبل أن نبادر بتأليف هذه الفئة، نحلل كيف نمثل تقديم خدمات في نطاق برنامج المحاكاة. بالنسبة لبرنامج المحاكاة، لا تهمنا نوعية الخدمة التي تقدمها المحطة للحريف؛ كل ما يهمنا هو مدة الوقت الذي تقضيه المحطة مسخرة للحريف. لكي نضبط طول هذه المدة، يجب أن نطلع على تفصيلين إثنين، و هما: سرعة المحطة؛ و متطلبات الحريف. نمثل سرعة المحطة بقيمة قارة يقع ضبطها عند تنفيذ الأسلوب الإفتتاحي؛ نلقي عليها إسم **servicerate**. أما عن متطلبات الحريف، فيقع ضبطها عشوائيا عند وصول الحريف إلى المحطة، حسب معدل يضبطه المستخدم. نمثل هذا المعدل بقيمة قارة يقع ضبطها عند تنفيذ الأسلوب الإفتتاحي؛ نلقي عليها إسم **serviceneed**. نحتاج لمتغيرتين إضافيتين لمتابعة حالة المحطة:

- أولا، رقم الحريف الذي يتلقى الخدمة حاليا؛ يقع إسناد هذه المتغيرة عندما يصل الحريف إلى المحطة، و يقع إستعمالها عندما يغادر الحريف المحطة.
- ثانيا، المدة الباقية في خدمته؛ يقع إسناد هذه المتغيرة عندما نضبط قيمة الخدمات التي يتطلبها الحريف، و يقع طرحها تدريجيا كل دقيقة. بكم يقع طرحها؟ بقيمة سرعة المحطة. إذا كانت المحطة بطيئة (مثلا: سرعتها 1)، نطرح قيمة ضئيلة (مثلا: 1) كل دقيقة، فيبقى الحريف مدة طويلة في المحطة. و إذا كانت المحطة سريعة (مثلا: سرعتها 6)، نطرح قيمة كبيرة (مثلا: 6) كل دقيقة، فيبقى الحريف مدة قصيرة في المحطة.

بناء على هذه التوضيحات، نقدم نص فئة محطة الخدمات، كما ألفناه في لغة جافا:

```
class serviceStation                                                    // 1.
    {                                                                   // 2.
        private int servicerate;                                        // 3.
                                                                        // 4.
        private int servicetime;                                        // 5.
                                                                        // 6.
        private int beingserved;                                        // 7.
                                                                        // 8.
        private int serviceneed;                                        // 9.
                                                                        // 10.
        private Random generator;                                       // 11.
                                                                        // 12.
        public serviceStation (int stationRate, int avgneed)            // 13.
        {                                                               // 14.
            servicetime = 0;                                            // 15.
            servicerate = stationRate;                                  // 16.
            serviceneed = avgneed;                                      // 17.
            generator = new Random();                                   // 18.
        }                                                               // 19.
                                                                        // 20.
        public void serve (int time)                                    // 21.
        {                                                               // 22.
            int hours = time/60;                                        // 23.
            int minutes = time%60;                                      // 24.
                                                                        // 25.
            if (servicetime>servicerate)                                // 26.
            {                                                           // 27.
                servicetime = servicetime - servicerate;                // 28.
                if (servicetime == 0)                                   // 29.
                {                                                       // 30.
                    //  customer has just finished                    // 31.
                    System.out.println("customer "+beingserved+        // 32.
                            "completed service at time "+               // 33.
                            hours+":"+minutes);                         // 34.
```

```
            }                                                       // 35.
        }                                                           // 36.
        else                                                        // 37.
        {                                                           // 38.
            if (servicetime>0)                                      // 39.
            {                                                       // 40.
                servicetime = 0;                                    // 41.
                System.out.println("customer "+beingserved+         // 42.
                        " completed service at time "               // 43.
                        +hours+":"+minutes);                        // 44.
            }                                                       // 45.
                                                                    // 46.
        }                                                           // 47.
    }                                                               // 48.
                                                                    // 49.
    public void loadCustomer (int custID)                           // 50.
    {                                                               // 51.
        servicetime = generator.nextInt(serviceneed*2)+1;           // 52.
        beingserved = custID;                                       // 53.
    }                                                               // 55.
    public boolean available ()                                     // 56.
    {                                                               // 57.
        return (servicetime ==0);                                   // 58.
    }                                                               // 59.
}                                                                   // 60.
```

نقدم بعض التعليقات على هذا النص:

- السطر 3: سرعة المحطة.
- السطر 5: الوقت الباقي للخدمة الحالية.
- السطر 7: الحريف الحالي.
- السطر 9: معدل متطلبات الحرفاء.
- السطر 11: مولد الأعداد العشوائية.
- الأسطر 16 و 17: تعيين سرعة المجطة و معدل المتطلبات حسب إختيار المستخدم.
- الأسطر 23 و 24: تحويل الوقت من حساب الدقائق إلى حساب الساعات و الدقائق.
- الأسطر 26 إلى 28: تحيين الوقت المتبقي للخدمة الحالية.
- السطر 29: الحريف الحالي أنهى خدمته.
- السطر 39: وقت الخدمة المتبقي أقل من سرعة المحطة.
- السطر 52: إذا أردنا أن يكون المعدل **serviceneed**، فنستخرج أعدادا عشوائية بين 0 و ضعفي هذا العدد.

5.14 مفاهيم عامة في البرمجة الموجهة للكائنات

1.5.14 إخفاء البيانات

من أهم مفاهيم البرمجة الموجهة للكائنات، نذكر مفهوم إخفاء البيانات. إن هذا المفهوم هو في الواقع مبدأ تصميمي ينص على أن كل وحدة من وحدات البرنامج (في لغة جافا، نمثل الوحدات بواسطة الفئات) تشمل ضمنها قرارات تصميمية لا تحتاج الوحدات الأخرى أن تطلع عليه. يمكن للمبرمج في لغة جافا أن يخفي بعض المتغيرات و بعض الأساليب بمجرد أن يصفها بأنها خفية (**private**). أما إذا أراد أن يضع المتغيرات و الأساليب على ذمة المتعاملين مع الفئة، فيصرح بأنها عمومية (**public**). فعادة نمثل حالة الفئة بواسطة متغيرات خفية و لا نسمح لأي فئة أخرى أن تغير حالة المفئة أو أن تطلع عليها إلا عن طريق الأساليب العمومية. لنقتنع من فائدة هذه القاعدة يكفي أن نتخيل ماذا يصير لو سمحنا لأي فئة خارج فئة الصف (أعلاه) أن تغير مؤشري الرأس و الذيل و الحجم و غيرها، بدون المرور عبر الأساليب العمومية.

2.5.14 مواصفة المفئات

علما أن كل فئة تخفي قرارات تصميمها عن الفئات الأخرى، فكيف يمكن للفئات الأخرى أن تتعامل معها؟ يمكن أن نتعامل مع فئة (مثلا: نستعملها) بدون أن نطلع على تصميمها، بمجرد أن نطلع على مواصفتها. هو ليس فقط ممكن بل مفضل، إذ أن المواصفة تعكس الصفات الهامة للفئة بدون أن تشمل تفاصيل ثانوية لا أهمية لها. فبالتالي نعرف كل فئة بواسطة مواصفتها. ما هي فوائد هذا المنهج؟

- يمكن لمنتج الفئة و مستخدمها أن يعملوا بصفة مستقلة، إذ أن المواصفة تقوم مقام عقد بينهما.
- يمكن لمنتج الفئة أن يغير تصميم الفئة بدون أن يضطر لإشعار كل من يستعملها: ما دامت الفئة تلبي مواصفاتها، فلا يهتم المستعملون أن يطلعوا على تصميمها.
- يمكن لمستعمل الفئة أن يغير كيفية إستعماله بدون أن يضطر لإشعار منتجها.

3.5.14 الوراثة

إن مفهوم الوراثة من أهم المفاهيم التي تأيدها البرمجة الموجهة للكائنات. نتأمل في مثال بسيط لتوضيح هذا المفهوم: نهتم بتمثيل طلبة في جامعة فنكون قائمة من المتغيرات لنصف طالب جامعي، و نكون قائمة من الأساليب تعكس العمليات التي قد نقوم بها بخصوص الطالب الجامعي.

- أمثلة في المتغيرات: تعريف الطالب (إسم، لقب، عنوان، حالة مدنية)؛ حالة مهنية (يشتغل كامل الوقت، يشتغل نصف الوقت)؛ برنامج الدراسة؛ مرحلة الدراسة؛ سنة الإنخراط؛ الحالة المالية (هل يتمتع بمنحة، هل يتمتع بإعفاء)، إلخ
- أمثلة في الأساليب: الإنخراط في الجامعة؛ الإنسحاب من الجامعة؛ التسجيل في دروس؛ الإنسحاب من دروس؛ التحصل على أعداد؛ دفع معاليم الدراسة؛ التخرج من الجامعة؛ إلخ.

نعتبر الآن أن هناك عدة أنواع من الطلبة:

- حسب مرحلة الدراسة: طلبة المرحلة الأولى، المرحلة الثانية، و المرحاة الثالثة.
- حسب أسلوب الدراسة: طلبة يتابعون دروسهم في مقام الجامعة، و طلبة يتابعون دروسهم عن بعد (عن طريق العنكبوتية).
- حسب سرعة الدراسة: طلبة يتابعون دروسهم وقتا كاملا، و طلبة يتابعون دروسهم نصف الوقت.
- حسب هوية الطالب: إذا كان الطلب أجنبيا فهو يخضع لقوانين تنص على عدد الدروس التي يمكن أن يتابعها، و على نوعية العمل الذي يمكن له القيام به، و معاليم الدراسة التي يدفعها، إلخ.

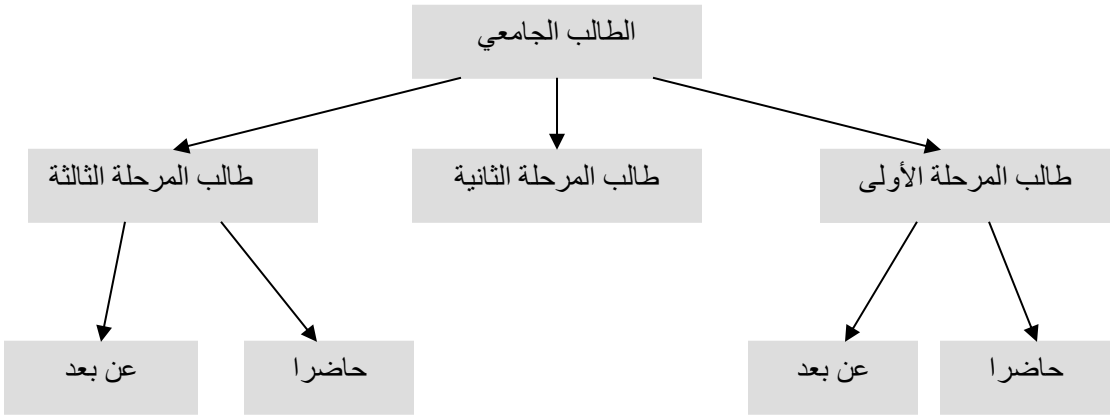
هل هذه الفروق هامة؟ طبعا هي هامة لأن العديد من المتغيرات التي تصف هؤلاء الطلبة و العديد من الأساليب التي ننفذها بخصوص هؤلاء الطلبة تتغير حسب صتف الطالب:

- مثلا: إذا كان الطالب في المرحلة الثالثة، يجب علينا أن نسجل في أي مخبر يتابع بحوثه، من يشرف عن إعماله، هل يتلقى منحة بحث من طرف المخبر، هل يقوم بمهام تعليمية، إلخ.
- أيضا: إذا كان الطالب في المرحلة الثالثة، فتختلف عملية تسجيله في الدروس لأنها تخضع لمراقبة المشرف على أعمال بحثه.

بحيث أن نوعية الطالب لها تأثير على المتغيرات و على الأساليب التابعة لهذا الطالب. عندئذ، كيف نكون فئات تسمح لنا أن نمثل كل هذه الأنواع من الطلبة. تسمح لنا لغة جافا أن نعين فئة كتفرع (أو كفئة فرعية) لفئة أخرى. إذا كانت فئة ف1 فئة فرعية لفئة ف2،

- ترث فئة ف1 كل المتغيرات و الأساليب التابعة لفئة ف2،
- يمكن لفئة ف1 أن تعين متغيرات إضافية خاصة بها،
- يمكن لفئة ف1 أن تعين أساليب إضافية خاصة بها،
- يمكن لفئة ف1 أن تعيد تعيين أساليب في ف2 لتخصصها لمتطلباتها.

في نطاق مثال الطلبة الذي درسناه سابقا، يمثل الرسم 6.14 صورة جزئية من علاقات الوراثة بين فئات الطلبة.



الرسم رقم 6.14: علاقات الوراثة بين فئات الطلبة (صورة جزئية)

توفر لغة جافا إمكانية تعيين فئة كفئة فرعية لفئة أخرى بواسطة وظيفة **extends**، كما يبين النص التالي حيث نعين فئة صف عن طريق عملية وراثة.

```
class intqueue                                                          // 1.
{                                                                       // 2.
    protected int maxqsize;              // buffer size                // 3.
                                                                        // 4.
    protected int [] carrier;            // holds queue elements       // 5.
                                                                        // 6.
    protected int head, tail;            // head and tail of queue     // 7.
                                                                        // 8.
    protected int qsize;                 // nbr of elements in the queue// 9.
                                                                        // 10.
    public intqueue ()                                                  // 11.
    {                                                                   // 12.
        head = 0; tail = 0; qsize = 0;                                  // 13.
        maxqsize = 30;                                                  // 14.
        carrier = new int [maxqsize];                                   // 15.
    }                                                                   // 16.
    public intqueue (int buffersize)                                    // 17.
    {                                                                   // 18.
        head = 0;  tail = 0;                                            // 19.
        qsize = 0;                                                      // 20.
        maxqsize = buffersize;                                          // 21.
        carrier = new int [maxqsize];                                   // 22.
    }                                                                   // 23.
                                                                        // 24.
    public void enqueue (int item)                                      // 25.
    {                                                                   // 26.
        // assumption:  always called when not full                     // 27.
        carrier[tail] = item;                                           // 28.
        tail = (tail + 1) % maxqsize;                                   // 29.
        qsize++;                                                        // 30.
    }                                                                   // 31.
                                                                        // 32.
    public int dequeue ()                                               // 33.
    {                                                                   // 34.
        // assumption: always called when not empty                     // 35.
        int val = carrier[head];                                        // 36.
        head = (head + 1) % maxqsize;                                   // 37.
```

```
        qsize--;                                                    // 38.
        return val;                                                 // 39.
    }                                                               // 40.
                                                                    // 41.
    public int size()                                               // 42.
    {                                                               // 43.
        return qsize;                                               // 44.
    }                                                               // 45.
                                                                    // 46.
    public boolean empty ()                                         // 47.
    {                                                               // 48.
        return (qsize == 0);                                        // 49.
    }                                                               // 50.
                                                                    // 51.
    public boolean full()                                           // 52.
    {                                                               // 53.
        return (qsize == maxqsize);                                 // 54.
    }                                                               // 55.
    }  // intqueue                                                  // 56.
                                                                    // 57.
class newqueuecls extends intqueue                                  // 58.
{                                                                   // 59.
    public int dequeue ()                                           // 60.
    {                                                               // 61.
        // assumption: always called when not empty                 // 62.
        return (carrier[head]);                                     // 63.
    }                                                               // 64.
                                                                    // 65.
    public void update ()                                           // 66.
    {                                                               // 67.
        head = (head + 1) % maxqsize;                               // 68.
        qsize--;                                                    // 69.
    }                                                               // 70.
 }                                                                  // 71.
```

نقدم بعض التعليقات حول هذا النص الذي يبين الفئة الأصلية (**intqueue**) و الفئة الفرعية (**newqueuecls**).

- الأسطر 3، 5، 7، و 9: بحكم عملية الوراثة، يجب أن نعطي لفئة **newqueuecls** إمكانية إستعمال المتغيرات الموجودة ضمن فئة **intqueue**، لكن هذا مستحيل ما دامت هذه المتغيرات مرفوقة بكلمة ، **private**، التي تمنع إستعمال هذه المتغيرات لكل الفئات الخارجية. فنعوض كلمة **private** بكلمة **protected** التي تمنع الإتصال بهذه المتغيرات على كل الفئات ما عدا الفئات الفرعية لهذه الفئة.
- الأسطر 11 إلى 15: تتطلب الوراثة أن تكون الفئة الأصلية لها أسلوب إفتتاحي بدون معلمات، يقع ندائه كلما وقع تجسيد كائنة من الفئة الفرعية. فأضفنا هذا الأسلوب الإفتتاحي، حيث نسند لحجم الجدول قيمة 30.
- الأسطر 17 إلى 56 تبقى على حالها.
- السطر 58: نصرح أن فئة **newqueuecls** هي فئة فرعية لفئة **intqueue**.
- الأسطر 60 إلى 64: نعيد تعيين أسلوب نزع العناصر من الصف، فنجعله يسترجع العنصر في رأس الصف لكن لا يغير الصف، فيصبح هذا الأسلوب أسلوب إطلاع بحت. كلما نادينا أسلوب **dequeue** على كائنة من فئة **newqueuecls**، يقع تنفيذ هذا الأسلوب عوضا عن الأسلوب الأصلي الموجود في الأسطر 33 إلى 40.
- الأسطر 66 إلى 70: تمكننا عملية الوراثة من تعيين أساليب خاصة بالفئة الفرعية ليست موجودة في الفئة الأصلية؛ هذا ما نقوم به في هذه الأسطر.

نبين في النص التالي كيف نغير البرنامج الرئيسي لكي نستعمل الفئة الفرعية عوضا عن الفئة الأصلية:

```
public static void main(String[] args) {                            // 1.
        // TODO code application logic here                         // 2.
                                                                    // 3.
        // work variables                                           // 4.
```

```
                                                                                    // 5.
    int customerid;                                                                 // 6.
    int hours, minutes;                                                             // 7.
                                                                                    // 8.
    // global variables                                                             // 9.
                                                                                    // 10.
    // int time;                                                                    // 11.
    int opening;                                                                    // 12.
    int closing;                                                                    // 13.
                                                                                    // 14.
    customerflow cflow;                                                             // 15.
    newqueuecls newqueue;                                                           // 16.
    serviceStation servestation;                                                    // 17.
                                                                                    // 18.
    //  initializations                                                             // 19.
                                                                                    // 20.
    opening = 540;                                                                  // 21.
    closing = 1020;                                                                 // 22.
    cflow = new customerflow(5);                                                    // 23.
    newqueue = new newqueuecls ();                                                  // 24.
    servestation = new serviceStation(2,10);                                        // 25.
                                                                                    // 26.
    // main loop                                                                    // 27.
                                                                                    // 28.
    for (int time =opening; time<(closing); time++)                                 // 29.
    {                                                                               // 30.
        hours = time / 60;                                                          // 31.
        minutes = time % 60;                                                        // 32.
        if (cflow.newcustomer())                                                    // 33.
        {                                                                           // 34.
            customerid = cflow.generatecustomer();                                  // 35.
            newqueue.enqueue(customerid);                                           // 36.
            System.out.println("customer "+customerid+" arrived at time "+      // 37.
                    hours+":"+minutes );                                            // 38.
        }                                                                           // 39.
        servestation.serve(time);                                                   // 40.
        if (!(newqueue.empty()) && servestation.available())                        // 41.
        {                                                                           // 42.
            customerid = newqueue.dequeue();                                        // 43.
            newqueue.update();                                                      // 44.
            servestation.loadCustomer(customerid);                                  // 45.
            System.out.println("customer "+customerid+" served at time "+       // 46.
                    hours+":"+minutes);                                             // 47.
        }                                                                           // 48.
    }                                                                               // 49.
    System.out.println("Queue Length at Closing: "+newqueue.size());               // 50.
    }                                                                               // 51.
}                                                                                   // 52.
```

نقدم بعض التعليقات الخاصة بالتغييرات عن البرنامج الرئيسي الأصلي:

- السطر 16: نصرح بصف من النوع الجديد (الفئة الفرعية للصف الأصلي).
- السطر 24: تنفيذ الأسلوب الإفتتاحي التابع للفئة الأصلية.
- السطر 36: أسلوب الإيداع لم يتغير.
- سطري 43 و 44: هنا يقع تنفيذ الأساليب التابعة للفئة الفرعية.

نقدم فيما يلي الأسطر الأخيرة الناتجة عن تنفيذ هذا البرنامج:

```
customer 80 completed service at time 16:50
customer 81 served at time 16:50
customer 81 completed service at time 16:54
customer 82 served at time 16:54
customer 82 completed service at time 16:55
customer 83 served at time 16:55
customer 86 arrived at time 16:58
Queue Length at Closing: 3
```

```
BUILD SUCCESSFUL (total time: 4 seconds)
```

6.14 تمارين

1 [ص] غيروا برنامج المحاكاة الذي درسناه في هذا الفصل لينمذج حركة الحرفاء في مركز خدمات ذات محطتين للخدمات. إستعملوا صفا واحدا يزود المحطتين.

2 [ص] غيروا برنامج المحاكاة الذي درسناه في هذا الفصل لينمذج حركة الحرفاء في مركز خدمات ذات محطتين للخدمات. إستعملوا صفين، كل صف يزود محطة. يقع إسناد الحرفاء الواردين إلى أقصر صف.

3 [ص] غيروا برنامج المحاكاة الذي درسناه في هذا الفصل لينمذج حركة الحرفاء في مركز خدمات ذات محطتين للخدمات. إستعملوا صفا واحدا يزود المحطتين. المطلوب منكم إحتساب معدل وقت الإنتظار.

4 [ص] غيروا برنامج المحاكاة الذي درسناه في هذا الفصل لينمذج حركة الحرفاء في مركز خدمات ذات محطتين للخدمات. إستعملوا صفين، كل صف يزود محطة. يقع إسناد الحرفاء الواردين إلى أقصر صف. المطلوب منكم إحتساب معدل وقت الإنتظار.

5 [م] المطلوب منكم تعويض أسلوب النزع في فئة الصف (الجزء 2.2.14) بأسلوبين، أحدهما أسلوب الإطلاع و الآخر أسلوب التغيير. ثم المطلوب منكم مراجعة برنامج المحاكة مستعملا الفئة الجديدة.

6 [م] في الجزء **3.4.14**، إستعملنا مؤشران (الرأس و الذيل) يدوران حول الجدول، فأضفنا لهما متغيرة تعكس عدد العناصر الموجودة في الصف. قد يكون هناك حل آخر يتمثل في أن المؤشران ينزادان دوما مع مرور العمليات (الذيل مع الإيداع و الرأس مع النزع) مع أننا لا نستعملهما مباشرة على الجدول بل نطبق عليهما عملية الباقي (%) فنكتب مثلا

```
carrier [tail % maxqsize] = item;
```

المطلوب منكم دراسة هذا الحل و محاولة تنفيذه.

7 [س] ما هو الغرق بين علاقة الوراثة و علاقة التجسيد. حللوا كل الفروق.

.

Towers of Hanoi	أبراج هانوي
Apple	آبل
Apple I	آبل الأول
Apple II	آبل الثاني
Range of representation	إتساع التمثيل
Coordonnees	إحداثيات
X coordinates	الإحداثية الأفقية
Y coordinates	الإحداثية العمودية
Program testing	إختبار البرنامج
Information hiding	إخفاء البيانات
Ada	آدا
ARPANET	آربانت
Software Crisis	أزمة البرمجيات
Reporting methods	اساليب الإطلاع
Updating methods	أساليب التغيير
Numerical Methods	الأساليب العددية
Object methods	أساليب الكائنات
Reporting methods	أساليب الكشف
Software Reuse	إستعمالية البرمجيات
Induction on multiple dimensions	الإستقراء المتعدد الأبعاد
Induction on two dimensions	الإستقراء على بعدين
Induction on data structures	الإستقراء على هياكل البيانات
Method (of a class)	أسلوب (فئة)
Name (expression)	إسم
Assignment (programming)	الإسناد
Sort trees	أشجار الترتيب
Binary trees	الأشجار الثنائية
Regulat binary trees	الأشجار الثنائية القانونية
Expression trees	أشجار العبارات
Syntax trees	الأشجار النحوية
Output	الإصدار
Insertion	الإضافة
Frames	الإطارات
Programming phases	أطوار البرمجة
Prime numbers	الأعداد الأولية
Real numbers	الأعداد الحقيقية
Double real number	الأعداد المثنات
Bernouilli Numbers	أعداد برنويي
Floating point number	الأعداد ذات النقطة العائمة
Fibonacci numbers	أعداد فيبوناشي
Hard Disks	الأقراص الصلبة،
Parentheses	الأقواس العادية

Square brackets	الأقواس المستقيمة
Curly brackets	الأقواس المنعرجة
Greatest common divisor	أكبر قاسم مشترك
Actuators	آلات التحكم
Sensors	آلات الحس
Alan Turing	آلان تورنغ
Actuator	آلة التحكم
Sensor	ألة الحسّ
Printer	آلة الطبع
Camera	آلة تصوير
Turing Machine	آلة تورنغ
Calculator	آلة حساب
Algol (Algorithmic Language)	آلغول
Robot	الآليات
America On Line (aol)	أمريكا أنلاين
NSF Net	أن أس أف نات
Vacuum tubes	أنابيب الفراغ
Vacuum tube	أنبوب فراغ
Resistor	أنبوب مقاومة
Intel	إنتال
Download	إنزال
Execution launch	إنطلاق التنفيذ
Compiler launch	إنطلاق المصرف
Operating Systems	أنظمة الإستخدام
Management Information Systems	الأنظمة التصرفية للمعلومات
Information Systems	أنظمة المعلومات
Enquire	إنكواير
Formats	الأنماط
Data types	أنواع البيانات
Object Attributes	أوصاف الكائنات
Augusta Ada Byron	أوغسطا آدا بايرون
IBM	أي بي أم
IBM 360	أي بي أم 360
IBM 7090	آي بي آم 7090
IBM PC	أي بي أم بي سي
Ethernet	إيثرنت
ENIAC	إينياك
Iowa State	آيُوا ستات
Pascaline	باسكالين
Packages	باقات
Palm Pilot	بالم بايلت
Powerbook 100	باوربوك 100
Presper Eckert	براسبر آكارت
Recursive programming	البرمجة الإستقرائية
Graphics programming	البرمجة الرسومية

Logic Programming	البرمجة المنطقية
Object Oriented Programming	البرمجة الموجهة للكائنات
Object oriented programming	البرمجة الموجهة للكائنات
Interactive program	برنامج تفاعلي
Iterative interactive program	برنامج تفاعلي تكراري
Iterative program	برنامج تكراري
Simulation program	برنامج محاكاة
Blaise Pascal	بلاز باسكال
Plan Kalkul	بلان كالكول
Paul Allen	بول آلان
Data	البيانات
Indexing	التأشير
Linking	تأليف الروابط
Data Abstraction	تجرد البيانات
Data abstraction	تجرد البيانات
Data abstraction	تجرد البيانات
Control abstraction	تجرد التعليمات
Instantiation	تجسيد
Left associativity	التجمع على اليسار
Specifying the range	تحديد الإتساع
Specifying the precision	تحديد الدقة
Programming	تحرير البرنامج
Numerical Analysis	التحليل العددي
Type conversion (programming)	تحويل الأنواع
Data processing	تحويل البيانات
Information processing	تحويل المعلومات
Transistor	ترانزيزتور
Operator priority	ترتيب الأولويات
Lexicographic ordering	الترتيب القاموصي
Linear list	التسلسل
Linked list	التسلسل
Program implementation	تسيير البرنامج
Initial configuration	التشكيلة الإفتتاحية
Declaration	التصريح
Declaration	تصريح
Private declarations (in Java)	التصريحات الخفية
Public declarations (in Java)	التصريحات المكشوفة
Internal product (of vectors)	التصريف العددي
Product of a matrix with a vector	تصريف المصفوفة بالمتوجه
Matrix product	تصريف مصفوفة بمصفوفة
Program design	تصميم البرنامج
Logic Design	التصميم المنطقي
Roundoff cumulation	تضاعف التقريب
Administrative applications	التطبيقات الإدارية
Educational applications	التطبيقات التربوية

Business Applications	تطبيقات التصرف
Industrial Applications	التطبيقات الصناعية
Scientific applications	التطبيقات العلمية
Artistic applications	التطبيقات الفنية
Financial Applications	التطبيقات المالية
Program evolution and maintenance	تطوير و صيانة البرنامج
Output statements	تعليمات الإصدار
Input statements	تعليمات التوريد
Alternation statement	تعليمات الخيار
Nested statements	التعليمات المتداخلة
Switch statement	تعليمة التوجيه
Switch statement	تعليمة التوزيع
Alternation statement	تعليمة الخيار
Conditional statement	تعليمة الشرط
Strict order	التفاوت الشديد
Strict ordering	التفاوت الشديد
Loose ordering	التفاوت اللين
Loose ordering	التفاوت المنحل
Information Technology	تقنيات المعلومات
Iteration	تكرار
For loop	التكرار المعدد
While statement (Java)	التكرار ذات الشرط السابق
Do-while statement (Java)	التكرار ذات الشرط اللاحق
Program Execution	تنفيذ البرنامج
Execution setup	تهيئة التنفيذ
Turn It In	تورنت إين
Import	التوريد
Input	التوريد
Input output	التوريد و الإصدار
TCP/IP	تي سي بي / أي بي
Tim Berners Lee	تيم بارنارس لي
Think Pad 700	ثنكباد 700
Java	جافا
Algebra	الجبر
Arrays	الجداول
Linear arrays	الجداول الخطية
One dimensional arrays	الجداول ذات البعد الواحد
Two dimensional arrays	الجداول ذات بعدين
Three dimensional arrays	جداول ذات ثلاثة أبعاد
Body of rule (BNF grammars)	جسد القاعدة
Mantissa	جسم (العدد)
Loop body	جسم المدار
Garbage collection	جمع الفواضل
Printer	جهاز الطبع

Joseph Marie Jackard	جوسف ماري جاكارد
John von Neumann	جون فون نويمان
John Atanasoff	جوهن آتاناسوف
John Mauchly	جوهن موشلي
John Napier	جوهن نابيير
Sine	الجيب
Sentinel	حارس
Computer	حاسوب
Personal Computer	الحاسوب الشخصي
Central computer, mainframe	الحاسوب المركزي
Default case	الحالة المهملة
Containers	الحاويات
Characters	الحروف
Fifth Generation Computers	حواسيب الجيل الخامس
Embedded computers	الحواسيب المتنقلة
Doundoff error	خطأ تقريب
Private (in Java)	خفية
Algorithms	الخوارزميات
Simulation algorithm	خوارزمية المحاكاة
disjunction	الخيار
Disjunction / logical or	الخيار المنطقي
Exponent	داعية (العدد)
DEC (Digital Equipment Corporation)	داك
Induction step	درجة الإستقراء
Precision of representation	دقة التمثيل
Semantics	الدلالة
Artificial Intelligence	الذكاء الإصطناعي
Artificial Intelligence	الذكاء الإصطناعي
Underflow	الذوبان
Head of rule (BNF grammars)	رأس القاعدة
Loader	رافع البرامج
Raytheon	رايثيون
Loading	رفع البرنامج
Grammar symbol	الرمز النحوي
Applied Mathematics	الرياضيات التطبيقية
Xerox Parc	زيروكس بارك
Saturn V	ساترن الخامس
Steven Jobs	ستيفن جوبس
Steven Wosniak	ستيفن فوسنياك
Buffer	السجل الوسيط
String	سلسة الحروف
String	سلسلة
Substring	سلسلة جزئية
Character string	سلسلة حروف

C	سي
C++	سي++
Silicon	السيليكون
Seymour Cray	سيمور كراي
Charles Babbage	شارل بابّج
Touch Screen	الشاشات الحسّاسة
Monitor	الشاشة
Screen	الشاشة
Hypertext	شبكة النصوص
Left subtree	الشجرة الجزئية اليسارية
Right subtree	الشجرة الجزئية اليمينية
Expression tree	شجرة العبارة
Object Tree	شجرة كائنات
Loop condition	شرط المدار
Stream	صادر
True	صحيح
true	صحيح
Program Compilation	صرف البرنامج
Waiting queues	صفوف الإنتظار
Analysis and capture of program specifications	ضبط و تحليل مواصفات البرنامج
Inclusive	ضمنا
Facteurs	ضوارب
Term	طرف
Tree traversal	الطواف على الشجرة
Run time	طور التنفيذ
Compile time	طور الصرف
Factor	عامل
Expressions	العبارات
Logical expressions	العبارات المنطقية
Unary logical expressions	العبارات المنطقية الفردية
Expression	عبارة
Case expression	عبارة البديل
Switch expression	عبارة التوجيه
Conditional expression	العبارة المشترطة
Transistor	العبورية
Literal (expression)	عدد
Prime number	عدد أولي
Floating point number	عدد حقيقي عائم
Double	عدد حقيقي مثنى
Integer	عدد كامل
Long integer	عدد كامل طويل
Short integer	عدد كامل قصير
Transistor	عدسة العبور
Conjunction	العطف

Conjunction / logical and	العطف المنطقي
Labels	العلامات
Computer science	علم الحاسوب
Computing Science	علم الحساب الآلي
Concatenation	عملية ربط
Public (in Java)	عمومية
JFrame (in Java)	عنصر الإطار
ImageIcon (in Java)	عنصر الصورة الرمزية
JLabel (in Java)	عنصر العلامة
JPanel (in Java)	عنصر اللوحة
Neutral element	العنصر المحايد
World Wide Web	العنكبوتية العالمية
Factorial	العوملة
False	غالط
False	غالط
Gotfried Wilhelm von Leibnitz	غوتفريد ويلهالم فون لايبنيتز
Gopher	غوفر
Classes	الفئات
Wrapper Classes	فئات التغليف
Generic class	الفئات العامة
Class	فئة
String class	فئة سلسة الحروف
Mouse	الفأرة
Content Pane	فضاء المحتوى
Fortran (FORmula TRANslation)	فورطران
Overflow	الفيضان
Object List	قائمة كائنات
File Reader (Java)	قارءة الملف
Basis of induction	قاعدة الإستقراء
Floppy Disk	القرص اللين
Compact Disk	القرص المدمج
Sub Class	قسم جزئي
Integer division	القسمة الكاملة
Databases	قواعد البيانات
Boolean values	القيمات المنطقية
Initial value	القيمة الإفتتاحية
Boolean	قيمة منطقية
Objects	الكائنات
Objects	الكائنات
Object	كائنة
Pront Writer (Java)	كاتبات الطبع
File Writer (Java)	كاتبة الملف
Ken Thomson	كان طومسن
Density of representation	كثافة التمثيل
Cray I	كراي الأول

Mouse	الكرة المتجولة
Clifford Berry	كلفورد باري
Konrad Zuse	كُنراد زوس
Cobol (Common Business Oriented Language)	كوبول
Stack	الكومة
Non-inclusive	لاضمنا
Inequality	اللامساواة
Inequality	اللامساواة
Automatic Formats	لأنماط الآلية
Programming Languages	لغات البرمجة
Panels	اللوحات
Mother Board	اللوحة الأم
Control panel (in Windows)	لوحة التحكم
Register Board	لوحة السجلات
Electronic Board	لوحة إلكترونية
Keyboard	لوحة المفاتيح
Keyboard	لوحة المفاتيح
Los Alamos	لوس آلاموس
Logarithm	اللوغاريثما
LISP (LISt Processing)	ليسب
Index	مؤشر
Formatter (in Java)	مؤلف الأنماط
Linker	مؤلف الروابط
Interpreters	مؤولات
Mark I	مارك الأول
Microsoft	مايكروسوفت
Musee des Arts et Metiers	متحف التقنيات و المهن
Variable	المتغيرات
Variables	المتغيرات
Iteration variable	متغيرة التكرار
Vectors	المتوجهات
Row vectors	المتوجهات الأفقية
Column vectors	المتوجهات العمودية
Rectangular triangle	المثلث المستطيل
Plain triangle	مثلث عادي
Isoceles triangle	مثلث عادي ذات جانبين متساويين
Equilateral triangle	مثلث متساوي الجوانب
Rectangular triangle	مثلث مستقيم عادي
Rectangular isoceles triangle	مثلث مستقيم متناسق
Simulation	محاكاة
Analytical Engine	محرك التحليل
Difference Engine	محرك الطرح
Central Memory	مخزن مركزي
Graphs	المخططات

Syntactic diagrams	المخططات النحوية
Limited degree graphs	مخططات ذات الدرجات المحدودة
Dense graphs	مخططات كثيفة
Loop	المدار
Integrated circuit	المدار المدمج
For loop	المدار المعدد
Infinite loop	مدار لا منتهي
Microchip	المدارات الدقيقة
Transport Control Protocol/ Internet Protocol	مراسم مراقبة التنقل / مراسم الإنترنت
Cosine	مرافق الجيب
Type checking	مراقبة الأنواع
Calculating rule	مساطر الحساب
Equality	المساواة
Equality. Equivalence	المساواة
Logical equivalence	المساواة المنطقية
Constant (in programming)	المستقرات
Constants	المستقرات
Compiler	المصرف
Java compiler	مصرّف جافا
Compilers	مصرفات
Matrices	المصفوفات
Speaker	مضخم الصوت
Microprocessor	المعالج الدقيق
Microprocessor	المعالج الصغير
Text editor	معالج النصوص
Data processing	معالجة البيانات
Central Processing	المعالجة المركزية
Information processing	معالجة المعلومات
Abacus	المعداد
Identifier	المعرّف
Identifier	المعرّف
Knowledge	المعرفة
Data	المعطيات
Parameters	معلمات
Actual parameters	المعلمات الحقيقية
Formal parameters	المعلمات الشكلية
Information	المعلومات
Informatics	المعلوماتية
Computer Architecture	معمارية الحاسوب
Computer Architecture	معمارية الحاسوب
Von Neumann Architecture	معمارية فون نويمان
Von Neumann Architecture	معمارية فون نويمان،
MIT (Massachusetts Institute of Technology)	المعهد التقني لماساشوستس

Method	مفهوم الأسلوب
Parameters	مفهوم المعلمات
Instantiation relation	مفهوم علاقة الإنتماء
Has Part	مفهوم علاقة التجزأ
Is-A relation	مفهوم علاقة التفاوت
Lexicographic comparison	مقارنة قاموصية
Components	المكونات
Actuator Adapter	ملائم آلة التحكم
Sensor Adapter	ملائم آلة الحس
Printer Adapter	ملائم آلة الطبع
Monitor Adapter	ملائم الشاشة
Network Adapter	ملائم العنكبوتية
Mouse Adapter	ملائم الفأرة
Keyboard Adapter	ملائم لوحة المفاتيح
Files	الملفات
Discriminant	مميز
Events Area	منطقة الأحداث
Edit Area	منطقة التحرير
Detail Area	منطقة التفاصيل
Virtual World Area	منطقة العالم الخيالي
Object Tree Area	منطقة شجرة الكائنات
Program specifications	مواصفات البرنامج
Class Specification	مواصفة الفئة
Pointer	موجّه
Application Domains	ميادين التطبيق
Scanner	الناسخ
Scanner	ناسخ
Keyboard scanner	ناسخة لوحة المفاتيح
Netscape	نتسكاب
Execution outcome	نتيجة التنفيذ
Syntax	النحو
(procedure) Call	نداء
Deletion	النزع
DOS (Disk Operating System)	نظام دوس
Theory of Computing	نظرية الحساب
Negation	النفي
Logical negation	النفي المنطقي
Data type	نوع البيانات
Niklaus Wirth	نيكلاوس ويرث
Harvard	هارفرد
Software Engineering	هندسة البرمجيات
Software engineering	هندسة البرمجيات
Data Engineering	هندسة البيانات
Information Engineering	هندسة المعلومات
Data Structures	هياكل البيانات

Bus	هياكل التنقل
Linear structures	الهياكل الخطية
Webboard	واب بورد
Webct	وابسيتي
Human Computer Interfaces	واجهات التعامل بين الحاسوب و المستخدم
Watt	واطّ
Input/ Output Devices	وحدات الإخراج و الإدخال
Input/ Output Devices	وحدات الإصدار و التوريد،
External memory units	وحدات ذاكرة خارجية
Control Unit	وحدة التحكم
Arithmetic and Logical Unit	وحدة الحساب العددي و المنطقي
Peripheral Memory	وحدة الذاكرة الفرعية
Central Memory	وحدة ذاكرة مركزية
Central Processing Unit	وحدة معالجة مركزية،
Inheritance	الوراثة
Buffered Reader (Java)	وسيطة القراءة
Buffered Writer (Java)	وسيطة الكتابة
Recursive functions	الوضائف الإستقرائية
Functions with side effects	وضائف ذات مفعول جانبي
Pure functions	وضائف صافية
Object functions	وظائف الكائنات
Square function	وظيفة المربع
Fibonacci function	وظيفة فيبوناشي
Windows 1995	وندوس 1995
Windows 1998	وندوس 1998
Windows 2000	وندوس 2000
Windows XP	وندوس أكس بي
Windows Vista	وندوس فيستا
William Oughtred	ويليام أوتراد
William Gates	ويليام غاتس
Yahoo	ياهو
Unix	يونكس

أنجليزي عربي

Abacus	المعداد
Actual parameters	المعلمات الحقيقية
Actuator	آلة التحكم
Actuator Adapter	ملائم آلة التحكم
Actuators	آلات التحكم
Ada	آدا
Administrative applications	التطبيقات الإدارية
Alan Turing	آلان تورنغ
Algebra	الجبر
Algol (Algorithmic Language)	آلغول
Algorithms	الخوارزميات
Alternation statement	تعليمات الخيار
Alternation statement	تعليمة الخيار
America On Line (aol)	أمريكا أنلاين
Analysis and capture of program specifications	ضبط و تحليل مواصفات البرنامج
Analytical Engine	محرك التحليل
Apple	آبل
Apple I	آبل الأول
Apple II	آبل الثاني
Application Domains	ميادين التطبيق
Applied Mathematics	الرياضيات التطبيقية
Arithmetic and Logical Unit	وحدة الحساب العددي و المنطقي
ARPANET	آربانت
Arrays	الجداول
Artificial Intelligence	الذكاء الإصطناعي
Artificial Intelligence	الذكاء الإصطناعي
Artistic applications	التطبيقات الفنية
Assignment (programming)	الإسناد
Augusta Ada Byron	أوغسطا آدا بايرون
Automatic Formats	لأنماط الآلية
Basis of induction	قاعدة الإستقراء
Bernouilli Numbers	أعداد برنويي
Binary trees	الأشجار الثنائية
Blaise Pascal	بلاز باسكال
Body of rule (BNF grammars)	جسد القاعدة
Boolean	قيمة منطقية
Boolean values	القيمات المنطقية
Buffer	السجل الوسيط
Buffered Reader (Java)	وسيطة القراءة
Buffered Writer (Java)	وسيطة الكتابة
Bus	هياكل التنقل

Business Applications	تطبيقات التصرف
C	سي
C++	سي++
Calculating rule	مساطر الحساب
Calculator	آلة حساب
Call (procedure)	نداء
Camera	آلة تصوير
Case expression	عبارة البديل
Central computer, mainframe	الحاسوب المركزي
Central Memory	مخزن مركزي
Central Memory	وحدة ذاكرة مركزية
Central Processing	المعالجة المركزية
Central Processing Unit	وحدة معالجة مركزية،
Character string	سلسلة حروف
Characters	الحروف
Charles Babbage	شارل بابّج
Class	فئة
Class Specification	مواصفة الفئة
Classes	الفئات
Clifford Berry	كلفورد باري
Cobol (Common Business Oriented Language)	كوبول
Column vectors	المتوجهات العمودية
Compact Disk	القرص المدمج
Compile time	طور الصرف
Compiler	المصرف
Compiler launch	إنطلاق المصرف
Compilers	مصرفات
Components	المكونات
Computer	حاسوب
Computer Architecture	معمارية الحاسوب
Computer Architecture	معمارية الحاسوب
Computer science	علم الحاسوب
Computing Science	علم الحساب الآلي
Concatenation	عملية ربط
Conditional expression	العبارة المشترطة
Conditional statement	تعليمة الشرط
Conjunction	العطف
Conjunction / logical and	العطف المنطقي
Constant (in programming)	المستقرات
Constants	المستقرات
Containers	الحاويات
Content Pane	فضاء المحتوى
Control abstraction	تجرد التعليمات
Control panel (in Windows)	لوحة التحكم

Control Unit	وحدة التحكم
Coordonnees	إحداثيات
Cosine	مرافق الجيب
Cray I	كراي الأول
Curly brackets	الأقواس المنعرجة
Data	البيانات
Data	المعطيات
Data Abstraction	تجرد البيانات
Data abstraction	تجرد البيانات
Data abstraction	تجرد البيانات
Data Engineering	هندسة البيانات
Data processing	تحويل البيانات
Data processing	معالجة البيانات
Data Structures	هياكل البيانات
Data type	نوع البيانات
Data types	أنواع البيانات
Databases	قواعد البيانات
DEC (Digital Equipment Corporation)	داك
Declaration	التصريح
Declaration	تصريح
Default case	الحالة المهملة
Deletion	النزع
Dense graphs	مخططات كثيفة
Density of representation	كثافة التمثيل
Detail Area	منطقة التفاصيل
Difference Engine	محرك الطرح
Discriminant	مميز
disjunction	الخيار
Disjunction / logical or	الخيار المنطقي
DOS (Disk Operating System)	نظام دوس
Double	عدد حقيقي مثنى
Double real number	الأعداد المثنات
Doundoff error	خطأ تقريب
Do-while statement (Java)	التكرار ذات الشرط اللاحق
Download	إنزال
Edit Area	منطقة التحرير
Educational applications	التطبيقات التربوية
Electronic Board	لوحة إلكترونية
Embedded computers	الحواسيب المتنقلة
ENIAC	إينياك
Enquire	إنكواير
Equality	المساواة
Equality. Equivalence	المساواة
Equilateral triangle	مثلث متساوي الجوانب

Ethernet	إيثرنت
Events Area	منطقة الأحداث
Execution launch	إنطلاق التنفيذ
Execution outcome	نتيجة التنفيذ
Execution setup	تهيئة التنفيذ
Exponent	داعية (العدد)
Expression	عبارة
Expression tree	شجرة العبارة
Expression trees	أشجار العبارات
Expressions	العبارات
External memory units	وحدات ذاكرة خارجية
Facteurs	ضوارب
Factor	عامل
Factorial	العوملة
False	غالط
False	غالط
Fibonacci function	وظيفة فيبوناشي
Fibonacci numbers	أعداد فيبوناشي
Fifth Generation Computers	حواسيب الجيل الخامس
File Reader (Java)	قارءة الملف
File Writer (Java)	كاتبة الملف
Files	الملفات
Financial Applications	التطبيقات المالية
Floating point number	الأعداد ذات النقطة العائمة
Floating point number	عدد حقيقي عائم
Floppy Disk	القرص اللين
For loop	التكرار المعدد
For loop	المدار المعدد
Formal parameters	المعلمات الشكلية
Formats	الأنماط
Formatter (in Java)	مؤلف الأنماط
Fortran (FORmula TRANslation)	فورطران
Frames	الإطارات
Functions with side effects	وضائف ذات مفعول جانبي
Garbage collection	جمع الفواضل
Generic class	الفئات العامة
Gopher	غوفر
Gotfried Wilhelm von Leibnitz	غوتفريد ويلهالم فون لايبنيتز
Grammar symbol	الرمز النحوي
Graphics programming	البرمجة الرسومية
Graphs	المخططات
Greatest common divisor	أكبر قاسم مشترك
Hard Disks	الأقراص الصلبة،
Harvard	هارفرد
Has Part	مفهوم علاقة التجزأ

Head of rule (BNF grammars)	رأس القاعدة
Human Computer Interfaces	واجهات التعامل بين الحاسوب و المستخدم
Hypertext	شبكة النصوص
IBM	أي بي أم
IBM 360	أي بي أم 360
IBM 7090	آي بي آم 7090
IBM PC	أي بي أم بي سي
Identifier	المعرّف
Identifier	المعرّف
ImageIcon (in Java)	عنصر الصورة الرمزية
Import	التوريد
Inclusive	ضمنا
Index	مؤشر
Indexing	التأشير
Induction on data structures	الإستقراء على هياكل البيانات
Induction on multiple dimensions	الإستقراء المتعدد الأبعاد
Induction on two dimensions	الإستقراء على بعدين
Induction step	درجة الإستقراء
Industrial Applications	التطبيقات الصناعية
Inequality	اللامساواة
Inequality	اللامساواة
Infinite loop	مدار لا منتهي
Informatics	المعلوماتية
Information	المعلومات
Information Engineering	هندسة المعلومات
Information hiding	إخفاء البيانات
Information processing	تحويل المعلومات
Information processing	معالجة المعلومات
Information Systems	أنظمة المعلومات
Information Technology	تقنيات المعلومات
Inheritance	الوراثة
Initial configuration	التشكيلة الإفتتاحية
Initial value	القيمة الإفتتاحية
Input	التوريد
Input output	التوريد و الإصدار
Input statements	تعليمات التوريد
Input/ Output Devices	وحدات الإخراج و الإدخال
Input/ Output Devices	وحدات الإصدار و التوريد،
Insertion	الإضافة
Instantiation	تجسيد
Instantiation relation	مفهوم علاقة الإنتماء
Integer	عدد كامل
Integer division	القسمة الكاملة
Integrated circuit	المدار المدمج
Intel	إنتال

Interactive program	برنامج تفاعلي
Internal product (of vectors)	التصريف العددي
Interpreters	مؤولات
Iowa State	آيُوَا ستات
Is-A relation	مفهوم علاقة التفاوت
Isoceles triangle	مثلث عادي ذات جانبين متساويين
Iteration	تكرار
Iteration variable	متغيرة التكرار
Iterative interactive program	برنامج تفاعلي تكراري
Iterative program	برنامج تكراري
Java	جافا
Java compiler	مصرّف جافا
JFrame (in Java)	عنصر الإطار
JLabel (in Java)	عنصر العلامة
John Atanasoff	جوهن آتاناسوف
John Mauchly	جوهن موشلي
John Napier	جوهن نابيير
John von Neumann	جون فون نويمان
Joseph Marie Jackard	جوسف ماري جاكارد
JPanel (in Java)	عنصر اللوحة
Ken Thomson	كان طومسن
Keyboard	لوحة المفاتيح
Keyboard	لوحة المفاتيح
Keyboard Adapter	ملائم لوحة المفاتيح
Keyboard scanner	ناسخة لوحة المفاتيح
Knowledge	المعرفة
Konrad Zuse	كُنراد زوس
Labels	العلامات
Left associativity	التجمع على اليسار
Left subtree	الشجرة الجزئية اليسارية
Lexicographic comparison	مقارنة قاموصية
Lexicographic ordering	الترتيب القاموصي
Limited degree graphs	مخططات ذات الدرجات المحدودة
Linear arrays	الجداول الخطية
Linear list	التسلسل
Linear structures	الهياكل الخطية
Linked list	التسلسل
Linker	مؤلف الروابط
Linking	تأليف الروابط
LISP (LISt Processing)	ليسب
Literal (expression)	عدد
Loader	رافع البرامج
Loading	رفع البرنامج
Logarithm	اللوغاريثما
Logic Design	التصميم المنطقي

Logic Programming	البرمجة المنطقية
Logical equivalence	المساواة المنطقية
Logical expressions	العبارات المنطقية
Logical negation	النفي المنطقي
Long integer	عدد كامل طويل
Loop	المدار
Loop body	جسم المدار
Loop condition	شرط المدار
Loose ordering	التفاوت اللين
Loose ordering	التفاوت المنحل
Los Alamos	لوس آلاموس
Management Information Systems	الأنظمة التصرفية للمعلومات
Mantissa	جسم (العدد)
Mark I	مارك الأول
Matrices	المصفوفات
Matrix product	تصريف مصفوفة بمصفوفة
Method	مفهوم الأسلوب
Method (of a class)	أسلوب (فئة)
Microchip	المدارات الدقيقة
Microprocessor	المعالج الدقيق
Microprocessor	المعالج الصغير
Microsoft	مايكروسوفت
MIT (Massachusetts Institute of Technology)	المعهد التقني لماساشوستس
Monitor	الشاشة
Monitor Adapter	ملائم الشاشة
Mother Board	اللوحة الأم
Mouse	الفأرة
Mouse	الكرة المتجولة
Mouse Adapter	ملائم الفأرة
Musee des Arts et Metiers	متحف التقنيات و المهن
Name (expression)	إسم
Negation	النفي
Nested statements	التعليمات المتداخلة
Netscape	نتسكاب
Network Adapter	ملائم العنكبوتية
Neutral element	العنصر المحايد
Niklaus Wirth	نيكلاوس ويرث
Non-inclusive	لاضمنا
NSF Net	أن أس أف نات
Numerical Analysis	التحليل العددي
Numerical Methods	الأساليب العددية
Object	كائنة
Object Attributes	أوصاف الكائنات
Object functions	وظائف الكائنات

Object List	قائمة كائنات
Object methods	أساليب الكائنات
Object Oriented Programming	البرمجة الموجهة للكائنات
Object oriented programming	البرمجة الموجهة للكائنات
Object Tree	شجرة كائنات
Object Tree Area	منطقة شجرة الكائنات
Objects	الكائنات
Objects	الكائنات
One dimensional arrays	الجداول ذات البعد الواحد
Operating Systems	أنظمة الإستخدام
Operator priority	ترتيب الأولويات
Output	الإصدار
Output statements	تعليمات الإصدار
Overflow	الفيضان
Packages	باقات
Palm Pilot	بالم بايلت
Panels	اللوحات
Parameters	معلمات
Parameters	مفهوم المعلمات
Parentheses	الأقواس العادية
Pascaline	باسكالين
Paul Allen	بول آلان
Peripheral Memory	وحدة الذاكرة الفرعية
Personal Computer	الحاسوب الشخصي
Plain triangle	مثلث عادي
Plan Kalkul	بلان كالكول
Pointer	موجّه
Powerbook 100	باوربوك 100
Precision of representation	دقة التمثيل
Presper Eckert	براسبر آكارت
Prime number	عدد أولي
Prime numbers	الأعداد الأولية
Printer	آلة الطبع
Printer	جهاز الطبع
Printer Adapter	ملائم آلة الطبع
Private (in Java)	خفية
Private declarations (in Java)	التصريحات الخفية
Product of a matrix with a vector	تصريف المصفوفة بالمتوجه
Program Compilation	صرف البرنامج
Program design	تصميم البرنامج
Program evolution and maintenance	تطوير و صيانة البرنامج
Program Execution	تنفيذ البرنامج
Program implementation	تسيير البرنامج
Program specifications	مواصفات البرنامج

Program testing	إختبار البرنامج
Programming	تحرير البرنامج
Programming Languages	لغات البرمجة
Programming phases	أطوار البرمجة
Pront Writer (Java)	كاتبات الطبع
Public (in Java)	عمومية
Public declarations (in Java)	التصريحات المكشوفة
Pure functions	وضائف صافية
Range of representation	إتساع التمثيل
Raytheon	رايثيون
Real numbers	الأعداد الحقيقية
Rectangular isoceles triangle	مثلث مستقيم متناسق
Rectangular triangle	المثلث المستطيل
Rectangular triangle	مثلث مستقيم عادي
Recursive functions	الوضائف الإستقرائية
Recursive programming	البرمجة الإستقرائية
Register Board	لوحة السجلات
Regulat binary trees	الأشجار الثنائية القانونية
Reporting methods	اساليب الإطلاع
Reporting methods	أساليب الكشف
Resistor	أنبوب مقاومة
Right subtree	الشجرة الجزئية اليمينية
Robot	الآليات
Roundoff cumulation	تضاعف التقريب
Row vectors	المتوجهات الأفقية
Run time	طور التنفيذ
Saturn V	ساترن الخامس
Scanner	الناسخ
Scanner	ناسخ
Scientific applications	التطبيقات العلمية
Screen	الشاشة
Semantics	الدلالة
Sensor	ألة الحسّ
Sensor Adapter	ملائم آلة الحس
Sensors	آلات الحس
Sentinel	حارس
Seymour Cray	سيمور كراي
Short integer	عدد كامل قصير
Silicon	السيليكون
Simulation	محاكاة
Simulation algorithm	خوارزمية المحاكاة
Simulation program	برنامج محاكاة
Sine	الجيب
Software Crisis	أزمة البرمجيات
Software Engineering	هندسة البرمجيات

Software engineering	هندسة البرمجيات
Software Reuse	إستعمالية البرمجيات
Sort trees	أشجار الترتيب
Speaker	مضخم الصوت
Specifying the precision	تحديد الدقة
Specifying the range	تحديد الإتساع
Square brackets	الأقواس المستقيمة
Square function	وظيفة المربع
Stack	الكومة
Steven Jobs	ستيفن جوبس
Steven Wosniak	ستيفن فوسنياك
Stream	صادر
Strict order	التفاوت الشديد
Strict ordering	التفاوت الشديد
String	سلسة الحروف
String	سلسلة
String class	فئة سلسة الحروف
Sub Class	قسم جزئي
Substring	سلسلة جزئية
Switch expression	عبارة التوجيه
Switch statement	تعليمة التوجيه
Switch statement	تعليمة التوزيع
Syntactic diagrams	المخططات النحوية
Syntax	النحو
Syntax trees	الأشجار النحوية
TCP/IP	تي سي بي / أي بي
Term	طرف
Text editor	معالج النصوص
Theory of Computing	نظرية الحساب
Think Pad 700	ثنكباد 700
Three dimensional arrays	جداول ذات ثلاثة أبعاد
Tim Berners Lee	تيم بارنارس لي
Touch Screen	الشاشات الحسّاسة
Towers of Hanoi	أبراج هانوي
Transistor	ترانزيزتور
Transistor	العبورية
Transistor	عدسة العبور
Transport Control Protocol/ Internet Protocol	مراسم مراقبة التنقل / مراسم الإنترنت
Tree traversal	الطواف على الشجرة
True	صحيح
true	صحيح
Turing Machine	آلة تورنغ
Turn It In	تورنت إين
Two dimensional arrays	الجداول ذات بعدين

Type checking	مراقبة الأنواع
Type conversion (programming)	تحويل الأنواع
Unary logical expressions	العبارات المنطقية الفردية
Underflow	الذوبان
Unix	يونكس
Updating methods	أساليب التغيير
Vacuum tube	أنبوب فراغ
Vacuum tubes	أنابيب الفراغ
Variable	المتغيرات
Variables	المتغيرات
Vectors	المتوجهات
Virtual World Area	منطقة العالم الخيالي
Von Neumann Architecture	معمارية فون نويمان
Von Neumann Architecture	معمارية فون نويمان،
Waiting queues	صفوف الإنتظار
Watt	واطّ
Webboard	واب بورد
Webct	وابسيتي
While statement (Java)	التكرار ذات الشرط السابق
William Gates	ويليام غاتس
William Oughtred	ويليام أوتراد
Windows 1995	وندوس 1995
Windows 1998	وندوس 1998
Windows 2000	وندوس 2000
Windows Vista	وندوس فيستا
Windows XP	وندوس أكس بي
World Wide Web	العنكبوتية العالمية
Wrapper Classes	فئات التغليف
X coordinates	الإحداثية الأفقية
Xerox Parc	زيروكس بارك
Y coordinates	الإحداثية العمودية
Yahoo	ياهو

الملحق أ:

تشفير الحروف العربية حسب مقياس

ISO-8859-6

ـــ	ـــ	ـه	ـص	ـسم	□	٣	٤	ـو	٪	٪	فـ	،	ء	ـه	ع
0600	0601	0602	0603	0604	0605	0606	0607	0608	0609	060A	060B	060C	060D	060E	060F
◌ّ	◌ٌ	◌ٍ	◌ً	◌َ	◌ُ	◌ِ	◌ْ	◌ٓ	◌ٔ	◌ٕ	؛	□	□	؞	؟
0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	061A	061B	061C	061D	061E	061F
ي	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د
0620	0621	0622	0623	0624	0625	0626	0627	0628	0629	062A	062B	062C	062D	062E	062F
ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	ـٰ	ـٱ	ـٹ	ـٺ	ـٽ
0630	0631	0632	0633	0634	0635	0636	0637	0638	0639	063A	063B	063C	063D	063E	063F
ـ	ف	ق	ك	ل	م	ن	ه	و	ى	ي	◌ً	◌ٌ	◌ٍ	◌َ	◌ُ
0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	064A	064B	064C	064D	064E	064F
◌ِ	◌ّ	◌ْ	◌ٓ	◌ٔ	◌ٕ	◌ٖ	◌ٗ	◌٘	◌ٙ	◌ٚ	◌ٛ	◌ٜ	◌ٝ	◌ٞ	◌ٟ
0650	0651	0652	0653	0654	0655	0656	0657	0658	0659	065A	065B	065C	065D	065E	065F
٠	١	٢	٣	٤	٥	٦	٧	٨	٩	٪	،	،	*	ـب	ف
0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	066A	066B	066C	066D	066E	066F
◌ٰ	ٱ	أ	إ	ء	ا	ؤ	ۇ	ى	ٹ	ن	ب	ټ	ٹ	پ	ټ
0670	0671	0672	0673	0674	0675	0676	0677	0678	0679	067A	067B	067C	067D	067E	067F
پ	ځ	خ	چ	چ	څ	چ	چ	ڈ	ډ	ب	ڈ	ت	ډ	ڈ	ڈ
0680	0681	0682	0683	0684	0685	0686	0687	0688	0689	068A	068B	068C	068D	068E	068F
ڟ	ڞ	ڟ	ڑ	ړ	ڑ	ږ	ڑ	ژ	ڑ	ښ	پس	پش	ص	ش	ڟ
0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	069A	069B	069C	069D	069E	069F
ڠ	ف	ڤ	ف	ڤ	ڤ	ڤ	ف	ق	ک	ک	ڪ	ګ	ګ	ڪ	گ
06A0	06A1	06A2	06A3	06A4	06A5	06A6	06A7	06A8	06A9	06AA	06AB	06AC	06AD	06AE	06AF
گ	گ	گ	گ	گ	ڵ	ڵ	ڵ	ڵ	ڼ	ں	ڻ	ڼ	ڼ	ھ	ڇ
06B0	06B1	06B2	06B3	06B4	06B5	06B6	06B7	06B8	06B9	06BA	06BB	06BC	06BD	06BE	06BF
ۀ	ه	ۀ	ة	ۆ	ۆ	ۆ	ۇ	ۈ	ۋ	ۋ	ۋ	ى	ۍ	ێ	ف
06C0	06C1	06C2	06C3	06C4	06C5	06C6	06C7	06C8	06C9	06CA	06CB	06CC	06CD	06CE	06CF

ۭ	ۮ	ۯ	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	ۺ	ۻ	ۼ	۽	۾	ۿ
06ED	06EE	06EF	06F0	06F1	06F2	06F3	06F4	06F5	06F6	06F7	06F8	06F9	06FA	06FB	06FC	06FD	06FE	06FF

الملحق ب:

تشفير الحروف اللاتينية حسب مقياس

ISO-8859-6

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
														-	
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	•
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F