جامعة البعث
الكلية التطبيقية
قسم تقنيات الحاسوب

# برمجة متقدمة
# السنة الثانية
# عملي
# محاضرة الخامسة

م. زكي نقولا م. بتول الليوس م. سارة معروف

# الاستثناءات ومعالجة الأخطاء في لغة سي شارب#C

هناك نوعان أساسيّان من الأخطاء التي قد تنتج عن البرنامج: النوع الأوّل هي أخطاء أثناء التصميم، أي أثناء كتابة الشيفرة، وهي أخطاء يمكن اكتشافها بسهولة من خلال مترجم سي شارب الذي يتعرّف عليها ويعطيك وصفًا عنها، وقد يزوّدك أيضًا ببعض المقترحات للتخلّص منها. بالنسبة للنوع الثاني من الأخطاء فهي التي تنتج أثناء تنفيذ البرنامج runtime errors. توجد الكثير من الأسباب لحدوث مثل هذه الأخطاء. فمن القسمة على صفر إلى القراءة من ملف غير موجود أو حتى استخدام متغيّر مصرّح على أنّه من نوع مرجعيّ ولكن لم تتم تهيئته بعد بمرجع إلى كائن. عند حدوث مثل هذه الأخطاء ترمي بيئة التنفيذ المشتركة CLR استثناءً exceptionيحتوي على معلومات بخصوص الخطأ الذي حدث، فإذا كنت مستعدًّا لالتقاط هذا الاستثناء فستنجو، وإلّا سيتوقّف برنامجك عن العمل بصورة مفاجئة.

## التقاط استثناء من خلال عبارةtry-catch

إذا صادفتك عبارة برمجيّة تحتوي على عمليّة حسابيّة أو على استدعاء لتابع آخر، أو أيّ شيء قد يثير الريبة في نفسك، فمن الممكن مراقبتها أثناء تنفيذ البرنامج باستخدام عبارة try-catch. تتألّف هذه العبارة من قسمين: القسم الأوّل هو قسم المراقبة try، والقسم الثاني هو قسم الالتقاط catch. الشكل "الأبسط" لهذه العبارة هو التالي:

```
try
{
        //عبارة برمجيّة مريبة
}
catch(Exception exp)
{
        //هنا يُلتقط الاستثناء وتتمّ معالجته
}
```

يمكن أن يحوي قسم tryعلى عبارات برمجيّة بقدر ما ترغب. عندما يصادف البرنامج أثناء التنفيذ خطأً ما، سيتوقّف التنفيذ عند العبارة التي سبّبت الخطأ، ثم ينتقل فورًا إلى قسم الالتقاط catch. لاحظ معي أنّ قسم catchسيُمرَّر إليه وسيط من الصنف Exception. الصنفExceptionموجود ضمن نطاق الاسم System، وهو الصنف الأب لجميع الاستثناءات. إذ أنّ أي استثناء مهما كانت صفته (بما فيها الاستثناءات التي يمكنك أن تكتبها أنت) يجب أن ترث من هذا الصنف.

بعد حدوث الاستثناء والانتقال إلى قسم catch، سيحتوي الوسيط expعلى معلومات حول مكان حدوث الاستثناء وسبب حدوثه، وغيرها من المعلومات التي قد تكون مفيدة لمستخدم البرنامج. تجدر الملاحظة بأنّ البرنامج لن يدخل إلى القسم catchأبدًا ما لم يحدث استثناء ضمن القسم try

```
1       using System;
2
3       namespace Lesson16_01
4       {
5           class Program
6           {
7               static void Main(string[] args)
8               {
9                   int x = 5;
10                  int y = 0;
11                  int result;
12
13                  try
14                  {
15                      result = x / y;
16                  }
17                  catch(Exception exp)
18                  {
19                      Console.WriteLine("The following error has occurred:");
20                      Console.WriteLine(exp.Message);
21                  }
22
23                  Console.WriteLine("Good Bye!");
24              }
25          }
26      }
```

من الواضح أنّ هذا البرنامج يتجّه لأن يجري عمليّة قسمة على صفر في السطر 15 ضمن القسم try. عند وصول البرنامج إلى السطر 15 وإجراء عمليّة القسمة هذه، سيتولّد استثناء يؤدّي إلى انتقال التنفيذ مباشرةً إلى قسم catch في السطر 17. في قسم catch يعرض البرنامج معلومات عن هذا الخطأ باستخدام الخاصيّة Message لكائن الحدث exp. في النهاية يعرض البرنامج في السطر 23 رسالة توديعيّة للمستخدم.

نفّذ البرنامج لتحصل على الخرج التالي:

```
The following error has occurred:
Attempted to divide by zero.
Good Bye!
```

جرّب الآن تغيير قيمة المتغيّر y لتصبح 1 مثلًا وأعد تنفيذ البرنامج. ستلاحظ ظهور الرسالة التوديعيّة فقط على الشاشة، أي أنّه لم يحدث أي استثناء هذه المرّة. على كلّ الأحوال لا ينصح باستخدام معالجة الاستثناءات من أجل حالة القسمة على صفر في البرنامج السابق.

**ملاحظة** :في الواقع يمكن الاستغناء عن الوسيط الذي يمرّر إلى قسم catch بشكل كامل، وفي هذه الحالة لن يكون بإمكانك الحصول على معلومات حول الاستثناء المُلتقط. سيبدو شكل عبارة try-catch على الشكل التالي:

```
try
{
        //عبارة برمجيّة مريبة
}
catch
{
        //هنا يُلتقط الاستثناء وتتمّ معالجته
}
```

من الممكن استخدام هذا الأسلوب إذا كنّا على يقين حول طبيعة الخطأ الذي سيحدث.

# عبارة try-catch أكثر تطورا

تصادفنا في بعض الأحيان حالات يكون من الضروري معها مراقبة أكثر من عبارة برمجيّة مريبة ضمن قسم try. لقد اتفقنا أنّه عند حدوث أيّ استثناء ضمن قسم try سينتقل التنفيذ إلى قسم catch. ولكن كيف سنميّز العبارة التي سبّبت هذا الاستثناء في قسم try؟

توجد العديد من الأصناف التي ترث من الصنف Exception والتي يُعتبر كلّ منها استثناءً مخصّصًا أكثر للمشكلة التي قد تحدث. فمثًلا كان من الممكن في البرنامج

السابق أن نستخدم الصنف DivideByZeroException بدلًا من الصنف Exception في عبارة catch، وذلك لأنّه يرث (بشكل غير مباشر) من الصنف Exception، وسيعمل البرنامج كما هو متوقّع. ولكن في هذه الحالة لن تستطيع catch التقاط سوى الاستثناءات التي تنتج عن القسمة على صفر.

في كثير من الحالات قد تتسبّب العبارات البرمجيّة الموجودة في قسم try باستثناءات متنوّعة لا توجد علاقة فيما بينها. مما يفرض علينا استخدام الصنف Exception لكي نلتقط بشكل مؤكّد أي استثناء قد يصدر عنها، أو أن تساعدنا سي شارب في هذا الخصوص! في الحقيقة الخيار الثاني هو الأفضل وهو جاهز. يمكننا في الواقع إضافة أقسام catch أخرى بقدر ما نرغب بعد قسم try.

فتح ملف نصي ومحاولة قراءة محتوياته. لاحظ أنّنا سنستخدم هنا نطاق الاسم System.IO.

```
1       using System;
2       using System.IO;
3
4       namespace Lesson16_02
5       {
6           class Program
7           {
8               static void Main(string[] args)
9               {
10                  string contents;
11
12                  Try
13                  {
14                      using (StreamReader sr = new StreamReader("myfile.txt"))
15                      {
16                          contents = sr.ReadLine();
17                      }
18
19                      Console.WriteLine("This file contains {0} characters.", contents.Length);
20                  }
21                  catch(NullReferenceException nullExp)
22                  {
23                      Console.WriteLine("The file does not contain any data.");
24                  }
25                  catch(FileNotFoundException notFoundExp)
26                  {
27                      Console.WriteLine("File: {0} not found!", notFoundExp.FileName);
28                  }
29              }
30          }
31      }
```

ستخدم هذا البرنامج قسميّ .catchالقسم الأوّل (الأسطر من 21 إلى 24) يلتقط استثناءً من النوع NullReferenceExceptionوهذا يحدث عند محاولة استدعاء تابع أو خاصيّة من متغيّر يحتوي على nullبدلًا من مرجع لكائن حقيقي. أمّا القسم الثاني (الأسطر من 25 إلى 28) فهو يلتقط استثناءً من النوع FileNotFoundExceptionوالذي يحدث عند محاولة القراءة من ملف غير موجود. نفّذ البرنامج السابق وستحصل على الرسالة التالية في الخرج:

```
File: C:\\Users\Husam\documents\visual studio
2015\Projects\Lesson16_02\Lesson16_02\bin\Debug\myfile.txt not found!
```

وهذا طبيعي تمامًا لأنّني لم أنشئ الملف myFile.txtفي هذا المسار. لاحظ كيف يضيف الصنف FileNotFoundExceptionخاصيّة جديدة له وهي FileName(السطر 27) من النوع stringالتي تحوي مسار الملف مع اسمه.

أنشئ الآن الملف myFile.txt واتركه فارغًا، ثمّ ضعه ضمن نفس المجلّد الذي يحوي الملف التنفيذي للبرنامج (موجود ضمن bin\Debug\).

أعد تنفيذ البرنامج وستحصل على الرسالة التالي:

```
The file does not contain any data.
```

السبب في ظهور هذه الرسالة هو الاستثناء NullReferenceException وذلك لأنّنا حاولنا الوصول إلى الخاصيّة Length (تعطينا عدد المحارف الموجودة ضمن متغيّر نصي) من المتغيّر النصي contents رغم أنّه يحتوي على null ) تذكّر بأنّنا تركنا الملف myFile.txt فارغًا.(

اذهب إلى الملف myFile.txt الذي أنشأناه قبل قليل، واكتب بعض الكلمات ضمنه واحفظ الملف، ثمّ أعد تنفيذ البرنامج Lesson16_02 من جديد. يجب الآن أن تحصل على رسالة تخبرك بعدد الأحرف التي كتبتها ضمن الملف.

**ملاحظة** :يجب الانتباه إلى ترتيب أقسام .catch فلو كان مثلًا أوّل قسم catch موجود بعد قسم try يلتقط استثناءً من النوع Exception فعندها لن يستطيع أي قسم لاحق التقاط أي استثناء، لأنّ جميع الاستثناءات سيلتقطها هذا القسم الأوّل. السبب في ذلك أنّ الصنف Exception هو الأب العام لجميع أصناف الاستثناءات الأخرى، فيمكن له التقاطها.

## عبارةtry-catch-final

يمكن إضافة قسم أخير لعبارة try-catch اسمه .final وكما يوحي اسمه، فهذا القسم يمكن له أن يحتوي على عبارات برمجيّة سيتمّ تنفيذها بعد أن يدخل البرنامج إلى القسم try العائد له. وذلك سواءً أحدث استثناء ضمن try أم لم يحدث. تكمن فائدة وجود هذا القسم، في أنّه قد نواجه أحيانًا بعض الحالات التي تتطلّب إجراء بعض المهام عندما نفرغ من قسم try مثل إغلاق بعض المصادر المفتوحة، أو تحرير الذاكرة بشكل فوري وغيرها. الشرط الوحيد لاستخدام هذا القسم الاختياري هو أن يكون آخر قسم في عبارةtry-catch

```
using System;
using System.IO;

namespace Lesson16_03
{
        class Program
        {
                static void Main(string[] args)
                {
                        string contents;

                        try
                        {
                                using (StreamReader sr = new StreamReader("myfile.txt"))
                                {
                                        contents = sr.ReadLine();
                                }
                                Console.WriteLine("This file contains {0} characters.", contents.Length);
                        }
                        catch(NullReferenceException nullExp)
                        {
                                Console.WriteLine("The file does not contain any data.");
                        }
                        catch(FileNotFoundException notFoundExp)
                        {
                                Console.WriteLine("File: {0} not found!", notFoundExp.FileName);
                        }
                        finally
                        {
                                Console.WriteLine("Good Bye!");
                        }
                }
        }
}
```

سواءً كان الملف `myFile.txt` موجودًا أم غير موجود، أو كان يحتوي على بيانات أم فارغاً، ستظهر العبارة Good Bye!على الشاشة.

**ملاحظة**: من الأفضل دومًا أن تحاول عدم استخدام عبارة try-catch وأن تستخدم عبارة `if` لاختبار الحالات التي تواجهك قبل تنفيذها. استخدم try-catch إذا كان ذلك ضروريًّا. والسبب في ذلك أنّ عمليّة معالجة الأخطاء بشكل عام تتطلّب المزيد من الموارد المخصّصة للبرنامج، مما قد يؤثّر على أداء البرنامج في حال تمّ استخدامها بشكل غير مدروس.

# مقدمة إلى المفهوم الكائني

تُعتبر لغة سي شارب لغة برمجة كائنيّة صرفة pure object oriented programming language فكلّ ما تشاهده أمامك في سي شارب عبارة عن كائن.

لكي نفهم ما هو الصنف وما هو الكائن اسمع منّي هذه القصّة: نتبع نحن البشر إلى ما يسمّى بالصنف الإنساني. يُعرّف هذا الصنف المزايا التي يجب أن يتمتّع بها كلّ إنسان. فمثلًا لكلّ إنسان اسم وطول ووزن ولون عينان وبصمة إبهام مميّزة تميّزه عن أيّ إنسان آخر. يُعرّف الصنف class الإنسانيّ هذه الصفات السابقة، بحيث أنّ كلّ كائن object إنسانيّ من هذا الصنف تكون له مثل هذه الصفات ولكنّ مع مجموعة خاصّة من القيم لها. فمثلًا الكائن من الصنف الإنساني هو إنسان قد يكون اسمه سعيد وطوله 180 سم ولون عينيه أسود وله بصمة إبهام مميّزة، وهذا الإنسان يختلف عن كائن إنسانيّ آخر، اسمه عمّار وطوله 175 سم ولون عينيه بنيّ وله أيضًا بصمة إبهام مميّزة خاصّة به، وهكذا. ندعو الصفات السابقة بالخصائص Properties، فالصنف Class يعرّف الخصائص، أمّا الكائن Object فيتمتّع بهذه الخصائص ولكن مع مجموعة قيم لها تميّزه عن كائنٍ آخر.

أمر آخر، يُعرّف الصنف الإنساني أيضًا سلوكيّات أو إجراءات معيّنة خاصّة للكائنات التي تعود للصنف الإنسانيّ. فهناك مثلًا سلوكيّات المشي والجري والضحك. وفي الغالب أنّ كل كائن يُعبّر عن هذه السلوكيّات بشكل يراعي خصوصيّته. فلكلّ منّا أسلوب مختلف في الضحك. كما يمتلك كلّ منّا أسلوب مختلف في المشي والجري، فقد تميّز إنسانًا لا ترى وجهه من خلال مشيته فقط وهذا أمر واردٌ جدًّا. مثل هذه السلوكيّات Methods نصطلح عليها في البرمجة بالتوابع. فالصنف الإنسانيّ يُعرّف وجود مثل هذه السلوكيّات ولكلّ كائن إنسانيّ الحريّة في التعبير عن هذه السلوكيّات بالشكل الذي يرغبه.

التابع في البرمجة يضم تعليمات برمجية يجري تنفيذها عند استدعائه. يعالج ويتعامل هذا التابع عادةً مع الخصائص والتوابع الأخرى الموجودة ضمن نفس الكائن. نسمي التوابع والخصائص بأعضاء الصنف class members

## المبادئ العامة للمفهوم كائني التوجه

هناك مبدآن أساسيّان ينبغي أن تتمتّع بها أيّ لغة برمجة تدعم المفهوم كائنيّ التوجّه وهما: التغليف Encapsulation والوراثة Inheritance.وهناك مفهوم مهم آخر يستند إلى الوراثة وهو التعدّديّة الشكلية.Polymorphism

### التغليف Encapsulation

وهو مبدأ جوهري في البرمجة كائنيّة التوجّه، وهو أحد أسباب ظهور هذا المفهوم. يُقرّر هذا المبدأ أنّه ليس من المفترض أن نطّلع على آلية العمل الداخلية للكائن. ما يهمنا هو استخدام الكائن وتحقيق الغرض المطلوب بصرف النظر عن التفاصيل الداخليّة له. تأمّل المثال البسيط التالي:

عندما نقود السيّارة ونريد زيادة سرعتها فإنّنا بكلّ بساطة نضغط على مدوسة الوقود. لا أعتقد أنّ أحدًا يهتمّ بالآلية الميكانيكيّة التي تقف وراء الضغط على مدوسة الوقود. فالمطلوب هو زيادة سرعة السيّارة فحسب دون الاهتمام بالتفاصيل الداخليّة. فالسيّارة تُغلّف encapsulate التفاصيل الميكانيكيّة الداخليّة التي تقف وراء زيادة سرعة السيّارة. السيّارة في هذا المثال هو كائن Object. وعمليّة زيادة السرعة هي سلوكيّة (تابع) Method من كائن السيّارة.

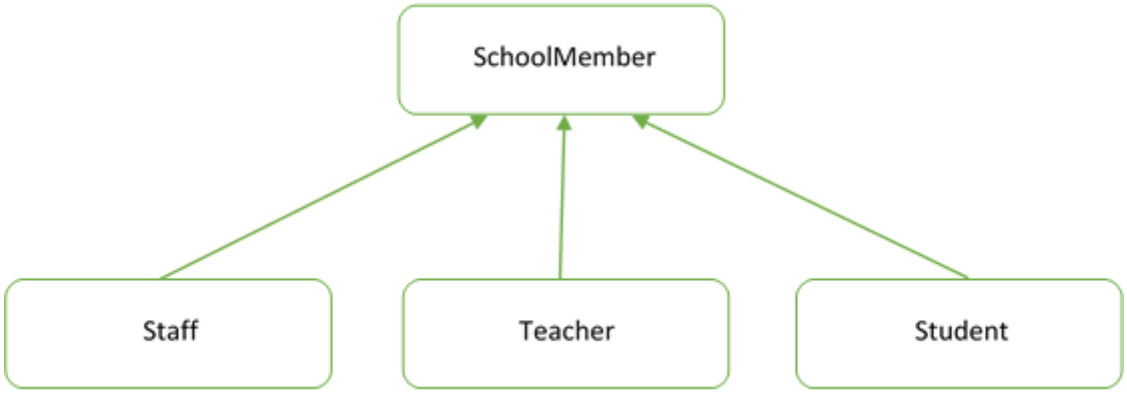
هناك مثال آخر كثيرًا ما نراه أثناء تجوّلنا في الشوارع ومحطّات القطار وصالات الانتظار، وهو آلات تحضير المشروبات الساخنة. نقف أمام الآلة نُدخل النقود ثمّ نضغط على زرّ محدّد لنحصل على المشروب الساخن الذي نرغب به. لا نهتمّ عادةً بالتفاصيل الداخليّة التي تحدث ضمن الآلة عندما نضغط أحد الأزرار للحصول على كوب من القهوة. فالآلة هنا تُعتبر كائنًا، وعمليّة الحصول على كوب من القهوة هي سلوكيّة Method من هذا الكائن. فهذه الآلة تعمل على تغليف encapsulate التفاصيل الداخليّة لعمليّة التحضير، فكلّ ما نفعله هو ضغط الزر ومن ثمّ نحصل على الناتج المطلوب. فإذا ما أُجري تعديل في الآلة بحيث تتغيّر طريقة تحضير مشروب ساخن لجعله أفضل وأكثر لذّة، فإنّ ذلك لن يؤثّر مطلقًا على أسلوب التعامل مع الآلة للحصول على نفس المشروب، ولن نلاحظ هذا التعديل إلّا بعد تذوّقنا للمشروب وملاحظة الفرق في المذاق.

### الوراثة Inheritance

تُعتبر الوراثة من أهم أشكال إعادة الاستخدام للمكوّنات البرمجيّة، حيث يعمل الصنف الجديد على الاستفادة من المكوّنات الموجودة مسبقًا ضمن الصنف الذي "يرث" منه ويجري عليها بعض التعديلات التي تناسبه على نحو مخصّص. فبدلًا من إنشاء صنف جديد من الصفر، يمكننا إنشاء صنف يعتمد على صنف آخر ويستفيد من خصائصه وسلوكيّاته (توابعه) الموجودة مسبقًا ثمّ يكيّفها أو يضيف عليها. نسمّي الصنف الأساسي الذي نرث منه بالصنف الأب. أمّا الصنف الذي يقوم بعمليّة الوراثة فنسمّيه بالصنف الابن أو بالصنف المشتق.

لتثبيت الفكرة لنتناول المثال التالي. في المدارس هناك ثلاثة أنواع أساسيّة من الأشخاص المتواجدين فيها: الطلاب والمدرّسون والإداريّون. يمكننا بناء صنف عام يُمثّل أي شخص يعمل في المدرسة وليكن SchoolMember يحتوي هذا الصنف على خصائص مثل: الاسم والكنية واسم الأب واسم الأم وتاريخ الميلاد ورقم الهاتف.

يمكننا البناء على هذا الصنف عندما نريد إنشاء أصناف أكثر "تخصّصًا" منه. مثل الصنف الذي يُعبّر عن الطلاب Student والصنف الذي يُعبّر عن المدرّسين Teacher، والصنف المُعبّر عن الإداريين Staff. يرث كلّ صنف منها من الصنف الأب SchoolMember فيصبح لكلّ منها نفس الخصائص الموجودة ضمن الصنف SchoolMember بشكل تلقائيّ.




من الواضح أنّ الصنف Student مخصّص أكثر من الصنف SchoolMember فهو يحتوي بالإضافة إلى الخصائص الموجودة في SchoolMember خصائص فريدة خاصّة به. فمثلًا من الممكن أن يحتوي على الخاصيّة التي تعبّر عن الصفّ الحالي Grade وعن السلوك العام Behavior للطالب، أمّا صنف المدرّس Teacher فمن الممكن أن يحتوي (بالإضافة إلى الخصائص الموجودة ضمن SchoolMember) على خاصيّة Course التي تُعبّر عن المقرّر الذي يدرّسه (رياضيّات، فيزياء ...الخ) والخاصيّة WeeklyHours التي تعبّر عن عدد الساعات التدريسيّة الأسبوعيّة المكلّف بها. وينطبق نفس المفهوم تمامًا على الصنف Staff الذي يعبّر عن الموظّفين الإداريين في المدرسة.

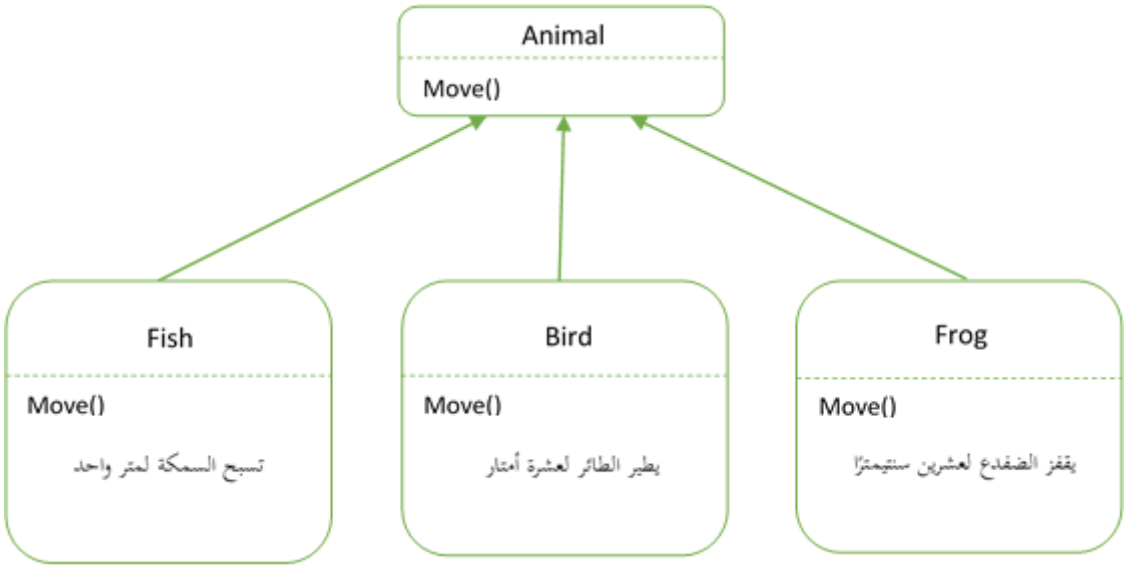
فالوراثة تنتقل بنا من الشكل الأكثر عموميّةً SchoolMember إلى الشكل الأكثر تخصيصًا مثل Student. وفي الحقيقة كان من الممكن أن نتابع عمليّة الوراثة اعتبارًا من الصنف Staff فهناك قسم التوجيّه وهناك أمانة السر والإدارة وغيرها، وكلّ منها يمكن أن يرث من الصنف Staff.

التعددية الشكلية Polymorphism

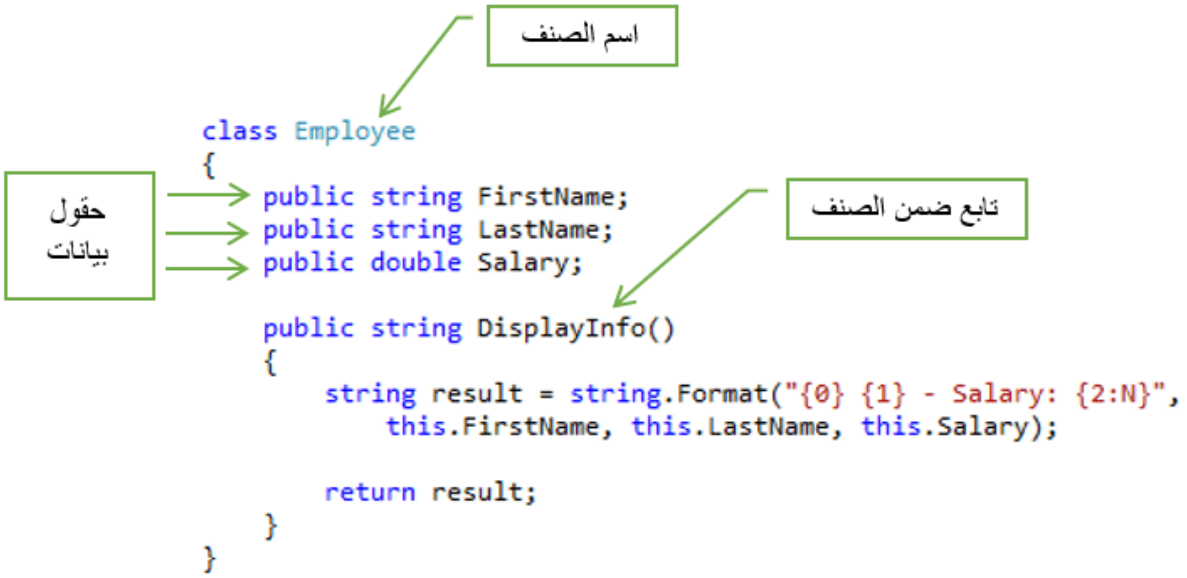
بفرض أنّنا نريد بناء برنامج يحاكي الحركة الانتقاليّة لعدّة أنواع من الحيوانات لدراسة حيويّة.

كلّ من أصناف السمكة Fish والطائر Bird والضفدع Frog ترث من الصنف Animal الذي يمثّل أيّ حيوان.

بفرض أنّ الصنف Animal يحتوي على سلوكيّة (تابع) اسمها Move (تُعبّر عن الانتقال)، فكما نعلم أنّ هذه السلوكيّة ستصبح وبشكل تلقائي موجودة ضمن أيّ صنف يرث من الصنف Animal، وهنا تكمن التعدديّة الشكليّة. فكل صنف من الأصناف Fish وBird وFrog يُعبّر عن عملية الانتقال Move بشكل مختلف. فالسمكة ربما تنتقل عن طريق السباحة مترًا واحدًا عند استدعاء التابع Move. أمّا الطائر Bird فمن الممكن أي يطير مسافة 10 متر عند كل استدعاء للتابع Move، وأخيرًا فإنّه من الممكن للضفدع أن يقفز مسافة 20 سنتيمتر كلّما استدعي التابع Move. فالتابع Move المعرّف ضمن الصنف Animal يمكن التعبير عنه بأشكال متعدّدة ضمن الأصناف الأبناء Fish وBird وFrog كلٌّ بحسب حاجته.




يمكن التصريح عن صنف في سي شارب باستخدام الكلمة المحجوزة `class` يليها اسم الصنف وهو يتبع لنفس قواعد التسمية للمتغيّرات، علمًا أنّه يفضّل أن يكون الحرف الأوّل من اسم الصنف حرفًا طباعيًّا كبيرًا. انظر إلى الشكل التالي حيث نرى الصنف البسيط `Employee` والذي يُعبّر عن موظّف في إحدى الشركات:




يحتوي هذا الصنف على ثلاثة حقول بيانات data fields هي:

- الاسم `FirstName`
- الكنية `LastName`
- الراتب `Salary`

تستطيع اعتبارها حاليًا أنّها تمثّل خصائص للصنف `Employee`، كما يحتوي هذا الصنف على تابع وحيد اسمه `DisplayInfo` الهدف منه هو الحصول على تمثيل نصيّ لكلّ كائن ننشئه من هذا الصنف كما سنرى بعد قليل، يشبه التابع إلى حدٍّ كبير الدّالة function في لغات البرمجة الأخرى. لا يتطلّب هذا التابع أيّ وسائط في حين أنّه يُرجع قيمة نصيّة من النوع string. هذه الحقول بالإضافة إلى التابع السابق تُعتبر أعضاء ضمن الصنف `Employee` كما ذكرنا ذلك مسبقًا. تقع أعضاء أيّ صنف ضمن حاضنتيه.

لاحظ الكلمة المحجوزة `public` والموجودة قبل كلّ تصريح لحقل أو تابع ضمن الصنف `Employee`. هذه الكلمة عبارة عن مُحدّد وصول access modifier.

تتحكّم محدّدات الوصول بقابلية الوصول إلى أعضاء الصنف من خارجه، سنتعامل مع نوعين آخرين من محدّدات الوصول وهما `private` و `protected`

إذا أردنا إنشاء كائن جديد من الصنف Employee فعلينا التصريح عن متغيّر مناسب من النوع Employee وذلك على الشكل التالي:

```
Employee empObject;
```

صرّحنا عن المتغيّر empObject على أنّه من النوع Employee. لاحظ التشابه في التصريح عن المتغيّرات بين أنواع موجودة ضمن سي شارب وبين أنواع ننشئها بأنفسنا.

التصريح السابق غير كافي لإنشاء الكائن. لإنشاء كائن من النوع Employee علينا استخدام العامل new الذي يعمل على إنشاء كائن من أيّ صنف نرغبه ويعمل على إعادة المرجع (العنوان) لذلك الكائن في الذاكرة. استخدام العامل new سهل حيث يمكننا كتابة ما يلي بعد عبارة التصريح السابقة:

```
empObject = new Employee();
```

يقوم العامل newبإنشاء كائن جديد من الصنف Employeeثمّ يُسند مرجع (عنوان) هذا الكائن ضمن المتغيّر empObject. لاحظ القوسين الموجودين بعد اسم الصنف Employee.في الحقيقة يُعبّر هذين القوسين عن استدعاء لبانية constructor الصنف Employeeعند إنشاء الكائن. ولكن أين هذه البانية؟ هذا ما سنراه بعد قليل. يمكن الآن الوصول إلى الحقول والتوابع الموجودة ضمن الكائن عن طريق كتابة المتغيّر الذي يحوي العنوان إلى الكائن (أي المتغيّر empObject)ثم نضع نقطة وبعدها اسم الحقل أو التابع الذي نريد الوصول إليه. في العبارة التالية سنسند القيمة "Mohammad" إلى الحقل FirstNameمن الكائن ( empObjectالكائن الذي يشير إليه :(empObject

```
empObject.FirstName = "Mohammad";
```

```
using System;

namespace Lesson06_01
{

    class Employee
    {
        public string FirstName;
        public string LastName;
        public double Salary;

        public string DisplayInfo()
        {
            string result = string.Format("{0} {1} - Salary: {2:N0}",
                this.FirstName, this.LastName, this.Salary);

            return result;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Employee employee1, employee2;

            employee1 = new Employee();
            employee1.FirstName = "Mohammad";
            employee1.LastName = "Mansoor";
            employee1.Salary = 1000;

            employee2 = new Employee();
            employee2.FirstName = "Saleh";
            employee2.LastName = "Mahmoud";
            employee2.Salary = 2500;

            Console.WriteLine("First Employee: {0}", employee1.DisplayInfo());
            Console.WriteLine("Second Employee: {0}", employee2.DisplayInfo());
        }
    }
}
```

عند تنفيذ البرنامج سنحصل على الخرج التالي:

```
First Employee: Mohammad Mansoor - Salary: 1,000.00
Second Employee: Saleh Mahmoud - Salary: 2,500.00
```

يقع التصريح عن الصنف Employee في الأسطر بين 6 و 19 ويحتوي هذا الصنف كما رأينا قبل قليل على أربعة أعضاء وهي عبارة عن ثلاثة حقول FirstName و LastName و Salary بالإضافة إلى التابع DisplayInfo الموجود بين السطرين 12 و18. تنحصر وظيفة هذا التابع في الحصول على التمثيل النصيّ لأيّ كائن ننشئه من الصنف Employee.

يحتوي التابع DisplayInfo على أسلوب جميل لتنسيق النصوص يشبه ذلك الأسلوب الذي كنّا نستخدمه مع التابع WriteLine. يحتوي الصنف string على تابع اسمه Format يقبل عدّة وسائط (السطر 14) أولها نصّ تنسيقي، أمّا الوسائط التالية فهي القيم التي ستجد لها أمكنةً ضمن النص التنسيقي، كما كنّا نستخدم التابع WriteLine بالضبط. يُرجع التابع Format نصًّا منسّقًا بحسب القيم الممرّرة له. الشيء الوحيد المختلف هو كيفيّة تنسيق قيمة الراتب Salary باستخدام مُحدّد التنسيق N0 : الموجود ضمن {N0:2}. يخبر هذا المحدّد التابع Format أنّ القيمة التي ستوضع في هذا المكان (وهي قيمة Salary) يجب أن تُنسّق على شكل رقم ذي فاصلة آلاف وبدون فاصلة عشريّة. يفيد مثل هذا التنسيق في الحصول على أرقام منسّقة بشكل محترف تُعبّر عن الراتب الذي يحصل عليه الموظّف وهي تبدو مثل 1,000 أو 2,500. جرّب استخدام التنسيق {N1:2} و {N2:2} ولاحظ الفرق.

لاحظ أنّني قد استخدمت الكلمة المحجوزة this متبوعةً بنقطة قبل اسم كل حقل. في الحقيقة تُشير هذه الكلمة إلى الكائن الحالي الذي يتمّ منه استدعاء التابع DisplayInfo كما سنرى ذلك بعد قليل. أمّا لإرجاع القيمة النصيّة من التابع DisplayInfo فإنّنا ببساطة نستخدم الكلمة المحجوزة return ونضع بعدها القيمة المراد إرجاعها.

الصنف Program المصرّح عنه في الأسطر بين 21 و 40 هو الصنف الذي تعاملنا معه في جميع البرامج التي كتبناها حتى الآن. يحتوي هذا الصنف على التابع Main الذي يمثّل نقطة الدخول للبرنامج كما نعلم.

يبدأ التابع Main بالتصريح عن متغيرين من النوع Employee وهما employee1 و employee2 ثمّ ينشئ كائنًا من النوع Employee باستخدام العامل new (السطر 27) ويسنده إلى المتغيّر employee1. بعد ذلك يمكن استخدام أيّ حقل أو تابع معرّف ضمن الصنف Employee عن طريق المتغيّر employee1 بشرط أن يكون له محدّد وصول public كما هو واضح في الأسطر من 28 حتى 30. يتكرّر نفس الأمر بالنسبة للمتغيّر employee2 الذي سيحمل كائنًا مختلفًا عن الكائن الموجود ضمن employee1.

أخيرًا وفي السطرين 37 و38 يتم طباعة التمثيل النصيّ لكلّ من الكائنين باستخدام التابع DisplayInfo. تجدر الإشارة إلى أنّه عند وصول تنفيذ البرنامج إلى السطر 37 وإلى الاستدعاء employee1.DisplayInfo() تحديدًا سيؤدّي ذلك إلى انتقال التنفيذ إلى السطر 14 ضمن هذا التابع لتنفيذ التعليمات البرمجيّة ضمنه ومن ثمّ الحصول على التمثيل النصيّ للكائن employee1 وإرجاعه إلى السطر

37 مرّة أخرى ليعمل البرنامج على تمرير هذه القيمة النصيّة للتابع `WriteLine` ومن ثمّ العرض على الشاشة، وبالطبع يتكرّر نفس الأمر تمامًا بالنسبة للكائن ضمن `employee2` في السطر 38.

إذا كنت تستخدم Visual Studio 2015 بأيّ إصدار فأنصحك أن تنفّذ هذا البرنامج بشكل خُطَوي لكي تتعرّف على آلية عمل هذا البرنامج بشمل عمليّ. اضغط على المفتاح **F11** (أو من القائمة **Debug > Step Into**) لتنفيذ البرنامج باستخدام منقّح الأخطاء debugger. ستلاحظ ظهور مستطيل أصفر يُشير إلى مكان التنفيذ الحالي، وكلما ضغطت المفتاح **F11** سينتقل تنفيذ البرنامج إلى العبارة البرمجيّة التالية خطوة بخطوة.

## البانية constructor ضمن الصنف

البانية constructor هي تابع من نوع خاص يجب أن تكون موجودة ضمن أيّ صنف في سي شارب. في حال تمّ إغفالها سيعمل المترجم على توليد واحدة افتراضيّة من أجلنا.

في الحقيقة وظيفة البانية هي بناء الكائن وحجز مكان مناسب له في الذاكرة، حيث يتم استدعاء البانية عند إنشاء الكائن باستخدام العامل `new`. لا يمكن للبواني إرجاع قيمة مخصّصة كما نفعل مع التوابع الأخرى عادةً، في الحقيقة هي تُرجع كائنًا من الصنف الموجودة ضمنه. ولكن يمكن أن تقبل وسائط نمرّرها إليها. استبدل الصنف `Employee` التالي بذلك الموجود ضمن البرنامج

```
class Employee
    {
        public string FirstName;
        public string LastName;
        public double Salary;

        public Employee()
        {
            Console.WriteLine("Hello, I'm in Employee's constructor!");
        }

        public string DisplayInfo()
        {
            string result = string.Format("{0} {1} - Salary: {2:N0}",
                this.FirstName, this.LastName, this.Salary);

            return result;
        }
    }
```

لقد أضفنا في هذه النسخة البانية `Employee()` للصنف `Employee`.

نفّذ البرنامج لتحصل على الخرج التالي:

```
*** Hello, I'm in Employee's constructor! ***
*** Hello, I'm in Employee's constructor! ***

First Employee: Mohammad Mansoor - Salary: 1,000
Second Employee: Saleh Mahmoud - Salary: 2,500
```

لاحظ أنّ العبارة:

```
*** Hello, I'm in Employee's constructor! ***
```

قد ظهرت مرّتين في الخرج، وذلك بسبب أنّنا أنشأنا كائنين حيث تُنفّذ هذه البانية من أجل كلّ عملية إنشاء. ولكن السؤال المطروح هنا، ماذا سنستفيد من هذه البانية؟

تُستخدم البواني عمومًا عندما نريد تهيئة الكائن ببعض القيم الضرورية لجعل حالته مستقرّة وذلك أثناء إنشائه وقبل محاولة الوصول إليه من أيّ مصدر خارجيّ. انظر الآن إلى الصنف `Employee` المعدّل الذي يحوي بانية تقوم ببعض الأعمال المفيدة:

```
class Employee
    {
        public string FirstName;
        public string LastName;
        public double Salary;

        public Employee(string firstName, string lastName, double salary)
        {
            this.FirstName = firstName;
            this.LastName = lastName;
            this.Salary = salary;
        }

        public string DisplayInfo()
        {
            string result = string.Format("{0} {1} - Salary: {2:N0}",
                this.FirstName, this.LastName, this.Salary);

            return result;
        }

 }
```

تتطلّب البانية هذه المرّة ثلاثة وسائط، تمثّل قيمًا سيتمّ إسنادها إلى الحقول. هذه الوسائط هي: `firstName` و `lastName` و `salary` (لاحظ أنّ اسم كلّ منها يبدأ بحرف طباعي صغير لتمييزها عن حقول الصنف).

إذا استبدلت هذا الصنف الجديد بالصنف القديم الموجود ضمن البرنامج وحاولت تنفيذ البرنامج فستحصل على خطأ. السبب في ذلك بسيط، وهو أنّ العبارتين في السطرين 27 و 32 من البرنامج تحاولان إنشاء كائنين من الصنف `Employee` عن طريق بانية لا تتطلّب أيّة وسائط وهذا ما لا يتوفّر في الصنف `Employee` الجديد. فعندما يلاحظ مترجم سي شارب وجود بانية واحدة على الأقل بصرف النظر عن عدد الوسائط التي تتطلّبها فإنّه يمتنع عن توليد بانية افتراضية بشكل تلقائي مثلما كان يفعل من قبل. يوجد حلّ سريع لهذه المشكلة يتمثّل في توفير بانية لا تحتاج لأيّة وسائط كما كان الوضع السابق. انظر إلى النسخة الأخيرة للصنف `Employee`:

```
class Employee
    {
        public string FirstName;
        public string LastName;
        public double Salary;

        public Employee(string firstName, string lastName, double salary)
        {
            this.FirstName = firstName;
            this.LastName = lastName;
            this.Salary = salary;
        }

        public Employee()
        {

        }
        public string DisplayInfo()
        {
            string result = string.Format("{0} {1} - Salary: {2:N0}",
                this.FirstName, this.LastName, this.Salary);

            return result;
        }

 }
```

بعد اعتماد هذا الصنف ضمن البرنامج ، سيعمل البرنامج الآن بشكل طبيعي ويظهر الخرج كما هو متوقّع. ولكن تأمّل معي هذا الصنف قليلًا، ألا تلاحظ وجود بانيتين له؟ هذا أمر طبيعي ووارد جدًّا في سي شارب حيث يمكن كتابة أكثر من تابع بنفس الاسم طالما اختلف عدد أو أنواع الوسائط الممرّرة لكلّ منهما. نسمي هذه الميزة بزيادة التحميل overloading للتوابع. فعند وجود استدعاء للتابع المزاد تحميله يتمّ اختيار الشكل المناسب بناءً على عدد وأنواع الوسائط الممرّرة.

لاحظ أنّ البانية عديمة الوسائط فارغة ولا بأس في ذلك. ولكنّ السؤال هنا كيف يمكن الاستفادة من البانية ذات الوسائط الثلاثة. الأمر بسيط، استبدل محتويات التابع `Main` في البرنامج بالشيفرة البسيطة المكافئة التالية:

```
Employee employee1, employee2;

employee1 = new Employee("Mohammad", "Mansoor", 1000);
employee2 = new Employee("Saleh", "Mahmoud", 2500);

Console.WriteLine("First Employee: {0}", employee1.DisplayInfo());
Console.WriteLine("Second Employee: {0}", employee2.DisplayInfo());
```

انظر كم أصبحت الشيفرة نظيفة وقصيرة ومريحة للعين.

إليك الآن البرنامج  كاملًا بعد التعديل:

```
using System;

namespace Lesson06_02
{

    class Employee
    {
        public string FirstName;
        public string LastName;
        public double Salary;

    public Employee(string firstName, string lastName, double salary)
        {
            this.FirstName = firstName;
            this.LastName = lastName;
            this.Salary = salary;
        }

        public Employee()
        {

        }

        public string DisplayInfo()
        {
            string result = string.Format("{0} {1} - Salary: {2:N0}",
                this.FirstName, this.LastName, this.Salary);

            return result;
        }


    }


    class Program
    {
        static void Main(string[] args)
        {
            Employee employee1, employee2;

             employee1 = new Employee("Mohammad", "Mansoor", 1000);
             employee2 = new Employee("Saleh", "Mahmoud", 2500);

             Console.WriteLine("First Employee: {0}", employee1.DisplayInfo());
             Console.WriteLine("Second Employee: {0}", employee2.DisplayInfo());
         }
     }
}
```