كتاب لغة برمجة الجافا Java

بسم الله الرحمان الرحيم

السلام عليكم ورحمة الله وبركاته,

بفضل من الله وتوفيقه أضع بين ايديكم كتابي المتواضع والذي يوضح ويشرح لغة برمجة الجافا. كما تعلمون أخوتي الاعزاء فهناك العديد من لغات البرمجة كالفيجوال بسيك والسي والسي شارب وغيرها. اخترت لغة الجافا لسببين أما الاول لكونها لغة تتوافق مع جميع أنظمة التشغيل كالويندوز والماك واللينوكس وأما السبب الثاني أنه ممكن استخدام هذه اللغه في عدة مجالات: مثل برمجة المواقع, برامج حاسوب وتطبيقات هواتف ذكية (اندرويد).

هدفي من هذا الكتاب هو النهوض بالامة الاسلامية والعربية والرقي في التعلم والعلم في الجانب التكنولوجي – البرمجة وعلم الحاسوب لنكون باذن الله في القمه وننافس باقي الامم. هذا الكتاب يفيد كل طالب علم في كلية علم الحاسوب أو هندسة البرمجيات وأيضا كل من لديه ميول لعالم البرمجة ويود أن يدخل في هذا المجال ويبدع به. طبعا الجافا لغه واسعه ودائما فيها تحديثات. حاولت أن اِشمل المواضيع الاكثر انتشارا والاكثر استخداما واستعمالا في مجال العمل. في هذا الكتاب استخدمت محرر الاكواد IntelliJ الخاص بشركة JetBrains.

ان شاء الله يكون عملي هذا متقبلا خالصا لوجه الله الكريم بنية علم ينتفع به وصدقة جارية. بارك الله بكم ووفقكم وجزاكم كل خير وآخر دعوانا أن الحمد لله رب العالمين.

التاريخ: 25/04/2020

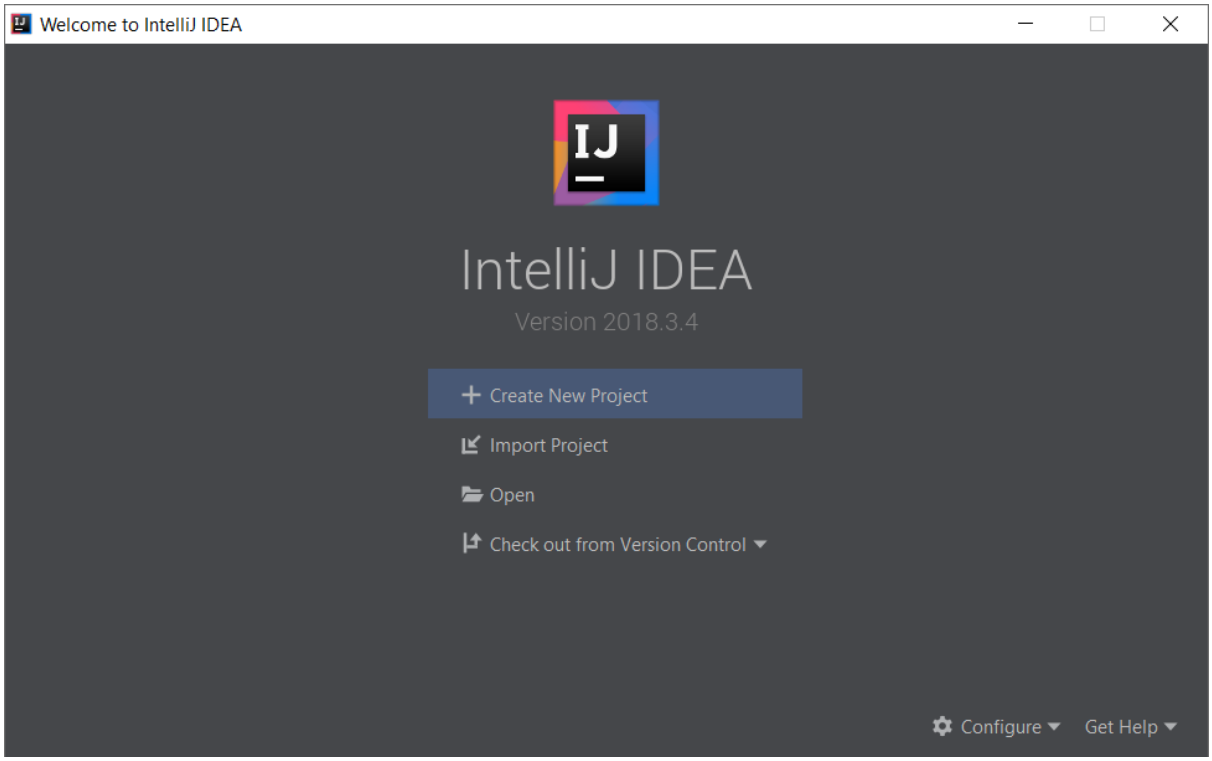
أخوكم محمد عويدات

eawedat@gmail.com

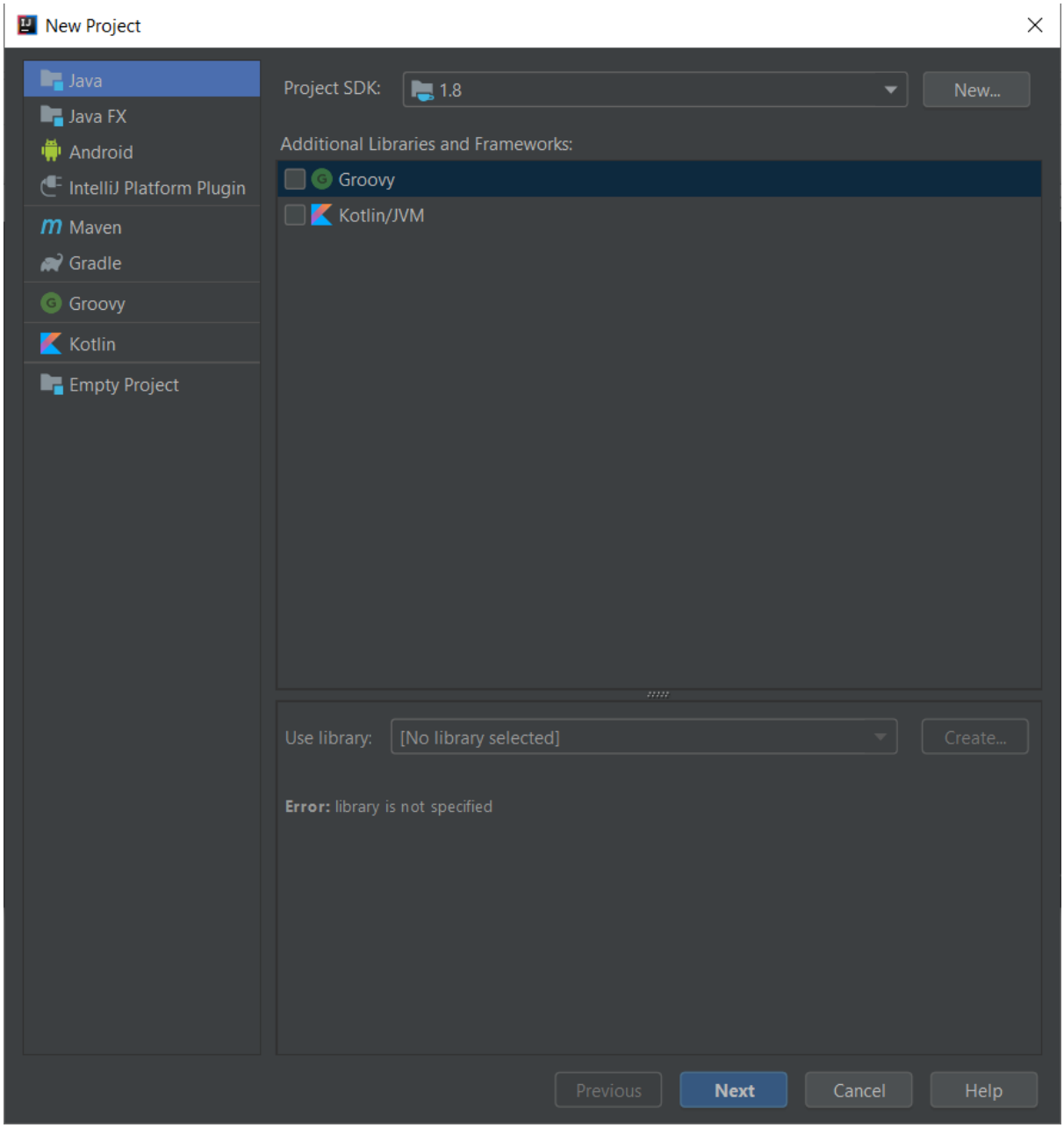
فهرست

تثبيت برنامج IntelliJ

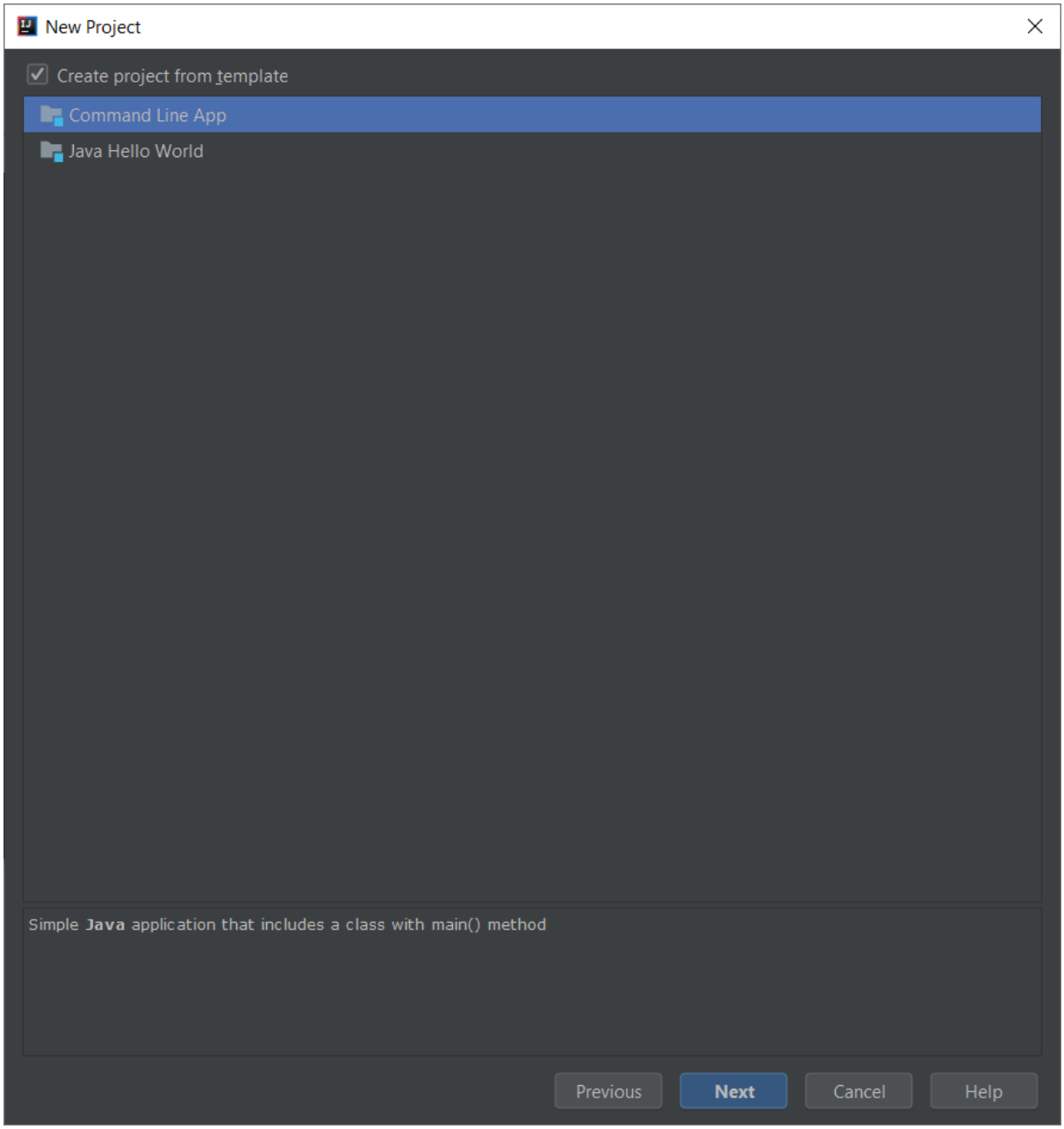
بعد تثبيت برنامج IntelliJ وتشغيله تظهر هذه الواجهة الرئيسية للبرنامج. نختار Create New Project.



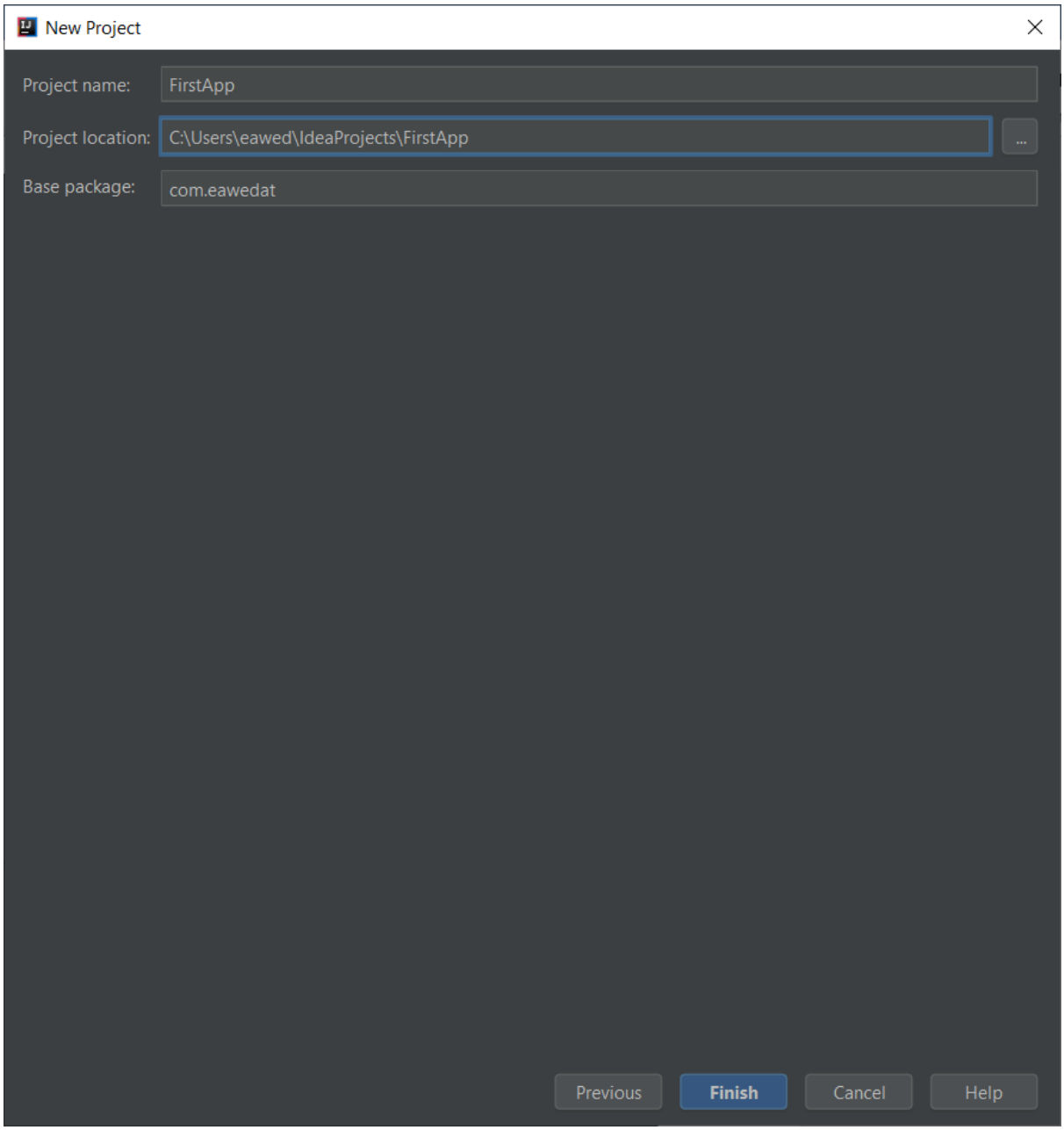
في هذه الواجهة نضغط الزر Next.



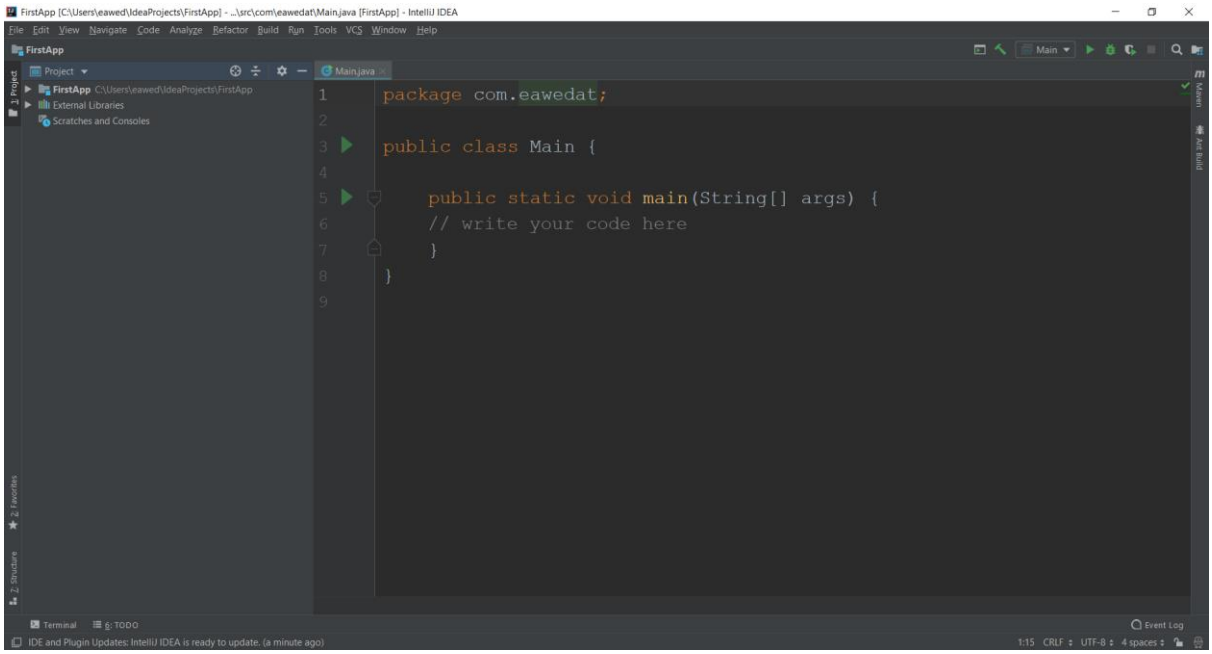
في هذه الواجهة نحدد الخيار Create project from template ثم نضغط الزر Next.



في خانة اسم المشروع Project name نكتب اسم فقط باللغة الانجليزية ويبدأ بحرف كبير.
يجب الانتباه ان مجلد المشروع Project location فقط باللغة الانجليزية ثم نضغط الزر Finish.



هذه الواجهة الرئيسية للبرنامج كما نلاحظ الجهة اليمنى لكتابة الاكواد والاوامر البرمجية واما القائمة اليسرى فهي تظهر اسماء الملفات.



بشكل عام نقوم بكتابة الاوامر والاكواد البرمجية داخل الدالة main وهي الدالة الرئيسية. بالنسبة للكلمات static و void سيتم شرحها لاحقا في الكتاب. الرمزين // هم للملاحظات. أي ان البرنامج compiler سوف يتخطى ويتجاهل كل سطر يحوي الرمزين // لانه سيفهم ان المبرمج أراد وضع ملاحظات وهي ليس اوامر واكواد برمجية فعلية يتم تطبيقها. يمكن أن نضع عدة أسطر كملاحظة من خلال الرمزين:

/*

These are comments,

*/

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {
    // write your code here
    }
}
```

تعريف متغيرات – variables

المتغير عبارة عن مكان في الذاكرة. يمكن من خلاله حفظ قيمة معينة واستخدامها في كل وقت. هناك عدة أنواع من المتغيرات: متغير نصي. متغير رقم صحيح. متغير رقم بفاصلة عشرية. متغير حرف وغيرها. فمثلا لو أردنا تعريف متغير من نوع رقم صحيح نستطيع ذلك من خلال الكلمة int ثم نكتب اسم المتغير. أسماء المتغيرات يجب أن تكون باللغة الانجليزية فقط وأن لا تبدأ برمز أو برقم.

int number;

لو أردنا على سبيل المثال تعريف متغير من نوع نص يمكن ذلك من خلال الكلمة String:

String name;

عندما نتحدث عن الذاكرة فهناك نوعين من الذاكرة يمكن التطرق لهما Heap و Stack.

ال Heap هي الذاكرة او المكان الذي يتم به حفظ الكائنات Objects , كائنات الكلاسات, سنتطرق لموضوع الكائنات لاحقا, على سبيل المثال String وأيضا المصفوفات arrays حتى لو كانت مصفوفات متغيرات ابتدائية مثل ;int[] numbers = new int[2].

ال Stack هي الذاكرة التي يتم بها حفظ المتغيرات البدائية كال int, double, float وغيرها boolean, char, long, short, byte وأيضا استدعاء الدوال.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

    int number;
    String name;
    double pi;
    float age;
    boolean flag;


    }
}
```

وضع قيم داخل هذه المتغيرات يتم بواسطة علامة يساوي =

مثلا لو أردنا إعطاء (تعويض) العدد 7 داخل متغير إسمه number نكتب ما يلي:

int number;

number = 7;

أي قمنا بمرحلتين (كتبنا سطرين): المرحلة الاولى (السطر الاول) تعريف نوع المتغير ثم إسمه وفي المرحلة الثانية (السطر الثاني) قمنا بتعويض أو إعطاء القيمة 7 داخل المتغير. كذلك الامر بالنسبة للمتغيرات النصية. مثلا لو أردنا إعطاء الاسم محمد عويدات داخل متغير نصي نكتب ما يلي:

String name;

name = “Muhamad Eawedat”;

يرجى الانتباه أن القيم النصية تكون بين علامتي " " أما فالقيم العددية فلا.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

    int number;
    String name;
    double pi;
    float age;
    boolean flag;

    number = 7;
    name = "Muhamad Eawedat";
    pi = 3.14;
    age = 32.11f;
    flag = true;

    }
}
```

يمكن اختصار الكتابة أي أن نقوم بتعريف متغير وتعويض قيمة داخله بنفس السطر (بنفس المرحلة)

مثلا: عوضا عن كتابة السطرين:

int number;

number = 7;

يمكن اختصار السطرين في سطر واحد وكتابة:

int number = 7;

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int number = 7;
        String name = "Muhamad Eawedat";
        double pi = 3.14;
        float age = 32.11f;
        boolean flag = true;

    }
}
```

يمكن أيضا تعريف أكثر من متغير في نفس السطر:

int number1, number2, result;

طبعا من الممكن أن تكون قيمة متغير معين حاصل جمع متغيرات مختلفه وأيضا ممكن عمل عمليات حسابية اخرى مثلا الضرب, القسمة, الجمع والطرح. كما نرى في هذا المثال فقد قمنا بجمع العددين (المتغيرين) ووضع قيمة جمعهما في المتغير result.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int number1, number2, result;

        number1 = 5;
        number2 = 10;
        result = number1 + number2;

    }
}
```

هناك عدة أشكال لاضافة العدد 1 الى قيمة متغيرمعين أبرزها

number++;

أو

++number;

فهي تقوم باضافة العدد 1 الى قيمة المتغير number فعلى سبيل المثال لو كانت قيمة المتغير 5 ثم قمنا بكتابة number++ ستكون القيمة الجديدة للمتغير 6.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int number = 1;

        number = number + 1;

        number += 1;

        number++;

        ++number;

    }
}
```

النوع boolean يقبل احدى القيمتين إما true وإما false.

السطر number1 == number2 يشمل الرمز == وذلك يعني سؤال فيما اذا كان المتغير number1 يساوي number2 اذا تحقق هذا الشرط النتيجة تكون true وبذلك ايضا قيمة المتغير result تكون true.

السطر الثاني: الرمز && يمثل AND وذلك يعني **وأيضا** أي **أن كلا الطرفين أو كلا الشرطين** يجب أن يتحقق لكي نحصل على القيمة true (طبعا ممكن أكثر من طرف أو شرط ويجب أن تتحقق جميع الشروط لارجاع القيمة true). على سبيل المثال قيمة المتغيرnumber1 هي 5 . العدد 5 أكبر من العدد 4 إذا تحقق الشرط الاول. الان نكمل للشرط الثاني ,قيمة المتغير number2 هي 3 والعدد 3 أكبر من العدد 2 وليس أصغر منه , اذا الشرط الثاني لم يتحقق وبهذا تكون النتيجة false.

السطر الثالث: الرمز || يمثل OR وذلك يعني **أو** أي **أن أحد الطرفين أو أحد الشرطين** يجب أن يتحقق لكي نحصل على القيمة true (طبعا ممكن أكثر من طرف أو شرط ويجب أن يتحقق شرط واحد على الاقل لارجاع القيمة true). على سبيل المثال قيمة المتغيرnumber1 هي 5 . العدد 5 أكبر من العدد 4 إذا تحقق الشرط الاول. الان نكمل للشرط الثاني ,قيمة المتغير number2 هي 3 والعدد 3 أكبر من العدد 2 وليس أصغر منه , اذا الشرط الثاني لم يتحقق وبهذا تكون النتيجة النهائية true لانه يكفي تحقق أحد الشرطين.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int number1=5;
        int number2=3;
        boolean result;

        result = number1 == number2;

        result = number1 > 4 && number2 < 2;

        result = number1 > 4 || number2 < 2;

    }
}
```

عملية الطباعة (طباعة على الشاشة) يتم من خلال الامر:

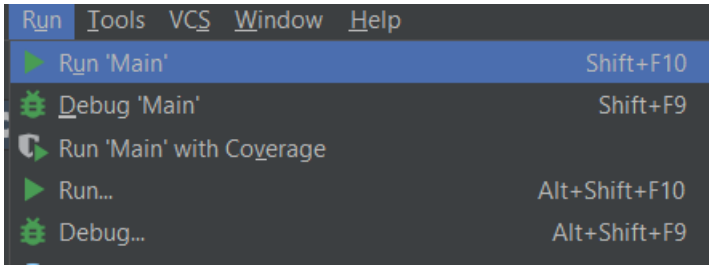
System.out.println();

ونكتب داخل الاقواس اما اسم المتغير لطباعة قيمته أو نص معين.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        System.out.println("Muhamad Eawedat");

        int number1 = 5;
        System.out.println(number1);

        double pi = 3.14;
        System.out.println(pi);

        String email = "eawedat@gmail.com";
        System.out.println(email);
    }
}
```

لتشغيل البرنامج ومشاهدة نتيجة الطباعة يجب الضغط على الزر Run (أيقونة السهم أو Play) باللون الاخضر.

إمكانية اخرى لتشغيل البرنامج من خلال الشريط العلوي ثم Run ثم 'Run 'Main'.



الشروط – If Conditionals

يتم كتابة الشروط من خلال الكلمة if والتي تعني إذا باللغه العربية. على سبيل المثال لو أردنا طباعة الجملة (: Very Good لطالب قد حصل على علامة أكبر من 85 في الامتحان:

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int grade = 90;

        if (grade > 85) {
            System.out.println("Very Good :)");
        }

    }
}
```

نتيجة الطباعة ستكون (: Very Good لان العلامة 90 هي أكبر من 85 وبذلك تحقق الشرط الاول.

هذه الصيغه تعد أبسط صيغ الشروط.

في المثال السابق قمنا بطباعة الجملة Very Good :) للطالب الذي حصل على علامة أكبر من 85 ولكن ماذا لو حصل على علامة أقل؟ في المثال السابق لن يقوم البرنامج بطباعة أي شيء. يمكننا تطوير الشرط واستخدام صيغة أكثر تركيبا من خلالها نستطيع أيضا التطرق للطالب الذي حصل على علامة أقل من 85 وذلك من خلال الكلمة else والتي تعني أي شيء آخر, أي كل عدد أقل من 85.

في هذا المثال نتيجة الطباعة ستكون Very Good :) لان العلامة 90 هي أكبر من 85 وبذلك تحقق الشرط الاول.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int grade = 90;

        if (grade > 85) {
            System.out.println("Very Good :)");
        } else {
            System.out.println("Try again!");
        }

    }
}
```

يمكن تطوير الشرط أكثر فأكثر واستخدام الكلمة else if والتي يقوم بفحصها اذا لم يتم تحقق الشرط الاول. اي في المثال التالي العلامة هي 90. يدخل للشرط الاول ويسال فيما اذا كانت العلامة أكبر من 85. نعم العلامة 90 أكبر من 85 . إذا تحقق الشرط الاول ولن يدخل الى الشرط الثاني (أكبر من 70). وهكذا سيتم طباعة الجملة (: Very Good.

فرضا ان العلامة كانت 80. عندها يدخل للشرط الاول, يسأل هل 80 أكبر من 85؟ كلا, اذا يفحص الشرط الثاني, هل 80 أكبر من 70؟ نعم, اذا تحقق الشرط الثاني. وعندها يطبع الجملة !Not bad.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int grade = 90;

        if (grade > 85) {
            System.out.println("Very Good :)");
        } else if (grade > 70) {
            System.out.println("Not bad!");
        } else {
            System.out.println("Try again!");
        }

    }
}
```

هناك صيغة شرطية اخرى تسمى Switch تشمل لائحة احتمالات:

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int grade = 90;

        switch (grade) {
            case 85:
                System.out.println("Very Good :)");
                break;

            case 70:
                System.out.println("Not bad!");
                break;

            default:
                System.out.println("Try again!");
                break;
        }
    }
}
```

حلقات التكرار – Loop

في كثير من الأحيان نريد تنفيذ أمر معين عدة مرات. فعوضا عن كتابة نفس الكود أو نفس السطر البرمجي عدة مرات نستطيع اختصار كتابة هذه السطور من خلال حلقة تكرار.

فمثلا لو أردنا طباعة الاسم Muhamad Eawedat 5 مرات يمكننا كتابة:

```
System.out.println(“Muhamad Eawedat”);

System.out.println(“Muhamad Eawedat”);

System.out.println(“Muhamad Eawedat”);

System.out.println(“Muhamad Eawedat”);

System.out.println(“Muhamad Eawedat”);
```

الان بدلا من كتابة نفس السطر يدويا 5 مرات نستطيع ادخال سطر واحد داخل حلقة تكرار وهي بدورها ستقوم بتكرار السطر:

```
for (int i=0;i<5;i++) {

        System.out.println(“Muhamad Eawedat”);

}
```

كل ما قمنا به هو تعريف متغير مع قيمة ابتدائية 0 ويتكرر طالما أن قيمته أصغر من 5 ويضيف واحد الى قيمته في كل مرة (عداد) حتى يصل الى 5 تتوقف الحلقة ويخرج منها لان الشرط لن يتحقق. مثال آخر, طباعة القيم من 0 الى 9:

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }

    }
}
```

حلقة تكرار while نفس فكرة الحلقة for مع فارق صيغة الكتابة syntax. في هذا المثال قمنا بتعريف متغير وعوضنا القيمة 0 كقيمة ابتدائية ثم قلنا طالما قيمة المتغير أصغر من 10 نفذ ما يلي:

اطبع قيمة المتغير i

أضف واحد لقيمة المتغير i

يجب الانتباه أن عملية اضافة واحد لقيمة المتغير في الحلقة while هي عملية ضرورية ليتم الخروج من حلقة التكرار والا سوف تكون الحلقة لا نهائية لان في ذلك الحين دائما 0 أصغر من 10 الشرط سيتحقق دائما.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int i = 0;

        while (i < 10) {
            System.out.println(i);
            i++;
        }
    }
}
```

الكلمة continue

عادة ما نجدها داخل حلقات التكرار. نكتبها عندما نريد تخطي قيمة معينة أي لتجاوز تنفيذ كود معين في الحلقة عند تحقق شرط معين. مثلا في هذا الكود عند الوصول الى العدد 2 سيتم تخطي جميع الاوامر التي تحت السطر if (i == 2) continue وسيتم الاستمرار مع الاعداد 3 , 4 وهكذا. في هذا المثال سيتم طباعة جميع الاعداد من 0 الى 9 عدا العدد 2.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {
            if (i == 2) continue;
            System.out.println(i);
        }
    }
}
```

الكلمة break

وظيفة هذه الكلمة هي الخروج كليا من الحلقة عند تحقق شرط معين بعكس الكلمة continue فهو لن يستمر مع باقي القيم او الاعداد. في هذا المثال سيتم طباعة العددين 0 و 1 فقط. فور وصوله للقيمة 2 سيتم الخروج كليا من الحلقة.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {
            if (i == 2) break;
            System.out.println(i);
        }
    }
}
```

المصفوفات – Arrays

المصفوفة عبارة عن عدة عناصر (متغيرات\قيم) من نفس النوع يتم تخزينها في الذاكرة تكون بجانب بعضها البعض. يتم تمييز المصفوفة عن متغير عادي من خلال القوسيين [] .في هذا المثال تم تعريف مصفوفة باسم names من نوع نص String وتحتوي على عنصرين (خليتين). في الخلية الاولة تم تخزين الاسم Muhamad وفي الخلية الثانية تم تخزين الاسم Eawedat. وأخيرا يتم المرور على جميع عنصار\خلايا المصفوفة names وطباعتها.

يتم الوصول لخلية معينة من خلال رقم ترتيبها في المصفوفة. هذا الرقم يسمى index وهو يشير لمكان العنصر في المصفوفة وهو يبدأ دائما من الرقم 0. مثلا الاسم Muhamad يتواجد في الخلية 0 وأما الاسم Eawedat فهو يتواجد في الخلية 1 وهكذا. يتم الوصول للعنصر من خلال كتابة اسم المصفوفة[رقم الخلية] .

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        String[] names = new String[2];
        names[0] = "Muhamad";
        names[1] = "Eawedat";

        for (int i = 0; i < names.length; i++) {
            System.out.println(names[i]);
        }
    }
}
```

طريقة أخرى لتعريف المصفوفة من خلال نفس السطر.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        String[] names = {"Muhamad", "Eawedat"};


        for (int i = 0; i < names.length; i++) {
            System.out.println(names[i]);
        }
    }
}
```

في هذا المثال نرى صيغة كتابة جديدة لحلقة التكرار for وهي تسمى حلقة التكرار المحسنة enhanced loop.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        String[] names = {"Muhamad", "Eawedat"};


        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

يمكن أيضا تعريف عدة أنواع من المصفوفات مثل مصفوفة من نوع أعداد صحيحة.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int[] numbers = new int[3];
        numbers[0] = 3;
        numbers[1] = 5;
        numbers[2] = 7;

        System.out.println(numbers[0]);
    }
}
```

ArrayList

هي مصفوفة ديناميكية بعكس المصفوفة العادية التي قمنا بتعريفها من خلال الاقوس [] . عندما نقول ديناميكية أي أن حجمها (عدد العناصر) التي تحتويها هذه المصفوفه هو متغير وغير ثابت أي أنه يتغير تلقائيا بحسب الحاجة. أي في كل مرة نقوم باضافة عنصر الى المصفوفة من خلال الدالة add اذا كانت تحتاج للمزيد من المساحه هي ستقوم بتوسيع نفسها بحسب الحاجة. كما نرى من خلال هذا المثال هناك العديد من الدوال التي يمكن استخدامها على المصفوفة مثل size و remove و contains و get و equals.

add() - اضافة عنصر أو قيمة معينة للمصفوفة

size() – العدد الكلي للعناصر في المصفوفة

get() – الحصول على قيمة معينة من خلال مكانها في المصفوفة

remove() – حذف عنصر\قيمة معينة من المصفوفة

contains() – فحص فيما اذا كانت المصفوفة تحوي قيمة معينة

equals() – فحص التشابه بين نصين

```
package com.eawedat;

import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {

        ArrayList<String> names = new ArrayList<>();
        names.add("Muhamad");
        names.add("Eawedat");
        names.add("eawedat@gmail.com");

        System.out.println(names.get(0));
        names.size();
        names.remove(0);
        names.contains("Muhamad");
        names.get(0).equals("Muhamad");

    }
}
```

الدوال – Functions / Methods

الدالة عبارة عن مكان يمكن الكتابة داخله. الهدف من الدوال هو تنظيم الكود, سهولة الاستخدام واختصار الكتابة. مثلا: لو أردنا طباعة حاصل جميع عددين ونحن نعمل ذلك كثيرا في برنامجنا فبدلا من كتابة الاوامر:

```
int number1, number2, result;

number1 = 5;

number2 = 7;

result = number1 + number2;

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

فبدلا من كتابة هذه الاكواد كل مرة نحتاج بها لطباعة حاصل جمع العددين يمكننا وضع هذه الاكواد داخل دالة نعطيها اسم وفي كل مرة نحتاج طباعة حاصل جمع العددين ما علينا فعله هو فقط استدعاء الدالة من خلال اسمها.

```
void printAddition() {

        int number1, number2, result;

        number1 = 5;

        number2 = 7;

        result = number1 + number2;

        System.out.println(result);

}
```

وعملية استدعاء الدالة يكون من خلال كتابة اسمها:

```
printAddition();
```

عندما نتحدث عن موضوع الدوال, فالدوال تقسم الى قسمين:

دوال لا تعيد قيمة وهي دوال تكون من نوع void.

دوال تعيد قيمة وهي دوال تكون من أي نوع مثلا: int, String, double, float, boolean, ArrayList وغيرها. يتم إعادة القيم بواسطة الكلمة المحفوظة return.

في هذا المثال: قمنا بتعريف عدة دوال.

الدالة الاولى من نوع void أي انها دالة لا تعيد قيمة معينة. تقوم فقط بتطبيق الاوامر مثل عمليات حسابية, طباعة وغيرها من أوامر. اسم الدالة show وهو اسم المبرمج يختاره.

قمنا بتعريف متغير من نوع رقم صحيح وقمنا بتعويض القيمة 5 فيه وأخيرا طبعنا قيمته.

الدالة الثانية من نوع نص String واسمها getName وتعيد النص Muhamad Eawedat

الدالة الثالثة من نوع رقم صحيح int واسمها calc تستقبل عددين صحيحين وتقوم بحساب حاصل جمع العددين وتعيد لنا حاصل الجمع. يجب الانتباه ان استدعاء الدالة يكون بهذه الطريقة:

calc(1,2);

calc2(100,500);

```
void show() {
    int x = 5;
    System.out.println(x);
}

String getName() {
    return "Muhamad Eawedat";
}

int calc(int num1, int num2) {
    int result = num1 + num2;
    return result;
}

int calc2(int num1, int num2) {
    return num1 + num2;
}

void setName(String name) {
    System.out.println(name);
}
```

مفهوم ال Overloading

Overloading هو مصطلح يعبر عن وجود عدة دوال بنفس الاسم والنوع لكنها تختلف **بعدد** المتغيرات (البارامترات) التي تستقبلها, أو **بنوع** المتغيرات التي تستقبلها, أو **بترتيب** المتغيرات التي تستقبلها.

نرى من خلال هذا المثال ان الدوال كلها باسم search. الاولى تستقبل متغير من نوع عدد صحيح, الثانية تستقبل متغير من نوع نص, الثالثة تستقبل متغيرين الاول من نوع عدد صحيح والثاني من نوع نص. الدالة الرابعة والاخيرة تستقبل متغيرين الاول من نوع نص والثاني من نوع عدد صحيح.

كما نلاحظ فالفرق بين الدالة الاولى والثانية هو **بنوع** المتغير الذي تستقبله.

الفرق بين الدالة الثالثة والرابعة هو **بترتيب** المتغيرات.

الفرق بين الدوال الاولى\الثانية وبين الدوال الثالثة\الرابعة هو **بعدد** المتغيرات.

```
//Overload - Overloading
void search(int id) {

}

void search(String name) {

}

void search(int id, String name) {

}

void search(String name, int id) {

}
```

هناك تسميات أخرى لمفهوم ال Overloading منها:

Static Polymorphism

Compile-time Polymorphism

دوال ال String

كما نرى فالمتغير النصي (المتغير من نوع نص String) له العديد من الدوال الجاهزة المكتوبة مسبقا والتي يمكن استدعائها واستخدامها. String بالاحرى هو نوع مبنى معين يسمى كلاس سوف نتطرق لموضوع المبنى (كلاس) لاحقا.

toUpperCase – دالة تقوم بتكبير حروف النص أي جعل حروفه كبيره

toLowerCase - دالة تجعل حروف النص صغيرة

length – دالة تقوم بارجاع طول النص أي عدد حروف النص

indexOf – دالة تقوم بارجاع مكان ظهور نص معين في نص آخر واذا لم تجده تعيد 1-

concat – دالة تقوم باضافة نص الى نص آخر

startsWith - دالة تقوم بفحص فيما اذا كان النص يبدأ بنص آخر

endsWith – دالة تقوم بفحص فيما اذا كان النص يتنهي بنص آخر

substring - دالة تقوم بقص جزء من النص

replace – دالة تقوم باستبدال نص معين بنص آخر

trim – دالة تقوم بازالة الفراغات من يمين ويسار النص

contains – دالة تفحص اذا كان النص يحتوي نص آخر

charAt – دالة تقوم بارجاع مكان حرف معين في النص

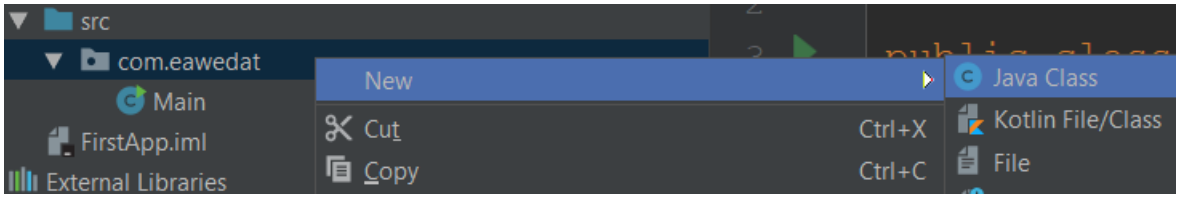
equals – دالة تقوم بفحص فيما اذا كان النص يساوي نص آخر

```java
public class Main {

    public static void main(String[] args) {

        String name = "Muhamad Eawedat";

        name.toUpperCase();
        name.toLowerCase();
        name.length();
        name.indexOf("am");
        name.concat(" Thank you");
        name.startsWith("Muh");
        name.endsWith("at");
        name.substring(0, 4);
        name.replace(" ", "_");
        name.trim();
        name.contains("Mu");
        name.charAt(0);
        name.equals("k");

    }
}
```

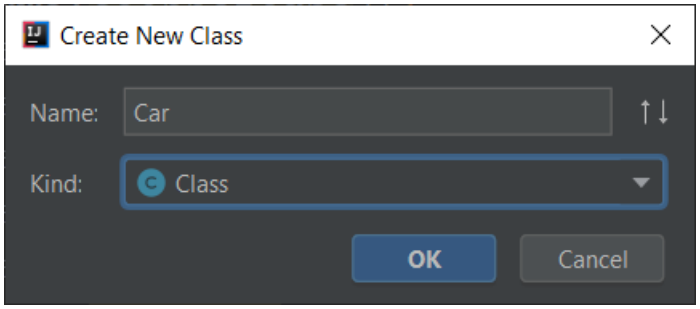
Class - كلاس

الكلاس عبارة عن هيكل أو مبنى معين داخله نستطيع تعريف متغيرات ودوال وكونستركتور Constructor. الهدف من الكلاس هو تنظيم الاكواد وترتيبها والاختصار في الكتابة. مثلا: يمكننا النظر الى شخص ككلاس في مفاهيم لغة الجافا, أو الى السيارة ككلاس وهكذا. لكل شخص خصائص وصفات, مثل الطول, الوزن, الاسم, لون الشعر, لون العينين وغيرها من صفات. لكل سيارة خصائص مثل: الشركة, الموديل, سنة الانتاج, السرعة القصوى للسيارة وغيرها من صفات.

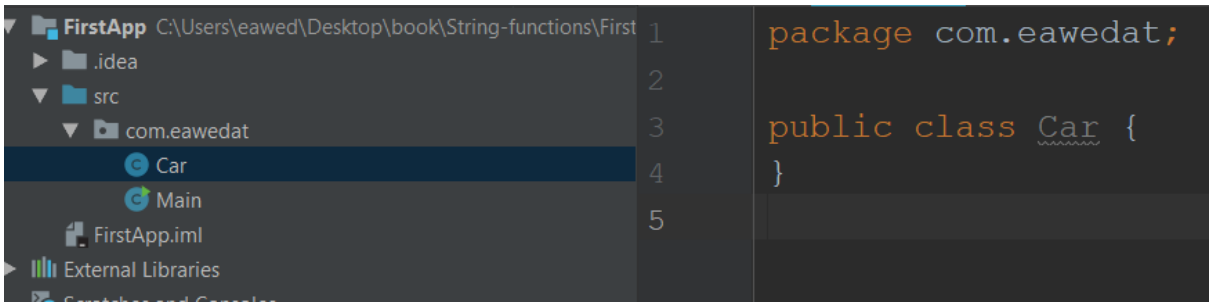
اولا: طريقة انشاء كلاس من خلال الضغط بالزر الايمن على اسم الرزمة Package, في هذا المثال com.eawedat ثم نختار New ثم Java Class.



تظهر هذه النافذه من خلالها نكتب اسم الكلاس باللغة الانجليزية بحيث ان الحرف الاول من الاسم حرف كبير.



كما نرى هنا يظهر الكلاس الجديد باسم Car وهو فارغ كليا من تعريفات متغيرات ومن دوال ومن Constructors.



يمكننا تعريف متغيرات كما تعلمنا في الدروس السابقة نكتب نوع المتغير واسمه. في هذا المثال قمنا بتعريف اسم السيارة كنص ثم موديل السيارة كرقم صحيح ولون السيارة كنص. هذه المتغيرات تمثل خصائص كلاس السيارة.

```
package com.eawedat;

public class Car {

    String name;
    int model;
    String color;

}
```

بعد تعريف المتغيرات في كلاس السيارة Car نذهب الى الملف او الكلاس الرئيسي Main ونقوم بتعريف كائن object من نوع سيارة من خلال هذا السطر:

```
Car car = new Car();
```

هذه الصيغة تسمى Static Binding لانه يتم التعرف على نوع الكائن في زمن ال compile. الكائن او ما يسمى ال object هو عبارة عن نسخة طبق الاصل من الكلاس او المبنى الاصلي Car أي عمل نسخه من Car وبذلك يمكننا انشاء العديد من النسخ , العديد من الكائنات وبهذا لن يتم تغيير الكلاس الاساسي, لاننا نعمل على نسخ منه وليس عليه بشكل مباشرة. فعلى سبيل المثال يمكن فعل ما يلي:

```
Car car1 = new Car();

Car car2 = new Car();

Car mazda = new Car();

Car bmw = new Car();
```

في المثال أعلاه قمنا بإنشاء 4 نسخ – 4 كائنات من الكلاس الاساسي Car ولكل نسخة لها خصائصها. أي تغيير في أي كائن من الكائنات الاربعة لا يؤثر على الاخر ولا على الكلاس الاساسي انما على نفسه. كما يظهر قمنا بتعريف مرجعية\نسخة\كائن باسم car من نوع Car (يجب الانتباه لحجم الحروف, فهناك فرق بين car و Car, car هي النسخة او المتغير او الكائن اما Car فهو الكلاس الاساسي او النوع) ثم قمنا باعطاء قيم لكل متغير موجود في الكائن car. مثلا قمنا باعطاء الاسم Mazda للمتغير name وقمنا باعطاء القيمة 323 للمتغير model وقمنا باعطاء القيمة White للمتغير color. بمجرد كتابة إسم الكائن car ثم وضع نقطة يتم إظهار جميع الخصائص المتاحة لهذا الكائن من أسماء متغيرات وأسماء دوال.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Car car = new Car();
        car.name = "Mazda";
        car.model = 323;
        car.color = "White";

    }
}
```

ال Constructor ودوال ال Getters / Setters

ال Constructor عبارة عن دالة خاصة تكون بنفس اسم الكلاس. فعلى سبيل المثال لدينا كلاس سيارة باسم Car ال Constructor يكون ايضا بنفس الاسم Car. هدف ال Constructor هو إعطاء قيم أولية للمتغيرات عند تعريف الكائن (النسخة).

دوال ال Getters هي دوال تكون من نفس أنواع المتغيرات وتعيد قيم هذه المتغيرات. مثلا:

قمنا بتعريف المتغير name من نوع نص وبذلك يمكننا تعريف دالة من نفس نوع المتغير String لتعيد لنا قيمة المتغير name.

```
String name;

String getName() {

        return name;

}
```

دوال ال Setters هي دوال تكون من نوع void ووظيفتها تحديد قيم المتغيرات التي تم استقبالها كبارامتر. مثلا:

```
String name;

void setName(String myname) {

        name = myname;

}
```

في حال كانت أسماء المتغيرات في الكلاس تتشابه مع أسماء البارامترات يجب إستخدام كلمة this وهي كلمة محفوظة وهي تقول لل compiler أنا اقصد متغير الكلاس وليس البارامتر. في حال كانت أسماء متغيرات الكلاس مختلفه عن أسماء البارامترات فلا حاجة للكلمة this.

```
package com.eawedat;

public class Car {

    String name;
    int model;
    String color;

    public Car(String name, int model, String color) {
        this.name = name;
        this.model = model;
        this.color = color;
    }

    public String getName() {
        return name;
    }

    public int getModel() {
        return model;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setModel(int model) {
        this.model = model;
    }
}
```

هذا مثال لكيفية إنشاء كائن أو نسخة من الكلاس Car مع إرسال قيم لل Constructor ليقوم ال Constructor باستقبال هذه القيم وتحديدها مباشرة للكائن (تعويضها في متغيرات النسخة). بمجرد كتابة إسم الكائن car ثم وضع نقطة يتم إظهار جميع الخصائص المتاحة لهذا الكائن من أسماء متغيرات وأسماء دوال.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Car car = new Car("Mazda", 323, "White");

        car.setModel(333);
        String carName = car.getName();

    }
}
```

كلمة this –

لكلمة this عدة استخدامات واستعمالات:

1. استدعاء Constructor في نفس الكلاس (بشرط ان يكون الاستدعاء من Constructor وان تكون في السطر الاول).
2. الوصول الى متغير في نفس الكلاس.
3. استدعاء دالة في نفس الكلاس.
4. تمرير كلمة this كبارامتر لدالة أخرى.
5. إعادة كلمة this كقيمة.

```
public class Person {

    String personName;

    Person() {
        this("Muhamad Eawedat"); //invoke constructor, must be first line
    }

    Person(String name) {
        personName = name;
    }

    void displayPerson() {
        System.out.println(this.personName); //invoke variable
    }

    void printName() {
        this.displayPerson(); //invoke method
    }

    void doSomething(Person person) {  }

    void anotherThing() {
        doSomething(this); //this as a parameter
    }

    Person getPerson() {
        return this; //this as a return type
    }
```

الوراثة – Inheritance

في لغة الجافا يمكن إنشاء كلاس معين وعدة كلاسات أخرى ترث من نفس الكلاس والهدف من ذلك هو الاختصار في الكتابة, تنظيم وترتيب الكود. مثلا يمكن إنشاء كلاس باسم Shape الذي يمثل شكل هندسي وإنشاء كلاسات دائرة, مستطيل, مربع وغيرها وأن نقوم باخبار كلاسات الدائرة والمربع والمستطيل انها ترث نفس صفات الشكل العام Shape. تخيلوا معي لو أن لدينا عشرات الكلاسات التي تمثل العشرات من الاشكال الهندسية ونحن نعلم ان للاشكال الهندسية طول وعرض فعوضا عن تعريف متغيرات الطول والعرض في كل كلاس من كلاسات الاشكال الهندسية يمكننا تعريف الطول والعرض في كلاس واحد الا وهو الكلاس الاساسي كلاس Shape ثم نقول لباقي الكلاسات ان ترث من كلاس ال Shape وبذلك جميع الكلاسات بدورها سترث مباشرة متغيرات الطول والعرض. في هذا المثال قمنا بانشاء كلاس Shape وقمنا بتعريف متغيرين عرض width وطول height

```
package com.eawedat;

public class Shape {

    int width;
    int height;

}
```

ثم قمنا بانشاء كلاس مربع باسم Square وقلنا له انك سترث صفات وخصائص ودوال الكلاس Shape بواسطة الكلمة المحفوظة extends. ثم قمنا بانشاء دالة باسم calcArea من نوع رقم صحيح (أي أنها ستعيد قيمة عدد صحيح). في الدالة calcArea قمنا بحساب المساحة واعادة قيمة المساحة.

```
package com.eawedat;

public class Square extends Shape {

    int calcArea() {
        return width * height;
    }

}
```

نذهب للكلاس Main الرئيسي ونقوم بانشاء كائن من كلاس المربع Square بمجرد كتابة اسم الكائن ثم وضع نقطة يتم إظهار قائمة بجميع المتغيرات والخصائص والدوال التي قام كلاس المربع Square بوراثتها من كلاس الشكل Shape. الجميل هنا مع اننا لم نقم بتعريف متغير العرض width والطول height في كلاس المربع Square الا انه بعد كتابة النقطه نستطيع رؤيتهم واستخدامهم لانه تم وراثتهم. عادة ما نجد الاوصاف Superclass أو parent للكلاس الاساسي الذي يتم الوراثه منه. والاسم Subclass أو child للكلاس الذي يرث من الكلاس الاساسي. أيضا يعرفون الوراثه على أنها:

Inheritance is an **IS-A** relationship. أي لو أخذنا الكلاسين Shape و Square.

Square is a Shape

نقول أن Square هو Shape. أي أن المربع هو عبارة عن شكل. وأيضا على سبيل المثال, لو أخذنا كلاسين Person و Human. نقول Person is a human. أي أن الشخص هو إنسان.

يمكن وراثة جميع المتغيرات والدوال التي من نوع public و protected عدا ال private لا يمكن وراثتها. إذا أردت منع وراثة كلاس معين عندها عليك تعريف الكلاس ك final كأنك تقول هذه النسخة نهائية من الكلاس لا يوجد من سيرثها.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Square square = new Square();
        square.width = 10;
        square.height = 10;

        System.out.println("Area of square is: " + square.calcArea());

    }
}
```

في موضوع الوراثه Inheritance يمكن عمل وراثة متعددة, أي عدة مستويات للوراثه على سبيل المثال:

```
class Human { }

class Person extends Human { }
class Student extends Person { }
```

في هذا المثال كلاس إنسان Human هو الكلاس الرئيسي. كلاس شخص Person يرث من كلاس إنسان Human. كلاس طالب Student يرث من كلاس شخص Person وأيضا من كلاس إنسان Human. أي أن كلاس طالب Student يرث جميع الدوال والمتغيرات الموجودة في كلاس Person وأيضا في كلاس Human عدا الدوال والمتغيرات التي تم تعريفها ك private. يجب الانتباه أننا لا نستطيع وراثه أكثر من كلاس في نفس الوقت, أي اننا لا نستطيع تعريف كلاس طالب Student هكذا:

```
Class Student extends Person, Human { }
```

يمكن وراثة كلاس واحد فقط في كل مرة. لحل هذه المشكلة يتم تعريف Interface والذي سنتطرق اليه لاحقا في الكتاب.

كلمة super -

كلمة super يكون لها معنى فقط عند عملية الوراثة. لكلمة super عدة استخدامات واستعمالات:

1. استدعاء Constructor في كلاس الاب – أي استدعاء ال Constructor للكلاس الذي ورثنا منه (بشرط ان يكون الاستدعاء من Constructor وان تكون في السطر الاول).
2. استدعاء دالة موجود في كلاس الاب. أي استدعاء دالة في الكلاس الذي ورثنا منه.
3. الوصول الى متغير في كلاس الاب. أي الوصول الى متغير تم تعريفه في الكلاس الذي ورثنا منه.

```
package com.eawedat;

public class Student extends Person {

    String studentName;

    Student() {
        super("Muhamad Eawedat"); //invoke superclass/parent constructor
    }

    Student(String name) {
        studentName = name;
    }

    void printPerson() {
        super.printName(); //invoke superclass/parent method
    }

    void displayPersonName() {
        System.out.println(super.personName); //invoke superclass/parent variable
    }
}
```

تعدد الاشكال – Polymorphism

تعدد الاشكال يعني أننا نستطيع تعريف كائن بقدرته التحول لنوع آخر يشتق منه. اولا لتحقيق مفهوم تعدد الاشكال لا بد من وجود وراثه Inheritance. مثلا نعرف 3 كلاسات: كلاس شخص وكلاس موظف وكلاس طالب. الموظف والطالب هم أشخاص. كلاس الموظف يرث من كلاس الشخص. وكلاس الطالب ايضا يرث من كلاس الشخص. الان يأتي دور مفهوم تعدد الاشكال أننا نستطيع تعريف كائن من نوع شخص Person لكنه يحمل كائن الموظف أو الطالب. مثلا:

```
Person p1 = new Student();

Person p2 = new Employee();
```

هذه الصيغ تسمى بال Dynamic binding لانه يتم التعرف على نوع الكائن في زمن التشغيل runtime. هذا هو مفهوم تعدد الاشكال أي ان Person يستطيع أن يحمل عدة أشكال ترث منه شكل (كائن) الطالب وشكل (كائن) الموظف. في هذا المثال نرى أننا قمنا بتعريف الكلاس Person الذي يمثل الشخص ثم عرفنا دالة باسم sayHello وقمنا بطباعة الجملة I am a person.

```java
package com.eawedat;

public class Person {

    void sayHello() {
        System.out.println("I am a person");
    }
}
```

تطبيق جديد للدوال – Overriding

نرى هنا أننا قمنا بتعريف كلاس باسم Employee والذي يمثل موظف ثم جعلناه يرث خصائص, متغيرات ودوال كلاس Person. انتبه للكلمة Override هنا قمنا بتطبيق مفهوم ال Overriding أي تطبيق جديد للدالة sayHello. كما نعلم فمسبقا قد قمنا بتعريف الدالة sayHello في كلاس ال Person وقلنا أيضا أن كلاس Employee يرث دوال ومتغيرات الكلاس Person مباشرة فلا حاجة لكتابة نفس الدالة في كلاس ال Employee لانه قد ورثها بشكل مباشر من Person وهذا صحيح لكن أحيانا نريد تطبيق أوامر مختلفه عن الكلاس الاساسي (ما يسمى Parent Class – Super Class) كأنك تقول أنا اريد وراثة جميع الدوال من الكلاس الاساسي Person عدا هذه الدالة تحديدا أريد أن اعمل لها تطبيق خاص في كلاسي الشخصي Employee وهذا هو مفهوم ال Overriding. لتحقيق او لتطبيق مفهوم ال Overriding يجب ان تكون وراثة (لا حاجة لتحقق تعدد الاشكال Polymorphism) وأيضا ان تكون الدالة معرفة ك public أو protected وليست private ولا final.

```
package com.eawedat;

public class Employee extends Person {

    @Override
    void sayHello() {
        System.out.println("I am an employee");
    }
}
```

نرى هنا أننا قمنا بتعريف كلاس باسم Student والذي يمثل طالب ثم جعلناه يرث خصائص, متغيرات ودوال كلاس Person. نرى أيضا أننا قمنا بتطبيق خاص للدالة sayHello.

```
package com.eawedat;

public class Student extends Person {

    @Override
    void sayHello() {
        System.out.println("I am a student");
    }
}
```

اذا لاحظنا الكلمة Override@ وهي عبارة عن ملاحظة أو حاشية وهي ليست إجبارية (ليست ضرورية) لكن البرنامج يقوم باضافتها بشكل مباشر. تسمى باللغه الانجليزية annotation ولها هدفين:

1. اذا قمنا بكتابة اسم غير صحيح للدالة بالكلاس الذي يرث ويطبق من جديد هذه الدالة, عندها ال compiler سيعطينا خطأ. مثلا لو كتبنا اسم الدالة sayHellw أو syHello بدلا من sayHello في احدى الكلاسين Student او Employee سيظهر لنا خطأ. وأيضا اذا كانت عدد أو انواع البارامترات غير ملائمة أو مناسبة للدالة الاصلية في كلاس الاب. مثلا اذا كتبنا sayHello(int number) في احدى الكلاسين Student أو Employee.

2. تسهيل القراءة للمبرمجين الاخرين الذي يعملون على نفس البرنامج أو الكود. مثلا لو أنت تعمل في شركة برمجة وعندك فريق مطورين ومبرمجين تعملون على نفس الكود. عندما يأتي مبرمج آخر ليعمل على نفس المشروع ويرى الكلمة Override@ يفهم أنك أردت انشاء تطبيق جديد للدالة الموروثه.

هناك تسميات أخرى لمفهوم ال Overriding منها:

Dynamic Polymorphism

Runtime Polymorphism

Dynamic Method Dispatch

مصفوفة متعددة الاشكال – Polymorphic Array

كما نرى هنا قمنا بتعريف مصفوفة من نوع Person بحجم 2 وقمنا بتعريف كائن من نوع موظف Employee في الخلية الاولى وقمنا بتعريف كائن طالب Student في الخلية الثانية ثم من خلال حلقة تكرار for قمنا باستدعاء الدالة sayHello الموجودة في جميع الكلاسات Person Employee, Student وكل كائن من الكائنات يقوم بتطبيق الدالة الخاصة به sayHello.

نتيجة الطباعة سوف تكون:

I am an employee

I am a student

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Person[] person = new Person[2];
        person[0] = new Employee();
        person[1] = new Student();

        for (int i = 0; i < person.length; i++) {
            person[i].sayHello();
        }

    }
}
```

مثال آخر لتعدد الاشكال – Polymorphism. قمنا بانشاء كلاس رئيسي باسم Animal الذي يمثل حيوان ما. ثم قمنا بانشاء كلاس فرعي Cat وكلاس فرعي Elephant وجعلنا الكلاسات Cat و Elephant ترث من الكلاس الرئيسي Animal. ثم قمنا بكتابة دالة باسم makeNoise في كلاس Animal وقمنا بعمل تطبيق جديد لهذه الدالة في كلا الكلاسين Cat و Elephant.

```java
package com.eawedat;

public class Animal {

    void makeNoise() {
        System.out.println("I am an animal");
    }
}
```

```java
package com.eawedat;

public class Cat extends Animal {

    @Override
    void makeNoise() {
        System.out.println("I am a cat");
    }
}
```

```java
package com.eawedat;

public class Elephant extends Animal {

    @Override
    void makeNoise() {
        System.out.println("I am an elephant");
    }
}
```

هنا نرى تطبيق مفهوم مصفوفة متعددة الاشكال – Polymorphic Array واستخدام حلقة التكرار المحسنه enhanced for loop لاستدعاء الدالة makeNoise الخاصة بكل كلاس والتي تطبع نتائج (جمل) مختلفة بحسب الكائن. نتيجة هذه الاوامر ستكون:

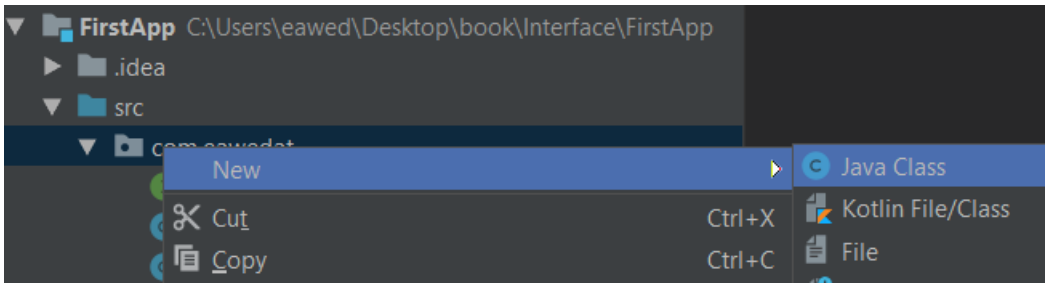
I am a cat

I am an elephant

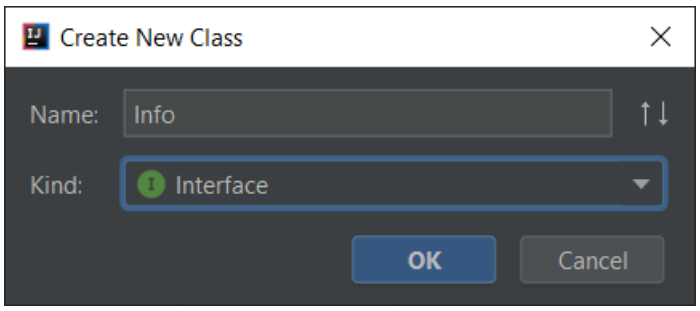
```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Animal[] animals = new Animal[2];

        Cat cat = new Cat();
        Elephant elephant = new Elephant();

        animals[0] = cat;
        animals[1] = elephant;

        for (Animal animal : animals) {
            animal.makeNoise();
        }

    }
}
```

مفهوم ال Interface

ال Interface وجد لايجاد حل لمشكلة الوراثة المتعددة. أولا لانشاء Interface نضغط على الزر الايمن للفأرة على اسم رزمة المشروع Package في هذا المثال اسم الرزمة هو com.eawedat ثم نختار New ثم Java Class.



في هذه النافذة نختار Interface في Kind ونكتب اسم ال Interface الذي نريده باللغه الانجليزيه بحيث أن اول حرف من الاسم يكون كبير.



نحصل على Interface فارغ نقوم مثلا بتعريف متغير باسم id من نوع عدد صحيح واعطائه القيمة 5 وأيضا نقوم بتعريف دالة نصية باسم getName من نوع String. يجب الانتباه ان جميع الدوال التي يتم تعريفها في ال Interface يجب ان لا تحوي على أوامر (أسطر أو أكواد) فقط تعريف وهي بشكل مباشر تعرف من نوع abstract public. أما المتغيرات فجميعها تكون في ال Interface من نوع public static final. متغير من نوع final يعني متغير مع قيمة ثابته لا نستطيع تغييرها لاحقا.

```
package com.eawedat;

public interface Info {

    int id = 5;
    String getName();

}
```

لكي نقوم بوراثة Interface يتوجب علينا كتابة الكلمة المحفوظة implements ويتوجب علينا أيضا تطبيق جميع الدوال الموجودة في ال Interface.

```
package com.eawedat;

public class Student implements Info {

    @Override
    public String getName() {
        return "Muhamad Eawedat";
    }
}
```

نتيجة طباعة البرنامج:

Muhamad Eawedat 5

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Student student = new Student();

        System.out.println(student.getName() + " " + student.id);

    }
}
```

طبعا الان يمكننا وراثة العديد من ال Interfaces, مثلا:

Interface A { }
interface B { }

Interface C { }

Class Student implements A, B, C { }

ملاحظة مهمة: لا يمكننا إنشاء كائن مباشرة من ال Interface: Info info = ~~new Info();~~

كلاس مجرد – Abstract

كلاس مجرد هو عبارة عن مبنى أو هيكل مجرد. الهدف من بناء كلاس من هذا النوع هو الوراثه واجبار كل من يرث هذا الكلاس تطبيق جميع الدوال التي تم تعريفها كمجرده. نوع من الالتزام أو ضمان بتطبيق دوال هذا الكلاس. كأن الكلاس يقول من الممكن أن ترثوا مني ولكن بشرط أن تقوموا بتطبيق الدوال التي عندي والتي تم تعريفها كمجرده abstract. طبعا كل كلاس يمكنه تطبيق الدالة بشكل مختلف, مثال طباعة نص مختلف أو تطبيق أوامر مختلفة عن الاخر. لا يمكن للدوال المعرفة ك abstract أن تحوي أكواد أو أوامر فهذه وظيفة الكلاسات الوارثه وليس الكلاس المجرد. هناك وجه تشابه بين كلاس مجرد abstract و interface فالكلاس المجرد يجب تطبيق الدوال التي عرفت ك abstract وأيضا في ال interface يجب على الكلاس الذي يرثه تطبيق جميع الدوال. تشابه آخر هو أن الكلاس المجرد abstract وأيضا ال Interface كلاهما لا نستطيع إنشاء كائنات منهما, نستطيع فقط الوراثه منهما. اما الاختلافات بينهما:

1. في ال Interface جميع المتغيرات هي فقط من نوع public static final أما في الكلاس المجرد يمكن تعريف أنواع متغيرات أخرى مختلفة.
2. جميع الكلاسات التي ترث من ال Interface عليها تطبيق الدوال جميعا بينما في الكلاس المجرد abstract يتوجب على الكلاسات التي ترثه تطبيق الدوال التي تم تعريفها ك abstract فقط.
3. ال Interface لا يحوي Constructor بينما الكلاس المجرد يحوي Constructor.
4. يتم وراثة ال Interface من خلال كلمة implements ويمكن وراثة أكثر من Interface في المرة الواحدة لكل كلاس, بينما الكلاس المجرد abstract يتم وراثته من خلال الكلمة extends وفقط وراثه واحد لكل كلاس.

```
package com.eawedat;

public abstract class Person {

    abstract void show(String message);

}
```

```java
package com.eawedat;

public class Employee extends Person {

    @Override
    void show(String message) {
        System.out.println("I am an employee");
    }
}
```

يجب التذكر أنه لا نستطيع إنشاء كائنات من كلاس مجرد abstract مثل:

Person person = ~~new Person();~~

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Person person = new Employee();
        person.show("Hello");

    }
}
```

متغيرات ودوال Static

متغير من نوع static هو متغير يخص الكلاس ولا يخص الكائن ككائن. أي اننا عندما نقوم بتعريف متغير من نوع static كأننا نقول هذا المتغير تابع للكلاس مباشرة وليس له علاقة للكائن وبذلك يمكن القول أن متغير من نوع static هو مشترك (قيمته مشتركه) لجميع الكائنات ويمكن الوصول اليه من خلال إسم الكلاس مباشرة دون الحاجة لكتابة اسم الكائن. طبعا دوال ال static تستطيع أن تقوم باستدعاء دوال static ولا تستطيع استدعاء دوال من نوع آخر. وأيضا بالنسبة للمتغيرات, دوال ال static تستطيع التعامل مع متغيرات instance variables من نوع static فقط.

دالة من نوع static هي أيضا دالة خاصة أو تابعه للكلاس وبذلك يمكن استدعائها مباشرة من خلال إسم الكلاس. كما يظهر في المثال: قمنا بتعريف متغير نصي باسم name من نوع static وأيضا دالة باسم display من نوع static.

```
package com.eawedat;

public class Data {

    public static final String name = "Muhamad Eawedat";

    public static void display() {
        System.out.println("I am a static method");
    }

}
```

كما نرى هنا, يمكننا الوصول الى المتغير مباشرة من خلال كتابة إسم الكلاس, ويمكن إستدعاء الدالة من خلال إسم الكلاس بدون حاجة لانشاء كائن من نفس الكلاس لان المتغيرالان مشترك لجميع الكائنات.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        System.out.println(Data.name);

        Data.display();

    }
}
```

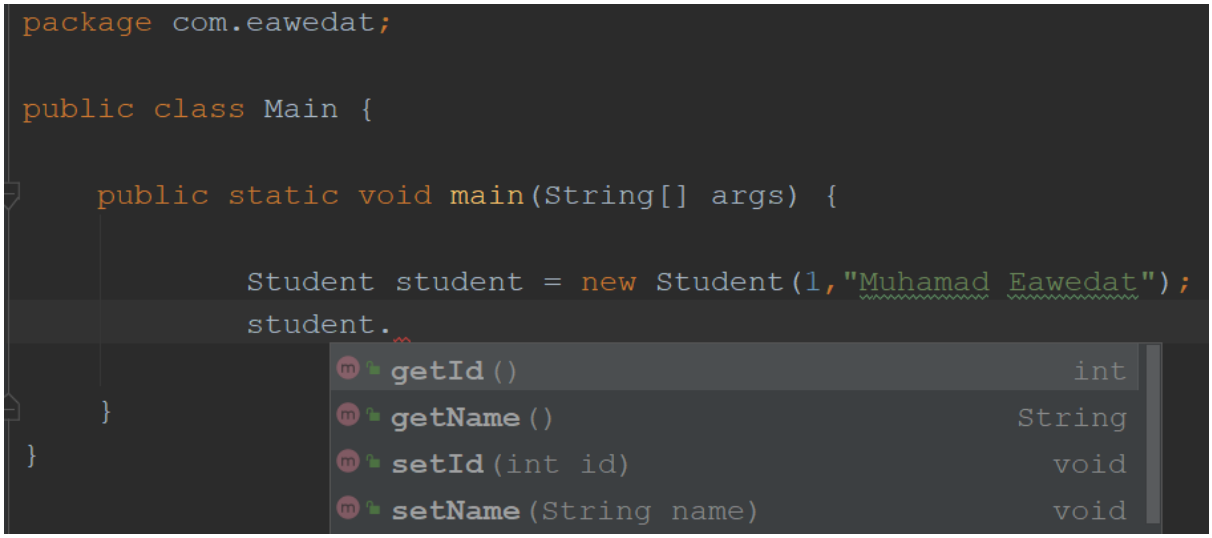
التغليف – Encapsulation

مفهوم التغليف عبارة عن نوع من أنواع إخفاء البيانات أو حماية البيانات Security, بذلك نمنع رؤية أو مشاهدة قائمة المتغيرات من خلال الكائن. لتطبيق مفهوم التغليف نحن بحاجة لتطبيق قاعدتين أو شرطين:

1. تعريف جميع المتغيرات من نوع private.
2. تعريف جميع دوال ال getters/setters ك public.

```
public class Student {

    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

اذا لاحظنا هنا, فعند كتابة اسم الكائن ثم نقطة لا تظهر أسماء المتغيرات لاننا قمنا بتعريفها ك private, مما يعني أننا نستطيع الوصول لهذه المتغيرات فقط من خلال دوال ال getters/setters أو طبعا من خلال الكلاس الاساسي Student نفسه.



تحويل أنواع البيانات – Type Casting

يمكن التحويل بين أنواع عديدة من المتغيرات. هنا سوف أتطرق لنوعين الاكثر شيوعا أو استخداما. يمكن التحويل من قيمة نصية الى قيمة عددية رقمية أو العكس من قيمة عددية رقمية الى قيمة نصية.

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        int num = 5;
        String str = "123";

        String y = String.valueOf(num);
        int num2 = Integer.parseInt(str);

    }
}
```

مفهوم Variable Length Argument

أحيانا كثيرة نريد استدعاء دالة معينة مع اختلاف في عدد البارامترات (عدد المتغيرات التي تستقبلها الدالة), في هذه الحالة يتوجب علينا إنشاء عدة دوال مع العدد المناسب للبارامترات لكل دالة ولكن هذا نوع من التكرار الذي يمكن توفيره من خلال variable length argument.

أي على سبيل المثال: لو أردنا عمل دالة تقوم بحساب وإعادة معدل عددين:

```
int avg(int num1, int num2) {

	return (num1+num2)/2;

}

avg(10,2);
```

ولو أردنا عمل دالة تقوم بحساب وإعادة معدل 3 أعداد:

```
int avg(int num1,int num2, int num3) {

	return (num1+num2+num3)/3;

}

avg(5,3,1);
```

ماذا لو أردنا انشاء المزيد من الدوال التي تحسب معدل 4 أعداد و 5 أعداد و 6 أعداد و 7 أعداد؟ يصبح نوع من التكرار الغير ضروري لاننا نعلم أن جميع الدوال تقوم بنفس الشي الا وهو حساب معدل الاعداد مع فارق عدد الاعداد. هنا يأتي دور ال variable length argument, يتم تعريفه من خلال 3 نقاط اضعها بين نوع البارامتر المستقبل واسمه.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        System.out.println(avg(100, 90));
        System.out.println(avg(100, 90, 90));
        System.out.println(avg(100, 90, 90, 80));
    }

    static int avg(int... numbers) {
        int total = 0;

        for (int i = 0; i < numbers.length; i++) {
            total += numbers[i];
        }

        return total / numbers.length;
    }
}
```

Generic Method

هي بكل بساطة دالة عمومية والتي تعني أنه يمكنها استقبال انواع مختلفة من البيانات أو الكائنات والتعامل معها. فمثلا لو أردنا طباعة أعداد صحيحة موجودة في مصفوفة يمكن تعريف التالي:

```
Integer[] numbers = {1,2,3,4};
printInteger(numbers);
void printInteger(Integer[] numbers) {
  for(int i=0;i<numbers.length;i++) {
        System.out.println(numbers[i]);
   }
}
```

ولو أردنا طباعة نصوص موجودة في مصفوفة يمكن تعريف التالي:

```
String[] names = {“Muhamad”,”Eawedat”,”eawedat@gmail.com”};
printNames(names);
void printNames(String[] names) {
  for(int i=0;i<names.length;i++) {
        System.out.println(names[i]);
   }
}
```

هنا يأتي دور الدالة العمومية Generic Method فعوضا عن كتابة عدة دوال وكل دالة تستقبل نوع مختلف من الكائنات نكتب دالة عمومية واحده لها القدرة على استيعاب أنواع مختلفة من الكائنات. نرى أنه يتم تعريف دالة عمومية من خلال كتابة الرمز <T> قبل اسم الدالة وأيضا كبارامتر.

```java
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        Integer[] numbers = {1, 2, 3};
        String[] names = {"Muhamad", "Eawedat"};

        printArray(numbers);

        System.out.println();

        printArray(names);

    }

    static <T> void printArray(T[] objects) {
        for (T obj : objects) {
            System.out.print(obj + " ");
        }
    }
}
```

معالجة الأخطاء – Error Handling – Try/Catch

أحيانا كثيرا نود تطبيق أو كتابة أمر معين لربما خلال تشغيل البرنامج هذا الامر يتغير مما قد يسبب حدوث خطأ, هذا الخطأ ممكن أن يؤدي الى اغلاق البرنامج بشكل مفاجئ, ما يسمى بعلم البرمجة Bug. إحدى الحلول لمعالجة هذه الانواع من الاخطاء هو أن نقوم بتغطية او تغليف الاكواد او الاوامر بكود try و catch. هذا النوع من معالجة الاخطاء نجده مثلا عند محاولتنا لقراءة ملف غير موجود أساسا أو انه لا توجد لدينا صلاحية قراءة الملف. أو مثلا قسمة عدد على صفر, المحاولة للوصول الى خلية أو عنصر في مصفوفة غير موجود أساسا. تحويل غير صحيح بين بيانات مختلفة وغيرها. فنكتب الاكواد داخل try . try تقول حاول فعل ما يلي من أكواد واوامر واذا لم تنجح في تحقيق هذه الاوامر بسبب خطأ ما لا تخرج من البرنامج و catch بدوره يصطاد الخطأ عند حدوثه. هناك كلمة finally وهي **ليست اجبارية**, ولكنها تعني الدخول اليها عند الانتهاء من try و catch بغض النظر اذا وقع خطأ أم لا فبكل الاحوال سيتم الدخول اليها بعد الانتهاء من تطبيق الاوامر في try و catch.

```
//Error handling
//For example: reading non-existing file

String s = null;
int i = s.length();

int num = 50;
num = num / 0;

int[] arr = new int[2];
arr[3] = 5;

String text = "abc";
int number = Integer.parseInt(text);


try {

} catch (Exception error) {

} finally {

}
```

تعدد المهام - Multithreading

الهدف من threading هو تشغيل أو تنفيذ عدة أمور في نفس الوقت. في كثير من الاحيان نريد أن نقوم بعدة أمور في برامجنا. مثلا فتح ملف نص والكتابة داخله وفي نفس الوقت الاتصال بقاعدة بيانات في الانترنت وجلب بيانات من القاعدة. في هذا الحال, مفضل أن يكون لكل عمل thread خاص به لكي لا يكون ضغط على البرنامج ويتم الخروج من البرنامج بشكل مفاجيء.

```
package com.eawedat;

public class MyThread extends Thread {

    @Override
    public void run() {

        int i = 0;

        while (i < 10) {

            System.out.println(i);
            i++;

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {

            }

        }
    }
}
```

```
package com.eawedat;

public class Main {

    public static void main(String[] args) {

        MyThread myThread = new MyThread();
        myThread.start();

    }

}
```

HashMap

HashMap عبارة عن كلاس من خلاله يتم تخزين مفتاح وقيمة بشكل key و value. لكل قيمة لها مفتاح خاص بها أي لا يمكن الوصول لقيمتين من خلال مفتاح واحد. المفاتيح لا يمكن أن تتكرر بينما القيم ممكن أن تتكرر. يمكن تشبيه ال HashMap بجدول يحوي عامودين بحيث كل عامود يمثل كائن, عامود يحوي المفاتيح وعامود يحوي القيم.

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "Muhamad");
map.put(2, "Eawedat");
map.put(3, "eawedat@gmail.com");
```

نرى هنا طرق مختلفة لطباعة المفاتيح والقيم المحفوظة في ال HashMap من خلال حلقات التكرار.

```
for (Map.Entry m : map.entrySet()) {
    System.out.println(m.getKey() + " " + m.getValue());
}

for (Map.Entry<Integer, String> m : map.entrySet()) {
    System.out.println(m.getKey() + " " + m.getValue());
}

for (String value : map.values()) {
    System.out.println(value);
}

for (Integer key : map.keySet()) {
    String value = map.get(key);
    System.out.println(value);
}
```

دوال شائعة الاستخدام في ال HashMap:

الحصول على قيمة من خلال المفتاح – get()

حذف قيمة من خلال المفتاح – remove()

فحص فيما اذا كانت هناك قيم أو أنه فارغ – isEmpty()

فحص فيما اذا كان هناك مفتاح معين – containsKey()

فحص فيما اذا كانت هناك قيمة معينة – containsValue()

عدد القيم الكلي – size()

حذف جميع العناصر: المفاتيح والقيم – clear()

```
map.get(1);
map.remove(1);
map.isEmpty();
map.containsKey(1);
map.containsValue("eawedat@gmail.com");
map.size();
map.clear();
```

HashSet

كلاس HashSet عبارة عن كلاس يتم من خلاله حفظ أو تخزين قيم. يمكن تشبيه HashSet بجدول يحوي عامود واحد فقط وهذا العامود يحوي قيم معينه لا تتكرر وغير مرتبة (غير منظمه).

```
HashSet<String> set = new HashSet<>();
set.add("Muhamad");
set.add("Eawedat");
set.add("eawedat@gmail.com");
```

نرى هنا طرق مختلفة لطباعة القيم المحفوظة في ال HashSet من خلال حلقات التكرار.

```
for (String temp : set) {
    System.out.println(temp);
}

Iterator<String> It = set.iterator();
while (It.hasNext()) {
    String St = It.next();
    System.out.println(St);
}
```

دوال شائعة الاستخدام في ال HashSet:

contains() – فحص فيما كانت قيمة معينة متواجدة

isEmpty() – فحص فيما اذا كانت هناك قيم أو أنه فارغ

remove() – حذف قيمة معينة

size() – عدد القيم الكلي

clear() – حذف جميع القيم

```
set.contains("eawedat@gmail.com");
set.isEmpty();
set.remove("eawedat@gmail.com");
set.size();
set.clear();
```