**الفصل الثاني**

**مشاريع Visual Basic**

في هذا الفصل سنتعمق أكثر في بناء البرامج، وسنحاول استخدام معظم العناصر الأساسية في اللغة.. إنّ هذا الفصل رؤيةٌ علويّةٌ لمدينتنا البرمجيّة، قبل أن نبدأ في الهبوط على قطاعاتها واحدا فواحدا.. هيا نبدأ فلا وقت نضيّعه!

* [**برنامج "محاسب القروض" Loan Calculator:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\02-%20المشاريع\1-%20برنامج.محاسب%20القروض.htm)

تطبيق لحساب معاملات القروض، نتعلّم من خلاله كيفيّة تصميم واجهة الاستخدام، وكيفيّة التفكير في البرمجة، وكيفيّة استخدام الدوالّ.

* [**إنشاء آلة حاسبة Building Math Calculator:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\02-%20المشاريع\2-%20إنشاء%20آلة%20حاسبة.htm)

غالبا يكون هذا هو أوّل تطبيق كامل ينشئه مبرمجو VB، فللآلة الحاسبة سحر لا يقاوم، فهي تجمع بين السهولة وجمال التصميم.. وبواسطتها نتعلّم كيف نصمّم واجهة استخدام متناسقة مزدحمةٍ بالأأدوات، وكيف نكتب كودا بسيطا يشغّل الأزرار المتشابهة في الوظيفة، وكيف نتّقي الأخطاء التي قد تحدث عند إجراء العمليّات الحسابيّة، وكيف نعالجها لو حدثت، وكيف نعالج الاستثناءات Exceptions.

* [**تشغيل محاسب القروض على الإنترنت:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\02-%20المشاريع\3-%20تشغيل%20محاسب%20القروض%20على%20الإنترنت.htm)

سترى في هذا التطبيق عجبا.. ستنسخ نفس الكود الموجود في محاسب القروض الذي أنشأناه كتطبيق ويندوز، إلى محاسب القروض الذي أنشأناه كتطبيق إنترنت.. وسيعمل.. وبدون أيّ تعديل!

* [**التعامل مع أكثر من نموذج Multiple Forms:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\02-%20المشاريع\4-%20التعامل%20مع%20أكثر%20من%20نموذج.htm)

سننشئ تطبيقا له واجهة نستطيع من خلالها أن نفتح نافذة محاسب القروض أو نافذة الآلة الحاسبة.. وبهذا نرى الاختلافات لتي حدثت في VB.Net عن VB6 في مجال التعامل مع نموذج من نموذج آخر.

* [**التعامل مع العديد من المشاريع:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\02-%20المشاريع\5-%20التعامل%20مع%20العديد%20من%20المشاريع.htm)

هذه خطوةٌ أخرى صغيرةٌ لمبرمج، ونتمنّى أن تكون خطوةً كبيرةً للبشريّة!.. سترى كيف يمكنك إضافة أكثر من مشروع لمتصفّح المشاريع، لتسهيل التعديل في كلّ منها لو تطلّب الأمر ذلك.

* [**بناء تطبيق اختبار Building Console Application:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\02-%20المشاريع\7-%20بناء%20تطبيق%20اختبار.htm)

سيبدو لك الآن أنّك تركب آلة الزمن للعودة إلى الماضي السحيق!.. نعم لقد تطوّر VB لدرجة أنّه صار ينشئ ـ لأوّل مرّة ـ تطبيقات اختبار ذات شاشة سوداء عقيمة تشبه شاشات الدوس المنقرض!

* [**الملفات التنفيذية Executable Files:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\06-%20Basic%20Controls%20الأدوات%20الأساسيّة.mht)

لا عجبَ إن تساءلت في دهشة: وماذا بقي من اللغة لم نتعلّمه في هذا الفصل؟!.. الآن نتعلّم كيف ننشئ برنامج إعداد، نستخدمه لإعداد تطبيقاتنا على أجهزة المستخدمين.

**برنامج "محاسب القروض" Loan Calculator**

* + [**كيف نصمّم محاسب القروض:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\1-%20برنامج.محاسب%20القروض\1-%20كيف%20نصمّم%20محاسب%20القروض.htm)
  + [**تصميم واجهة المستخدم Designing User Interface:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\1-%20برنامج.محاسب%20القروض\2-%20تصميم%20واجهة%20المستخدم.htm)
  + [**برمجة تطبيق القروض Programming Loan Application:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\1-%20برنامج.محاسب%20القروض\3-%20برمجة%20تطبيق%20القروض.htm)
  + [**إجازة البيانات Validating Data:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\1-%20برنامج.محاسب%20القروض\4-%20إجازة%20البيانات.htm)
  + [**كتابة تطبيق حسن الأداء Writing Well-Behaved Applications:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\1-%20برنامج.محاسب%20القروض\5-%20كتابة%20تطبيق%20حسن%20الأداء.htm)

**كيف نصمّم محاسب القروض:**

واحد من البرامج العملية سهلة التنفيذ، هو البرنامج الذي يحسب القروض ومعاملاتها.

إن Visual Basic يمنحك دوالَّ جاهزةً لأداء أنواعٍ كثيرةٍ من الحسابات الماليّة، حتّى إنّك ـ مثلا ـ لن تحتاجُ لأكثر من سطرٍ واحدٍ من الكود لحساب المدفوعات الشهرية بمعرفة قيمة القسط ومدته ومعدّل الفائدة.

(ملحوظة لا بدّ منها: يبدو والله أعلم، أنّ الفائدة هي اسم الدلع للربا في العصر الحديث!!)

وبغض النظر عن لغة البرمجة التي تستخدمها، فإن عليك أن تتّبع هذه الخطوات لتصميم برنامجك:

1. حدّد مهامّ برنامجك، وكيف سيتفاعل مع المستخدم لتأدية هذه المهامّ.
2. صمّم واجهة المستخدم تبعًا للخطوة الأولى.
3. اكتب الكود الذي يستجيب للأحداث المختلفة التي تعنيك (وقد عرفنا في الفصل السابق الخطوات الأساسية للتفكير عند تصميم الكود).

**تصميم واجهة المستخدم Designing User Interface**

* [**الأدوات المطلوبة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20تصميم%20واجهة%20المستخدم\1-%20الأدوات%20المطلوبة.htm)
* [**وضع الأدوات:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20تصميم%20واجهة%20المستخدم\2-%20وضع%20الأدوات.htm)
* [**محاذاة الأدوات Aligning Controls:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20تصميم%20واجهة%20المستخدم\3-%20محاذاة%20الأدوات.htm)

**الأدوات المطلوبة:**

تبعًا للخطوة الأولى، ستجد أن المستخدم يحتاج لتحديد قيمة القسط ومعدّل الفائدة ومدة القرض (بالشهور).. لهذا يتوجّب عليك أن توفّر للمستخدم ثلاثة مربعات نصوص TextBoxes، ليُدخل فيها هذه القيم.



أيضا نحتاج لذكر إذا ما كان الدفع سيتم في أول الشهر أم في آخره.. ستجد أن أنسب وسيلة لإدخال هذه المعلومة هي مربع الاختيار CheckBox، وهو يغيّر حالته Togglesعند ضغطه، فلو كانت عليه علامة الاختيار "✓"، فستؤّدي ضغطة بالفأرة أو مسطرة المسافات إلى إزالتها، وضغطة أخرى إلى إعادتها.

إن المستخدم لا يدخل أي بيانات في هذه الأداة، ممّا سيريحك من احتمال أن تكون المدخلات خاطئة.. كما أن هذه أبسط وسيلة لتحديد قيمة متغيّر له حالتان فحسب (نعم / لا).

وعندما يفرغ المستخدم من كتابة البيانات المطلوبة، يمكنه أن يضغط زر "عرض المدفوعات" Show Payment، لحساب المدفوع شهريا، وعرضه في مربع النص أسفل النافذة.

إذن فالبرنامج كله يندرج تحت حدث ضغط هذا الزر!

**ملحوظة هامّة:**

سنصمّم واجهة هذا البرنامج بالإنجليزية، ولكن بإمكانك تعريبه بمنتهى البساطة، بكتابة جميع اللافتات والعناوين والرسائل بالعربية، مع استخدام خاصيّة "من اليمين لليسار" RightToLeft، وجعلها "صواب" True في كل أداة، لتظهر وتتصرّفَ بطريقةٍ تناسب اللغةَ العربيّة.

**وضع الأدوات:**

ابدأ مشروعًا جديدًا، وامنحه الاسم LoanCalculator.. وحينما يظهر لك النموذج، غيّر اسمه إلى LoanForm باستخدام نافذة الخصائص.

أوّل ما سنفعلُه، هو أن نحدّد الخط الذي سنستخدمه في النموذج، حيثُ سيصبحُ هذا الخط هو الخطَّ الافتراضيَّ لكل ما سنضعه على النموذج من أدوات فيما بعد، مع ملاحظة أنّ بإمكانك تغيير خطّ أيّ أداةٍ بعدَ ذلك في أيّ وقتٍ.

**نصيحة:**

إذا كنت ستغيّر خطّ كل أداةٍ، فلا تحاول محاذاتها ببعضها قبلَ أن تنتهيَ من تغيير الخطوط، حتّى لا يضيع مجهودك في المحاذاة هباءً، وذلك لأنّ تغيير حجم الخطّ قد يؤدّى لتغيير حجم الأداة.

وفي تطبيقنا هذا، سنستخدم خط Verdana، بحجم 10 نقاط.

ولتصميم النموذج، اتبع الخطوات التالية:

1- ضع أربع لافتات Labels على النموذج، واكتب بها النصوص التالية:

|  |  |
| --- | --- |
| خاصية النص Text | Name اسم اللافتة |
| مقدار القرض | Label1 |
| المدة (بالشهر) | Label2 |
| معدل الفائدة | Label3 |
| المدفوع شهريا | Label4 |

وعيك أن تجعل كل لافتة عريضة بما يكفي لتلائم النص المكتوب بها.. أو ببساطة، قم بتغيير خاصيّة "تحجيم تلقائي" AutoSize لكلّ لافتة لتصبح قيمتها True، وبذلك سيتمّ تغيير حجم اللافتة تلقائيّا ليتناسب مع حجم النصّ المكتوب.

ولعلك تتساءل: لماذا لم نُسمِّ لافتاتنا بأسماء برمجية تتماشى مع القواعد التي أوضحناها في الفصل الأوّل؟

والإجابة بسيطة: إن كلَّ ما نحتاجه من هذه اللافتات هو ما تعرضه من عناوين (نصوص)، ولن نستخدمَ أسماءَها في كود هذا البرنامج.

2- ضع مربع نصٍ بجوار كل لافتة، وعدل اسم ونص كل منها كالتالي:

|  |  |  |
| --- | --- | --- |
| مربع النص | الاسم | النص المبدئي |
| TextBox1 | TxtAmount | 25000 |
| TextBox2 | TxtDuration | 48 |
| TextBox3 | TxtRate | 14.5 |
| TextBox4 | TxtPayment |  |

3- في مربع النص الرابع ستظهر قيمة المدفوعات الشهرية كما سيحسبها البرنامج، لهذا فليس من المنطقيِّ أن يكتب المستخدم أيَّ شيءٍ في هذه الخانة.. لحسن الحظِّ أن لدينا خاصيّة لعمل ذلك، وهي خاصيّة "للقراءة فقط" ReadOnly.. اجعل لها القيمة "صواب" True، وحينئذٍ لن يمكنَ الكتابةُ في مربع النصِّ إلا باستخدام خاصية "نص" Text من نافذة الخصائصِ أو من الكود، وهو ما لن يستطيع المستخدم فعلَه!

**ملحوظتان:**

1- طبعًا ستتساءلُ في لماضة: ولماذا لا نستخدم لافتةً Label بدلا من مربع النصّ، وهي تعرض البيانات، ولكنها لا تسمح للمستخدم بالكتابة فيها؟

هذا سؤالٌ وجيه، وهو حلٌّ جيّدٌ فعلا، وإن كان التناسقُ الجماليُّ للنموذجِ سيختلُّ قليلا ـ وهو ما يمكن حلّه، بمنح اللافتة لون خلفية BackColor أبيض، ومنحها مظهرًا مجسّمًا، بوضع القيمة "ثلاثيّ الأبعاد" Fixed3D في خاصيّة "شكل الإطار" BorderStyle، وبهذا تصبح شبيهةً بمربع النص تمامًا!

ولكن ستتبقّى مشكلةٌ واحدة، هي أن المستخدم لن يستطيع نسخ القيمة المكتوبة في اللافتة، لو أراد أن ينقلها إلى تطبيق آخر أو إلى ملفّ.. هذه هي فائدة مربع النصّ في هذه الحالة.

2- اسم خاصية "للقراءة فقط" ReadOnly هو الاسم الجديد لخاصية "مغلق" Locked في VB6.. الجدير بالذكرِ أنَّ خاصية Locked ما زالت موجودة، وإن أصبحت لها وظيفةٌ أخرى، فبجعل هذه الخاصيّة "صواب" True، يتم تثبيت الأداة في موضعها، بحيثُ لا يمكنُك تحريكها أو تغيير حجمها بالفأرة.. ويحضرنا أن نذكر هنا بالمناسبة، أنّ بإمكانك تثبيت كل الأدوات على النموذج مرّة واحدة، بضغط "تنسيق" / "تثبيت الأدوات" Format.Lock Controls.

4- ضع مربع اختيار CheckBox على النموذج، وسمِّه ChkPayEarly.. سيكون عنوانه الافتراضي Check1، وسيظهر على اليمين.

5- نظرًا لبعضِ المعايير الشكلية، نفضّل أن يكون عنوان مربع الاختيار على اليسار (بمعنى أن تكون خانة الاختيار على اليمين).. سنستخدم لذلك خاصيّة "محاذاة خانة الاختيار" CheckAlign.. ستجد أنّ لها القيمة "يسار الوسط" MiddleLeft.. اضغط زر إسدال القائمة.. ستظهر لك القيم المختلفة لهذه الخاصية مرسومة على هيئة مستطيلات.



اضغط المستطيل الموجود في الصفّ الأوسطِ والعمودِ الأيمن.. ستجدُ أن قيمة الخاصية قد صارت "يمين الوسط" MiddleRight.. واضح أن المقطع الأوّل Middle يشيرُ إلى كيفية المحاذاة رأسيًّا (في المنتصف)، بينما المقطع الثاني Right يشيرُ إلى كيفية المحاذاة أفقيًّا (على اليمين).

6- غيّر عنوان مربع الاختيار، بوضع النص "Early Payment" في خاصيّة Text.

7- ضع زرًّا على النموذج، وسمّه BtShowPayment، واجعل عنوانه "Show Payment".

8- وأخيرا، ضع زرا آخر على النموذج، وسمّه BtExit، واجعل عنوانه Exit.

**محاذاة الأدوات Aligning the Controls:**

تأكّد أولا أنّ النصوص على كل اللافتات ظاهرةٌ بالكامل، وذلك بمنح اللافتة عرضًا مناسبًا، حتّى لا يلتفّ النص إلى سطر ثانٍ فلا تتمكن من رؤيته (إلا بزيادة ارتفاع اللافتة).

نصل الآن لمحاولة تنسيق الأدواتِ على النموذج، لتظهر في شكلٍ مقبولٍ للمستخدم.

إنّ بيئة التطوير تمنحك الأوامرَ اللازمة لمحاذاة الأدوات، وكلّها تندرجُ تحت القائمة الرئيسية "تنسيق" Format.

لكي تحاذي الأدوات الموجودة على نموذجنا الحاليّ، اتّبع ما يأتي:

1- حددّ اللافتات الأربعَ على النموذج، وذلك بإحدى الطريقتين التاليتين:

أ. اضغط زر الفأرةِ الأيمن على النموذج وتحرّك به دون ترك الزر.. ستجد أنّ إطارًا مستطيلا يرتسم على النموذجِ، إحدى رءوسِه حيثُ بدأتَ الضغط، ورأسه المقابلة للرأس الأولى، توجد حيثُ يوجدُ مؤشّرُ الفأرةِ الآن.. حاولْ أن تتحّركَ بهذا الإطارِ حتّى يحتوي لافتاتنا الأربع، ثمّ اترك زرّ الفأرة.. ستجدُ أنّ اللافتاتِ الأربعَِ قد حّدّدت.

ب. اضغط زر الفأرةِ الأيمنَ على إحدى اللافتات لتحديدها، ثم اضغط زر Ctrl من لوحة المفاتيحِ ولا تتركه، واضغط باقي اللافتات بالفأرة.. بهذه الطريقة سيتم تحديد كلّ لافتة تضغطها، دونَ إزالة التحديد من على ما سبقها من لافتات.. بعدَ انتهاء تحديد اللافتات، لا تنسَ رفع إصبعك عن زر Ctrl!!.. ولو أردت إزالةَ تحديد إحداها، فاضغطها بالفأرة مرّةً أخرى مع ضغط زر Ctrl.

لعلك لاحظتَ أنَّ هاتينِ الطريقتينِ مماثلتانِ لما نفعلُه لتحديد مجموعة من الملفاتِ أو المجلدات في مستكشف ويندوز Windows Explorer.. هذه هي روعةُ الويندوز، حيثُ أنّ أساليبه قياسيّة، بحيث تستطيع أن تطبّق معظم ما تعرفه في تطبيق ما، على باقي التطبيقات.

2- والآن اضغط "تنسيق" / "محاذاة" / "الحواف اليسرى" Format.Align.Lefts.. واضحٌ أنّ هذه العمليةُ تصفُ نفسَها، فالأسماء واضحة: ستتمُّ محاذاةُ الحواف اليسرى للافتاتِ.. ولكن أيّها ستثبتُ في مكانها، لتتمّ المحاذاة معه؟

ستلاحظُ أنَّ ثلاثَ لافتات ذات مقابض[[1]](#footnote-1)Handles بيضاء، بينما الرابعة ذات مقابض سوداء.. هذه الأخيرةُ هي الأداةُ المرجعيّة، التي تتمُّ المحاذاة معها.. ولتغييرها، اضغط بالفأرةِ على الأداةِ التي تريدها مرجعا للمحاذاة.

3- حدد مربعات النصوص الأربعة، واضغط "تنسيق" / "محاذاة" / "الحواف اليسرى" Format.Align.Lefts.

**إرشاد:**

حاول أن تستكشف وتجرّب بنفسك باقي أوامر القائمة الرئيسية "تنسيق".

**برمجة تطبيق القروض Programming Loan Application**

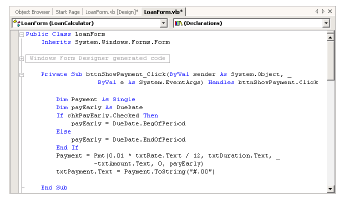
* [**زرّ إنهاء التطبيق:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20برمجة%20تطبيق%20القروض\1-%20زرّ%20إنهاء%20التطبيق.htm)
* [**الدالة الأساسية:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20برمجة%20تطبيق%20القروض\2-%20الدالة%20الأساسيّة.htm)
* [**إجراء حساب المدفوعات:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20برمجة%20تطبيق%20القروض\3-%20إجراء%20حساب%20المدفوعات.htm)

**زرّ إنهاء التطبيق:**

والآن شغّل البرنامج، واختبره.. أدخل بعض النصوص في مربعات النصوص.. وغيّر حالة مربّع الاختيار Check box.

بالتأكيد لن يؤّدي ضغط زر "حساب المدفوعات" Show Payment إلى حدوث أي تأثير، فنحن لم نكتب أي كود حتى الآن.

ابتهج بما يكفي بهذه الواجهة، ثم أغلق البرنامج.. انقر زر "حساب المدفوعات" مرتين، لتفتح نافذة الكود على الإجراء المستجيب لضغط الزر، وقد تكلمنا عن الأحداث ومجيباتها بما يكفي في الفصل الأول.



ستلاحظ أن الجملتين التاليتين تسبقان مجيب ضغط الزر:

**Public Class LoanForm**

**Inherits System.Windows.Forms.Form**

والجملة الأولى تعرّف خليّة Class النموذج الموجود في البرنامج، فكما اتفقنا سابقا، فإن كل شيء سنقابله هو كائن Object، وكل كائن عبارة عن نسخة Instant من خليّة تحتوي على خواصه ووسائله وأحداثه.. هذه الطريقة تمكّننا من استخدام أكثر من نسخةٍ من الخليّة (أكثر من كائن) في نفس اللحظة، لكل منها سماتها المختلفة، بدون بذل مجهودٍ لإعادة إنشاء كلٍّ منها.

كما تمكّننا هذه الطريقة من استخدام الوراثة، وهذا هو دور الجملة الثانية، فهي تورّث الكائن جميع الخصائص والوسائل والأحداث المرتبطة بالنماذج، والتي تمنحها لنا اللغة جاهزة، لرسم النموذج على الشاشة والتحكمِ في شكله ووظيفته.

وعليك ألا تحاول تغيير هاتين الجملتين، إلى أن تفهم مغزاهما جيدًا عندما نشرح الوراثة بإذن الله.

**ملحوظة:**

على عكس VB6، الذي كان لكل نوعٍ من ملفاته امتداد مختلف، فالنموذج مثلا كان له الامتداد ".frm"، ووحدة الكود Module كان لها الامتداد ".bas"، فإن كل ملفات عناصر VB.NET الأساسيّة لها نفس الامتداد، وهو ".VB".. وتعتبر هذه ميزة كبيرة، لأنَّ الملف الواحد يستطيع الآن أن يحتوي على أكثر من نوعٍ مختلفٍ من العناصر، بمجرد كتابة مقطع تعريف كلّ منها.

تعال أولا نكتب حدث ضغط زر إغلاق البرنامج.. إنّه سهل جدًّا، فسوف تكتب به سطرًا واحدًا، به كلمة واحدة، تنهي البرنامج في الحال:

**Private Sub BtExit\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtExit.Click**

**End 'أنهِ التطبيق في الحال**

**End Sub**

والآن سنتفرّغ لكتابة كود زرّ المدفوعات.. تعال نتعرّف على كيفيّة ذلك.

**الدالة الأساسيّة:**

يمكنك استخدام الدالة Pmt() لحساب المدفوعات الشهرية، وكما هو واضح فإنّ حروفها هي اختصار لكلمة **P**ay**m**en**t** أي دفع أو سداد، وهي تُستدعي بجملة كالتالية:

**المدفوع\_شهريّا = Pmt(معدّل الفائدة, عدد الشهور, مقدار القرض \_**

**, القيمة المستقبليّة, استحقاق الدفع)**

عرفنا من قبل أن الدالة تعتبر بمثابة عملية (حسابية)، ينتج عنها قيمة ما.

وعرفنا أيضا أن معظم الدوال تحتاج لمعاملات، يتم إرسالها بين قوسيها، لتتحكم في الطريقة التي تعمل بها الدالة، أو لتكون بمثابة المدخلات التي تنفّذ الدالة حساباتها عليها.

كما عرفنا أن شق الجملة الموجود على يمين علامة التساوي ينفّذ أولا، حيث توضع القيمة الناتجة عنه في المتغير الموجود على يسار علامة التساوي.

إذن فلا بد أنّك تألف تلك الصيغة، حتّى لو كانت الدالة ومعاملاتها جديدة عليك.

**شرح معاملات الدالة:**

**ملحوظة:**

إذا واجهتك أيّ صعوبة في فهم هذا المقطع فلا تبتئس، فليس معنى هذا أنّ اللغة معقّدة، وليس معناه كذلك أنّك غير مؤهّل لفهمهما.. كلّ ما هناك هو أنّ الدالة Pmt التي نستخدمها دالّة إحصائيّة ذات استخدام خاصّ، ولا أدري وجهة نظر مؤلّف المرجع الأصليّ ـ سامحه الله ـ في البدء بدالة كهذه!!.. وإن كانَ الرجلُ مشكورًا قد نصح من لا يستسيغونَ هذا النوع من التطبيقات التجاريّة أن يتجاوزوا هذا المشروعَ وينتقلوا إلى المشروع التالي.. وأنا كذلك لا أعفي نفسي من اللائمة، فقد كان من المفروض أن أغيّر هذا المثال إلى مثالٍ أبسط.. ولقد تردّدت في ذلك طويلا، ثمّ قدّرت أن أمنحك الفرصة للتعرّف على الدوالّ الإحصائيّة لأنّك لن تسمع عنها شيئا بعد ذلك في هذا المرجع (طبعا ستتنفّسُ الصُعَداء!!).

عامّةً لا تَخَفْ.. ليست كلّ دوال VB كهذه الدالة، وإن كان هذا لا يعني ألا تحاول أن تقرأ الجزء القادم، ففيه معلومات هامّة حبذا أن تعرفها، كما أنّك ببعض التركيز ستجد كلّ شيءٍ في غاية الوضوح والبساطة ـ إن شاء الله.

**معدّل الفائدة InterestRate:**

هذا المعامل يمثّل معدل الفائدة المئويّة شهريا.. فمثلا: لو كانت الفائدة **السنوية** 16.5%، يجب أن يكتب المستخدم في مربع معدل الفائدة 16.5، فتكون الفائدة **الشهرية** 0.165 ÷ 12.

**مدة القرض Periods:**

عدد الشهور التي يتم التقسيط عليها.

**قيمة القرض Amount:**

وهو هنا يكون سالبًا، نظرًا لأنّك مدين به.. ولو جعلته بقيمة موجبة، فستكون القيمة الناتجة من الدالة سالبة، لأنّها في هذه الحالة تمثّل ما تقبضه أنت من نقود، لا ما تدفعه.. باختصار: ما هو مستحقٌّ لك من نقود له قيمة موجبة، وما هو مستحقٌّ عليك له قيمة سالبة.

**القيمة المستقبلية FutureValue:**

وهي خاصة بحسابات الاستثمار، وفي حالتنا هذه قيمتها صفر.. ويمكن ألا ترسل هذا المعامل، فهو معامل اختياريّ Optional، حيثُ ستقوم الدالة في هذه الحالة باستخدام قيمة افتراضية Default Value ، وهي الصفر.

**استحقاق الدفع Due:**

وهو أيضًا معاملٌ اختياريّ، فيمكنك عدم إرساله.. وهو يخبر الدالةَ متى تكون المدفوعات مستحقة الدفع، وهي تأخذ واحدةً من القيمتين:

- بداية المدّة DueDate.BegOfPeriod.. وهذه هي القيمة الافتراضية، التي تستخدمها الدالة لو لم تُرسل لها هذا المعامل.

- نهاية المدة DueDate.EndOfPeriod.

وهاتان القيمتان هما تعبيران من صميم اللغة Built-in، ولكل تعبير منهما رقم يناظره.. إن وجود الأرقام في الكود قد لا يدل على معنى محدد، ممّا يشكّل صعوبةً في فهم الكود عند مراجعته.. لهذا يمكنك استخدام هذه التعبيرات الواضحة ـ والتي تسمّى المُرَقَّمات Enumerations ـ لتجعل برنامجك مقروءًا وواضحًا.. وسنتعرف فيما بعد على كيفية بناء المُرَقّمات الخاصة بنا.

وليس عليك معرفة كيف تقوم دالة المدفوعات Pmt() بالعمليات الحسابية.. كل ما عليك هو استدعاؤها ومنحها المعاملات المناسبة، وستعيد هي الناتج إليك.

**مثال:**

لحساب المدفوعات الشهرية على قرض قيمته 25000، بفائدة 14.5%، يسدّد على 48 شهرًا، في نهاية كل شهر، يمكنك استدعاء دالة المدفوعات Pmt() كالتالي:

**Console.WriteLine(Pmt(0.145/12, 48, -25000, 0, DueDate.EndOfPeriod))**

وبما أنَّ المعاملين الأخيرين اختياريّان ـ كما ذكرنا، فيمكنك أن تستدعي دالة المدفوعات كالتالي:

**Console.WriteLine(Pmt(0.145 / 12, 48, -25000))**

ستجد أن القيمة 689.4488212872186 قد كُتبتْ في شاشة المخرجات Output Window.

والآن تعال نكتب حدث ضغط زر الحسابات.

**إجراء حساب المدفوعات:**

اضغط زرّ حساب المدفوعات مرّتين بالفأرة، حتّى تفتح نافذة محرّر الكود على الإجراء المستجيب لضغط هذا الزرّ.. اجعل هذا الإجراء يبدو كما يلي:

**Private Sub BtShowPayment\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtShowPayment.Click**

**Dim Payment As Single**

**Dim PayEarly As DueDate**

**If ChkPayEarly.Checked Then**

**PayEarly = DueDate.BegOfPeriod**

**Else**

**PayEarly = DueDate.EndOfPeriod**

**End If**

**Payment = Pmt(0.01 \* TxtRate.Text / 12, TxtDuration.Text, \_**

**-TxtAmount.Text, 0, PayEarly)**

**TxtPayment.Text = Payment.ToString("#.00")**

**End Sub**

**شرح الإجراء:**

- تعرفُ بالطبع أنّ أوّل سطرين في هذا الإجراء يقومان بتعريف متغيّرين.. الجديد هنا هو نوعا هذين المتغيرين:

فالمتغير الأوّل من النوع "مفرد" Single، ممّا يعني أنّه متغيّر رقميٌّ يقبل العلامة العشرية (مثل 23.5).

أما المتغير الثاني، فهو من النوع DueDate.. وهذا النوع ليس من أنواع المتغيرات البسيطة مثل النصوص Strings والأعداد الصحيحة Integers وغيرها، ولكّنه أحد المرقِّمات Enums، وقد صادفنا متغيرا من هذا النوع من قبل في أحد معاملات الدالة Pmt()، وهو المعامل الأخير، المسئول عن تحديد ميعاد الدفع.. ونحنُ هنا نحتاجُ لمتغيّرٍ من نفس هذا النوعِ حتّى نرسلَه للدالةِ في موضع المعامل الأخير، حيثُ يجبُ أن تتناسق أنواع المتغيرات المرسلة، مع أنواع المعاملات التي تستقبلها الدالة، وإلا حدث الخطأ البرمجيّ "الأنواع غير متوافقة" Type mismatch.

- بعد ذلك لدينا جملة شرط، لمعرفة قيمة مربع الاختيار ChkPayEarly:

- فإذا كانت قيمة الخاصية "مختارًا" Checked "صواب" True، فهذا يعني أنّ الدفع في أول كل شهر.. هنا سنضع القيمة DueDate.BegOfPeriod في المتغير PayEarly، لندل على ذلك.

- وإذا كانت قيمة تلك الخاصية غير ذلك Else، ممّا يعني أنّها خطأ (فليس لدينا هنا سوى قيمتين هما: "صواب" و"خطأ")، فإنَّ هذا يعني أنَّ الدفع في آخر كلّ شهر.. هنا سنضع القيمة DueDate.EndOfPeriod في المتغير PayEarly، لندل على ذلك.

**ملاحظة:**

إنَّ القيمة "صواب" True هي القيمة الافتراضية للشرط Condition الذي نختبره في جملة الشرط، لهذا يمكن عدم كتابتها، أي أنَّ الجملة:

**If ChkPayEarly.Checked Then**

هي اختصار للجملة:

**If ChkPayEarly.Checked = True Then**

وكلتاهما صحيحةٌ ويمكنُ استخدامها.

- الآنَ نستطيع استدعاء دالة المدفوعات Pmt()، مع إمدادها بالمعاملات اللازمة، وهي القيم التي أدخلها المستخدم في مربعات النصوص، واستقبال القيمة المعادة Return Value من الدالة في المتغيّر Payment.

الملاحظة الوحيدة هنا، هي أنّنا قد ضربنا قيمة مربع النص TxtRate في 0.01 (أي قسمناه على 100) ثم قسمناه على 12، وذلك حتّى نحوّل الرقم الصحيح المدخل في خانة النسبة المئويّة السنويّة إلى رقم عشري يدل على النسبة المئويّة الشهريّة.

- بقيَ أن نعرضَ الناتجَ في مربّع النص الخاصّ به، ولكن علينا أولا أن نحدد الدقة العشرية بخانتين، لأن النقود لا تُدفع بأقل من السنت (أو القرش).. هنا سنستخدم دالة التحويل إلى نص ToString، الموجودة في كل الكائنات الرقمية، مثل المتغير Payment في حالتنا هذه.. هذه الدالة تحوّل الأرقامَ إلى نصوصٍ، بالتنسيق الذي نريده.. والعلامة "#" تدل على الجزء الصحيح من الرقم، وتليها العلامة العشريّة "."، ويليها صفران، لإخبار الدالة أنّ الرقم سيتم تقريبه لأقرب خانتين عشريتين.

**إجازة البيانات Validating the Data:**

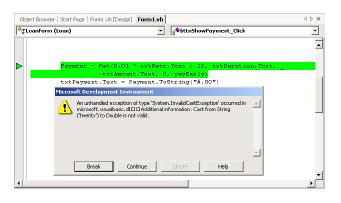
ماذا سيحدث لو كتب المستخدم اسمه مثلا، بدلا من الأرقام في أي مربع نص ونفذ البرنامج؟

بالتأكيد سيحدث خطأ برمجيّ وسينهار البرنامج ويغلق.. حينئذ سيستاء منك المستخدم كثيرا، رغم أنّه هو الذي تسبّب في الخطإ نتيجةً للمدخلات الخاطئة.

معنى هذا أنَّ عليك أن تتوقع أنواع الخطإ المختلفة، التي يمكن أن تتسبّب فيها البيانات الخاطئة، بحيث لا تسمح للمستخدم بإدخالها، أو تطلب منه تصحيحها، دون أن يُغلَق البرنامج بسبب خطإ مفاجئ.

**إرشاد:**

اكتب كلمة "عشرون" في مربع النص الأول، واضغط زر حساب المدفوعات.. ستظهر لك رسالة خطأ.. اضغط زر "التوقف المؤقت" Break.. ستجد أن نافذة الكود قد فتحت، وأن الجملة المتسببة في الخطإ قد تم تحديدها باللون الأخضر.. يمكنك الآن أن تشير بمؤشر الفأرة على أي متغير، ليظهر لك تلميح يعرض قيمته.. كما يمكنك كتابة أي أمر في نافذة الأوامر Command Window، وتنفيذه بضغط زر الإدخال Enter، وكأنه جزء من البرنامج نفسه.



تعالَ نرى كيف نعالج مثل هذه الأخطاء:

واحدة من الطرق، هي اختبار محتوى كل أداةٍ قبل تنفيذ البرنامج.. فإذا كانت إحدى القيم غير صالحة، يتم عرض رسالة للمستخدم تطالبه بإعادة إدخالها.. وبالطبع لا يتم تنفيذ أي حسابات، قبل أن تكون كل البيانات صالحة.. هذا الكود الذي يفعل ذلك:

**Private Sub BtShowPayment\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtShowPayment.Click**

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Validating Data إجازة البيانات \_\_\_\_\_\_\_\_\_\_\_\_\_**

**' تمت إضافة هذا الجزء للتأكد من صلاحية البيانات المدخلة قبل تنفيذ البرنامج.**

**'-------------------------------------------------------------------------------------**

**' سنعرّف الآن بعض المتغيرات، لنضع فيها القيم الموجودة في مربعات النصوص**

**Dim LoanIRate As Single**

**Dim LoanDuration As Integer**

**Dim LoanAmount As Integer**

**'--------------------------------------------------------**

**' Validate amount التأكد من صلاحية قيمة القرض**

**'--------------------------------------------------------**

**If IsNumeric(TxtAmount.Text) Then**

**LoanAmount = TxtAmount.Text**

**Else**

**MsgBox("قيمة القرض غير مناسبة")**

**Exit Sub 'غادر الإجراء بدون إكمال تنفيذه**

**End If**

**'---------------------------------------------------------------**

**' Validate interest rate التأكد من صلاحية معدل الفائدة**

**'---------------------------------------------------------------**

**If IsNumeric(TxtRate.Text) Then**

**LoanIRate = 0.01 \* TxtRate.Text / 12**

**Else**

**MsgBox("قيمة الفائدة غير مقبولة")**

**Exit Sub**

**End If**

**'------------------------------------------------------------------**

**' Validate loan’s duration التأكد من صلاحية مدة القرض**

**'------------------------------------------------------------------**

**If IsNumeric(TxtDuration.Text) Then**

**LoanDuration = TxtDuration.Text**

**Else**

**MsgBox("حدد مدّة القرض بالشهور")**

**Exit Sub**

**End If**

**'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''**

**' لعلك لاحظت أن الكود الخاص بإجازة البيانات أطول بالفعل من الكود الذي**

**' ينفّذ وظيفة البرنامج نفسه.. هذا هو العالم الحقيقيّ يا صديقي!**

**'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''**

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ إجراء الحسابات \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**' لو كانت كل القيم المدخلة صحيحة، فسيصل البرنامج إلى هذا الموضع، وسيستمر**

**' تنفيذه.. لاحظ أن هذا هو نفس الكود الذي كتبناه من قبل، لحساب المدفوعات الشهرية**

**' ولكن مع استخدام المتغيرات التي تحمل قيم مربعات النصوص،**

**' لإرسالها كمعاملات للدالة، بدلا من مربعات النصوص نفسها.**

**'-------------------------------------------------------------------------------------**

**'لاحظ أن جملة تعريف المتغيرات تأتي في أي موضع من الإجراء**

**Dim Payment As Single**

**Dim PayEarly As DueDate**

**If ChkPayEarly.Checked Then**

**PayEarly = DueDate.BegOfPeriod**

**Else**

**PayEarly = DueDate.EndOfPeriod**

**End If**

**Payment = Pmt(LoanIRate, LoanDuration, -LoanAmount, 0, PayEarly)**

**TxtPayment.Text = Payment.ToString("#.00")**

**End Sub**

لقد استخدمنا منهجًا بسيطًا للتحقّق من صحّة المدخلات، وذلك بالتأكّد فحسب، ممّا إذا كانت قيمًا رقمية أم لا.

إن الدالة "إنّه رقميّ" IsNumeric()، هي واحدة من الدوال الجاهزة التي يمنحها لك VB.. وهي تأخذ معاملا واحدا، هو ذلك النصّ ـ أو المتغيّر أو الكائن أو الخاصيّة التي تحمل النصّ ـ الذي تريد أن تتأكّد من أنّه يتكوّن فقط من مجموعة من الأرقام، بحيثُ لا يحتوي على أي حرفٍ أو رمز غيرها، وبهذا يمكن معاملته كعدد وليس كنص.. لو كان النص عددا، تُرجع الدالة "صواب" True.. وإلا أرجعت "خطأ" False.

فإذا كانت قيمة مربع النص رقميّة، نضع قيمته في المتغير الذي خصصناه لذلك، وإلا عرضنا رسالة لتحذير المستخدم، وأنهينا تنفيذ الإجراء في الحال، باستخدام جملة Exit Sub.

**ملاحظة:**

التعبير Exit من التعبيرات الهامة جدا، حيث إنّه يمكنك من الخروج من المقاطع التي لا تريد الاستمرار في تنفيذها.. ولقد رأينا بالفعل جملة Exit Sub.. ولكنّها ليستِ الوحيدة.. هناك أيضًا:

|  |  |
| --- | --- |
| صيغة المقطع | تعبير الخروج من المقطع |
| **Function**  **End Function** | **Exit Function** |
| **Do**  **Loop** | **Exit Do** |
| **While**  **End While** | **Exit While** |
| **For**  **Next** | **Exit For** |

وسنتعرف على المقاطع Do..Loop و While.. End While و For..Next في الفصل الرابع إن شاء الله.

والآن شغّل البرنامج، واختبر أداءه في حالة إدخال بيانات خاطئة.

**ملحوظة:**

هناك أداة يمكن إضافتها للنموذج، لتعطي للمستخدم تنبيها بوجود خطإ في قيمة أحد مربعات النصوص.. وستجد هذه الأداة في صندوق الأدوات باسم: ErrorProvider.. ولمزيد من التفاصيل، راجع [أدوات التحقّق من الصحّة](file:///C:\Users\Administrator\Desktop\05-%20برمجة%20الإنترت\1-%20مقدّمة%20لبرمجة%20الإنترنت\6-%20أدوات%20الإنترنت\2-%20أدوات%20التحقّق%20من%20الصحّة\9-%20هل%20توجد%20أدوات%20ويندوز%20للتحقّق%20من%20الصحّة.htm) في الفصل رقم 19.

**كتابة تطبيق حسن الأداء Writing Well-Behaved Applications:**

بالنسبة لتطبيق صغير كمحاسب القروض، لن تشعر بخطورةِ انهيار البرنامج وانغلاقه فجأةً، حيث ستقول في نفسك: وماذا على المستخدم لو أعاد فتح البرنامج ليُدخل البيانات صحيحةً تلك المرة؟!

ولكنّ الوضع سيختلفُ تمامًا، لو حدث مثل هذا الأمر في برنامجٍ ضخم، استغرق إدخال البيانات فيه وقتًا طويلا من المستخدم.. في هذه الحالةِ سيستشيط المستخدم غضبًا، وسينعت مصمم البرنامج بالغباء ـ وهو الوصف المفضّل لديّ شخصيّا في مثل هذه المواقف! ـ وربما لو كنتَ بجواره لقام بخنقك بيديه!!

إن إنتاج برنامجٍ راسخ الأداء هي مسئوليتك بالدرجة الأولى، ويجب عليك أن تراعي فيه ميل المستخدمِ للكسل، ورغبته في أن يفعل الكمبيوتر كل شيء، وفي الحصول على واجهة استخدامٍ جذّابة وسهلة الاستعمال، ولا تسمح له بالخطإ.. إنَّ هذه المهامَّ ستستهلك منك أضعافَ الوقتِ الذي يستغرقه إنجاز وظيفة برنامجك الأساسيّة.. ولكن ما باليد حيلة.. أعانك الله!

ولِتفهمَ ما أعنيه، لا تنصدم إذا أخبرتك أنَّ برنامج محاسب القروض لم ينته بعد، فما زالت به بعض إمكانيات الخطإ!

شغّل البرنامج، واكتب رقمًا هائلا كقيمةٍ للقرض، وليكن 99999999999999999.. اضغط زر حساب المدفوعات.. ستجدُ أن رسالة خطإٍ قد ظهرت من نوعية "تجاوز الحد" Overflow، وهو خطأ يحدث عندما تحاول وضع رقمٍ ضخمٍ جدا في متغير رقميّ لا يستطيع استيعابه.. اضغط زر "التوقف" Break، لتتوقف عند السطر الذي سبّب الخطإ.

في الواقع، إن إدخال أيّ قيمة أكبر من 2.147.483.647 في أي مربع نصٍ في برنامجنا، سيؤدي إلى حدوث حالة "تجاوز الحد "Overflow، وذلك لأنّ هذه القيم توضع في متغيراتٍ من النوع "عدد صحيح" Integer، وأقصى عدد يقبله هذا النوع هو ذلك المذكور سالفا.. ولتلافي هذه الحالة، غيّر تعريف المتغيرات من النوع "عدد صحيح" إلى النوع "عدد مفرد" Single، حتّى تستطيع استيعاب قيمٍ أكبر، وحتّى تقبلَ الأعداد العشريّة.

**قاعدة:**

لا يمكن التخلص من مسببات حالة "تجاوز الحد" Overflow عن طريق "إجازة البيانات" Data Validation، وهناك احتمال أن تنتج أخطاء رياضية كثيرة في برنامجك، وذلك ببساطةٍ لأنّك لا تستطيع معرفة النتائج قبل أن تنفّذ الحسابات!

في هذه الحالة، نحتاج لاستخدام ما يسمّى: "معالجة الأخطاء" Error Handling، وهو كود إضافيّ، بإمكانه معالجةُ الأخطاء **بعد** أن تحدث بالفعل، وفي هذه الحالة لن يكونَ منطقيًّا أن تطالب VB بالتوقف عن تنفيذ البرنامج، ولكنك ستطالبه بأن ينفذ بعض الأوامر التي يمكنها معالجة الخطإ الذي حدث.

كيف؟.. سنعرف هذا في المشروع التالي.

**إنشاء آلة حاسبة Building Math Calculator**

* + [**تصميم واجهة الاستخدام Designing the User Interface:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20إنشاء%20آلة%20حاسبة\1-%20تصميم%20واجهة%20الاستخدام.htm)
  + [**برمجة تطبيق الآلة الحاسبة Programming the MathCalculator App:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20إنشاء%20آلة%20حاسبة\2-%20برمجة%20تطبيق%20الآلة%20الحاسبة.htm)
  + [**أدوات التصحيح Debugging Tools:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20إنشاء%20آلة%20حاسبة\3-%20أدوات%20التصحيح.htm)
  + [**إضافة المزيد من الميزات Adding More Features:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20إنشاء%20آلة%20حاسبة\4-%20إضافة%20المزيد%20من%20الميزات.htm)
  + [**معالجة الاستثناءات Exception Handling:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20إنشاء%20آلة%20حاسبة\5-%20معالجة%20الاستثناءات.htm)

**تصميم واجهة الاستخدام Designing the User Interface:**

تطبيقنا هذا أكثر تقدّمًا من سابقه، وهو سيظهر لنا مدى السهولة التي يمنحها لنا VB، في إنشاء تطبيقات ذات واجهات تبدو معقدةً كالتي في الشكل:



ومهمّة تطبيقنا، هو أن يعمل كآلة حاسبة، تقوم بالعمليات الرياضية البسيطة: الجمع والطرح والضرب والقسمة.. ويمكنك فيما بعد أن تضيف عمليات أعقد، كالدوال المثلثية واللوغاريتمات، وهي سهلة البرمجة، لأن VB يمنحك الدوال الجاهزة التي تحسبها ـ كما سنرى فيما بعد.

وبرغم سهولته، إلا أن تصميم واجهة الاستخدام سيتطلب بعض الوقت والجهد، لأنك ستتعامل مع العديد من الأزرار، وستحتاج لتنسيقها في شكل جماليّ، لتبدوَ أقرب ما تكون للآلة الحاسبة المعروفة.

ابدأ مشروعًا جديدًا وأسمه MathCalculator، وسمِّ نموذجه الرئيسيّ CalculatorForm.

والآن اتبع هذه الخطوات:

1- اجعل للنموذج الخط Verdana بحجم 10 نقاط، لترثه أي أداةٍ توضع عليه فيما بعد.

2- أضف لافتةً لاستخدامها كشاشة الآلة الحاسبة، وسمّهاLblDisplay ، وغيّر خاصية "طراز الإطار" BorderStyle إلى "ثلاثي البعد" Fixed 3D، وغير كذلك لون خلفيتها BackColor ولون الخط ForeColor.

3- ضع زرًّا واجعله يعرض النص "1"، وسمّه Bt1.. غير حجمه بعنايةٍ حتّى يصبح العنوان الذي يعرضه في مركزه تمامًا.. إن باقي الأزرار ستكون نسخًا من هذا الزر، فاتعب قليلا في تصميمه، حتّى يسهل عليك باقي العمل.. بعد أن تنتهي، حرّك الزر إلى موضعه النهائيّ على النموذج.

4- اضغط الزر بزر الفأرة الأيمن، واختر "نسخ" Copy من القائمة الموضعية.. لقد تم نسخ الزر إلى لوحة القصاصات Clipboard، ويمكنك لصق أي عددٍ تريده منه إلى النموذج.

5- اضغط في أي موضع على النموذج بزر الفأرة الأيمن، واختر "لصق" Paste من القائمة الموضعية.. ستجد أن نسخةً جديدةً من الزر قد لصقت على النموذج، فوق الزر الأصليّ، ولها نفس العنوان ـ "1" ـ ولكن اسمها Button1.

6- سمِّ الزر الجديد Bt2 وضع فيه النص "2".. والآن حرّك هذا الزر إلى يمين الزر الأوّل، ولا تهتمّ بضبط محاذاتهما الآن.. أجّل ذلك لحين وضع كل الأزرار.

7- كرّر الخطوتين 5 و 6 عددًا كافيًا من المرات، حتّى تنتهي من وضع أزرار لكل الأرقام، مع مراعاة وضع اسم وعنوان الزر بما يناسب الرقم الذي سيعرضه.. عامّةً يمكنك تسمية الأزرار كما تحبّ، لأن المهم هو نص كل منها، لأنه هو الذي سيعرض في شاشة الآلة الحاسبة.

8- أضف زرين جديدين، أحدهما للمسح Clear، واجعل له النص "C" وسمّه BtClear، والآخر للعلامة العشريّة، وسمّه BtPeriod.. ونظرًا لأن العلامة العشرية صغيرة، حاول أن تكبّر حجم خطها، حتّى يمكن رؤيتها بوضوح.

9- الآن نحاذي الأزرار، باستخدام القائمة الرئيسية "تنسيق" Format:

أ. أولا سنحاذي أزرار الصف الأوّل.. وسنبدأ المحاذاة، بمحاذاة الزر "1" مع الحافة اليسرى من اللافتة.. ثم بعد ذلك، حدد كل أزرار الصف الأول، واضغط "تنسيق" / "المسافات الأفقية" / "اجعلها متساوية" Format.Horizontal Spacing.Make Equal، وذلك لمساواة الأبعاد بين كل الأزرار.. بعد ذلك افعل المثل مع أزرار العمود الأوّل، مع جعل الأبعاد الرأسية متساوية، بضغط "تنسيق" / "المسافات الرأسية" / "اجعلها متساوية" Format.Vertical Spacing.Make Equal.

ب. اختر الأزرار الثلاثة في الصف الثاني، وحاذِ حوافّها العليا Top باستخدام الزر الأوّل كمرجع للمحاذاة.. افعل المثل مع الصفين الثالث والرابع.

9- والآن أضف أزرار العمليات الحسابية، وحازِها.

10- أضف زر الناتج "="، وأسمه BtEquals، واجعله في عرض الأزرار الثلاثة التي تعلوه.

**ملاحظة:**

عند إجراء أي عملية محاذاة، يمكن التراجع عنها بضغط "تحرير" / "تراجع" Edit.Undo أو ضغط زري Ctrl+z من لوحة المفاتيح.. في الواقع، إن أيّ شيءٍ تفعله في تطبيقك يمكن التراجع عنه بهذه الطريقة، مثل التراجع عن الكتابة في محرر الكود، أو حتّى التراجع عن تغيير قيمة إحدى الخصائص في نافذة الخصائص.. ولإعادة ما تم التراجع عنه، اضغط "تحرير" / "إعادة" Edit.ReDo.

**برمجة تطبيق الآلة الحاسبة**

* [**برمجة الأرقام Digits:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20برمجة%20تطبيق%20الآلة%20الحاسبة\1-%20برمجة%20الأرقام.htm)
* [**العمليات الحسابية Math Operations:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\2-%20برمجة%20تطبيق%20الآلة%20الحاسبة\2-%20برمجة%20العمليات%20الحسابية.htm)

**برمجة الأرقام Digits:**

انقر مرتين على الزر "1"، ليظهر لك المستجيب لحدث ضغطه.

**Private Sub Bt1\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Bt1.Click**

**End Sub**

أول ما سيجول بخاطرك، هو أننا سنكتب الكود الخاص بكل زر من أزرار الأرقام العشرة.. لا لن نفعل ذلك، فبخلاف أنه شاق، فإن من العبث أن نعيد نفس الكود عشر مرات، في الوقت الذي تتماثل فيه وظيفة كل الأزرار العشرة.

إذن ماذا سنفعل؟

كل ما علينا، هو أن نضيف أسماء الأزرار العشرة تفصلها فاصلة "," بعد كلمة "يستجيب لـ" Handles في تعريف الإجراء السابق.. في هذه الحالة، سيستجيب هذا الإجراء لحادث ضغط أيٍّ من الأزرار العشرة.

والآن نحتاج اسمًا أكثر منطقيةً للإجراء، بدلا من Bt1\_Click.. فلنسمه "ضغط رقم" Digit\_Click.

والآن عدل تعريف الإجراء ليبدو كالتالي:

**Private Sub Digit\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Bt1.Click, Bt2.Click, \_**

**Bt3.Click, Bt4.Click, Bt5.Click, Bt6.Click, \_**

**Bt7.Click, Bt8.Click, Bt9.Click, Bt0.Click**

**End Sub**

والمطلوب الآن هو ما يلي:

عند ضغط أحد الأرقام، لا بد أن يظهر على شاشة الآلة الحاسبة.

هذه هي كيفية فعل ذلك:

**LbDisplay.Text += Sender.Text**

وهذا السطر يُضيف الرقم المضغوط إلى نهاية ما هو مكتوب في اللافتة.

**ملحوظة:**

الجملة السابقة مكافئة للجملة التالية:

**LbDisplay.Text = LbDisplay.Text + Sender.Text**

وهي كذلك مكافئة للجملة التالية:

**LbDisplay.Text = LbDisplay.Text & Sender.Text**

أي أن الرمز "&" ـ الذي يستخدم مع النصوص فقط ـ يكافئ علامة الجمع "+".. بينما الرمز "+=" هو اختصار لعملية إضافة نص جديد على النص الموجود حاليا في المتغيّر، وهو لا يستخدم مع النصوص فقط، بل يستخدم في عمليات جمع الأرقام أيضا، فمثلا، لو لدينا متغير رقمي I، وأردنا أن نجمع 3 على قيمته الحالية ـ ولتكن 5 ـ يمكننا استخدام الجملة:

**I += 3**

والتي تكافئ:

**I = I + 3**

وبعد تنفيذ أي من الجملتين، ستصبح قيمة I = 8.

ولكن ما معنى "المرسِل" Sender؟

إن هذا هو أحد المعاملات التي يستقبلها الإجراء المستجيب للحدث، وهو يعبّر عن الأداة التي تسبّبت في إطلاق الحدثِ واستدعاء الإجراء.. وفي حالتنا هذه، هو الزر الذي تم ضغطه.

وهنا يجب أن تعرف شيئًا هامًّا: إن كائن "المرسل" Sender هو كائن عام، يمكن أن يكون أيَّ شيء، نموذجا أو أداة.. وفي هذه الحالة لا يستطيع مترجم الكود Compiler أن يعرف نوع هذا الكائن على وجه الخصوص، قبل تنفيذ البرنامج.. لهذا عندما تكتب Sender.، وتظهر لك قائمة الأعضاء، لن تجد فيها خاصية Text.. بل لن تجد فيها إلا دالة واحدة هي "اقرأ نوع الكائن" GetType.. لهذا فإن استخدام التعبير Sender.Text يعتبر استخدامًا خطيرًا، ولا ينصح به ـ رغم أنّه سيعمل، والأفضل أن توضّح لمترجم الكود النوع الذي تتعامل معه أولا، بجملةٍ كالتالية:

**CType(sender, Button).Text = "1"**

والدالة CType تقوم بتحويل نوع "المرسل" من كائن عام غير محدد (وهو المعامل الأول للدالة)، إلى النوع الذي نريده (وهو المعامل الثاني للدالة)، وهو هنا الزّرّ.. واضح أن اسم هذه الدالة هو اختصار لجملة Convert Type، أي "تحويل النوع".

هذه الطريقة هي ما ننصحك باتّباعه دائمًا.. إن أول فائدة ستجنيها، هي أن قائمةً بكل أعضاء مربع النص، ستظهر لك فور أن تكتب النقطة "." بعد إغلاق قوس دالة التحويل، ومن هذه القائمة، يمكنك اختيار الخاصية Text.. تمامًا كأنّك تتعامل مع مربع النصّ.

ما زال العمل في كتابة كود أزرار الأرقام لم ينته بعد.. فعند ضغط زر أي عملية حسابيّة، يكون الحدّ الأوّل قد كتبَ بالكامل، ونتوقع أن ما يُكتب بعد ذلك هو الحد الثاني للعملية، وعلى هذا يجب أن نمسح ما على الشاشة بمجرد ضغط أول رقم يلي زر العملية الحسابية، بحيث يكون هذا الرقم هو الوحيد الذي يظهر على الشاشة.. هذا هو الإجراء الذي ينفذ هذا:

**Private Sub Digit\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Bt1.Click, Bt2.Click, \_**

**Bt3.Click, Bt4.Click, Bt5.Click, Bt6.Click, \_**

**Bt7.Click, Bt8.Click, Bt9.Click, Bt0.Click**

**If clearDisplay Then**

**LbDisplay.Text = ""**

**' علامتي تنصيص فارغتين، معناها عرض نص طوله صفر: أي مسح النص.**

**ClearDisplay = False**

**End If**

**LbDisplay.Text += CType(Sender, Button).Text**

**End Sub**

سيدور بيننا هذا الحوار:

ستسألني:

- ماذا يعني ClearDisplay الذي تستخدمه في الكود؟

وسأقول لك باقتضاب:

- إنه متغير.

وسترد في استياء:

- لقد لاحظت ذلك.. هل تظنني غبيا؟.. وهو كذلك من النوع الذي يقبل القيمتين "صواب" True و"خطأ" False.

* هذا النوع يسمّى "منطقي" Boolean، نسبةً إلى العالم الرياضيّ (جورج بوول).

- ولكنك لم تقل ذلك من قبل.

- هاأنذا قد قلته!

- ما زال هناك شيءٌ يحيّرني.. أين تعريف هذا المتغيّر؟

وهو سؤالٌ في محلّه.. هذا المتغير مرتبط بأحداثٍ كثيرة في البرنامج، ولا يمكن تعريفه في هذا الإجراء.. إنّ كل متغيّر له نطاق Scope، خارج هذا النطاق يصبح المتغير غير مرئيٍّ، وكأنّه ليس موجودًا.. لهذا فالمتغيرات التي يتم تعريفها داخل إجراء ما، هي متغيرات خاصة بهذا الإجراء فحسب، ولا يمكن التعامل معها من أي موضعٍ آخر في البرنامج.. ولكن المتغير الذي يتم تعريفه على مستوى الخليّة Class، يمكن التعامل معه على مستوى جميع إجراءات ودوال وخصائص الخليّة.. لهذا سنعرّف المتغير ClearDisplay بعد جملة الوراثة Inherits في خليّة النموذج، وبهذا يصبح مرئيًا للنموذج كله.

**Public Class LoanForm**

**Inherits System.Windows.Forms.Form**

**Dim ClearDisplay As Boolean**

لاحظ أنّ برنامجنا به نموذج واحد فقط، ولكن لو كان به أكثر من نموذج، فإنّ هذا المتغيّر لن يكون مرئيا ولا يمكن استخدامه من أي نموذج غير الذي تم تعريفه به.

والآن، ما على إجرائنا، سوى أن يتأكّد من قيمة هذا المتغيّر، ليعرف إن كان من المحتّم أن يمسح الأرقام الموجودة على الشاشة أم لا.. وطبعًا بعد أن يمسحها، عليه أن يجعل قيمة ClearDisplay "خطأ"، لأنّك لن تحتاج لمسح الشاشة بعد مسحها بالفعل.. أمّا بالنسبة لجعل هذا المتغير "صواب"، فهي مهمة تقع على عاتق أزرار العمليات الحسابية وزر إظهار النتيجة "="، لأن ضغط أي منها يشير إلى انتظار إدخال عدد جديد على الشاشة.

والآن سنكتب الكود الذي يمسح ما على الشاشة من أرقام، ليسمح للمستخدم بتصحيح الخطإ.. انقر الزر "C" مرتين، واكتب فيه ما يلي:

**Private Sub BtClear\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtClear.Click**

**LbDisplay.Text = ""**

**End Sub**

نفرغ الآن لزر العلامة العشريّة.. وهي هنا تعمل كأيّ رقم، ولكن مع استثناء واحد: أنّها لا يمكن أن تُكتب في أي عدد أكثر من مرة واحدة.

هذا هو الكود الخاص بها:

**Private Sub BtPeriod\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtPeriod.Click**

**If LbDisplay.Text.IndexOf(".") = -1 Then**

**' لا توجد علامة عشرية**

**'إضافة علامة عشرية في نهاية النص الحالي**

**LbDisplay.Text = LbDisplay.Text & "."**

**End If**

**End Sub**

لدينا هنا وسيلة جديدة هي "موضع" IndexOf، وهي من الوسائل الخاصة بالمتغيرات والخصائص النصّيّة، ووظيفتها أن تبحث عن مكان وجود الحرف المرسل لها كمعامل، في النص الذي تنتمي إليه.. فإن وجدته، أرجعت لنا رقم أول موضع يوجد فيه، وإن لم تجده أرجعت -1 (وذلك لأنّ موضع أوّل حرف في النصّ رقمه صفر وليس 1!!).

**برمجة العمليات الحسابية Math Operations:**

الآن سنعرّف ثلاثة متغيرات:

|  |  |
| --- | --- |
| الحد الأوّل للعملية الحسابية. | Operand1 |
| علامة العملية الحسابية | Operator |
| الحدّ الثاني للعملية الحسابية. | Operand2 |

- عندما يضغط المستخدم أحد أزرار العمليات الحسابية، يتم تخزين العدد المعروض على الشاشة حاليا في "الحد الأول" Operand1.

- أيضًا، يجب أن يحتفظ البرنامج بنوع العملية الحسابية التي يجريها، لذلك سيتم تخزين نوع العملية الحسابية في "العملية" Operator.

- سيدخل المستخدم عددًا آخر، ثم سيضغط زر الناتج "=".. في هذه اللحظة سيقوم برنامجنا بالتالي:

1. تخزين الرقم المعروض على الشاشة في "الحد الثاني" Operand2.

2. تنفيذ العملية الحسابية على الحدين الأول والثاني.

3. عرض الناتج على الشاشة، مع ملاحظة أنّ هذا الناتج يمكن أن يكون الحد الأوّل في عملية حسابية جديدة.

واضح من طريقة عملها، أن متغيراتنا الثلاثة Operand1, Operator, Operand2 يجب أن تُعرّف على مستوى النموذج كلّه (لهذا تسمّى "متغيرات النموذج" Form-wide، وليس على مستوى إجراء واحد، وذلك لأنّها ستستخدم من أكثر من إجراء مختلف (ضغط أزرار العمليات، وضغط زر الناتج).. هكذا يجب أن يبدو الكود:

**Public Class LoanForm**

**Inherits System.Windows.Forms.Form**

**Dim ClearDisplay As Boolean**

**Dim Operand1 As Double**

**Dim Operand2 As Double**

**Dim Operator As String**

**'………………………**

**' باقي محتويات النموذج**

**'……………………..**

**End Class**

تلاحظ هنا أننا نستخدم نوعا جديدا من المتغيرات: "مزدوج" Double.. إنّه متغيّر رقميّ أيضًا، ولكنه يتفوّق على "المفرد" Single في أن دقته العشريّة أعلى، بمعنى أنّه يستوعب خانات عشرية أكثر بكثير.

والآن انقر زر "+" واكتب به ما يلي:

**Private Sub BtPlus\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtPlus.Click**

**Operand1 = Val(LbDisplay.Text)**

**Operator = "+"**

**ClearDisplay = True**

**End Sub**

تعرضنا من قبل لمفهوم تحويل الأنواع.. عندنا هنا حالة مشابهة:

إن اللافتة تحتوي على نص، حتى لو كان هذا النص يمثّل أرقاما.. وفي السطر الأوّل من الإجراء، نريد تخزين هذا النص في متغيّر رقمي.. لهذا نستخدم الدالة "قيمة" Val() ـ واسمها اختصار لكلمة Value ـ لتحويل النص الذي يمثّل الأرقام إلى أرقام فعليّة، يمكن إجراء العمليات الحسابية عليها.

ولن أكتب هنا الكود الخاص بباقي العمليات الحسابية، فهي مشابهة لعملية الجمع، وعليك أنت كتدريبٍ أن تكتبه من تلقاء نفسك.

سنقفزُ مباشرةً لكود إظهار الناتج:

**Private Sub BtEquals\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtEquals.Click**

**Dim Result As Double**

**Operand2 = Val(LbDisplay.Text)**

**Select Case Operator**

**Case "+"**

**Result = Operand1 + Operand2**

**Case "-"**

**Result = Operand1 - Operand2**

**Case "\*"**

**Result = Operand1 \* Operand2**

**Case "/"**

**' لاحظ أن القسمة على صفر ليس لها معنى رياضيّ، وهي تؤدي لخطإ برمجيّ**

**' لهذا يجب تلافي حدوث هذا الحالة**

**If Operand2 <> "0" Then Result = Operand1 / Operand2**

**End Select**

**LbDisplay.Text = Result**

**'يمكن أن يكون الناتج المعروض على الشاشة هو الحد الأوّل في العملية التالية**

**' كما أننا نتوقع أن أي رقم سيكتبه المستخدم لن يضاف للناتج،**

**' بل سيظهر بمفرده على الشاشة.. أي أن على زر الرقم القادم أن يمسح الشاشة**

**ClearDisplay = True**

**End Sub**

هذا الإجراء يعرّضنا لمقطع Block جديد، ألا وهو جملة "اختيار الحالة" Select Case statement، والتي تستخدم لمقارنة قيمة متغيّر ـ ذلك الذي يلي تعبير Select Case مباشرة، بمجموعة محددة من القيم ـ تلك التي تلي كل كلمة "حالة" Case.. وعند حدوث تطابق، يتم تنفيذ الكود الخاص بهذه الحالة، ثم القفز إلى جملة "نهاية الاختيار" End Select.

فمثلا، لو كانت قيمة العملية Operator هي "+"، فسيتم تنفيذ الحالة الأولى، وتجاهل باقي الحالات.. وهكذا.

الجدير بالذكر، أنّ جملة "اختيار الحالة" Select Case مكافئة نوعًا لجملة الشرط If..Then، وإن كانت جملة الشرط هي الحالة العامة، بمعنى أن كل جملة "اختيار الحالة" يمكن كتابتها في صورة جملة شرط ـ وإن كانت ستكون أطول ـ ولكن ليست كل جملة شرط يمكن كتابتها في صورة "اختيار الحالة".

وها هي ذي جملة الشرط المناظرة لجملة "اختيار الحالة" في الإجراء السابق:

**If Operator = "+" Then**

**Result = Operand1 + Operand2**

**ElseIf Operator = "-" Then**

**Result = Operand1 - Operand2**

**ElseIf Operator = "\*" Then**

**Result = Operand1 \* Operand2**

**ElseIf Operator = "/" Then**

**If Operand2 <> "0" Then Result = Operand1 / Operand2**

**End If**

واضحٌ تمامًا أن استخدام جملة "اختيار الحالة" هو الأفضل، عند التعامل مع متغير واحد نريد اختبار تساوي قيمته مع واحدة من مجموعة قيم، لأن جملة الشرط في هذه الحالة مرهقة وطويلة ـ كما يمكنك أن ترى.

الجدير بالذكر أن المتغير الذي نختبر قيمته بجملة "اختيار الحالة" يمكن أن يكون رقميا أو نصّيًا.

**ملاحظة:**

لعلك لاحظت أن جملة الشرط :

**If Operand2 <> "0" Then Result = Operand1 / Operand2**

ليس لها End If.. إنّ ذلك ممكنٌ في الحالة التي يكون فيها جواب الشرط جملة واحدة، حيث يتم كتابتها مع جملة الشرط في سطر واحد، فلا يوجد هنا مقطع، ولا حاجة لاستخدام End if.. وإن كان بإمكانك كتابة نفس الجملة السابقة كالتالي:

**If Operand2 <> "0" Then**

**Result = Operand1 / Operand2**

**End If**

والآن شغّل البرنامج، واختبره.. تلاحظ أنّنا لا نحتاج لإجازة البيانات في هذا البرنامج، لأن كل المدخلات تتّم عن طريق الأزرار، فلا فرصة لدى المستخدم لإدخال بيانات مغلوطة.

الخطأ الوحيد المحتمل هو القسمة على صفر، وقد تمت معالجته برمجيا كما رأيت.

**أدوات التصحيح Debugging Tools**

نظرًا لبساطة البرنامج السابق، ستجد أنّه يعمل بطريقةٍ صحيحة.. حّتى لو كان به أي خطإ، فسيمكنك اكتشاف سببه سريعا.

ولكن للأسف، ليس الأمر دائمًا بهذه البساطة، فكلما تعمّقت في البرمجة، ستواجهك الكثير من الأخطاء المعقّدة، والتي يمكن أن تقضيَ أيامًا كاملةً حائرًا في سببها.

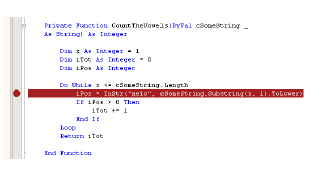
هنا تبرز أهمية عملية اصطياد الأخطاء وتصحيحها Debugging، والتي يمنحك VS .NET الأدوات اللازمة لتسهيلها.

ولا بدّ أن نتطرق هنا لموضوع تصحيح الأخطاء، لأنّه سيكون حيويّا بالنسبة لك في أيّ كود ستكتبه فيما بعد.

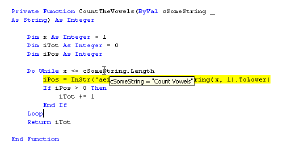
* [**"نقاط التوقّف" Breakpoints:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20أدوات%20التصحيح\1-%20نقاط%20التوقّف.htm)
* [**نافذة المتغيّرات الخاصّة The Local Windows:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20أدوات%20التصحيح\2-%20نافذة%20المتغيّرات%20الخاصّة.htm)
* [**نافذة المراقبة الفوريّة Quick Watch ونافذة المراقبة Watch window:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20أدوات%20التصحيح\3-%20نافذة%20المراقبة%20الفوريّة%20ونافذة%20المراقبة.htm)
* [**نافذة الأوامر Command Window:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20أدوات%20التصحيح\4-%20نافذة%20الأوامر.htm)
* [**نافذة المخرجات Output window:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20أدوات%20التصحيح\5-%20نافذة%20المخرجات.htm)

**"نقاط التوقّف" Breakpoints:**

قف بمؤشر الكتابة داخل الإجراء BtEquals\_Click، على السطر الذي ينفذ عملية الطرح، واضغط F9.. ستلاحظ أن السطر قد تم تحديده باللون البنّي، مع ظهور دائرة في هامش الصفحة الأيسر، بمحاذاة هذا السطر.. لقد صار هذا السطر "نقطة توقّف" Breakpoint، أي أن تنفيذ البرنامج سيتوقف مؤقتًا عندما يصل إليه.



شغّل البرنامج، وأجر عملية طرح، واضغط زر الناتج "=".. ستلاحظ أن التنفيذ سيتوقف، وستفتح لك نافذة محرر الكود، حيث ستظهر لك نقطة التوقف وقد تم تحديدها باللون الأصفر، دليلا على أن هذا السطر هو السطر الذي سيتم تنفيذه عند استكمال تشغيل البرنامج.



حلّق بمؤشر الفأرةِ فوق المتغير Operand1، وتوقّف لحظة.. سيظهر لك تلميح على الشاشة، مكتوب فيه اسم المتغير والقيمة التي يحملها في هذه اللحظة.. كرّر نفس العملية مع المتغيّر Operand2.. هذه الخاصية ليست مرتبطةً بالمتغيرات الموجودة في سطر نقطة التوقف فقط، بل تشمل أي متغيّر آخر داخل الإجراء الحالي.

استطلع قيمة المتغيّر Result.. ستجدها صفرًا، نظرًا لأنّ السطر المتوقَّف عنده لم ينفّذ بعد، أي أن عملية الطرح لم تتمّ بعد.

إن هذه العملية مفيدة جدًّا، فلو كان هناك خطأ بالبرنامج، ولاحظتَ أن قيم المتغيرات غريبة وغير متوقعة، فمن المحتمل أنّ الخطأ موجودٌ في هذا السطر أو ما سبقه.

ويمكنك أن تضغط F8 لتنفيذ سطر واحد من الكود، حيث سيتوقف البرنامج بعدها على السطر التالي له، وهو في حالتنا هذه: End Select.

**ملحوظة:**

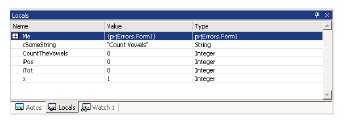
لو لم يعمل زر F8، فاضغط "تصحيح" / "تنفيذ خطوة" Debug.StepInto.. واقرأ اختصار الأمر StepInto على القائمة، حيث ستجده مكتوبا بجواره، كالتالي: StepInto F8.

استطلع الآن قيمة المتغير Result.. ستجد أنّها لم تعد صفرا، فقد تم تنفيذ عملية الطرح، وتم وضع الناتج في المتغير Result.

بإمكانك الآن أن تضغط F8 لتنفيذ سطر آخر، أو أن تضغط F5 ليتم استكمال تشغيل البرنامج بالطريقة الطبيعية.

**نافذة المتغيّرات الخاصّة The Local Windows:**

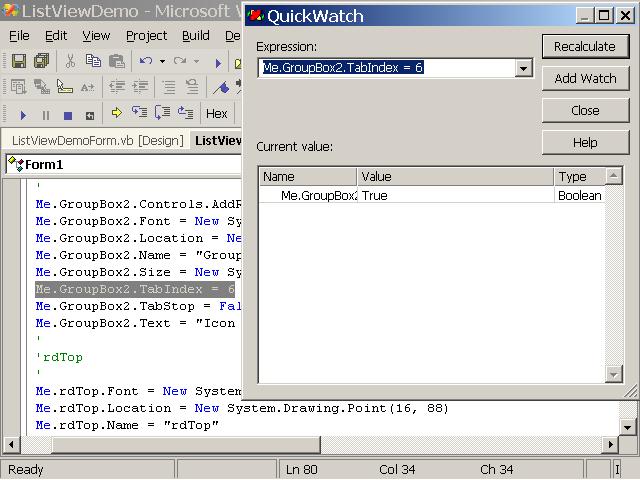
بينما أنت في وقت التوقّف Break Time، إذا أردت أن تستعرض جميع قيم المتغيّرات الخاصّة بالإجراء الحالي، اضغط القائمة "تصحيح" / "نوافذ" / "المتغيّرات الخاصّة" Debug.Windows.Locals، لتظهر لك نافذة المتغيّرات الخاصّة Local Windows.



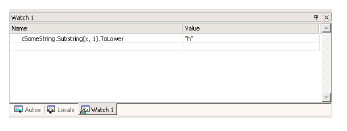
حينئذ سيمكنك أن تشاهد قيم كلّ متغيّرات الإجراء الحالي، وكلّما نفّذت خطوة من البرنامج (بضغط F8) سيتمّ تحديث قيم ما تغيّر من هذه المتغيّرات.

**نافذة المراقبة الفوريّة Quick Watch ونافذة المراقبة Watch window:**

هذه النافذة مشابهة للنافذة السابقة، إلا إنّها تمكّنك من تتبّع قيم تعبيرات معقّدة بدلا من الاكتفاء بقيمة متغيّر منفرد.. ولاستخدام هذه النافذة، حدّد أيّ متغيّر أو تعبير (أيّ عمليّة تحتوى على متغيّر أو أكثر)، واضغط زرّ الفأرة الأيمن، ومن القائمة الموضعيّة اضغط "مراقبة فوريّة" QuickWatch.. ستظهر لك نافذة "المراقبة الفوريّة" Quick Watch الموضّحة في الصورة:



وستجد أنّ التعبير الذي حدّدته مكتوب في مربّع النصّ المعنون بـ "تعبير" Expression، وستجد قيمته معروضة في القائمة التي تشغل نصف النافذة السفلي.. فإذا كنت تريد إضافة هذا التعبير لنافذة المراقبة، بحيث يمكنك متابعة قيمته دائما أثناء التنفيذ، فاضغط زر "إضافة مراقبة" Add Watch.. ويمكن أن تقوم بهذه الخطوة مباشرة دونَ فتح نافذة المراقبة الفوريّة، وذلك بتحديد التعبير في الكود، وضغط زرّ الفأرة الأيمن، ومن القائمة الموضعيّة تضغط Add Watch.. عندئذ ستظهر لك "نافذة المراقبة" أسفل الشاشة، حيث ستعرض لك قيمة التعبير المضاف مع تحديثها كلّما تغيّرت.



**نافذة الأوامر Command Window:**

ليست الطريقة السابقة هي الطريقة الوحيدة لمعرفة قيم المتغيّرات، فكما ذكرنا سابقا، يمكننا استخدام نافذة الأوامر Command Window لفعل ذلك.

لو لم تكن نافذة الأوامر ظاهرة، فاضغط "عرض" / "نوافذ أخرى" / "نافذة الأوامر" View.Other Windows.Command Window.. تلاحظ أن السطر الحالي في نافذة الأوامر يبدأ بعلامة ">".. ضع مؤشر الكتابة بجوار هذه العلامة، واكتب ما يلي:

**? Operand1 / Operand2**

اضغط زر الإدخال Enter.. ستجد أن الناتج ظهر لك في السطر التالي بين علامتي تنصيص.

**ملاحظة:**

لو كنت في وقت التصميم Design Time، فستظهر لك رسالة في السطر التالي، تخبرك بأنه لا يمكن تنفيذ هذا الأمر في وقت التصميم.. إذن عليك أن تكون في وقت التوقّف Break Time، بوضع علامة توقّف في أي موضع من الكود ليتوقف التنفيذ عندها.. حينئذ يمكنك أن تستخدم نافذة الأوامر لتنفيذ السطر السابق.

وإذا أردت أن تعرف العدد الذي تعرضه شاشة الآلة الحاسبة، فاكتب:

**? LbDisplay.Text**

لاحظ أن علامة الاستفهام في بداية السطر، هي اختصار لأمر طباعة قيمة المتغير أو العملية.

والآن أزل علامة التوقف التي سبق أن وضعتها، وذلك بضغط F9 مرةً أخرى فوق نفس السطر.

**نافذة المخرجات Output window:**

طريقة أخرى من طرق اختبار قيم المتغيرات ونواتج الدوال، هي طباعة قيمها في نافذة المخرجات Output window.. ويمكنك عمل ذلك بجملةٍ كالتالية:

**Console.WriteLine(Operand1)**

لو لم تكن هذه النافذة مفتوحة، فاضغط "عرض" / "نوافذ أخرى" / "المخرجات" View.Other Windows.Output.

**إضافة المزيد من الميزات Adding More Features:**

دعنا نضيف زرين جديدين لآلتنا الحاسبة:

زر المعكوس الجمعي "+/-"، والذي يعكس إشارة الرقم، وزر المعكوس الضربي "1/x"، والذي يحسب مقلوب الرقم (ناتج قسمة الواحد عليه).

أضفِ الزرين، ونسّقهما.. سمّ الأوّل BtNegate، وسمّ الثاني BtInverse .. ثم اكتب لهما الكود التالي:

**Private Sub BtNegate\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtNegate.Click**

**LbDisplay.Text = - Val(LbDisplay.Text)**

**ClearDisplay = True**

**End Sub**

**Private Sub BtInverse\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtInverse.Click**

**' لاحظ أن قلب الصفر معناها القسمة على صفر، وهي ليست ممكنة**

**If Val(LbDisplay.Text) <> 0 Then \_**

**LbDisplay.Text = 1 / Val(LbDisplay.Text)**

**ClearDisplay = True**

**End Sub**

لاحظ أن الجملة:

**If Val(LbDisplay.Text) <> 0 Then \_**

**LbDisplay.Text = 1 / Val(LbDisplay.Text)**

مكافئة للجملتين:

**TempValue = Val(LbDisplay.Text)**

**If TempValue <> 0 Then LbDisplay.Text = 1 / TempValue**

حيثُ إن TempValueهو متغير رقميّ (عليك تعريفه أولا بالطبع).

ما أريدك أن تعرفه هنا، هو أن استخدام الجملة الأولى، التي تتعامل مع خاصية نص اللافتة مباشرةً، تعتبر أسرع وأسهل، ولا بأس بها في البرامج القصيرة.. ولكنّ مثل هذه الطريقة في كتابة الكود مرفوضة في البرامج الطويلة ـ خاصة داخل الجمل التكراريّة Loops ـ لأن بها سببين لتضييع الوقت وإبطاء البرنامج:

1- استخدام خاصيّة النص Text مرتين، مع أن قيمتها ثابتة في كلتا المرتين، حيث إنّ استخدام المتغيرات أسرع بمراحل من استخدام خواص الأدوات والكائنات.

2- استدعاء دالة Val() مرتين بنفس المعامل، أي أنها تعطي نفس الناتج في كلتا المرتين، وهو تضييع للوقت لا مبرر له.

إن أجزاءً مئوية من الثانية تضيع بمثل هذه الطريقة، لن يتأثر بها المستخدم في البرامج القصيرة، ولكنها قد تجعل المستخدم ينتظر لدقائق، بسبب تراكمها في البرامج الطويلة.

**لهذا خذها كقاعدة:**

كل خاصيّة أو دالة أو عملية، يتم تكرارها بنفس الصيغة لتعطي نفس النتائج في كل مرّة، استخدمها مرة واحدة، وضع ناتجها في متغيّر، وكرّر استخدام المتغيّر بعدها كما يحلو لك.

أريدك أن تفعل هذا حتّى في البرامج الصغيرة، حتّى يصبح من عاداتك الحميدة في البرمجة.

والآن، حاول أن تضيف لآلتك المزيد من الإمكانيات.. إن VS .NET تمدّك بخليّة Class يسمّى "رياضيات" Math، يحتوي على عدد وفير من الدوال الرياضية الجاهزة، مثل "جتا" Cos()، و"الجذر التربيعي" Sqrt().. وهذان مثالان لكيفية استخدامهما:

* **LbDisplay.Text = Math.Cos(Val(LbDisplay.Text))**
* **If LbDisplay.Text < 0 Then**

**MsgBox("لا يمكن حساب الجذر التربيعي لعدد سالب")**

**Else**

**LbDisplay.Text = Math.Sqrt(Val(LbDisplay.Text))**

**End If**

حاول أن تضيف أزرارًا جديدةً تؤدّى هذه الإمكانيات.

**ملاحظة:**

قد يصيبك الملل من كثرة كتابة Math. قبل اسم الدالة الرياضية.. هناك حل لهذا.. يمكنك أن تستدعي اسم الخليّة لبرنامجك، بكتابة جملة الاستيراد التالية بأعلى الصفحة، قبل الجملة التي تعرّف خليّة النموذج نفسه:

**Imports System.Math**

بهذه الطريقة يمكنك كتابة اسم الدالة مباشرة.. مثال:

**LbDisplay.Text =Cos(Val(LbDisplay.Text))**

**معالجة الاستثناءات Exception Handling:**

حاول أن تضرب عددين هائلين، وكلما ظهرت لك نتيجة، استمر في ضربها في عدد كبير جدا، حتّى تظهر لك كلمة "ما لا نهاية" Infinity على الشاشة.. إن هذا معناه أنّ الرقم الناتج تجاوز أقصى قيمة يقبلها المتغير المزدوج Double.. ولكننا لم نكتب في البرنامج أي معالجةٍ لخطإ "تجاوز الحد" OverFlow.. فمن أين إذن جاءت كلمة Infinity؟

هذه هي طريقة VB لإخبارك بأنّه لا يستطيع التعامل مع رقم بهذه الضخامة.. إنّ هذه ليس محدوديّةً في قدرات اللغة، بل هي من سمات الكمبيوتر ككل، حيث إنّه يخصّص عددا محدودًا من وحدات الذاكرة Bytes لتخزين الأرقام.

ورغم أنّ تطبيق الآلة الحاسبة يبدو راسخًا، ولا يمكن أن يتسبّب في أيّة أخطاء، إلا أنّنا لا نضمن ما يمكن أن يتسبّب فيه المستخدم من أخطاء، لهذا سوف نضيف بعض الكود لمعالجة أي خطإ غير متوقّع.

**ملحوظة:**

الأخطاء الآن تسمّى "استثناءات" Exceptions، ويمكن أن تفكّر فيها على أنّها "استثناء" من حالة البرنامج الطبيعية.. عامّةً لا تجعل هذه التسمية تحيّرك كثيرا، واعتبر أن من ابتدعها كان يعمل بالمثل القائل: "الملافظ سعد"، حيث إنّه من الألطف أن تظهر لك ـ أو للمستخدم ـ رسالةٌ تخبركما بحدوث "استثناء" في البرنامج، بدلا من أن تلطمكما رسالة تقول إنَّ البرنامج به خطأ يا مغفّل!

وفي حالة حدوث أي استثناء، لا بد أن ينفّذ البرنامج بعض الأوامر "الاستثنائيّة" لمعالجة هذه الحالة.

ولتنظيم عملية معالجة الاستثناءات، أضافت ميكروسوفت للغة ما أسمته "المعالجة الهيكليّة للاستثناء" Structured exception handling.. والفكرة كلّها، أنّ البرنامج في حالة حدوث خطإ، بدلا من أن يعرض رسالة خطإ ويتم إغلاقه، يقوم بتنفيذ جزء من الكود، ذلك الذي تكتبه أنت لمثل هذه الحالات.

بالنسبة لبرنامج الآلة الحاسبة، المكان الوحيد الذي يمكن أن تضيف فيه معالجةً للاستثناءات، هو إجراء ضغط زر الناتج، حيث تتمّ فيه كل الحسابات.

ولكن كيف يبدو مقطع معالجة الاستثناء؟.. إنه ـ في صيغته العامّة ـ يبدو كالتالي:

**Try**

**{ مقطع من الكود }**

**Catch اسم\_الاستثناء As نوع الاستثناء**

**{ مقطع معالجة هذا النوع من الاستثناءات }**

**……….**

**Catch يمكن أن يتكرّر مقطع**

**لأيّ عدد من المرّات، بأنواع مختلفة من الاستثناءات**

**……….**

**Catch اسم\_الاستثناء As Exception**

**{ مقطع معالجة باقي أنواع الاستثناءات }**

**Finally**

**{ مقطع إلزامي، ينفّذ في كلّ الأحوال}**

**End Try**

في البداية سيحاول البرنامج تنفيذ مقطعٍ من الكود (ذلك الموجود في جملة "حاول" Try).. فإذا نجح تنفيذ هذا المقطع بدون مشاكل، فإن التنفيذ سينتقل لجملة "في النهاية" Finally، حيث سينفّذ بعض العمليات المترتّبة على تنفيذ مقطع الكود (مثل إغلاق ملف مفتوح، أو محو بعض البيانات من الذاكرة... إلخ).. عامّةً، فإنّ جملة Finally يتمّ تنفيذها سواء حدث استثناء أم لم يحدث، وهي جملة اختياريّة، بمعني أنّك تستطيع ألا تكتبها.. وفي حالة عدم وجود هذه الجملة، سينتقل التنفيذ لجملة "إنهاء المحاولة" End Try.

ولكن لو حدث خطأ في مقطع "حاول" Try، فإنّ التنفيذ ينتقل مباشرةً إلى المقطع "قنص الاستثناء" Catch Exception، ليتمّ تنفيذ الجمل الخاصّة بمعالجة الاستثناء.

ما يجب أن تعلمه هو أنّ للاستثناءات أنواعا كثيرة.. وكلّ استثناء هو عبارة عن خليّة Class، ستجد المعلومات الكافية عنها في ملفّات استعلام اللغة، فعند شرح أيّ خاصيّة أو وسيلة، ستجد في ملفات الاستعلام معلومات عن نوعية الأخطاء التي يمكن أن تحدث عند استخدام هذه الخاصيّة أو الوسيلة.. ومن أمثلة أنواع الاستثناءات NullReferenceException و ArgumentOutOfRangeException وغيرها.. ولحسن حظّك، لديك النوع العام Exception، الذي يستطيع أن يتعامل مع أيّ استثناء.. لذلك يجب أن تجعله في آخر جملة من جمل Catch، ليستقبل كلّ أنواع الخطإ التي لم تتعامل معها جمل Catch السابقة.

ويمكنك أن تجد أهمّ أنواع الاستثناءات في الملحق رقم1، حيث يمكنك الرجوع إليه لاستخدامها في الحالات المختلفة.

ولجميع خلايا الاستثناء نفس الخصائص، وهي تحتوي على جميع المعلومات المتاحة حول الاستعلام، مثل المشروع الذي حدث به الاستثناء، والدالة التي حدث بها، وأسباب حدوثه.. (راجع ملحق 1).

وبالنسبة لبرنامجنا، فليس أمامك الكثير لتفعله مع أخطاء الحسابات.. كل ما ستفعله هو عرض رسالة تحذير للمستخدم، تطالبه فيها بإعادة إدخال أرقام أكثر دقّة.

ولكنّ دنيا معالجة الأخطاء ذاخرة، ولا تقتصر على مثل هذه المعالجة الهزيلة في برنامج آلتنا الحاسبة!

فمثلا، لو لم يستطعْ برنامجك أن يقرأ ملفّا من أسطوانة، يجب عليك أن تعرض رسالة للمستخدم، تطالبه فيها بشراء محرّك أسطوانات!!، أو بإدخال الأسطوانة فيه لو كان عنده واحد، ثم يكرر المحاولة.

وفي حالات أخرى، يمكنك أن تخبر المستخدم بغياب قيمةٍ ما، ثم تواصل التنفيذ.

**عامةً، إليك هذا الخبر السعيد:**

ليس هناك طريقة عامّة لمعالجة كلّ الاستثناءات مرّةً واحدة.. إنّ هذه هي مسئوليّتك الشخصيّة، فكل برنامج يختصّ بنوع من الأخطاء، تحدّده طريقة عمله.. هنا يأتي دور الخبرة، فكلما كتبت برامج أكثر، كلما ظهرت لك نوعيات عجيبة غير متوقعة من الأخطاء، والتي سيتم طحنك في البداية حتّى تعالجها.. بعد هذا سيكون الأمر روتينيًّا بالنسبة لك.

والآن، عدّل إجراء ضغط زر الناتج في آلتنا الحاسبة، ليصبح كالتالي:

**Private Sub BtEquals\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtEquals.Click**

**Dim Result As Double**

**Operand2 = Val(LbDisplay.Text)**

**Try**

**Select Case Operator**

**Case "+"**

**Result = Operand1 + Operand2**

**Case "-"**

**Result = Operand1 - Operand2**

**Case "\*"**

**Result = Operand1 \* Operand2**

**Case "/"**

**If Operand2 <> "0" Then \_**

**Result = Operand1 / Operand2**

**End Select**

**LbDisplay.Text = Result**

**Catch Exc As Exception**

**MsgBox(Exc.Message)**

**LbDisplay.Text= "ERROR"**

**Finally**

**' هذه الجملة سيتم تنفيذها في كل الأحوال، سواء حدث استثناء أم لا**

**' وذلك لأنّنا نريد مسح محتويات الشاشة لاستقبال الأرقام الجديدة**

**ClearDisplay = True**

**End Try**

**End Sub**

لاحظ أن جملة Catch تم فيها تعريف متغير ـ بأي اسم تختاره ـ من النوع استثناء Exception، حيث ستضع فيه اللغة كل المعلومات المرتبطة بالاستثناء الذي حدث.. ولقد استخدمنا بالفعل إحدى خواص هذا المتغيّر، وهي الخاصيّة "وصف" Message، والتي تحتوي على رسالة الخطإ الذي حدث، حيث عرضناها للمستخدم في رسالة.

الجدير بالذكر أنّ الطريقة القديمة لمعالجة الخطإ، والتي دأب مبرمجو VB6 على استخدامها، ما زالت قابلة للاستخدام، وهي تتلخّص في استخدام إحدى الجملتين التاليتين في بداية الإجراء:

**الجملة الأولى:**

**On Error Resume Next ' عند حدوث خطأ واصل تنفيذ الجملة التالية**

حيث في هذه الجملة تخبر VB أنّ عليه الاستمرار في تنفيذ السطر التالي للسطر الذي حدث به الخطأ، دون عرض رسالة خطإ.

ولكم كيف تعرف بأنّ خطأ قد وقع؟.. في هذه الحالة يمكنك استخدام كائن الخطإ Err Object، الذي يمنحك الخصائص التالية:

**رقم الخطإ Number:**

لكلّ خطإ رقم، وبهذا تستطيع أن تعرف نوع الخطإ الذي حدث، وذلك بقراءة ملفّات إرشادات اللغة.. ما يهمّنا في هذه الخاصيّة هو أنّنا سنستخدمها لمعرفة حدوث الخطإ من عدمه، فلو لم يحدث أيّ خطإ، فإنّ قيمة هذه الخاصيّة ستكون صفرا.

**الوصف Description:**

تعيد لك نصّا يصف الخطأ الذي حدث.

كما يمتلك كائن الخطإ بعض الوسائل، من أهمّها:

**محو Clear:**

تمكّنك من محو معلومات الخطإ بعد أن تعالجه، وبذلك ستعود خاصيّة رقم الخطإ إلى القيمة صفر، حتّى تستطيع أن تعرف بحدوث أيّ خطإ جديد.

**معرفة الاستثناء GetException:**

تعيد لك كائنا من النوع Exception، يحتوى على معلومات عن الاستثناء الذي حدث.

والآن سأريك كيف يمكن استخدام هذه الطريقة في معالجة الخطأ في زرّ "=":

**Private Sub BtEquals\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtEquals.Click**

**On Error Resume Next**

**Dim Result As Double**

**Operand2 = Val(LbDisplay.Text)**

**Select Case Operator**

**Case "+"**

**Result = Operand1 + Operand2**

**If Err.Number <> 0 Then**

**MsgBox(Err.Description)**

**ClearDisplay = True**

**Exit Sub**

**End If**

**Case "-"**

**Result = Operand1 - Operand2**

**If Err.Number <> 0 Then**

**MsgBox(Err.Description)**

**ClearDisplay = True**

**Exit Sub**

**End If**

**Case "\*"**

**Result = Operand1 \* Operand2**

**If Err.Number <> 0 Then**

**MsgBox(Err.Description)**

**ClearDisplay = True**

**Exit Sub**

**End If**

**Case "/"**

**If Operand2 <> "0" Then**

**Result = Operand1 / Operand2**

**If Err.Number <> 0 Then**

**MsgBox(Err.Description)**

**ClearDisplay = True**

**Exit Sub**

**End If**

**End If**

**End Select**

**LbDisplay.Text = Result**

**ClearDisplay = True**

**End Sub**

لاحظ أنّنا كرّرنا مقطع معالجة الخطإ في كلّ موضع نتوقّع فيه حدوث الخطأ، ممّا جعل الكود أطول.. إنّ ذلك إجباريّ، فلو حدث خطأ في أيّ حالة Case، فسيقوم VB بتنفيذ السطر التالي، ممّا يعني أنّه سينتقل إلى اختبار الحالة التالية، ممّا لا نضمن معه النتائج.. من هنا كانت الجملة الثانية لمعالجة الخطإ أكثر فعالية، خاصّة حينما تتوقّع حدوث الأخطاء في أكثر من موضع.

**الجملة الثانية:**

**On Error GoTo LineErr ' عند حدوث خطإ اقفز إلى سطر معيّن**

وأنت هنا تخبر VB أنّ عليه القفز إلى سطر معيّن عند حدوث خطإ في أيّ موضع من الإجراء.. وفي الغالب سيكون هذا السطر تاليا لآخر سطر في الإجراء، حيث ستقوم بكتابة الكود الذي يعالج الخطإ.. لاحظ أنّ الاسم LineErr اختياريّ، ويمكنك ان تسمّيه كما تشاء.. المهمّ أن تكتب هذا الاسم متبوعا بنقطتين متعامدتين ":" قبل السطر الذي تريد الانتقال إليه.. انظر كيف سنعالج أخطاء زرّ "=" بهذه الطريقة:

**Private Sub BtEquals\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtEquals.Click**

**On Error GoTo LineErr**

**Dim Result As Double**

**Operand2 = Val(LbDisplay.Text)**

**Select Case Operator**

**Case "+"**

**Result = Operand1 + Operand2**

**Case "-"**

**Result = Operand1 - Operand2**

**Case "\*"**

**Result = Operand1 \* Operand2**

**Case "/"**

**If Operand2 <> "0" Then**

**Result = Operand1 / Operand2**

**End If**

**End Select**

**LbDisplay.Text = Result**

**ClearDisplay = True**

**' حتّى لا يتمّ تنفيذ مقطع معالجة الخطإ بدون حدوث خطأ، يجب أن تكتب الجملة التالية**

**Exit Sub**

**' مقطع معالجة الخطإ**

**' عند حدوث أيّ خطإ سينتقل التنفيذ مباشرةً إلى هنا**

**LineErr:**

**MsgBox(Err.Description)**

**ClearDisplay = True**

**End Sub**

الآن لديك كامل الحرّيّة في اختيار الطريقة التي تناسبك لمعالجة أخطاء برنامجك.

**تشغيل محاسب القروض على الإنترنت**

* + [**تطبيقات الإنترنت:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20تشغيل%20محاسب%20القروض%20على%20الإنترنت\0-%20تشغيل%20محاسب%20القروض%20على%20الإنترنت.htm)
  + [**أدوات الإنترنت:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20تشغيل%20محاسب%20القروض%20على%20الإنترنت\1-%20أدوات%20الإنترنت.htm)
  + [**والآن فلنبدأ تطبيقنا:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20تشغيل%20محاسب%20القروض%20على%20الإنترنت\2-%20والآن%20فلنبدأ%20تطبيقنا.htm)
  + [**حان وقت البرمجة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\3-%20تشغيل%20محاسب%20القروض%20على%20الإنترنت\3-%20حان%20وقت%20البرمجة.htm)

**تشغيل محاسب القروض على الإنترنت:**

الآن سنحاول أن ننشئ تطبيقا جديدا لحساب القروض، ولكنّه سيعمل هذه المرّة على متصفّح الإنترنت Browser، بحيث يستطيع أيّ مستخدم أن يتصل بخادمك Server ليستخدمه، بدون أن يضطر لإنزاله على جهازه.

باختصار: أنت بصدد تحويل محاسب القروض من تطبيق ويندوز إلى تطبيق إنترنت، وتأكّد أنّك سرعان ما ستكتشف أنّهما متماثلان كثيرًا.

إن Visual Studio.NET يحاول أن يجعل تطوير برامج الإنترنت في نفس سهولة برامج VB.. إنّ باستطاعتك الآن أن تنشئ واجهة تطبيقات الإنترنت (صفحة HTML عليها أدوات تتفاعل مع المستخدم) تمامًا بنفس الطريقة التي تنشئ بها نموذج الويندوز التقليديّ.. ليس هذا فحسب، بل تكتب كذلك كود VB المعروف للاستجابة للأحداث، كما في نموذج الويندوز العاديّ.

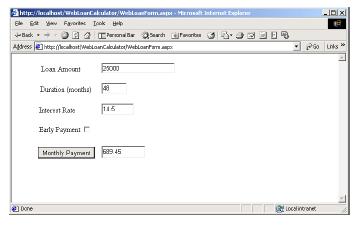
**ملاحظة:**

لكتابة واختبار تطبيقات الإنترنت، يجب أن يكون "خادم معلومات الإنترنت"Internet Information Server (IIS) متوافرا ويعمل على جهازك (راجع كيفيّة إعداده في مقدّمة الكتاب).. هذا هو السبب الذي يجعل هذه اللغة لا تعمل إلا على Win NT أو Win 2000 أو Win XP، فهي التي تسمح بإنزال هذا الخادم على الجهاز.. ويجب هنا أن أحذّرك من خطإ شائع، وهو إعداد إطار العمل .Net Framework أولا ثمّ إعداد IIS.. في هذه الحالة لن يتمّ تشغيل تطبيقات الإنترنت، ويجب إزالة إطار العمل (من Add Or Remove Programs من لوحة التحكّم Control Panel)، ثمّ إعادة إعداده مرّة أخرى (من أسطوانة Upgrade Windows Components) بعد أن يكون IIS قد تمّ إعداده.

ابدأ مشروعًا جديدًا، وفي مربع حوار "مشروع جديد" اضغط أيقونة "تطبيق إنترنت" ASP.NET Web Application.. سمّ التطبيق WebLoanCalculator، وأغلق مربع الحوار.

ستشاهد نافذةً مماثلة للنموذج العاديّ.. هذه هي صفحة الإنترنت، أو نموذج الإنترنت، وستجد أنّها تتّخذ الاسم الافتراضيّ WebForm1.aspx.

إنّ نموذج الإنترنت مماثل تمامًا لنموذج الويندوز، إلا إنّه يظهر كصفحة HTML على أحد متصفّحات الإنترنت، مثل Internet Explorer.



ويتم حفظ ملفات مشروع الإنترنت في المجلد الذي حدّدناه، ولكنّ هذا المجلّد يتمّ إنشاؤه داخل المجلّد الرئيسيّ لخادم الإنترنت Web server’s root folder، الذي يكون عادةً "C:\Inetpub\wwwroot folder".

وعلى هذا، إذا أردت أن تفتح تطبيق WebLoanCalculator المرفق مع هذا الفصل، فعليك بنسخه أولا إلى المجلد المذكور سالفًا، ثمّ شغّل VS .NET وافتح التطبيق.

عامّةً، سنشرح الآن، كيف ننشئ هذا التطبيق من الصفر.

سمّ النموذج الرئيسيّ للتطبيق WebLoanForm.aspx.

ولكي تشغّل هذا التطبيق، ببساطة اضغط F5.. أو يمكنك اتباع طريقة أخرى، بأن تفتح متصفّح الإنترنت Internet Explorer، وتكتب العنوان التالي في مربع العناوين Address Box في أعلى النافذة:

<http://localhost/WebLoanCalculator/WebLoanForm.aspx>

ثم تضغط زر الإدخال Enter.

**أدوات الإنترنت:**

افتح صندوق الأدوات.. ستجد أن شريط "نماذج الإنترنت" Web Forms Tab نشيط، بدلا من شريط "نماذج ويندوز" Windows Forms Tab.

وهذا الشريط يحتوي على الأدوات التي يمكن أن تستخدمها على نماذج الإنترنت، وهي شبيهة بأدوات نماذج ويندوز، ولكنّها ليست بجودة ولا ثراء وظائف أدوات نماذج ويندوز، وذلك لأنّ صفحات الإنترنت تستخدم واجهة أبسط بكثير من واجهة الويندوز.. ونظرًا لأنّ المتصفّح لا يتعامل إلا مع ملفات HTML، يتم تحويل واجهة التطبيق إلى صفحات HTML.

وهناك شريط هام آخر في صندوق الأدوات، وهو شريط HTML، الذي يحتوي على أدوات HTML القياسية، التي يمكنك استخدامها على أي صفحة إنترنت.. إنّ بعض أدوات الإنترنت Web Controls تعتبر متقدّمةً نوعًا ما، بالمقارنة بالقدرات المحدودة لأدوات HTML.

ولكن هل يعني هذا أنّ الصفحات التي تحتوي على أدوات الإنترنت، لا يمكن عرضها على أي متصفح آخر غير Internet Explorer؟.. كلا بالطبع، فهي يتمّ تحويلها آليًّا إلى كود HTML قياسي، يمكن تسليمه إلى أيّ متصفّح.

مثال: ستجد الأداة "العرض المتفرّع" TreeView على شريط أدوات الإنترنت، وهي لا مثيل لها ـ ولا حتّى ما يقاربها ـ بين أدوات HTML.. ورغم هذا، يمكن عرض هذه الأداة على أيّ متصفّح، لأنّ مصمّم نماذج الإنترنت سيضيف تركيبات HTML المناسبة، لإنشاء شيءٍ يبدو ويتصرف بما يشبه أداة "العرض المتفرّع"، ولكنّه ليس هي!

**والآن فلنبدأ تطبيقنا:**

ضع أربع لافتات على النموذج.. ورتبها كما في الشكل.

حدّد كل لافتة بالفأرة، وابحث عن خاصية "نص" Text لها في نافذة الخواص.. كما تلاحظ، فإنّ معظم الخواص الأساسيّة لأدوات الإنترنت، لها نفس أسماء خصائص أدوات الويندوز.

غير نصوص اللافتات الأربع إلى: Loan Amount وDuration (months)" و Interest و Monthly Payment.

لاحظ أن لافتة الإنترنت يتغير حجمها تلقائيا، ليتناسب مع عرض النص المعروض بها.

والآن أضف أربعة مربعات نصوص، كل منها بجوار لافتته، ثم حدّد كل منها، لتغيير خاصية "المعرّف" ID إلى TxtAmount و TxtDuration و TxtRate و TxtPayment.

**ملاحظة:**

خاصية ID، تناظر خاصية الاسم Name في أدوات ويندوز، وهي التي سيتم استخدامها في الكود للتعامل مع خواص ووسائل الأداة.

أضف مربع اختيار CheckBox، وضع فيه النص "Early Payment"، وسمّه ChkPayEarly.

**ملاحظة:**

ستجد أن النص في مربع الاختيار، ملتصق بخانة الاختيار.. ضع بعض المسافات في بداية النص، حتّى يتباعدا قليلا.

وأخيرا، ضع زرا على النموذج، واكتب عليه النص "Monthly Payment"، وغير اسمه إلى BtShowPayment.. هذا هو الزر الذي سيسلّم معلومات القرض إلى الخادم Server، حيث سيقوم الكود المناسب بحساب المدفوع شهريًّا، وإرجاع الناتج للعميل Client.

والآن اشرع في محاذاة الأدوات على النموذج، باستخدام القائمة الرئيسيّة "تنسيق" Format، تمامًا كما فعلت من قبل في نماذج ويندوز.

**حان وقت البرمجة:**

انقر الزر مرتين، لتفتح نافذة محرّر الكود، حيث سيظهر لك الإجراء الذي يستجيب لحدث ضغط الزر.. أضف إليه نفس الكود الذي كتبناه في برنامج محاسب القروض من قبل.. اضغط F5 لتشغيل الإجراء.. ثوانيَ وتظهر لك نافذة متصفّح الإنترنتInternet Explorer ، عارضةً لك الصفحة التي صمّمتها.. املأ مربعات النصوص ثم اضغط الزر.. ما هي إلا ثوانيَ ويظهر لك ناتج العمليّة، ولكن مع ملاحظة أنّ صفحةً جديدة تمامًا قد وصلت للمتصفّح، حاويةً القيم التي أدخلتها كما هي بالإضافة للناتج.

**إرشاد:**

إذا حدثت أيّ أخطاء ورفض VB تشغيل التطبيق الذي نسخته من مجلّد برامج هذا الفصل، فصمّم التطبيق بنفسك من البداية.

**ملحوظة:**

رغم أن VB لن يعترض على دالة MsgBox الموجودة في الكود، إلا إنّها لن تعمل، لأنّه من غير المسموح عرض نموذج إجباري Modal على متصفّح الإنترنت.. ولحلّ هذه المشكلة، سنستخدم طريقة أخرى لعرض الرسالة، وذلك عن طريق خليّة الرسالة MessageBox Class.. هذه الخليّة تمنحك الوسيلة Show لعرض الرسالة، وتتيح لك بعض الخيارات التي تسمح لك بعرض الرسالة على متصفّح الإنترنت.. ونظرا لأنّ خليّة الرسالة موجودة في تطبيقات الويندوز، فإنّ استخدامها في تطبيق الويب يحتاج لاتّباع الخطوات التالية:

1- اضغط بزّر الفأرة الأيمن عنصر المراجع References في متصفّح المشاريع Solution Explore، ومن القائمة الموضعيّة اضغط الأمر Add Reference.

2- في مربّع الحوار الذي سيظهر، ابحث في القائمة عن الاسم System.Wendiows.Forms.. انقرّه مرّتين بالفأرة.. ستجد أنّه انتقل للقائمة السفليّة.. اضغط زرّ OK لإغلاق مربّع الحوار.

3- في أقصى بداية كود النموذج، أضف جملة الاستيراد التالية:

**Imports System.Windows.Forms**

4- الآن يمكنك عرض رسالة الخطأ كالتالي:

**MessageBox.Show( \_**

**"مقدار القرض يجب أن يكون رقما", \_ 'نصّ الرسالة**

**"بيانات غي صالحة", \_ ' عنوان الرسالة**

**MessageBoxButtons.OK, \_ ' OK ستعرض الرسالة زرّ**

**MessageBoxIcon.Error, \_ ' ستعرض الرسالة أيقونة الخطإ**

**MessageBoxDefaultButton.Button1, \_ ' الزرّ الافتراضيّ هو الزرّ الأوّل**

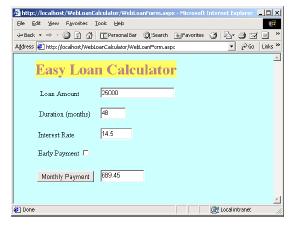
**MessageBoxOptions.ServiceNotification \_ ' عرض الرسالة على متصفّح الإنترنت**

**)**

ويجب ألا تنسى أنّ نموذج الإنترنت هو صفحة HTML، فإذا ما نظرت إلى كود الصفحة المعروضة في متصفّح الإنترنت، لوجدت أنّه HTML، بمعنى أنّك تستطيع معاملته كوثيقة HTML، مضيفًا ما تريد من الألوان والسمات.

ليس هذا فحسب، بل إن بيئة التطوير تسمح لك بتصميم نماذج الإنترنت بطريقة مرئيّة، أو بكتابة كود HTML.

تعال نضيف عنوانا ملونا، ونغير خلفية الصفحة:



في نافذة الخصائص، حدد خاصية "مظهر الصفحة" pageLayout للنموذج.. ستجد أن قيمتها هي "عرض الشبكة" GridLayout، وهو الوضع الذي يسمح لك بوضع الأدوات ومحاذاتها.. غير هذه القيمة إلى "عرض عادي" FlowLayout.. بهذا سيتخذ النموذج شكل صفحة HTML تقليديّة.. ضع مؤشّر الكتابة في بداية الصفحة وابدأ في الكتابة.. اكتب النص: "Easy Loan Calculator**"** ثم حدّده بالفأرة.. ستلاحظ أن أزرار تنسيق النصوص على شريط الأدوات قد صارت فعّالة.. اجعل حجم الخط 6 نقاط، وغير لونه ولون خلفيته.. أدخل بعض المسافات على يسار النص، لتوسيطه فوق الأدوات.

**ملحوظة:**

رغم أنّ متصفّحات الإنترنت تتجاهل المسافات الابتدائية، إلا أن VB يقوم بالواجب، ويحوّلها إلى ما يجعلها تظهر على المتصفّحات.

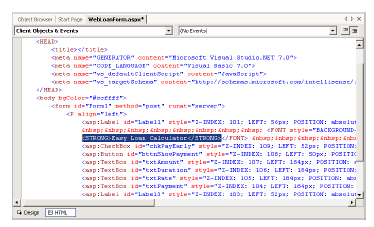
يمكنك أيضًا أن تغيّر لون الصفحة، وذلك بتغيير خاصيّة bgColor إلى أي لون فاتح تختاره.. عندما يظهر لك مربع اختيار الألوان، ستجد عليه الشريط "ألوان الإنترنت" Web colors، وهي الألوان التي يمكن للمتصفّحات أن تعرضها بأمان Safe colors.

ولترى كيف يتعامل مصمّم النماذج مع عناصر HTML الموجودة على الصفحة، اضغط زر HTML أسفل نافذة المصمّم.. ستجد أنّ مصمّم النماذج قد أضاف جمل HTML التالية، لينشئ رأس الصفحة Header:

**<FONT style = "BACKGROUND-COLOR: #ffff66" color = "#996666" size = "6">**

**<STRONG>Easy Loan Calculator</STRONG></FONT>**

وهو كود HTML المباشر، الذي يمكن أن تجده في أي صفحة إنترنت، وهو لا يحتوي على أيٍّ من أدوات الإنترنت Web Controls.



حدد تعريف الخط <FONT> Tag واحذفه، ثم ارجع لعرض التصميم.. ستجد أن العنوان قد اختفى من الصفحة.. ارجع مرة أخرى إلى عرض HTML، وضع هذه الجملة مباشرةً بعد التعريف <body> ، وقبل التعريف <form>:

**<h1>Easy Loan Calculator</h1>**

اضغط F5 لتشغيل البرنامج، عندما يظهر متصفح الإنترنت، املأ مربعات النصوص واضغط الزر.

ستجد أنّ التطبيق يعمل بطريقة مماثلةٍ لذلك الذي أنشأناه على الويندوز، ولكنّ واجهته تعمل على المتصفّح (على جهاز المستخدم)، بينما تتمّ الحسابات على الخادم Server، حيث يقع برنامجك.

**التعامل مع أكثر من نموذج Working with Multiple Forms**

* + [**إنشاء تطبيق به أكثر من نموذج:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\4-%20التعامل%20مع%20أكثر%20من%20نموذج\2-%20إنشاء%20تطبيق%20به%20أكثر%20من%20نموذج.htm)
  + [**عرض نموذج من نموذج آخر:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\4-%20التعامل%20مع%20أكثر%20من%20نموذج\3-%20عرض%20نموذج%20من%20نموذج%20آخر.htm)
  + [**اختيار نموذج بدء التشغيل StartUp Form:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\4-%20التعامل%20مع%20أكثر%20من%20نموذج\4-%20اختيار%20نموذج%20بدء%20التشغيل.htm)
  + [**لعبة طريفة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\4-%20التعامل%20مع%20أكثر%20من%20نموذج\5-%20لعبة%20طريفة.htm)

**إنشاء تطبيق به أكثر من نموذج:**

بعد أن أوضحنا التشابه الجمَّ بين تطبيقات الويندوز والإنترنت، سنعود إلى تطبيقات الويندوز مرّةً أخرى.

إن غالبية التطبيقات تتكّون من نماذج عديدة، يناظر كل منها جزءًا من التطبيق مختلف.

في هذا الجزء، سننشئ برنامجا ذا ثلاثة نماذج، وسنسمح للمستخدم بالتنقّل بينها.

وسيكون تطبيقنا ببساطة، هو برنامجا يضمّ محاسب القروض والآلة الحاسبة معًا، حيث سننشئ نموذجا جديدا، ليصبح هو نقطة الانطلاق، التي يمكن الوصول منها للبرنامجين، عن طريق ضغط زر أيّ منهما.

ابدأ مشروعًا جديدًا، وسمّه Calculators، وسمّ نموذجه الرئيسيّ CalculatorsForm.

أضف للنموذج زرّين، وسمّها BtMath و BtLoan، وامنحهما النصّين "Simple Math" و "Simple Loan".

أضف زرا ثالثا للنموذج، وأسمه BtGame.



اضغط بزر الفأرة الأيمن على اسم التطبيق في متصفّح المشاريع، ومن القائمة الموضعية التي ستظهر، اختر "إضافة عنصر موجود" Add Existing Item.. في مربع الحوار الذي سيظهر لك، تصفّح المجلّدات حتّى تصل إلى مجلّد البرنامج MathCalculator، ومنه اختر الملف MathForm.vb، واضغط "موافق" OK.

ستجد أنّك بهذا قد أضفت نموذج الآلة الحاسبة للبرنامج الجديد.

افعل المثل لإضافة النموذج LoanForm من المجلّد LoanCalculator.

**ملحوظة:**

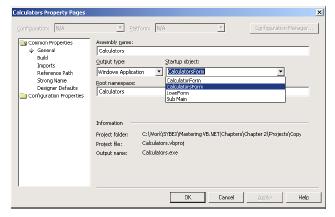
كل مشروع يتكون من مجموعةٍ من الملفّات.. هذه الملفّات تظهر لك في نافذة متصفّح المشاريع، وكلها موجودة في مجلّد المشروع، حتّى إنّك لو فتحت المجلّد Calculators، فستجد أنّه يحتوي على النموذجين CalculatorForm و LoanForm.. لقد تمّ نسخهما من النموذجين الأصليّين عند إضافتهما للبرنامج.

ولنقل مشروع إلى مكان آخر، ما عليك إلا أن تنقل مجلّده إلى هذا المكان.. ولعمل نسخة جديدة من البرنامج، انسخ مجلّده إلى مكان آخر.

**اختيار نموذج بدء التشغيل:**

افترض أنّنا نريد تغيير النموذج الذي يُعرض عند بداية تشغيل البرنامج، فكيف نفعل هذا؟

اضغط زر الفأرة الأيمن فوق اسم المشروع في متصفّح المشاريع، وفي القائمة الموضعية اضغط "خصائص" Properties.. سيظهر لك مربع حوار "صفحات الخصائص" Property Pages.. ستجد عليه قائمّة مركّبة ComboBox عنوانها "كائن بداية البرنامج" StartUp Object.



أسدل هذه القائمة.. سترى أسماء النماذج الثلاثة التي يحتويها البرنامج.. اختر منها النموذج الذي تريد عرضه عند بدء التشغيل، والذي في حالتنا هذه بالطبع يجب أن يكون CalculatorsForm.

**عرض نموذج من نموذج آخر:**

والآن نحتاج لإضافة الكود اللازم لعرض هذين النموذجين الجديدين.

سيصدم مبرمجو VB6 حينما يعرفون أن استخدام اسم النموذج للتعامل مع أدواته وخصائصه ووسائله مباشرةً لم يعد ممكنا!.. وبدلا من ذلك لا بدّ أن تعرّف متغيّرًا جديدًا تشير به إلى النموذج، وعن طريقه يمكنك أن تتعامل مع ما يحلو لك من أعضائه!

هذا هو ما سنفعله عند ضغط أي من الزرين: نعرف متغيّرا من نوع النموذج المطلوب عرضه، من خلال هذا المتغيّر سنستخدم الوسيلة "عرض" Show() لإظهار النموذج.

**Private Sub BtMath\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtMath.Click**

**Dim CalcForm As New CalculatorForm**

**' حمّل النموذج واعرضه على الشاشة**

**CalcForm.Show()**

**End Sub**

**التغليف Encapsulation:**

كيف نستخدم اسم النموذج كنوع من أنواع المتغيرات؟

لقد رأينا من قبل أن النموذج قد تم تمثيله بخليّة Class.. هذا هو ما يمنحك القدرة على تعريف كائنات Objects من هذا النوع.

إن الكائن هو ببساطة، نسخة مستقلّة من الخليّة، يتمّ تحميلها في الذاكرة، بحيث يمكن تغيير قيم خصائصها واستخدام وسائلها، دون التأثير على الخليّة الأصليّة، أو على قيم أو أداء أيّ كائنات أخرى تم تعريفها من نفس الخليّة.

باختصار: رغم أنّ عددا من الكائنات يمكن أن تكون نسخًا من نفس الخليّة، إلا إنّها معزولة تمامًا عن بعضها، لا يتدخّل أيّ منها في عمل الآخر.

هذه الخاصيّة من خصائص البرمجة بالعناصر تسمّى "التغليف" Encapsulation، حيث إنّ خصائص كل كائن ووسائله وأحداثه مغلّفة بداخله، بحيث يعمل الكائن بطريقة مستقلّة تمامًا عن كل الكائنات الأخرى التي أنشئت معه من نفس الخليّة.

ولكنّك ما زلت تتساءل: ماذا تعني كلمة "جديد" New في جملة التعريف Dim؟

إنّها تشرح نفسها، فهي تعني ببساطة: إنشاء نسخة جديدة من خليّة النموذج، والإشارة إليها بالكائن CalcForm.

وهنا ستعاود التساؤل: وماذا لو لم أضع كلمة New في جملة التعريف؟.. ماذا لو كتبتها:

**Dim CalcForm As CalculatorForm**

في هذه الحالة أيضًا الجملة صحيحة، ولكنّ وظيفتها اختلفت، فقد صار معناها: تعريف متغيّر **فارغ** من نوع الكائن CalculatorForm.. هذا المتغيّر غير مرتبطٍ بأي نسخةٍ من النموذج.. إنّه مجرّد هيكل فارغ، ولو استخدمت خصائصه أو وسائله هكذا مباشرةً، فسيحدث خطأ برمجيّ، وستظهر لك رسالة تخبرك بأنّ هذا المتغيّر لم يُنشأ بعد!

ما زالت نظرةُ الحيرةِ تعربدُ في عينيك!.. حسنا.. انظر للأمر كالتالي:

اعتبر أنّ لديك إنسانًا آليّا متطورّا، خرج لتوّه من المصنع.. هذا الإنسان الآليّ تامٌّ ومكتمل التصنيع، ولكنّه رغم هذا لن يعمل.. لماذا؟.. لأنّه بدون برنامج تشغيل!

هذا بالضبط هو حال المتغيّر CalcForm.. إنّه يمتلك كلّ سمات النماذج، ولكنّه لا يعمل، تماما كالآليّ الخامل.. هنا يأتي دور كلّمة New، لتضع فيه "برنامج التشغيل"، الذي يجعل الحياةَ تدبّ فيه، فيتمّ حجز نسخة له في الذاكرة، ويبدأ في أداء وظيفته كنموذجٍ حقيقيّ مستقلٍّ بذاته.

والآن لو شغّلت البرنامج، وضغطت الزرّ الأوّل، فستجد أنّ الآلة الحاسبة قد ظهرت.. ولو ضغطت نفس الزر مرة أخرى، فستجد أن نسخةً جديدةً من الآلة الحاسبة قد ظهرت على الشاشة أيضًا، لأنّك في كلّ مرّة تعرّف نسخةً جديدةً من النموذج عند تنفيذ كلمة New!

والآن يمكنك استخدام كل من نسختَيِ الآلة الحاسبة المعروضتين بطريقة مستقلّة عن الأخرى، فيمكنك مثلا عرض نتيجتين مختلفتين على شاشتيهما في نفس اللحظة.. هذه هي خاصيّة "التغليف" Encapsulation التي شرحناها من قبل.

ولكنّ هذه الطريقة في الأداء غير مرغوبة، فليس من المنطقيّ أن نسمح للمستخدم بعرض أكثر من آلة حاسبة واحدة على الشاشة.

واحد من الحلول، هو نقل جملة تعريف المتغيّر من مستوى حادثة ضغط الزرّ، إلى مستوى النموذج نفسه، وبهذا يتم تعريف المتغيّر مرة واحدةً بغضّ النظر عن عدد مرات ضغط الزرّ.. وطبعًا تعريفه على مستوى النموذج، يجعله قابلا للاستخدام من أي إجراء داخل النموذج، كما ذكرنا من قبل.

صدمةٌ أخرى لمبرمجي VB6: إن عرض أي نموذج على الشاشة، لا يجعله تلقائيّا النموذج النشيط Active، كما كان الحال فيما مضى.

فعند ظهور الآلة الحاسبة، يتحتّم على المستخدم أن يضغطها بالفأرة لتنشيطها أولا قبل استخدامها.. ما عدا ذلك، سيظل النموذج الرئيسيّ هو النموذج النشيط!

ولحلّ هذه المشكلة، سنستخدم وسيلة "تنشيط" Activate الخاصة بالنموذج المعروض، لجعله النموذج النشيط في البرنامج.

والآن هذا هو الكود المعدّل، ليس للإجراء وحده، بل للنموذج الرئيسيّ كلّه، حتّى يمكنك رؤية كيف قمنا بتعريف المتغيرين على مستوى النموذج.

**Public Class CalculatorsForm**

**Inherits System.Windows.Forms.Form**

**Dim CalcForm As New CalculatorForm()**

**Dim LoanForm As New LoanForm()**

**' لاحظ أن اسم المتغير يمكن أن يكون هو اسم الخليّة نفسها.. لا توجد أيّ مشكلة في هذا**

**Private Sub BtMath\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtMath.Click**

**CalcForm.Show()**

**CalcForm.Activate()**

**End Sub**

**Private Sub BtLoan\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtLoan.Click**

**LoanForm.Show()**

**LoanForm.Activate()**

**End Sub**

**End Class**

**لعبة طريفة:**

بقي لدينا الزرّ الثالث.. امنحه النصّ "Play A Game".. دعنا نؤّدي به شيئًا طريفًا:

حينما تضغط هذا الزرّ، سيقفز إلى موضعٍ آخر على النموذج.. هذه هي الكيفيّة:

**Private Sub BtGame\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles BtGame.Click**

**' تغيير موضع الحافّة اليسرى للزر كدالة عشوائية في العرض**

**BtGame.Left = Rnd() \* Me.Width \* 0.8**

**' تغيير موضع الحافّة العليا للزرّ كدالة عشوائية في الارتفاع**

**BtGame.Top = Rnd() \* Me.Height \* 0.8**

**End Sub**

واضح أنّنا نحرّك الزرّ، بتغيير موضع حافتيه اليسرى والعليا.. هذه الحركة ستكون عشوائيّة، نظرًا لأنّنا نستخدم الدالة "عشوائي" Rnd()، التي تنتجُ رقمًا عشوائّيا أكبر من "0" وأصغر من "1"، وحتّى نجعل الرقم كبيرًا، ضربناه في 80% من عرض الزر (أو ارتفاعه).

**التعامل مع العديد من المشاريع:**

يمكن أن يحتوي متصفّح المشاريع على أكثر من مشروع، كلّ منها غير مرتبط بالآخر.

هنا يتبادر سؤال هامّ إلى الذهن: ما فائدة خلطة المشاريع هذه، إذا لم يكن بينها رابط معيّن؟

يحدث أحيانا أن تعمل في أكثر من مشروع بينها مكوّنات مشتركة.. في هذه الحالة يمكنك أن تفتح كل هذه المشاريع معا، حتّى يسهل عليك التنقل بينها.

ولكن الفائدة الحقيقيّة تتّضحُ أثناء إنشاء المكوّنات Components والأدوات الخاصّة User Controls، فهي لا تعمل بمفردها، وإنّما يتم استخدامها داخل المشاريع الأخرى.. في هذه الحالة تحتاج إلى برنامج اختبار، لتجربة هذه الأدوات عليه.

والآن سنحاول أن نتعامل مع مشروعين في وقت واحد: مشروع محاسب القروض، ومشروع الآلة الحاسبة.

ابدأ مشروعا فارغا بضغط "ملف" / "جديد" / "مشروع فارغ" File.New.Blank Solution.. سيظهر لك مربع حوار "مشروع جديد".. سمّ المشروع Calculators واضغط OK.

في نافذة متصفّح المشاريع، ستجد اسم المشروع ولا شيءَ غيره.. ولا حتّى قائمة بالمراجع References، والتي تجدها في كل مشروع آخر.

لإضافة مشاريع، اضغط "ملف" /"إضافة مشروع" / "مشروع موجود" File.Add Project.Existing Project.. يمكنك أيضا أن تضغط بزر الفأرة الأيمن، اسم مجموعة المشاريع Solution’s name في متصفّح المشاريع، ومن القائمة الموضعيّة تختار "إضافة عنصر موجود" / "مشروع" Add Existing Item.Project.

سيظهر لك مربع حوار، استخدمه لإضافة مشروع الآلة الحاسبة Calculator.

كرّر نفس الخطوات لإضافة مشروع محاسب القروض LoanCalculator.

**ملحوظة:**

لو فتحت مجلّد المشروع، فستجد أنّه يحتوي على عدد قليل من الملفّات على غير العادة، ففي هذه المرّة ما زال كلّ من المشروعين يقبع في مجلّده المستقلّ، ممّا يعني أنّ أيّ تغيير يتمّ في المشروعين ضمن مجموعة المشاريع، سيظهر أثره في المشروع الأصليّ.

لديك الآن مجموعة مشاريع تسمّى Calculators، تحتوي على مشروعين.

ماذا إذا أردت تشغيل واحد منهما؟.. إن بيئة التطوير لا تعرف من منهما تعني، وستعرض لك رسالة خطإ.

في هذه الحالة عليك أن تحدّد من منهما هو مشروع بدء التشغيل StartUp Project.

اضغط اسم أحد المشروعين بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط "اجعله مشروع بدء التشغيل" Set As StartUp project.

والآن، اعتبر أنّك تريد كتابة تقرير عن المشروعين.. أفضل اختيار لكتابة الوثائق السريعة، هو صفحات HTML.

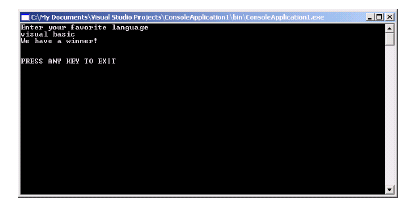
اضغط بزر الفأرة الأيمن، اسم مجموعة المشاريع في متصفّح المشاريع، ومن القائمة الموضعيّة اختر "إضافة عنصر جديد" Add New Item.. في مربع الحوار حدّد "صفحة HTML" HTML Page، وأدخل لها الاسم الذي تريد.. ستجد أن صفحةً فارغة قد ظهرت في المصمّم Designer.. ضع مؤشّر الكتابة في بدايتها، وابدإ الكتابة.. لتنسيق النصوص، استخدم الأزرار على شريط الأدوات.

هذا هو عرض التصميم Design view، وفيه يتم كتابة كود HTML لك آليا، بينما أنت تستخدم الأدوات المرئيّة بسهولة.. ولكن يمكنك أن تنتقل إلى عرض HTML، لكتابة كود HTML يدويا.

**بناء تطبيق اختبار Building Console Application:**

رأينا أمثلةً لتطبيقات الويندوز والإنترنت، وسنرى فيما يلي من فصولٍ أمثلةً لباقي أنواع المشاريع، ما عدا تطبيق الاختبار، فسنتعرف عليه ها هنا، لأوّل وآخر مرّة!

إنّ واحدةً من الخصائص الجديدة في VB.NET، هي قدرتك على إنشاء تطبيقات يتم تنفيذها في شاشة حث الأوامر Command Prompt Window، تلك الشاشة السوداء التي تظهر بها بعض السطور تطالبك بضغط زر ما، أو كتابة شيءٍ ما وضغط زر الإدخال لمواصلة التنفيذ.. ورغم أنها تشبه الدوس DOS كثيرًا، إلا أنها لا تنتمي إلى الدوس.. إنها فحسب مجرد نافذة لإدخال النصوص.



ملحوظة:

هذا التطبيق هو تطبيق عريق، ينتمي للطريقة القديمة التي كان تنفيذ البرنامج فيها يتم على التوالي، دون أن يمنح المستخدم أي حرية في الحركة.. كل ما عليه هو أن يُدخل المطلوب منه ويقرأ النتائج.. هذا قبل أن تظهر البرمجة الموجهة بالأحداث Event Driven Programming، التي يتم فيها تنفيذ أجزاء البرنامج المختلفة تفاعليا مع المستخدم، بالاستجابة لضغطات الأزرار وحركة الفأرة... إلى آخر ما تعلمناه من قبل.

والآن، سنبني تطبيق اختبار ConsoleApplication1، يطلب من المستخدم إدخال اسم لغته المفضلة، وبناء عليه سيعرض رسالة في سطر جديد، لتقييم هذه اللغة.

ابدأ مشروعًا جديدًا، واختر القالب "تطبيق اختبار" Console Application.

ولأنه لا واجهة له، ستجد أن أول ما تراه في تطبيق الاختبار، هو نافذة محرر الكود، وبها ما يلي:

**Module Module1**

**Sub Main()**

**End Sub**

**End Module**

خلافا لتطبيقات الويندوز، التي هي خليّة Class، فإن التطبيق الاختياريّ هو وحدة كود Module .

والإجراء الرئيسيّ Sub Main، هو نقطة دخول التطبيق، حيث إنّه الإجراء الذي ُيستدعى تلقائيًّا عند تشغيل البرنامج، لهذا فلا بد للكود الذي تريد تنفيذه أن يوضع داخل هذا الإجراء.

اكتب هذه الأوامر في الإجراء الرئيسيّ:

**Module Module1**

**Sub Main()**

**Console.WriteLine("أدخل اسم لغتك المفضّلة")**

**Dim language As String**

**language = Console.ReadLine()**

**language = language.ToUpper**

**If language = "VISUAL BASIC" Or language = "VB" \_**

**Or language = "VB.NET" Then**

**Console.WriteLine("Excellent Choice")**

**Else**

**Console.WriteLine("Good Choice")**

**End If**

**Console.WriteLine()**

**Console.WriteLine()**

**Console.WriteLine("PRESS ANY KEY TO EXIT")**

**Console.ReadLine()**

**End Sub**

**End Module**

هذا الكود مماثل لتطبيق الويندوز الذي كتبناه من قبل، ما عدا أنّه يرسل المخرجات لنافذة محث الأوامر، بدلا من عرض رسالة للمستخدم Message Box، وإن كان من الممكن استخدامها هي وبعض عناصر الويندوز الأخرى.. فلو استبدلنا السطر:

**Console.WriteLine("Excellent Choice")**

بالسطر:

**MsgBox ("Excellent Choice")**

فسيتم عرض رسالة للمستخدم، تماما كما عهدناها في تطبيقات الويندوز.

**لماذا هذا العذاب؟!:**

لعلك الآن تتساءل: ماذا يدفع المرء لكتابة مثل هذا التطبيق العقيم؟.. لا شيء أكثر من استخدامه في اختبار بعض أجزاء من الكود، فمثل هذا التطبيق لا يحتاج لواجهة، وليس به أحداث ولا مجيبات أحداث، مما يعني سرعة كتابته.

فمثلا لو أردت اختبار الدالة "الفرق بين التاريخين" DateDiff()، فاكتب هذا التطبيق:

**Sub Main()**

**Console.WriteLine ( \_**

**DateDiff(DateInterval.Day, #3/9/2000#, #5/15/2004#))**

**Console.WriteLine("PRESS ANY KEY TO EXIT")**

**Console.ReadLine()**

**End Sub**

لعلك لاحظت أن السطرين الأخيرين في كل تطبيقات الاختبار ثابتان بنصّهما، ووظيفتهما أن يعطلا تنفيذ البرنامج إلى أن يضغط المستخدم أي زر من لوحة المفاتيح، وذلك حتّى تتاحَ له فرصة مشاهدة النتائج التي أخرجها له البرنامج.

وبخلاف اختبار أجزاءٍ قصيرة من الكود، فإن مثل هذه التطبيقات عديمة النفع، والأفضل أن تتفرغَ ـ بكامل قواك العقليّة ـ لتطبيقات الويندوز والإنترنت، ففيهما الإبداع ـ كلّ الإبداع.

خذ نفسًا عميقًا إذن، استعدادًا للفصول القادمة.

**الملفات التنفيذية Executable Files**

* + [**أين يوجد الملفّ التنفيذيّ للتطبيق؟:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\6-%20الملفات%20التنفيذية\0-%20أين%20الملفّ%20التنفيذيّ.htm)
  + [**توزيع التطبيق Distributing an Application:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\6-%20الملفات%20التنفيذية\1-%20توزيع%20التطبيق.htm)
  + [**إنشاء برنامج ويندوز للإعداد Creating a Windows Installer:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\6-%20الملفات%20التنفيذية\2-%20إنشاء%20برنامج%20ويندوز%20للإعداد.htm)
  + [**خصائص برنامج الإعداد:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\6-%20الملفات%20التنفيذية\3-%20خصائص%20برنامج%20الإعداد.htm)
  + [**أزرار متصفّح المشاريع Solution Explorer Buttons:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\6-%20الملفات%20التنفيذية\4-%20أزرار%20متصفّح%20المشاريع.htm)
  + [**تشغيل برنامج الإعداد Running the Windows Installer:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\02-%20المشاريع\6-%20الملفات%20التنفيذية\5-%20تشغيل%20برنامج%20الإعداد.htm)

**أين الملفّ التنفيذيّ:**

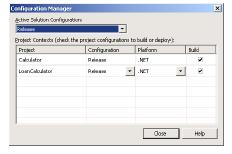
ماذا لو أردت أن تبنى برنامجين تنفيذيين Executable لكل من المشروعين السابقين؟

اضغط "بناء" / "بناء مجموعة المشاريع" Build.Build Solution أو "بناء" / "إعادة بناء مجموعة المشاريع" Build.Rebuild Solution.

سيتمّ إنشاء الملفين التنفيذيّين (.Exe) في مجلّد باسم Bin، داخل مجلّد كل من المشروعين.

تحت مجلّد البرنامج، ستجد أيضًا مجلدا اسمه Obj، وهو يحتوي على مجلّد فرعيّ يسمّى Debug.. ستجد فيه نسخةً أخرى من الملفّ التنفيذي للمشروع.. ولو نقرت هذا الملفّ مرتين بالفأرة ـ مع التأكّد من أن VS .NET غير مفتوحة ـ فستجد أن التطبيق سيُشغّل ـ خارج بيئة التطوير ـ حيث يمكنك استخدامه، تمامًا كأيّ تطبيق ويندوز آخر، ويمكنك أيضا أن تضيف اختصارا Short Cut له على سطح المكتب Desk Top.

إن المجلّد Debug يحتوي على "نسخة التصحيح" Debug version من البرنامج التنفيذيّ، والطبيعيّ أن تقوم بعد أن تنتهي من تنفيذ البرنامج، بتغيير الإعدادات الافتراضيّة للمشروع من "تصحيح" Debug إلى "إصدار" Release.. ولفعل ذلك اضغط "بناء" / "مدير الإعدادات" Build.Configuration Manager، ليظهر لك مربع حوار الإعدادات.



ستجد أن الإعدادات التلقائيّة هي "تصحيح" Debug، وهي تؤدّى إلى إنتاج كود أكثر ملاءمة لتتبّع الأخطاء والتصحيح.

الاختيار الآخر للإعدادات هو "إصدار" Release.. اختره وأغلق مربع الحوار.

الآن لو أعدت بناء المشروع (أو مجموعة المشاريع)، فستجد أن المجلّد Obj يحتوي على مجلّد يسمّى Release، حيث سيحتوي على نسخة جديدة من البرنامج التنفيذيّ، تختلفُ عن نسخة التصحيح، في أنّها لا تحتوي على معلومات رمزيّة للتصحيح، ممّا يعني أنّ تشغيل نسخة الإصدار سيكون أسرع.

**توزيع التطبيق Distributing an Application:**

إن توزيع الملفّ التنفيذيّ بمفرده ليس تصرّفا موفّقا، لأنّه يحتاج إلى ملفّات مساندة Support Files، فإذا لم توزّع هذه الملفات مع برنامجك، فلا تتوقّع أن يعمل الملفّ التنفيذيّ على أي جهاز آخر، إلا في حالة واحدة: إذا كانت VS .NET موجودةً على هذا الجهاز!

إذن فأنت تحتاج إلى برنامج إعداد SetUp، ليوزّع ملفّات المساندة مع البرنامج، فإذا كان بعضها موجودًا على الجهاز، لا ينزله برنامج الإعداد (إلا إذا كانت إصدارا أحدث من الموجود حاليا).

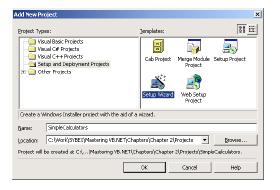
إن مشروع الإعداد يُنشئ ملف ويندوز للإعداد Windows installer file، وله الامتداد .msi، ويحتوي على الملف التنفيذيّ والملفات المساندة له، بالإضافة لقيم التسجيل Registry entries لو كانت مطلوبة، وتعليمات الإنزال.. هذا هو الملفّ الذي توزّعه للمستخدمين، والذي بنقره مرتين بالفأرة، سيقوم بإعداد برنامجك على أجهزتهم.

إن موضوع إنشاء برنامج ويندوز للإعداد Windows installers ضخم بحق، وله مراجع كاملة بمفرده، لهذا فسنكتفي في هذا الفصل بنبش هذا الموضوع فحسب.

**إنشاء برنامج ويندوز للإعداد Creating a Windows Installer:**

لإنشاء مثل هذا البرنامج، يجب عليك أن تضيف مشروع إعداد لمشاريعك.

افتح مجموعة المشاريع المسمّاة Calculators، واضغط "ملف" / "إضافة مشروع" / "مشروع جديد" File.Add Project.New، وفي مربع الحوار اختر "مشاريع الإعداد والتوزيع" Setup and Deployment Projects.

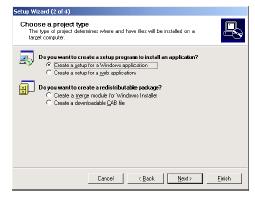


في قسم القوالب Templates Pane سترى خمسة أنواع مختلفة من مشاريع إعداد، أسهلها هو "الإعداد السحري" Setup Wizard، والذي سيحملك عبر عدد من الخطوات المتتابعة لإنشاء مشروع إعداد، الذي هو بدوره، برنامج سحريّ لمساعدة المستخدم لإعداد تطبيقك على جهازه.

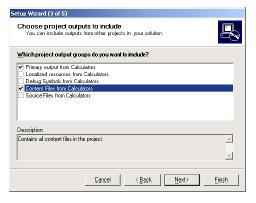
اختر هذا النوع، وأسم المشروع Simple-Calculators، ثم اضغط OK.

- ستظهر لك نافذة ترحيب البرنامج السحريّ.. اضغط "التالية" Next لتجاوزها.

- في النافذة التالية سيُطلب منك اختيار نوع المشروع: فإمّا أن ننشئ برنامج إعداد لتطبيق، أو برنامج إعداد يضيف مكوّنات جديدة لتطبيق سبق إعداده.. طبعا سنختار الحالة الأولى، حيث سنجد لدينا خيارين: هل تطبيقنا هو تطبيق ويندوز أم إنترنت؟.. اختر الأوّل أيضا، ثم اضغط "التالية" Next.



- في النافذة التالية سيُطلب منك إضافة ما تريد من الملفات لبرنامج الإعداد.. طبعًا يجب أن يكون الملفّ التنفيذيّ بين هذه الملفات.. وإذا كان برنامجك يحتوي على "مصادر إعدادات محلّيّة" Localized Resources، حتّى يتمكّن تطبيقك من تغيير اللغة تبعًا لموقع المستخدم، فأضف الاختيار الثاني.. اضغط "التالية" Next.



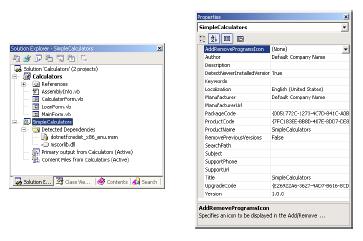
- النافذة التالية تمكّنك من إضافة أية ملفات إضافيّة ليست موجودة في المشروع.. اضغط "التالية" Next.

- ستظهر لك النافذة الأخيرة، وبها ملخص المواصفات التي اخترتَها لمشروع الإعداد.. اضغط "إنهاء" لإغلاق البرنامج السحريّ وإنشاء مشروع الإعداد.

اختر مشروع الإعداد من متصفّح المشاريع، واضغطه بزر الفأرة الأيمن، واختر "خصائص" Properties من القائمة الموضعيّة، لترى خصائصه.

الخبر السعيد هو أنّك لن تحتاج لكتابة أي كود لهذا المشروع.. كلّ ما عليك فعله هو ضبط بعض الخصائص.. دعنا نرى.

**خصائص برنامج الإعداد:**



**خاصية "أيقونة برنامجي الإعداد والإزالة" AddRemoveProgramsIcon:**

وظيفتها واضحة طبعا من اسمها.. ولكن ما يستحقّ التعليق هنا، هو أن VB لا ينشئ فقط برنامج إعداد، ولكنّه ينشئ كذلك برنامجا يتيح للمستخدم إزالة برنامجك ومكوناته من على جهازه.

**خاصية "لاحظ وجود نسخ أحدث معدّة سابقا" DetectNewerInstalledVersion:**

قيمة هذه الخاصيّة في الوضع التلقائيّ "نعم"، وذلك حتّى يتأكّد برنامج الإعداد أولا، مّما إذا كانت هناك نسخة أحدث من تطبيقك معّدة سلفا على الجهاز، فلا يقوم بإنزال نسخة أقدم بدلا منها.

**خاصيّة "المُصنّع" Manufacturer:**

وبها تختار اسم المجلّد الذي سيتمّ إنزال التطبيق فيه، على المسار الذي يختاره المستخدم.

**خاصيّة "المؤلّف" Author:**

ويمكنك أن تكتب بها اسمك.

**خاصية "اسم المنتَج" ProductName:**

الاسم الافتراضيّ لبرنامج الإعداد.

**خاصيّة "العنوان" Title:**

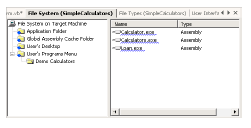
العنوان الذي يظهر على نافذة برنامج الإعداد.

**أزرار متصفّح المشاريع The Solution Explorer Buttons:**

ستلاحظ أن فتح مشروع الإعداد، أدّى إلى استبدال ستة أزرار جديدة بالأزرار التقليدية لمتصفّح المشاريع، هي التي سنصفها حالا:

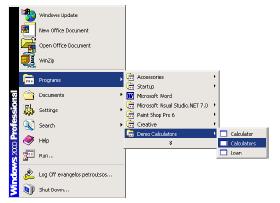
**زر "محرّر نظام الملفات" File System Editor:**

اضغط هذا الزرّ، ليظهر لك محرّر نظام الملفّات على جهاز المستخدم.. على اليسار ستجد قائمةً بها العناصر التي يمكن التأثير عليها في جهاز المستخدم، اثنان منها هما:



"سطح مكتب المستخدم" User's Desktop و"قائمة برامج المستخدم" User's Programs menu، وعن طريقهما تستطيع أن تحدّد إذا ما كنت تريد برنامجك أن يظهر على سطح المكتب أو قائمة البرامج أم لا.. اضغط أيهما بزر الفأرة الأيمن، واضغط "إضافة" Add من القائمة الموضعيّة.. سيبرز لك قائمة فرعيّة بها أربعة عناصر، يمكنك أن تنشئها على جهاز المستخدم، من خلال برنامج الإعداد: "مجلّد" Folder و"مخرجات البرنامج" Project Output و"ملف" File و"تجميع" Assembly.

افرض مثلا أنّنا نريد إنزال برنامجَيِ الآلة الحاسبة ومحاسب القروض، بالإضافة لبرنامج الحواسب Calculator.. في هذه الحالة نريد إضافة ثلاثة أوامر جديدة لقائمة برامج المستخدم Programs menu.. هنا سيكون من الأفضل أن نجمع هذه الأوامر معا في مجلّد خاص بها.



اضغط "قائمة برامج المستخدم" User’s Programs Menu بزر الفأرة الأيمن، واضغط "إضافة" / "مجلّد" Add.Folder.. سيظهر لك مجلد جديد تحت "قائمة برامج المستخدم".. غيّر اسمه إلى Demo Calculators، ثم حدّده وانظر في خصائصه.. تأكّد من أن خاصيّة "أنشئه دائما" AlwaysCreate لها القيمة "صواب" True، وإلا فلن يقوم برنامج الإعداد بإضافة هذا المجلّد لقائمة برامج المستخدم.

بعد ذلك اضغط المجلّد الجديد بزر الفأرة الأيمن، واضغط "إضافة" / "ملف" Add.File.. سيظهر لك مربع حوار، يمكنك بواسطته اختيار الملف التنفيذي لبرنامج الآلة الحاسبة.. كرّر نفس الخطوات لإضافة الملف التنفيذي لمحاسب القروض وللحواسب Calculators.

**زر "محرّر مسجّل القيم " Registry Editor:**

يعرض لك هذا الزر محرّر مسجّل القيم، الذي يمكّنك من إضافة مفاتيح جديدة New Keys لمسجّل قيم الويندوز Registry على جهاز المستخدم.

مثلا يمكنك أن تسجّل تاريخا مشفّرا، تتوقّف بعده النسخة التجريبية Demo Version عن العمل على جهاز المستخدم.

ولكن عليك قبل أن تحاول كتابة أي قيمة في مسجّل القيم، أن تفهمه جيدا أولا، وتتعلم كيف تستخدمه برمجيا من داخل VB.

**زر "محرّر أنواع الملفّات" File Types Editor:**

إذا كان برنامجك يستخدم نوعًا خاصًا بك من الملفات، فيمكنك استخدام هذا المحرّر لجعل برنامجك يعمل، بمجرّد النقر مرتين على أي ملفٍ من هذا النوع.

**تحذير:**

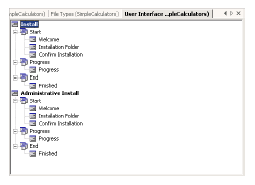
هذه هي خير طريقة لتدمير ارتباطات امتدادات الملفات بالبرامج التي تفتحها على جهاز المستخدم، لهذا حذار أن تفكّر في ربط أيّ نوعٍ من الملفّات التقليديّة ببرنامجك.. استخدم هذه الطريقة فقط مع الملفات ذات الامتدادات التي تبتكرها أنت.

اضغط هذا الزر.. في نافذة أنواع الملفات ستجد عنصرا واحدا: File Types On Target Machine.. اضغطه بزر الفأرة الأيمن، واضغط "إضافة نوع ملف" Add File Type.. سيضيف هذا الأمر نوع ملف جديد، وتحته الفعل "فتح" Verb &Open.. اضغط النوع الجديد وشاهد خصائصه في نافذة الخصائص.. يمكنك أن تتعامل مع الخصائص التالية:

"الوصف" Description، "الامتدادات" Extensions، "الأمر" Command (وهو اسم البرنامج التنفيذي الذي سيفتح هذا الملف).

**زرّ "محرّر واجهة المستخدم" User Interface Editor:**

اضغط هذا الزر لترى خطوات الإعداد.. كل مرحلة في عملية الإعداد تتمّ في خطوة أو خطوتين، حيث تُعرض مربعات حوار مختلفة في كل منها.



بعض مربعات الحوار يحتوي على رسائل.. يمكنك أن تعدّل نصوص هذه الرسائل من خصائص مربع النص، كما تظهر لك في نافذة الخصائص.

ويمكنك أن تضيف مربعات الحوار الخاصة بك، ولكن مع كتابة بعض الكود الذي يتفاعل مع اختيارات المستخدم في مثل هذه الحواريات.

**زرا "أفعال خاصة" Custom Actions و"شروط الانطلاق" Launch Conditions:**

يمكنك هذان الزران من تحديد أفعال خاصّة ومتطلبات النظام.. فمثلا: يمكنك أن تحدّد ألا يتم إعداد برنامجك، إلا على جهاز تتوافر عليه مكوّنات معيّنة.

**تشغيل برنامج الإعداد Running the Windows Installer:**

اضغط "بناء" / "بناء مجموعة المشاريع" Build.Build Solution، ليقوم VB بإنشاء برنامج الإعداد.. لا تجزع إذا استغرق ذلك دقائق عديدة!

في النهاية، سينتج الملف SimpleCalculators.msi، داخل المجلّد Release في مجلّد البرنامج SimpleCalculators.. سيكون حجمه حوالي 15 ميجا بايت!

تتساءل عن تلك المحتويات التي أوصلت هذا الملفّ لهذا الحجم؟.. ستجدها مسرودةً في نافذة المخرجات Output window.

انسخ برنامج الإعداد إلى جهاز لا يحتوي على VS .NET وحاول تجربته.. أو جرّبه على جهازك إذا تعذّر الخيار الأوّل.

انقر أيقونة برنامج الإعداد مرتين، وتابع خطوات التثبيت.

**نصيحة:**

نظرًا لأن الملفات المساندة للبرنامج التنفيذيّ يتم وضعها في المجلد الذي يحمل اسم الشركة CompanyName، حاول أن تجعل هذا المجلّد ثابتا لبرامجك، حتّى لا يتمّ نسخ ملفات المساندة المشتركة بينها أكثر من مرة على جهاز المستخدم.

1. المقابض: هي المربعات الصغيرة على الحوافّ، التي يتمّ منها تغييرُ حجم الأداة. [↑](#footnote-ref-1)