الجامعة الافتراضية السورية
SYRIAN VIRTUAL UNIVERSITY

التصميم والبرمجة غرضية التوجه

الدكتور سامي خيمي

ISSN: 2617-989X

Books & Refrences

التصميم والبرمجة غرضية التوجه

الدكتور سامي خيمي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Object Oriented Programming

Sami Khiami

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

ISSN: 2617-989X

الفهرس

الفصل الأول
الصفوف والأغراض

عنوان الموضوع:

الصفوف والأغراض باستخدام Visual Studio 2013

الكلمات المفتاحية:

التصريح عن صف، الحقول Fields، الخصائص Properties، متغيرات المنتسخات Instance variables. نمط القيمة Value Type ونمط المرجع Reference Type. باني الصف Constructor.

ملخص:

نُبين في هذا الفصل أساسيات البرمجة غرضية التوجه. حيث نعرض كيفية التصريح عن الصف وتعريف أعضاء الصف: الحقول والخصائص والطرق. كما نُبين استخدام الصف كنمط مرجع جديد يسمح بإنشاء أغراض من هذا الصف.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- التصريح عن الصفوف.
- إنشاء الأغراض من الصفوف.
- الحقول.
- الخصائص.
- الخصائص التلقائية.
- الطرق.
- نمط القيمة ونمط المرجع.
- باني الصف.

المخطط:

الصفوف والأغراض باستخدام Visual Studio 2013

- 4 وحدات (Learning Objects)

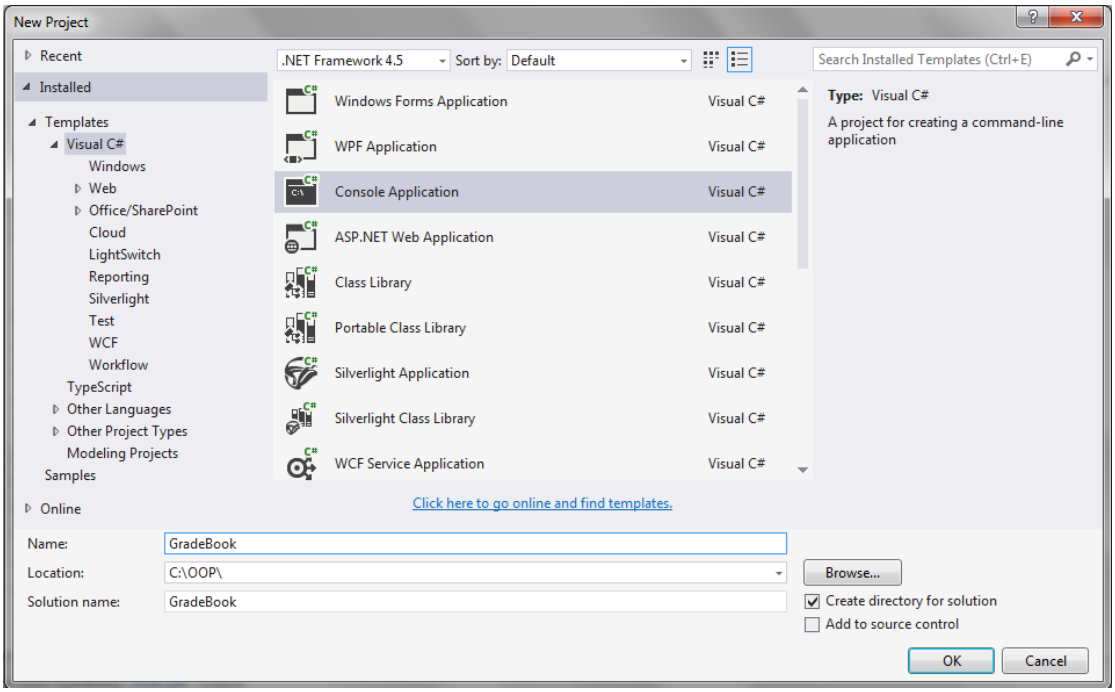
1- إنشاء صف يحوي طريقة وإنشاء غرض من الصف

- يسمح محيط العمل Visual Studio 2013 بإنشاء التطبيقات بشكل بسيط وسريع:

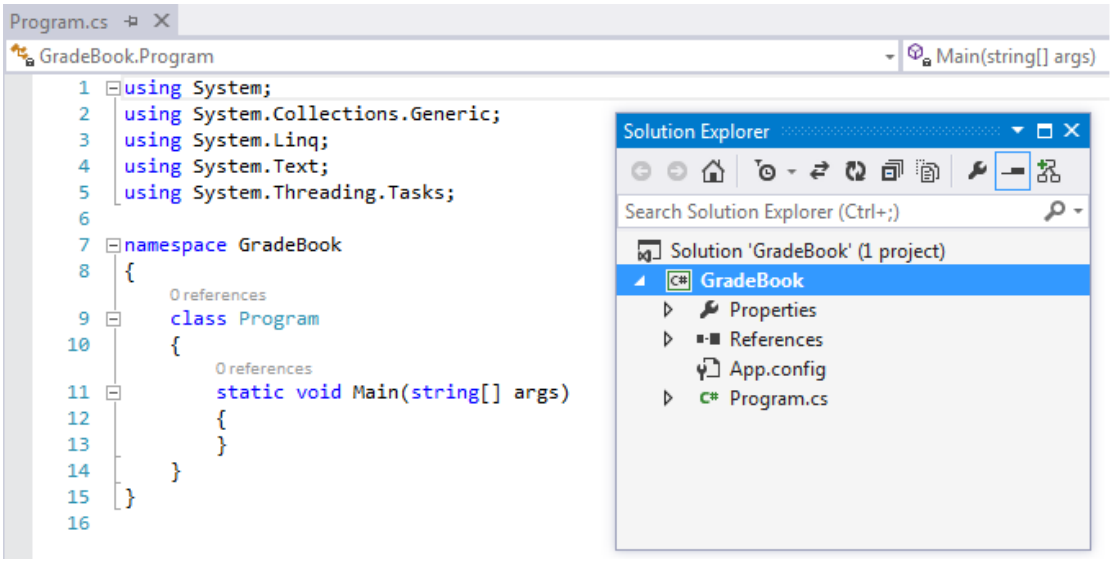
1. افتح محيط العمل Visual Studio 2013.
2. أنشئ مشروع جديد:

File → New → Project

3. اختر Visual C# / Console Application، وقم بإدخال اسم المشروع GradeBook ومسار التخزين:

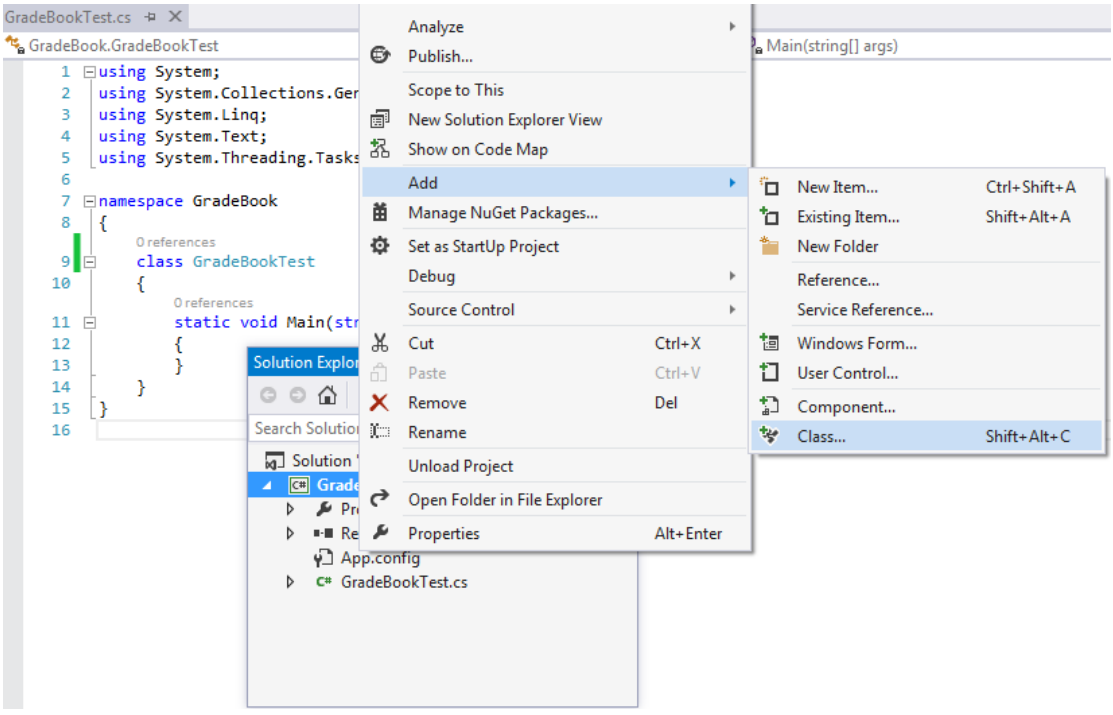


4. يتم فتح مشروع جديد يحوي الملف Program.cs والذي بداخلة الطريقة (Main) والتي هي نقطة البدء بالتنفيذ:



5. قم بتغيير اسم الملف Program.cs إلى GradeBookTest.cs. لاحظ أن اسم الصف سيُصبح تلقائياً أيضاً `GradeBookTest`.

6. انقر بالزر الأيمن على أيقونة المشروع GradeBook ثم اختر إضافة Add صف Class:



7. قم بتسمية الصف الجديد `GradeBook`.

8. قم بكتابة الطريقة البسيطة `DisplayMessage()` في الصف والتي تُظهر رسالة ترحيبية:

```
// GradeBook.cs
// Class declaration with one method.
using System;
```

```
public class GradeBook
{
// display a welcome message to the GradeBook user
public void DisplayMessage()
{
 Console.WriteLine( "Welcome to the Grade Book!" );
} // end method DisplayMessage
} // end class GradeBook
```

9. افتح الملف GradeBookTest.cs لإنشاء غرض من الصف واستدعاء طريقة الصف:

```
//  GradeBookTest.cs
// Create a GradeBook object and call its DisplayMessage method.
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
 GradeBook myGradeBook;
 myGradeBook = new GradeBook();
// call myGradeBook's DisplayMessage method
 myGradeBook.DisplayMessage();
} // end Main
} // end class GradeBookTest
```

10. قم بالتنفيذ:

```
Welcome to the Grade Book!
Press any key to continue . . .
```

11. لاحظ أنك بعد إضافتك للصف الجديد GradeBook أصبح لديك نمط بيانات جديد يُمكنك تعريف متغيرات منه وإنشاء أغراض جديدة.

12. يكون المتغير myGradeBook مؤشر (مرجع) على الغرض.

13. يتم استخدام المعامل new لإنشاء غرض object (مُنتسخ instance) جديد من الصف GradeBook.

- لاحظ الأقواس بعد اسم الصف في المعامل new للدلالة على استدعاء باني الصف الافتراضي.
- لاحظ المعامل (.) بعد اسم المتغير لاستدعاء طريقة الصف.

التصريح عن طريقة لها معاملات

- نقوم فيما يلي بإضافة معامل الدخل courseName (اسم المادة) لطريقة الصف السابق ليُصبح الصف:

```
// GradeBook.cs
// Class declaration with a method that has a parameter.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public void DisplayMessage( string courseName )
{
 Console.WriteLine( "Welcome to the grade book for\n{0}!", courseName );
} // end method DisplayMessage
} // end class GradeBook
```

- ثم نقوم بإنشاء غرض من الصف السابق واستدعاء الطريقة DisplayMessage مع تمرير قيمة لمعامل الدخل. يتم الطلب من المستخدم بإدخال سلسلة نصية (اسم المادة) ومن ثم تمرير القيمة كمعامل دخل للطريقة:

```
// GradeBookTest.cs
// Create a GradeBook object and pass a string to
// its DisplayMessage method.
using System;
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
 GradeBook myGradeBook = new GradeBook();
// prompt for and input course name
Console.WriteLine( "Please enter the course name:" );
string nameOfCourse = Console.ReadLine(); // read a line of text
Console.WriteLine(); // output a blank line
// call myGradeBook's DisplayMessage method
// and pass nameOfCourse as an argument
myGradeBook.DisplayMessage(nameOfCourse);
} // end Main
} // end class GradeBookTest
```

- يكون التنفيذ مثلاً:

```
Please enter the course name:
OOP

Welcome to the grade book for
OOP!
Press any key to continue . . .
```

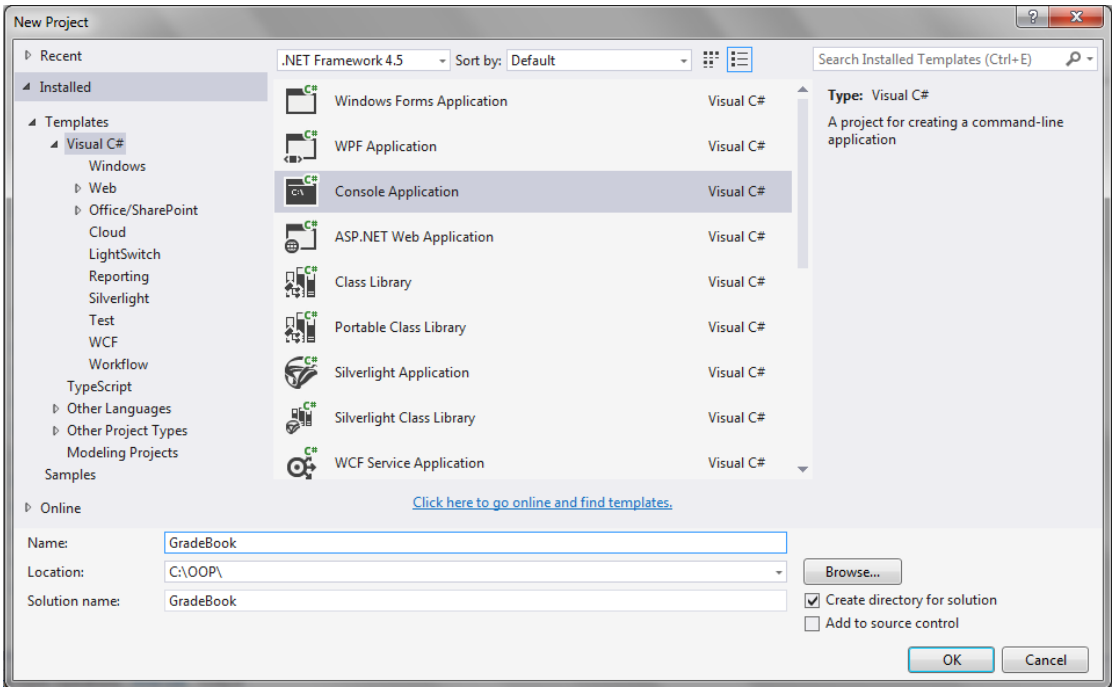
إنشاء صف يحوي طريقة وإنشاء غرض من الصف

- يسمح محيط العمل Visual Studio 2013 بإنشاء التطبيقات بشكل بسيط وسريع:

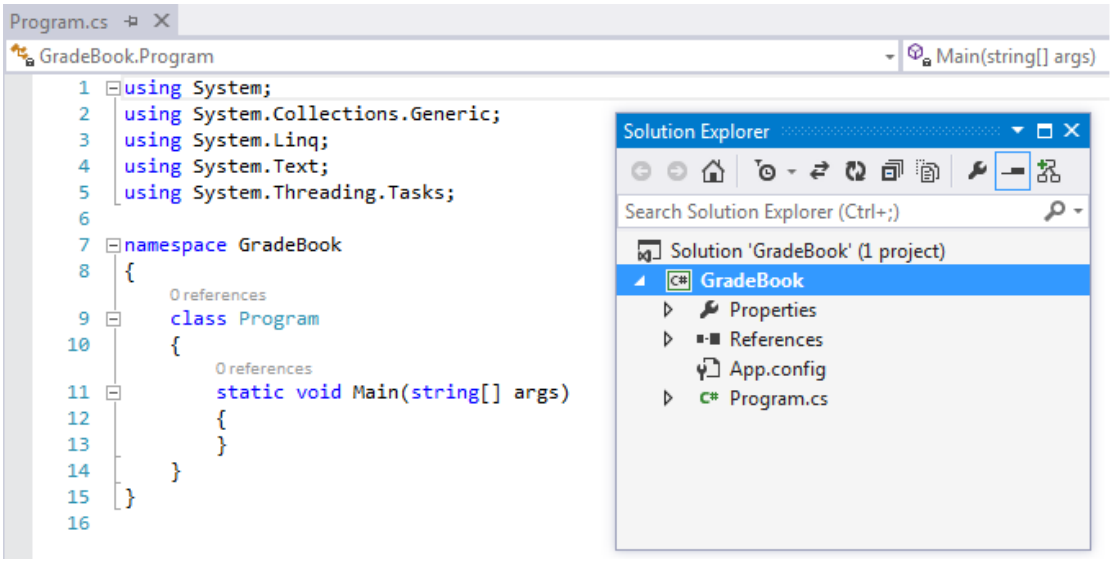
1. افتح محيط العمل Visual Studio 2013.
2. أنشئ مشروع جديد:

File → New → Project

3. اختر Visual C# / Console Application، وقم بإدخال اسم المشروع GradeBook ومسار التخزين:

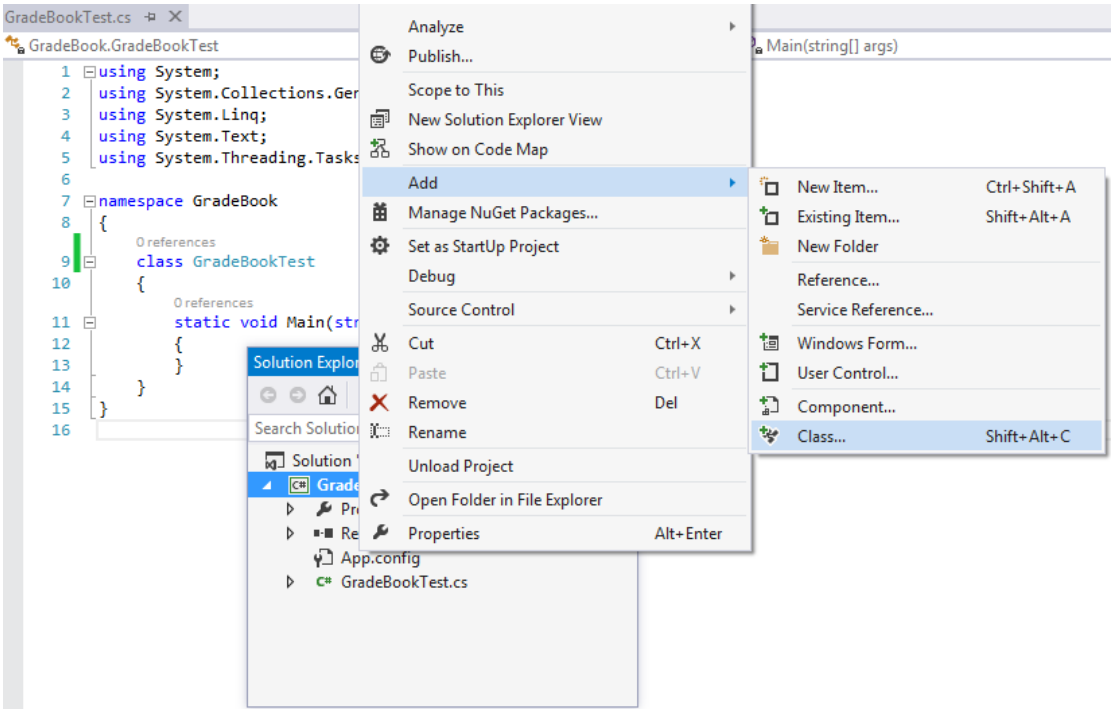


4. يتم فتح مشروع جديد يحوي الملف Program.cs والذي بداخلة الطريقة (Main) والتي هي نقطة البدء بالتنفيذ:



5. قم بتغيير اسم الملف Program.cs إلى GradeBookTest.cs. لاحظ أن اسم الصف سيُصبح تلقائياً أيضاً `GradeBookTest`.

6. انقر بالزر الأيمن على أيقونة المشروع GradeBook ثم اختر إضافة Add صف Class:



7. قم بتسمية الصف الجديد `GradeBook`.

8. قم بكتابة الطريقة البسيطة `DisplayMessage()` في الصف والتي تُظهر رسالة ترحيبية:

```
// GradeBook.cs
// Class declaration with one method.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public void DisplayMessage()
{
 Console.WriteLine( "Welcome to the Grade Book!" );
} // end method DisplayMessage
} // end class GradeBook
```

9. افتح الملف GradeBookTest.cs لإنشاء غرض من الصف واستدعاء طريقة الصف:

```
//  GradeBookTest.cs
// Create a GradeBook object and call its DisplayMessage method.
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
 GradeBook myGradeBook;
 myGradeBook = new GradeBook();
// call myGradeBook's DisplayMessage method
 myGradeBook.DisplayMessage();
} // end Main
} // end class GradeBookTest
```

10. قم بالتنفيذ:

```
Welcome to the Grade Book!
Press any key to continue . . .
```

11. لاحظ أنك بعد إضافتك للصف الجديد GradeBook أصبح لديك نمط بيانات جديد يُمكنك تعريف متغيرات منه وإنشاء أغراض جديدة.
12. يكون المتغير myGradeBook (كما سنرى لاحقاً) مؤشر (مرجع) على الغرض.
13. يتم استخدام المعامل new لإنشاء غرض object (منتسخ instance) جديد من الصف GradeBook.

- لاحظ الأقواس بعد اسم الصف في المعامل new للدلالة على استدعاء باني الصف الافتراضي (نستعرض مفهوم الباني لاحقاً).
- لاحظ المعامل (.) بعد اسم المتغير لاستدعاء طريقة الصف.

التصريح عن طريقة لها معاملات

- نقوم فيما يلي بإضافة معامل الدخل courseName (اسم المادة) لطريقة الصف السابق ليُصبح الصف:

```
// GradeBook.cs
// Class declaration with a method that has a parameter.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public void DisplayMessage( string courseName )
{
 Console.WriteLine( "Welcome to the grade book for\n{0}!", courseName );
} // end method DisplayMessage
} // end class GradeBook
```

- ثم نقوم بإنشاء غرض من الصف السابق واستدعاء الطريقة DisplayMessage مع تمرير قيمة لمعامل الدخل. يتم الطلب من المستخدم بإدخال سلسلة نصية (اسم المادة) ومن ثم تمرير القيمة كمعامل دخل للطريقة:

```
// GradeBookTest.cs
// Create a GradeBook object and pass a string to
// its DisplayMessage method.
using System;
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
 GradeBook myGradeBook = new GradeBook();
// prompt for and input course name
Console.WriteLine( "Please enter the course name:" );
string nameOfCourse = Console.ReadLine(); // read a line of text
Console.WriteLine(); // output a blank line
// call myGradeBook's DisplayMessage method
// and pass nameOfCourse as an argument
myGradeBook.DisplayMessage(nameOfCourse);
} // end Main
} // end class GradeBookTest
```

- يكون التنفيذ مثلاً:

```
Please enter the course name:
OOP

Welcome to the grade book for
OOP!
Press any key to continue . . .
```

2- الحقول والخصائص

الحقول Fields، الخصائص Properties، متغيرات المنتسخات Instance variables

- يُمكن تعريف متغيرات في الصف. تُدعى هذه المتغيرات بحقول الصف. عندما يتم إنشاء منتسخات من الصف، سيكون لكل مُنتسخ نسخه الخاصة من هذه المتغيرات.
- تُعرّف الحقول عادةً على أنها خاصة private. بمعنى أنه لا يُمكن الوصول إليها والتعامل معها من خارج الصف.
- نقوم عادةً بتعريف خاصية Property لكل حقل. يتمّ من خلال الخاصية تعامل الأغراض المُنشأة من الصف مع الحقول. يكون للخاصية مُحدّد الوصول عام public وبالتالي يُمكن للأغراض الوصول لهذه الخصائص.
- يكون لكل خاصية عادةً الموصلين (Accessors) get و set.
- يتم استدعاء الموصل get عند طلب الوصول لقيمة الحقل الموافق للخاصية.
- يتم استدعاء الموصل set عند طلب كتابة قيمة value في الحقل الموافق للخاصية.
- نقوم في الصف التالي بتعريف الحقل (اسم المادة) courseName الخاص private ، والخاصية الموافقة (اسم المادة) CourseName العامة public. يقوم الموصل get بإرجاع قيمة الحقل، ويقوم الموصل set بإسناد القيمة المُمرره value إلى الحقل.

لاحظ أننا قمنا بتعديل الطريقة DisplayMessage لتقوم بإظهار قيمة الخاصية CourseName في الرسالة الترحيبية.

```
//  GradeBook.cs
// GradeBook class that contains a private instance variable,courseName,
// and a public property to get and set its value.
using System;
public class GradeBook
{
private string courseName; // course name for this GradeBook
// property to get and set the course name
public string CourseName
{
get
{
 return courseName;
} // end get
set
{
 courseName = value;
} // end set
} // end property CourseName
// display a welcome message to the GradeBook user
```

```
public void DisplayMessage()
{
// use property CourseName to get the
// name of the course that this GradeBook represents
Console.WriteLine( "Welcome to the grade book for\n{0}!",
           CourseName ); // display property CourseName
} // end method DisplayMessage
} // end class GradeBook
```

- نستخدم فيما يلي الصف السابق حيث نقوم بطلب قيمة من المستخدم. نقوم بإسناد هذه القيمة إلى الخاصية العامة CourseName ومن ثم استدعاء الطريقة DisplayMessage().

```
//  GradeBookTest.cs
// Create and manipulate a GradeBook object.
using System;
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
GradeBook myGradeBook = new GradeBook();
// display initial value of CourseName
Console.WriteLine( "Initial course name is: '{0}'\n", myGradeBook.CourseName );
// prompt for and read course name
Console.WriteLine( "Please enter the course name:" );
myGradeBook.CourseName = Console.ReadLine(); // set CourseName
Console.WriteLine(); // output a blank line
// display welcome message after specifying course name
 myGradeBook.DisplayMessage();
} // end Main
} // end class GradeBookTest
```

- يُعطي التنفيذ:

```
Initial course name is: "

Please enter the course name:
OOP

Welcome to the grade book for
OOP!
Press any key to continue . . .
```

- لاحظ أنه وبخلاف المتغيرات البسيطة، تقوم C# بإعطاء قيم ابتدائية لحقول الصف عند إنشاء كائن من الصف. نقوم بطباعة قيمة الحقل CourseName والذي لم نقم بإعطاء قيمة

له بعد. تكون القيمة الابتدائية لحقل من النوع string هي السلسلة الفارغة ولذا تُظهر الطباعة ". (تضع C# القيمة 0 للحقول الرقمية).

الخصائص التلقائية Auto-Implemented Properties

- لاحظ أننا في المثال السابق، استخدمنا الموصل get لاسترجاع قيمة الحقل، والموصل set لإسناد قيمة للحقل.
- يُمكن في مثل هذه الحالة استخدام الخصائص التلقائية بحيث نكتب مباشرةً:

```
public string CourseName { get; set; }
```

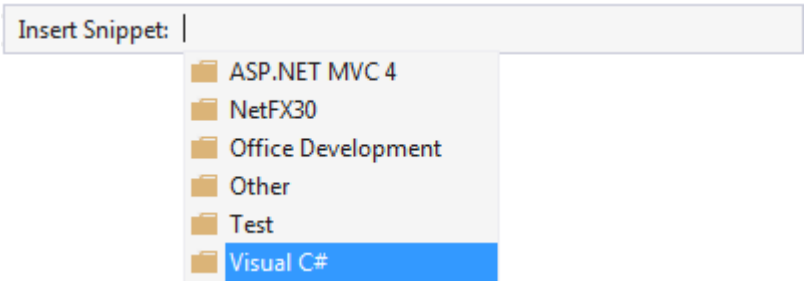
- سيقوم المترجم بعدها بإنشاء حقل private موافق تلقائياً وإنشاء الموصلات get وset اللازمة.
- يصبح الصف GradeBook في حالتنا:

```
//  GradeBook.cs
// GradeBook class that contains a private instance variable, courseName,
// and a public property to get and set its value.
using System;
public class GradeBook
{
   // display a welcome message to the GradeBook user
   public string CourseName { get; set; }
   public void DisplayMessage()
   {
      // use property CourseName to get the
      // name of the course that this GradeBook represents
      Console.WriteLine( "Welcome to the grade book for\n{0}!",
         CourseName ); // display property CourseName
   } // end method DisplayMessage
} // end class GradeBook
```

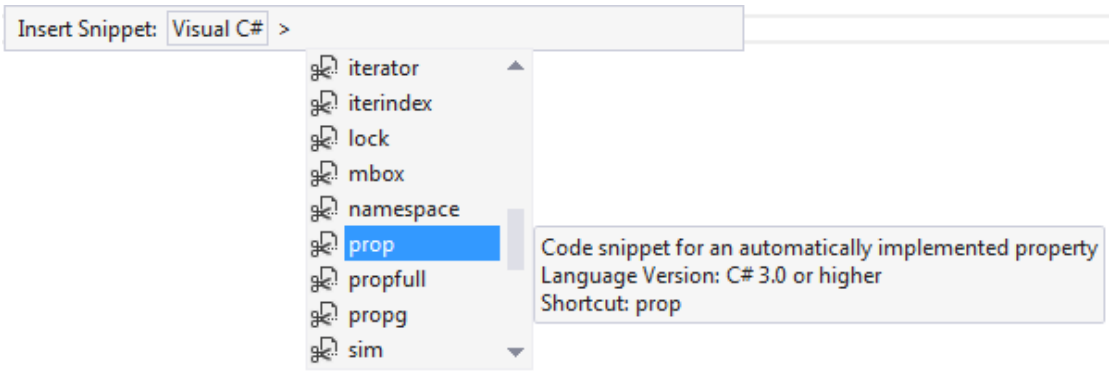
Code Snippets for Auto-Implemented استخدام قصاصات الكود الجاهزة Properties

- يُمكن استخدام قصاصات الكود الجاهزة لإنشاء خاصية تلقائية:

1. اضغط المفتاحين Ctrl+k ثم Ctrl+x:



2. اختر prop:



3. سيتم إضافة السطر التالي:

```
public int MyProperty { get; set; }
```

4. قم بالتعديلات المطلوبة.

3- أنماط القيم وأنماط المرجع Value Types vs. Reference Types

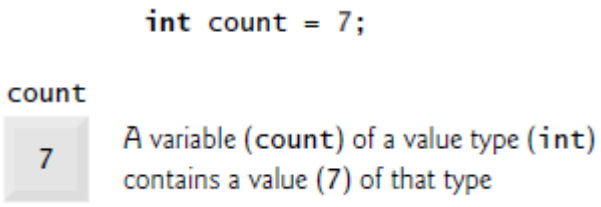
تنقسم الأنماط في C# إلى نوعين: أنماط القيمة و أنماط المرجع.

أنماط القيمة Value Types

- تكون جميع الأنماط البسيطة في C# أنماط قيمة مثل `int` و`double`.
- يحوي المتغير من هذا النمط قيمة معينة من النمط الموافق. فمثلاً حين نكتب:

```
int count = 7;
```

فإن المتغير count والذي هو من النوع البسيط `int` يحوي القيمة 7:

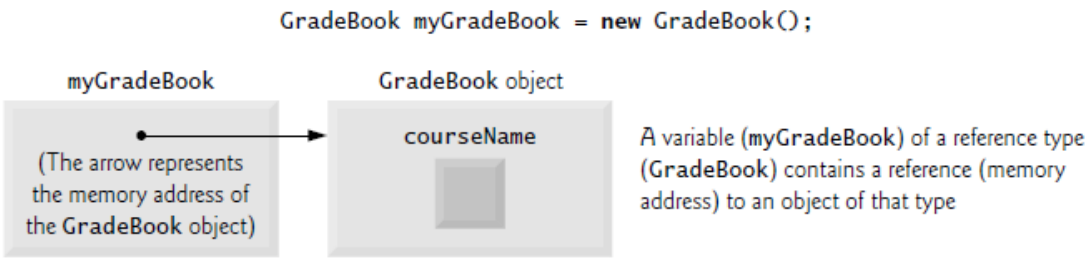


أنماط المرجع Reference Types

- يحوي متغير المرجع عنوان لمكان في الذاكرة يتمّ فيه تخزين البيانات التي يؤشر عليها المتغير.
- فمثلاً عندما كتبنا في المثال السابق:

```
GradeBook myGradeBook = new GradeBook();
```

سيؤدي ذلك إلى إنشاء غرض من الصف `GradeBook` وتخزينه في الذاكرة. سيحوي المتغير myGradeBook والذي هو من النمط `GradeBook` عنوان الكائن في الذاكرة:



تهيئة الأغراض باستخدام باني الصف Initializing Objects with Constructors

- قمنا في الأمثلة السابقة باستخدام الباني الافتراضي الذي تُنشئه C# تلقائياً واستخدامه مع المعامل new. لايكون لهذا الباني متغيرات دخل.
- يُمكننا إضافة باني إلى الصف مع متغيرات دخل لإعطاء قيم ابتدائية لحقول الصف.
- نقوم في المثال التالي بإضافة باني للصف له معامل دخل يقوم الباني بتمريره إلى الخاصية GradeBook:

```
//  GradeBook.cs
// GradeBook class with a constructor to initialize the course name.
using System;
public class GradeBook
{
// auto-implemented property CourseName implicitly creates an
// instance variable for this GradeBook's course name
public string CourseName { get; set; }
// constructor initializes auto-implemented property
// CourseName with string supplied as argument
public GradeBook( string name )
{
 CourseName = name; // set CourseName to name
} // end constructor
// display a welcome message to the GradeBook user
public void DisplayMessage()
{
// use auto-implemented property CourseName to get the
// name of the course that this GradeBook represents
Console.WriteLine( "Welcome to the grade book for\n{0}!", CourseName );
} // end method DisplayMessage
} // end class GradeBook
```

- نقوم فيما يلي باستخدام الباني الجديد السابق لإنشاء غرضين من الصف: gradeBook1 و gradeBook2.

```
//  GradeBookTest.cs
// GradeBook constructor used to specify the course name at the
// time each GradeBook object is created.
using System;
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create GradeBook object
GradeBook gradeBook1 = new GradeBook( // invokes constructor
   "CS101 Introduction to C# Programming" );
GradeBook gradeBook2 = new GradeBook( // invokes constructor
   "CS102 Data Structures in C#" );
// display initial value of courseName for each GradeBook
Console.WriteLine( "gradeBook1 course name is: {0}",
gradeBook1.CourseName );
Console.WriteLine( "gradeBook2 course name is: {0}",
gradeBook2.CourseName );
} // end Main
} // end class GradeBookTest
```

- يكون ناتج التنفيذ:

```
gradeBook1 course name is: CS101 Introduction to C# Programming
gradeBook2 course name is: CS102 Data Structures in C#
Press any key to continue . . .
```

4- مثال تعليمي

- نقوم في المثال التعليمي التالي بالتصريح عن الصف `Account` لإدارة حساب بنكي. يحوي هذا الصف حقل واحد لتخزين رصيد الحساب `balance`.

```
// Account.cs
// Account class with a constructor to
// initialize instance variable balance.
public class Account
{
private decimal balance; // instance variable that stores the balance
// constructor
public Account( decimal initialBalance )
{
 Balance = initialBalance; // set balance using property
} // end Account constructor
// credit (add) an amount to the account
public void Credit( decimal amount )
{
 Balance = Balance + amount; // add amount to balance
} // end method Credit
// a property to get and set the account balance
public decimal Balance
{
get
{
  return balance;
} // end get
set
{
 // validate that value is greater than or equal to 0;
 // if it is not, balance is left unchanged
 if ( value >= 0 )
    balance = value;
} // end set
} // end property Balance
} // end class Account
```

- نقوم في الموصل `get` في الخاصية `Balance` بإرجاع قيمة الرصيد.

- نقوم في الموصل set في الخاصية Balance باختبار فيما إذا كانت القيمة المُمرره value موجبة أم سالبة. في حال كون القيمة سالبة نضع 0 في الحقل عوضاً عنها.
- لاحظ أننا كتبنا باني للصف لإسناد قيمة ابتدائية للرصيد. لاحظ أننا استخدمنا في الباني الخاصية Balance في عملية الإسناد وذلك للتحقق من أن القيمة المسندة موجبة. أي قيمة ممرره سالبة للرصيد سوف تُعتبر 0.
- تقوم الطريقة (Credit) بإضافة قيمة إلى الرصيد (موجبة أو سالبة) ولا تُعيد شيئاً void. لاحظ أننا استخدمنا أيضاً الخاصية Balance في الطريقة سواءً لقراءة قيمة الرصيد أو لإسناد القيمة الناتجة عن جمع الرصيد مع القيمة المُمررة. أيضاً هنا لا يُمكن أن نُسند نتيجة سالبة إلى الرصيد طالما أننا نستخدم الخاصية Balance.
- نقوم فيما يلي باختبار الصف السابق:

```
// AccountTest.cs
// Create and manipulate Account objects.
using System;
public class AccountTest
{
// Main method begins execution of C# application
public static void Main( string[] args )
{
Account account1 = new Account( 50.00M ); // create Account object
Account account2 = new Account( -7.53M ); // create Account object
// display initial balance of each object using a property
Console.WriteLine( "account1 balance: {0:C}",account1.Balance );
// display Balance property
Console.WriteLine( "account2 balance: {0:C}\n",account2.Balance );
// display Balance property
decimal depositAmount; // deposit amount read from user
// prompt and obtain user input
Console.Write( "Enter deposit amount for account1: " );
 depositAmount = Convert.ToDecimal( Console.ReadLine() );
Console.WriteLine( "adding {0:C} to account1 balance\n", depositAmount );
account1.Credit( depositAmount ); // add to account1 balance
// display balances
Console.WriteLine( "account1 balance: {0:C}", account1.Balance );
Console.WriteLine( "account2 balance: {0:C}\n", account2.Balance );
// prompt and obtain user input
Console.Write( "Enter deposit amount for account2: " );
depositAmount = Convert.ToDecimal( Console.ReadLine() );
Console.WriteLine( "adding {0:C} to account2 balance\n", depositAmount );
account2.Credit( depositAmount ); // add to account2 balance
// display balances
Console.WriteLine( "account1 balance: {0:C}", account1.Balance );
Console.WriteLine( "account2 balance: {0:C}", account2.Balance );
} // end Main
} // end class AccountTest
```

- يكون التنفيذ مثلاً:

```
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 49.99
adding $49.99 to account1 balance

account1 balance: $99.99
account2 balance: $0.00

Enter deposit amount for account2: 123.21
adding $123.21 to account2 balance

account1 balance: $99.99
account2 balance: $123.21
Press any key to continue . . .
```

- نقوم في المثال التعليمي التالي بالتصريح عن الصف Account لإدارة حساب بنكي. يحوي هذا الصف حقل واحد لتخزين رصيد الحساب balance.
- تمّ اختيار نمط الرصيد decimal لأن هذا النمط يقوم بتخزين الأرقام العشرية بدقة محدّدة وبدون أي تقريب. بينما تقوم بقية الأنماط float و double والتي تعتمد الفاصلة العائمة بتقريب الأرقام بعد الفاصلة حسب الحاجة.
- عند استخدام قيمة من النمط decimal في الكود يجب وضع M أو m بعدها (اختصار كلمة العملة Money). مثال:

```
decimal x;
 x = 44.44M;
```

- في حال عدم وضع اللاحقة M، سيقوم المترجم بإعطاء خطأ لأنه يعتبر القيمة من النوع double.

```
// Account.cs
// Account class with a constructor to
// initialize instance variable balance.
public class Account
{
private decimal balance; // instance variable that stores the balance
// constructor
public Account( decimal initialBalance )
{
 Balance = initialBalance; // set balance using property
} // end Account constructor
// credit (add) an amount to the account
public void Credit( decimal amount )
{
 Balance = Balance + amount; // add amount to balance
} // end method Credit
// a property to get and set the account balance
public decimal Balance
{
get
{
  return balance;
} // end get
set
{
 // validate that value is greater than or equal to 0;
 // if it is not, balance is left unchanged
 if ( value >= 0 )
    balance = value;
} // end set
} // end property Balance
} // end class Account
```

- نقوم في الموصل get في الخاصية Balance بإرجاع قيمة الرصيد.
- نقوم في الموصل set في الخاصية Balance باختبار فيما إذا كانت القيمة المُمرره value موجبة أم سالبة. في حال كون القيمة سالبة نضع 0 في الحقل عوضاً عنها.
- لاحظ أننا كتبنا باني للصف لإسناد قيمة ابتدائية للرصيد. لاحظ أننا استخدمنا في الباني الخاصية Balance في عملية الإسناد وذلك للتحقق من أن القيمة المسندة موجبة. أي قيمة مُمرره سالبة للرصيد سوف تُعتبر 0.
- تقوم الطريقة (Credit) بإضافة قيمة إلى الرصيد (موجبة أو سالبة) ولا تُعيد شيئاً void. لاحظ أننا استخدمنا أيضاً الخاصية Balance في الطريقة سواءً لقراءة قيمة الرصيد أو لإسناد القيمة الناتجة عن جمع الرصيد مع القيمة المُمرره. أيضاً هنا لا يُمكن أن نُسند نتيجة سالبة إلى الرصيد طالما أننا نستخدم الخاصية Balance.
- نقوم فيما يلي باختبار الصف السابق:

```
// AccountTest.cs
// Create and manipulate Account objects.
using System;
public class AccountTest
{
// Main method begins execution of C# application
public static void Main( string[] args )
{
Account account1 = new Account( 50.00M ); // create Account object
Account account2 = new Account( -7.53M ); // create Account object
// display initial balance of each object using a property
Console.WriteLine( "account1 balance: {0:C}",account1.Balance );
// display Balance property
Console.WriteLine( "account2 balance: {0:C}\n",account2.Balance );
// display Balance property
decimal depositAmount; // deposit amount read from user
// prompt and obtain user input
Console.Write( "Enter deposit amount for account1: " );
 depositAmount = Convert.ToDecimal( Console.ReadLine() );
Console.WriteLine( "adding {0:C} to account1 balance\n", depositAmount );
account1.Credit( depositAmount ); // add to account1 balance
// display balances
Console.WriteLine( "account1 balance: {0:C}", account1.Balance );
Console.WriteLine( "account2 balance: {0:C}\n", account2.Balance );
// prompt and obtain user input
Console.Write( "Enter deposit amount for account2: " );
depositAmount = Convert.ToDecimal( Console.ReadLine() );
Console.WriteLine( "adding {0:C} to account2 balance\n", depositAmount );
account2.Credit( depositAmount ); // add to account2 balance
// display balances
Console.WriteLine( "account1 balance: {0:C}", account1.Balance );
Console.WriteLine( "account2 balance: {0:C}", account2.Balance );
} // end Main
} // end class AccountTest
```

- لاحظ أننا نستخدم التنسيق {0:C} وذلك لإظهار الرصيد بتنسيق العملة.

- يكون التنفيذ مثلاً:

```
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 49.99
adding $49.99 to account1 balance

account1 balance: $99.99
account2 balance: $0.00

Enter deposit amount for account2: 123.21
adding $123.21 to account2 balance

account1 balance: $99.99
account2 balance: $123.21
Press any key to continue . . .
```

- يُبين الجدول التالي التنسيقات المختلفة التي يُمكن استخدامها:

التنسيق	الرمز
تنسيق العملة	C,c
تنسيق الأعداد الطبيعية	D,d
تنسيق وضع فاصلة الآلاف مع خانتين بعد الفاصلة العشرية	N,n
التنسيق العلمي مع ست خانات بعد الفاصلة العشرية	E,e
تنسيق خانات محدّدة بعد الفاصلة العشرية(خانتين افتراضياً)	F,f
التنسيق العام مع الخانات العشرية	G,g
تنسيق السداسي عشر	X,x

5- اقتراحات وتمارين

تمرين 1: صف الدائرة Circle

- قم بالتصريح عن صف الدائرة Circle. يحوي هذا الصف الحقل الخاص radius لتخزين نصف القطر. كما يحوي الخاصية العامة Radius للتعامل مع الحقل radius.
- يجب التحقق من أن القيمة المُمررة لنصف القطر موجبة وإلا فيتم إسناد القيمة 0.
- اكتب الطريقة Circumference() لحساب محيط الدائرة، والطريقة Area() لحساب مساحتها.
- قم باختبار الصف السابق بإنشاء عدة كائنات منه واستخدام الخصائص والطرق فيه.

تمرين 2: صف المستطيل Rectangle

- قم بالتصريح عن صف المستطيل Rectangle. يحوي هذا الصفين الحقلين الطول length والعرض width. قم بالتصريح عن الخاصيتين العامتين الطول Length والعرض Width. قم بالتحقق من أن القيم المسندة للحقلين هما قيم موجبة.
- اكتب الطريقة Perimeter() لحساب محيط المستطيل والطريقة Area() لحساب مساحة المستطيل.
- قم باختبار الصف السابق بإنشاء عدة كائنات منه واستخدام الخصائص والطرق فيه.

الفصل الثاني
الصفوف (1)

عنوان الموضوع:

الصفوف (1)

الكلمات المفتاحية:

throw، ToString()، override ، this.

ملخص:

نعرض في هذا الفصل لبعض المواضيع المتقدمة في الصفوف.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- إطلاق الاستثناءات.
- ركوب الطريقة ToString().
- المرجع للصف نفسه this.
- التحميل الزائد لباني الصف.

المخطط:

الصفوف (1)

- 3 وحدات (Learning Objects)

1- مواضيع متقدمة في الصفوف

- ليكن المطلوب كتاب صف يُمثل الوقت. يتألف الوقت من الساعة والدقائق والثواني.
- ليكن الصف Time1 التالي:

```
// Time1.cs
// Time1 class declaration maintains the time in 24-hour format.
using System; // namespace containing ArgumentOutOfRangeException
public class Time1
{
 private int hour; // 0 - 23
 private int minute; // 0 - 59
 private int second; // 0 - 59
// set a new time value using universal time; throw an
// exception if the hour, minute or second is invalid
public void SetTime( int h, int m, int s )
{
// validate hour, minute and second
if ( (h >= 0 && h < 24 ) && (m >= 0 && m < 60 ) && (s >= 0 && s < 60 ) )
{
 hour = h;
 minute = m;
 second = s;
} // end if
else
 throw new ArgumentOutOfRangeException();
} // end method SetTime
// convert to string in universal-time format (HH:MM:SS)
public string ToUniversalString()
{
 return string.Format( "{0:D2}:{1:D2}:{2:D2}", hour, minute, second );
} // end method ToUniversalString
// convert to string in standard-time format (H:MM:SS AM or PM)
public override string ToString()
{
return string.Format( "{0}:{1:D2}:{2:D2} {3}",
 ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
   minute, second, ( hour < 12 ? "AM" : "PM" ) );
} // end method ToString
} // end class Time1
```

- نقوم فيما يلي باستخدام الصف السابق:

```
// Time1Test.cs
// Time1 object used in an application.
using System;
public class Time1Test
{
public static void Main( string[] args )
{
// create and initialize a Time1 object
Time1 time = new Time1(); // invokes Time1 constructor
// output string representations of the time
Console.Write( "The initial universal time is: " );
Console.WriteLine( time.ToUniversalString() );
Console.Write( "The initial standard time is: " );
Console.WriteLine( time.ToString() );
```

```
Console.WriteLine(); // output a blank line
// change time and output updated time
time.SetTime( 13, 27, 6 );
Console.Write( "Universal time after SetTime is: " );
Console.WriteLine( time.ToUniversalString() );
Console.Write( "Standard time after SetTime is: " );
Console.WriteLine( time.ToString() );
Console.WriteLine(); // output a blank line
// attempt to set time with invalid values
try
{
 time.SetTime( 99, 99, 99 );
} // end try
 catch ( ArgumentOutOfRangeException ex )
{
Console.WriteLine( ex.Message + "\n" );
} // end catch
// display time after attempt to set invalid values
Console.WriteLine( "After attempting invalid settings:" );
Console.Write( "Universal time: " );
Console.WriteLine( time.ToUniversalString() );
Console.Write( "Standard time: " );
Console.WriteLine( time.ToString() );
} // end Main
} // end class Time1Test
```

- يكون ناتج التنفيذ:

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM


Universal time after SetTime is: 13:27:06
Standard time after SetTime is: 1:27:06 PM


Specified argument was out of the range of valid values.


After attempting invalid settings:
Universal time: 13:27:06
Standard time: 1:27:06 PM
Press any key to continue . . .
```

الطريقة SetTime

- لهذه الطريقة ثلاثة معاملات تُمثّل الساعة والدقائق والثواني. تختبر هذه الطريقة المجالات الصحيحة للقيم الممرة وفي حال عدم موافقتها تقوم برفع استثناء من النمط معامل خارج المجال ArgumentOutOfRangeException وذلك باستخدام الكلمة المفتاحية throw.

الطريقة ToUniversalString

- تقوم هذه الطريقة بإرجاع سلسلة نصية تُمثل التاريخ بالتنسيق العالمي.

الطريقة ToString

- تقوم هذه الطريقة بإرجاع سلسلة نصية تُمثل التاريخ بالتنسيق القياسي. تقوم هذه الطريقة بركوب override الطريقة (ToString) المُعرّفة على كل الأغراض. (ندرس لاحقاً الكلمة المفتاحية override).
- نقوم فيما يلي باستخدام الصف السابق:

```
// Time1Test.cs
// Time1 object used in an application.
using System;
public class Time1Test
{
public static void Main( string[] args )
{
// create and initialize a Time1 object
Time1 time = new Time1(); // invokes Time1 constructor
// output string representations of the time
Console.Write( "The initial universal time is: " );
Console.WriteLine( time.ToUniversalString() );
Console.Write( "The initial standard time is: " );
Console.WriteLine( time.ToString() );
Console.WriteLine(); // output a blank line
// change time and output updated time
time.SetTime( 13, 27, 6 );
Console.Write( "Universal time after SetTime is: " );
Console.WriteLine( time.ToUniversalString() );
Console.Write( "Standard time after SetTime is: " );
Console.WriteLine( time.ToString() );
Console.WriteLine(); // output a blank line
// attempt to set time with invalid values
try
{
 time.SetTime( 99, 99, 99 );
} // end try
 catch ( ArgumentOutOfRangeException ex )
{
```

```
Console.WriteLine( ex.Message + "\n" );
} // end catch
// display time after attempt to set invalid values
Console.WriteLine( "After attempting invalid settings:" );
Console.Write( "Universal time: " );
Console.WriteLine( time.ToUniversalString() );
Console.Write( "Standard time: " );
Console.WriteLine( time.ToString() );
} // end Main
} // end class Time1Test
```

- يكون ناتج التنفيذ:

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM


Universal time after SetTime is: 13:27:06
Standard time after SetTime is: 1:27:06 PM


Specified argument was out of the range of valid values.


After attempting invalid settings:
Universal time: 13:27:06
Standard time: 1:27:06 PM
Press any key to continue . . .
```

2- الكلمة المفتاحية this

- يُمكن للصف التعامل مع مرجع لنفسه باستخدام الكلمة المفتاحية `this`.
- ليكن الصف `SimpleTime` التالي:

```
//  ThisTest.cs
// this used implicitly and explicitly to refer to members of an object.
using System;
public class ThisTest
{
public static void Main( string[] args )
{
 SimpleTime time = new SimpleTime( 15, 30, 19 );
 Console.WriteLine( time.BuildString() );
} // end Main
} // end class ThisTest
// class SimpleTime demonstrates the "this" reference
public class SimpleTime
{
 private int hour; // 0-23
 private int minute; // 0-59
 private int second; // 0-59
```

```
// if the constructor uses parameter names identical to
// instance variable names the "this" reference is
// required to distinguish between names
public SimpleTime( int hour, int minute, int second )
{
 this.hour = hour; // set "this" object's hour instance variable
 this.minute = minute; // set "this" object's minute
 this.second = second; // set "this" object's second
} // end SimpleTime constructor
// use explicit and implicit "this" to call ToUniversalString
public string BuildString()
{
return string.Format( "{0,24}: {1}\n{2,24}: {3}",
"this.ToUniversalString()", this.ToUniversalString(),
"ToUniversalString()", ToUniversalString() );
} // end method BuildString
// convert to string in universal-time format (HH:MM:SS)
public string ToUniversalString()
{
// "this" is not required here to access instance variables,
// because method does not have local variables with same
// names as instance variables
return string.Format( "{0:D2}:{1:D2}:{2:D2}",
 this.hour, this.minute, this.second );
} // end method ToUniversalString
} // end class SimpleTime
```

- تستخدم الطريقة `SimpleTime` ثلاثة معاملات للساعة `hour` والدقائق `minute` والثواني `second`. لهذه المعاملات نفس أسماء حقول الصف.
- للتمييز في جسم الطريقة بين المعامل والحقل يجب استخدام الكلمة المفتاحية `this` قبل اسم الحقل.
- لا يكون استخدام الكلمة المفتاحية `this` واجباً بالضرورة (الطريقة (`ToUniversalString`)) لأنه لا يوجد أي التباس.
- يكون التنفيذ:

```
this.ToUniversalString(): 15:30:19
    ToUniversalString(): 15:30:19
Press any key to continue . . .
```

الطريقة SimpleTime

- تستخدم الطريقة ثلاثة معاملات للساعة hour والدقائق minute والثواني second. لهذه المعاملات نفس أسماء حقول الصف.
- للتمييز في جسم الطريقة بين المعامل والحقل يجب استخدام الكلمة المفتاحية this قبل اسم الحقل.
- لا يكون استخدام الكلمة المفتاحية this واجباً بالضرورة (الطريقة (ToUniversalString)) لأنه لا يوجد أي التباس.
- يكون التنفيذ:

```
this.ToUniversalString(): 15:30:19
   ToUniversalString(): 15:30:19
Press any key to continue . . .
```

3- التحميل الزائد لباني الصف Overloaded Constructors

- يُمكن كتابة أكثر من باني للصف. يكون لكل باني توقيعه signature المختلف.
- نقوم في المثال التالي بكتابة أكثر من باني واحد للصف.
- يكون للباني الأول ثلاثة معاملات للساعة والدقائق والثواني. في حال عدم تمرير قيمة للمعامل نضع القيمة 0 له.
- ندعو الباني الثاني بباني النسخ Copy Constructor حيث نُمرر له معامل من نفس نمط الصف ليقوم بتهيئة الغرض المنشأ بنفس قيم حقول الغرض المُمرر للباني.

```
// Time2.cs
// Time2 class declaration with overloaded constructors.
using System; // for class ArgumentOutOfRangeException

public class Time2
{
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59
   // constructor can be called with zero, one, two or three arguments
   public Time2( int h = 0, int m = 0, int s = 0 )
   {
      SetTime( h, m, s ); // invoke SetTime to validate time
   } // end Time2 three-argument constructor
   // Time2 constructor: another Time2 object supplied as an argument
   public Time2( Time2 time )
      : this( time.Hour, time.Minute, time.Second ) { }

   // set a new time value using universal time; ensure that
```

```
   // the data remains consistent by setting invalid values to zero
   public void SetTime( int h, int m, int s )
   {
      Hour = h; // set the Hour property
      Minute = m; // set the Minute property
      Second = s; // set the Second property
   } // end method SetTime

   // property that gets and sets the hour
   public int Hour
   {
      get
      {
         return hour;
      } // end get
      set
      {
         if ( value >= 0 && value < 24 )
            hour = value;
         else
            throw new ArgumentOutOfRangeException(
               "Hour", value, "Hour must be 0-23" );
      } // end set
   } // end property Hour
   // property that gets and sets the minute
   public int Minute
   {
      get
      {
         return minute;
      } // end get
      set
      {
         if ( value >= 0 && value < 60 )
            minute = value;
         else
            throw new ArgumentOutOfRangeException(
               "Minute", value, "Minute must be 0-59" );
      } // end set
   } // end property Minute
   // property that gets and sets the second
   public int Second
   {
      get
      {
         return second;
      } // end get
      set
      {
         if ( value >= 0 && value < 60 )
            second = value;
         else
            throw new ArgumentOutOfRangeException(
               "Second", value, "Second must be 0-59" );
      } // end set
   } // end property Second

   // convert to string in universal-time format (HH:MM:SS)
   public string ToUniversalString()
   {
      return string.Format(
         "{0:D2}:{1:D2}:{2:D2}", Hour, Minute, Second );
```

```
   } // end method ToUniversalString

   // convert to string in standard-time format (H:MM:SS AM or PM)
   public override string ToString()
   {
      return string.Format( "{0}:{1:D2}:{2:D2} {3}",
         ( ( Hour == 0 || Hour == 12 ) ? 12 : Hour % 12 ),
         Minute, Second, ( Hour < 12 ? "AM" : "PM" ) );
   } // end method ToString
} // end class Time2
```

- نُبين فيما يلي استخدامات مختلفة لباني الصف السابق:

```
//Time2Test.cs
// Overloaded constructors used to initialize Time2 objects.
using System;
public class Time2Test
{
   public static void Main( string[] args )
   {
      Time2 t1 = new Time2(); // 00:00:00
      Time2 t2 = new Time2( 2 ); // 02:00:00
      Time2 t3 = new Time2( 21, 34 ); // 21:34:00
      Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
      Time2 t5 = new Time2( t4 ); // 12:25:42
      Time2 t6; // initialized later in the program

      Console.WriteLine( "Constructed with:\n" );
      Console.WriteLine( "t1: all arguments defaulted" );
      Console.WriteLine( "   {0}", t1.ToUniversalString() ); // 00:00:00
      Console.WriteLine( "   {0}\n", t1.ToString() ); // 12:00:00 AM

      Console.WriteLine("t2: hour specified; minute and second defaulted" );
      Console.WriteLine( "   {0}", t2.ToUniversalString() ); // 02:00:00
      Console.WriteLine( "   {0}\n", t2.ToString() ); // 2:00:00 AM

      Console.WriteLine("t3: hour and minute specified; second defaulted" );
      Console.WriteLine( "   {0}", t3.ToUniversalString() ); // 21:34:00
      Console.WriteLine( "   {0}\n", t3.ToString() ); // 9:34:00 PM

      Console.WriteLine( "t4: hour, minute and second specified" );
      Console.WriteLine( "   {0}", t4.ToUniversalString() ); // 12:25:42
      Console.WriteLine( "   {0}\n", t4.ToString() ); // 12:25:42 PM

      Console.WriteLine( "t5: Time2 object t4 specified" );
      Console.WriteLine( "   {0}", t5.ToUniversalString() ); // 12:25:42
      Console.WriteLine( "   {0}", t5.ToString() ); // 12:25:42 PM

      // attempt to initialize t6 with invalid values
      try
      {
         t6 = new Time2( 27, 74, 99 ); // invalid values
      } // end try
      catch ( ArgumentOutOfRangeException ex )
      {
         Console.WriteLine( "\nException while initializing t6:" );
         Console.WriteLine( ex.Message );
      } // end catch
```

```
   } // end Main
} // end class Time2Test
```

- يكون ناتج التنفيذ:

```
Constructed with:

t1: all arguments defaulted
   00:00:00
   12:00:00 AM

t2: hour specified; minute and second defaulted
   02:00:00
   2:00:00 AM

t3: hour and minute specified; second defaulted
   21:34:00
   9:34:00 PM

t4: hour, minute and second specified
   12:25:42
   12:25:42 PM

t5: Time2 object t4 specified
   12:25:42
   12:25:42 PM

Exception while initializing t6:
Hour must be 0-23
Parameter name: Hour
Actual value was 27.
Press any key to continue . . .
```

الباني الافتراضي بدون معاملات

- في حال عدم التصريح عن أي باني للصف. يقوم المترجم تلقائياً بإنشاء باني افتراضي للصف بدون معاملات يُمكنك استخدامه مع new لإنشاء أغراض من الصف.
- أما في حال كتابتك لأي باني للصف فلن يقوم المترجم بإنشاء الباني الافتراضي بدون معاملات وعليك كتابته بنفسك إن أردت استخدامه.

4- اقتراحات وتمارين

تمرين 1: صف الدائرة Circle

قم بتعديل صف الدائرة Circle الذي قُمت بإنشائه سابقاً:

- أضف باني جديد له معامل دخل بنفس اسم الخاصية Radius.
- أضف باني نسخ له معامل دخل من النمط Circle.
- أضف الطريقة (ToString) للصف لتقوم بإظهار معلومات الدائرة.
- قم باختبار الصف السابق مع عدة كائنات منه.

تمرين 2: صف المستطيل Rectangle

قم بتعديل صف المستطيل Rectangle الذي قُمت بإنشائه سابقاً:

- أضف باني جديد له معاملي دخل: Length وWidth.
- أضف باني نسخ له معامل دخل من النمط Rectangle.
- أضف الطريقة (ToString) للصف لتقوم بإظهار معلومات المستطيل.
- قم باختبار الصف السابق مع عدة كائنات منه.

الفصل الثالث
الصفوف (2)

عنوان الموضوع:

الصفوف (2).

الكلمات المفتاحية:

التركيب Composition، الأعضاء الساكنة static.

ملخص:

نتابع في هذا الفصل استعراض المواضيع الأساسية في الصفوف. حيث نعرض أولاً لتركيب الصفوف ومن ثم لاستخدام الأعضاء الساكنة.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- تركيب الصفوف Composition.
- الأعضاء الساكنة Static.

المخطط:

الصفوف (2)

- 3 وحدة (Learning Objects)

1- التركيب Composition

- يُمكن أن يكون للصف أعضاء members من صفوف أخرى. يُدعى هذا الاستخدام بالتركيب composition.
- سنقوم فيما يلي بالتصريح عن صف التاريخ Date، ومن ثم استخدام هذا الصف في صف الموظف Employee من أجل تاريخ الميلاد وتاريخ التعيين للموظف.
- نُصرح أولاً عن صف التاريخ Date والذي يحوي ثلاثة خصائص عامة للعام Year والشهر Month واليوم Day.

```
// Date.cs
// Date class declaration.
using System;

public class Date
{
   private int month; // 1-12
   private int day; // 1-31 based on month

   // auto-implemented property Year
   public int Year { get; private set; }

   // constructor: use property Month to confirm proper value for month;
   // use property Day to confirm proper value for day
   public Date( int theMonth, int theDay, int theYear )
   {
      Month = theMonth; // validate month
      Year = theYear; // could validate year
      Day = theDay; // validate day
      Console.WriteLine( "Date object constructor for date {0}", this );
   } // end Date constructor

   // property that gets and sets the month
   public int Month
   {
      get
      {
         return month;
      } // end get
      private set // make writing inaccessible outside the class
      {
         if ( value > 0 && value <= 12 ) // validate month
            month = value;
         else // month is invalid
            throw new ArgumentOutOfRangeException(
               "Month", value, "Month must be 1-12" );
      } // end set
   } // end property Month

   // property that gets and sets the day
   public int Day
   {
      get
      {
         return day;
```

```
      } // end get
      private set // make writing inaccessible outside the class
      {
         int[] daysPerMonth = { 0, 31, 28, 31, 30, 31, 30,
                                31, 31, 30, 31, 30, 31 };

         // check if day in range for month
         if ( value > 0 && value <= daysPerMonth[ Month ] )
            day = value;
         // check for leap year
         else if ( Month == 2 && value == 29 &&
            ( Year % 400 == 0 || ( Year % 4 == 0 && Year % 100 != 0 ) ) )
            day = value;
         else // day is invalid
            throw new ArgumentOutOfRangeException(
               "Day", value, "Day out of range for current month/year" );
      } // end set
   } // end property Day

   // return a string of the form month/day/year
   public override string ToString()
   {
      return string.Format( "{0}/{1}/{2}", Month, Day, Year );
   } // end method ToString
} // end class Date
```

- نقوم في صف الموظف `Employee` التالي باستخدام الصف السابق من أجل تاريخ الميلاد `BirthDate` وتاريخ التعيين `HireDate`:

```
// Employee.cs
// Employee class with references to other objects.
public class Employee
{
   public string FirstName { get; private set; }
   public string LastName { get; private set; }
   public Date BirthDate { get; private set; }
   public Date HireDate { get; private set; }

   // constructor to initialize name, birth date and hire date
   public Employee( string first, string last,
      Date dateOfBirth, Date dateOfHire )
   {
      FirstName = first;
      LastName = last;
      BirthDate = dateOfBirth;
      HireDate = dateOfHire;
   } // end Employee constructor

   // convert Employee to string format
   public override string ToString()
   {
      return string.Format( "{0}, {1}  Hired: {2}  Birthday: {3}",
         LastName, FirstName, HireDate, BirthDate );
   } // end method ToString
} // end class Employee
```

- نقوم فيما يلي باستخدام الصفين السابقين:

```
// EmployeeTest.cs
// Composition demonstration.
using System;

public class EmployeeTest
{
   public static void Main( string[] args )
   {
      Date birth = new Date( 7, 24, 1949 );
      Date hire = new Date( 3, 12, 1988 );
      Employee employee = new Employee( "Bob", "Blue", birth, hire );

      Console.WriteLine( employee );
   } // end Main
} // end class EmployeeTest
```

- يكون ناتج التنفيذ:

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949
Press any key to continue. . .
```

2- الحقول الساكنة static

- يكون لكل غرض نسخه الخاصة من حقول الصف.
- نحتاج في بعض الحالات إلى تشارك جميع أغراض الصف لنفس البيانات.
- يسمح التصريح عن حقل باستخدام الكلمة المفتاحية static بمشاركة هذا الحقل من قبل جميع أغراض الصف.
- يتم الوصول لحقل ساكن عام من خارج الصف باستخدام اسم الصف متبوعاً بالمعامل (.) ومن ثم اسم الحقل الساكن. مثلاً: Employee.Count.
- نقوم في الصف التالي بتعريف الحقل الساكن العام Count. سيقوم باني الصف في كل مرة يتم إنشاء غرض من الصف بزيادة هذا العداد بمقدار واحد.

```
// Employee.cs
// Static variable used to maintain a count of the number of
// Employee objects that have been created.
using System;

public class Employee
{
   public static int Count { get; private set; } // objects in memory

   // read-only auto-implemented property FirstName
```

```
   public string FirstName { get; private set; }

   // read-only auto-implemented property LastName
   public string LastName { get; private set; }

   // initialize employee, add 1 to static Count and
   // output string indicating that constructor was called
   public Employee( string first, string last )
   {
      FirstName = first;
      LastName = last;
      ++Count; // increment static count of employees
      Console.WriteLine( "Employee constructor: {0} {1}; Count = {2}",
         FirstName, LastName, Count );
   } // end Employee constructor
} // end class Employee
```

- نقوم فيما يلي بإنشاء غرضين من الصف السابق ومن ثم إظهار عدد الموظفين:

```
//  EmployeeTest.cs
// Static member demonstration.
using System;

public class EmployeeTest
{
   public static void Main( string[] args )
   {
      // show that Count is 0 before creating Employees
      Console.WriteLine( "Employees before instantiation: {0}",
         Employee.Count );

      // create two Employees; Count should become 2
      Employee e1 = new Employee( "Susan", "Baker" );
      Employee e2 = new Employee( "Bob", "Blue" );

      // show that Count is 2 after creating two Employees
      Console.WriteLine( "\nEmployees after instantiation: {0}",
         Employee.Count );

      // get names of Employees
      Console.WriteLine( "\nEmployee 1: {0} {1}\nEmployee 2: {2} {3}\n",
         e1.FirstName, e1.LastName,
         e2.FirstName, e2.LastName );

   } // end Main
} // end class EmployeeTest
```

- يكون ناتج التنفيذ:

```
Employees before instantiation: 0
Employee constructor: Susan Baker; Count = 1
Employee constructor: Bob Blue; Count = 2

Employees after instantiation: 2
```

```
Employee 1: Susan Baker
Employee 2: Bob Blue

Press any key to continue . . .
```

الأعضاء الساكنة static

- يكون لكل غرض نسخه الخاصة من حقول الصف.
- نحتاج في بعض الحالات إلى تشارك جميع أغراض الصف لنفس البيانات.
- يسمح التصريح عن حقل باستخدام الكلمة المفتاحية `static` بمشاركة هذا الحقل من قبل جميع أغراض الصف.
- يتم الوصول لحقل ساكن عام من خارج الصف باستخدام اسم الصف متبوعاً بالمعامل (.) ومن ثم اسم الحقل الساكن. مثلاً: Employee.Count.
- نقوم في الصف التالي بتعريف الحقل الساكن العام Count. سيقوم باني الصف في كل مرة يتم إنشاء غرض من الصف بزيادة هذا العداد بقيمة واحد (وبالتالي سيكون لدينا دائماً في هذا العداد عدد الأغراض المنشأة من الصف).

```
// Employee.cs
// Static variable used to maintain a count of the number of
// Employee objects that have been created.
using System;

public class Employee
{
   public static int Count { get; private set; } // objects in memory

   // read-only auto-implemented property FirstName
   public string FirstName { get; private set; }

   // read-only auto-implemented property LastName
   public string LastName { get; private set; }

   // initialize employee, add 1 to static Count and
   // output string indicating that constructor was called
```

```
   public Employee( string first, string last )
   {
      FirstName = first;
      LastName = last;
      ++Count; // increment static count of employees
      Console.WriteLine( "Employee constructor: {0} {1}; Count = {2}",
         FirstName, LastName, Count );
   } // end Employee constructor
} // end class Employee
```

- نقوم فيما يلي بإنشاء غرضين من الصف السابق ومن ثم إظهار عدد الموظفين:

```
//  EmployeeTest.cs
// Static member demonstration.
using System;

public class EmployeeTest
{
   public static void Main( string[] args )
   {
      // show that Count is 0 before creating Employees
      Console.WriteLine( "Employees before instantiation: {0}",
         Employee.Count );

      // create two Employees; Count should become 2
      Employee e1 = new Employee( "Susan", "Baker" );
      Employee e2 = new Employee( "Bob", "Blue" );

      // show that Count is 2 after creating two Employees
      Console.WriteLine( "\nEmployees after instantiation: {0}",
         Employee.Count );

      // get names of Employees
      Console.WriteLine( "\nEmployee 1: {0} {1}\nEmployee 2: {2} {3}\n",
         e1.FirstName, e1.LastName,
         e2.FirstName, e2.LastName );

   } // end Main
} // end class EmployeeTest
```

- يكون ناتج التنفيذ:

```
Employees before instantiation: 0
Employee constructor: Susan Baker; Count = 1
Employee constructor: Bob Blue; Count = 2

Employees after instantiation: 2

Employee 1: Susan Baker
Employee 2: Bob Blue

Press any key to continue . . .
```

3- الطرق الساكنة static methods

- تُستخدم الطرق الساكنة عند الحاجة لتعريف طرق لا ترتبط بأغراض الصف. عادةً، تكون طرق عامة مثل حساب القاسم المشترك الأعظم لعددين.
- لا يُمكن في الطرق الساكنة استخدام أعضاء غير ساكنة من الصف.
- لا يُمكن في الطرق الساكنة استخدام الكلمة المفتاحية this.
- نحتاج في بعض الحالات إلى تشارك جميع أغراض الصف لنفس البيانات. يسمح التصريح عن حقل باستخدام الكلمة المفتاحية static بمشاركة هذا الحقل من قبل جميع أغراض الصف.
- نقوم في المثال التالي بالتصريح عن الحقل الساكن instances. يتشارك جميع أغراض هذا الصف الحقل الساكن. يقوم باني الصف بزيادة 1 لهذا الحقل.
- نُصرح في الصف عن الطريقة الساكنة (HowManyCats) والتي تقوم بإظهار قيمة الحقل الساكن instances.

```
public class Cat
{
    private static int instances = 0;
    public Cat()
    {
        instances++;
    }
    public static void HowManyCats()
    {
        Console.WriteLine("{0} cats adopted", instances);
    }
}
```

- نقوم فيما يلي بإنشاء ثلاثة أغراض من الصف السابق:

```
public class StaticTest
{
   public static void Main( string[] args )
   {
       Cat.HowManyCats();
       Cat frisky = new Cat();
       Cat.HowManyCats();
       Cat whiskers = new Cat();
       Cat.HowManyCats();


   } // end Main
} // end
```

- يُعطي التنفيذ:

```
0 cats adopted
1 cats adopted
2 cats adopted
Press any key to continue . . .
```

الصف Math

- يحوي الصف Math مجموعة من الطرق الساكنة للعمليات الرياضية.
- يُبين الجدول التالي أهم طرق الصف Math:

الطريقة	الوصف	أمثلة

`Abs( x )`	القيمة المطلقة لـ x	`Abs( 23.7 )` is `23.7`
		`Abs( 0 )` is `0`
		`Abs( -23.7 )` is `23.7`

`Ceiling( x )`	التقريب لأصغر عدد طبيعي ليس أصغر من x	`Ceiling( 9.2 )` is `10.0` `Ceiling( -9.8 )` is `-9.0`
`Cos( x )`	تجيب x (x بالراديان) (x in radians)	`Cos( 0.0 )` is `1.0`

`Exp( x )`	الرفع لقوة العدد e	`Exp( 1.0 )` is approximately
		`2.7182818284590451`
		`Exp( 2.0 )` is approximately
		`7.3890560989306504`

`Floor( x )`	التقريب لأكبر عدد طبيعي ليس أكبر من x	`Floor( 9.2 )` is `9.0` `Floor( -9.8 )` is `-10.0`

`Log( x )`	اللوغاريتم الطبيعي لـ x (القاعدة e)	`Log( 2.7182818284590451 )`
		is approximately `1.0`
		`Log( 7.3890560989306504 )`
		is approximately `2.0`

`Max( x, y )`	أكبر قيمة	`Max( 2.3, 12.7 )` is `12.7` `Max( -2.3, -12.7 )` is `-2.3`
`Min( x, y )`	أصغر قيمة	`Min( 2.3, 12.7 )` is `2.3` `Min( -2.3, -12.7 )` is `-12.7`
`Pow( x, y )`	x مرفوع للقوة y	`Pow( 2.0, 7.0 )` is `128.0` `Pow( 9.0, .5 )` is `3.0`
`Sin( x )`	جيب x (x بالراديان) (x in radians)	`Sin( 0.0 )` is `0.0`
`Sqrt( x )`	الجذر التربيعي لـ x	`Sqrt( 900.0 )` is `30.0` `Sqrt( 9.0 )` is `3.0`

- كما يحوي الصف Math الثوابت:
  - Math.PI = 3.1415926535…
  - Math.E = 2.7182818285…

4- اقتراحات وتمارين

تمرين: صف حساب التوفير *Savings-Account*

- قم بالتصريح عن الصف حساب التوفير SavingsAccount.
- استخدم الحقل الساكن annualInterestRate لتخزين الفائدة السنوية لجميع الحسابات من هذا الصف.
- يحوي هذا الصفين الحقل الخاص savingsBalance لتخزين قيمة رصيد الحساب.
- يحوي الصف الطريقة CalculateMonthlyInterest التي تُستخدم لحساب الفائدة الشهرية وذلك بضرب الرصيد CalculateMonthlyInterest بمعدل الفائدة السنوية annualInterestRate مقسوماً على 12. يجب إضافة هذه الفائدة إلى الرصيد.
- يكون للصف الطريقة الساكنة ModifyInterestRate لإسناد قيمة جديدة إلى الحقل الساكن annualInterestRate.
- قم باختبار الصف السابق عن طريق إنشاء غرضين saver1 وsaver2 مع رصيد ابتدائي 2000 و3000. قم بإسناد القيمة 4% إلى الفائدة السنوية annualInterestRate. ثم قم بحساب الفائدة الشهرية ومن ثم طباعة الأرصدة الجديدة.

الفصل الرابع
الوراثة Inheritance.

عنوان الموضوع:

الوراثة Inheritance.

الكلمات المفتاحية:

الصف الأساسي، الصف المشتق.

ملخص:

نستعرض في هذا الفصل الإمكانيات المختلفة التي تُقدّمها الوراثة بين الصفوف.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- الصف الأساسي.
- الصف المشتق.

المخطط:

الوراثة Inheritance.

- 3 وحدات (Learning Objects)

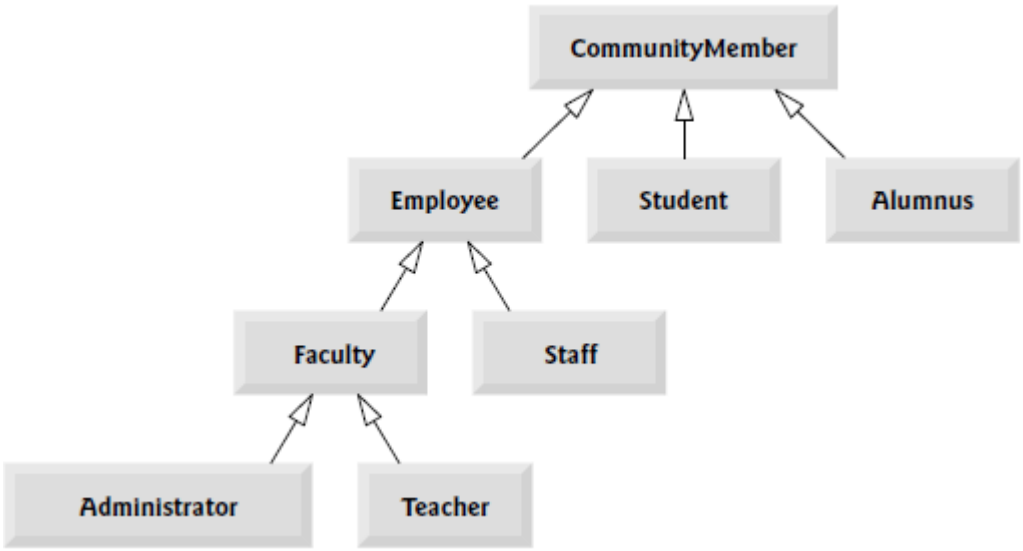
1- الصفوف الأساسية والصفوف المشتقة Base Classes and Derived Classes

- يُمكن في الكثير من الأحيان أن يكون غرض من صف ما هو غرض من صف آخر بنفس الوقت. فمثلاً، في الهندسة، يكون كل مستطيل شكل رباعي (مثله مثل المربع والمعين). إلا أنه بالطبع ليس من الضرورة أن يكون أي شكل رباعي مستطيل.
- نقول في هذه الحالة أن الشكل الرباعي هو صف أساسي Base Class وأن المستطيل هو صف مشتق Derived Class.
- يُبين الجدول التالي بعض الأمثلة لصفوف أساسية وصفوف مشتقة منها:

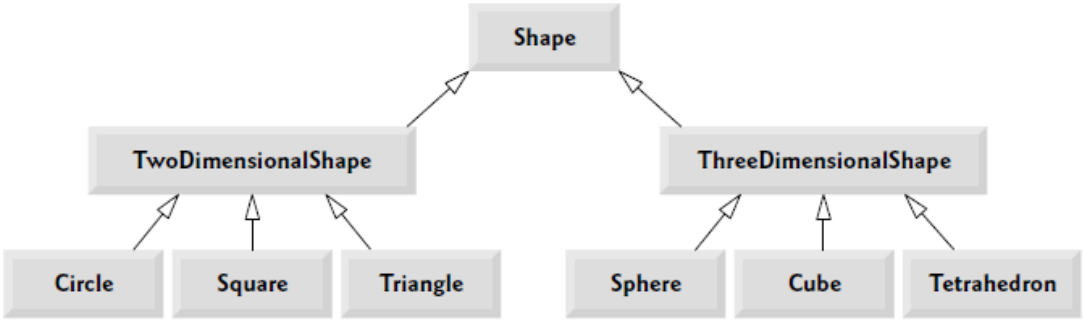
الصف الأساسي	الصفوف المشتقة
طالب Student	طالب متخرج، طالب غير متخرج GraduateStudent, UndergraduateStudent
شكل هندسي Shape	دائرة، مثلث، مستطيل Circle, Triangle, Rectangle
موظف Employee	هيئة تعليمية، هيئة إدارية، عامل بالساعة، عامل بالعمولة Faculty, Staff, HourlyWorker, CommissionWorker
حساب بنكي Account	حساب شيك، حساب توفير CheckingAccount, SavingsAccount
قرض Loan	قرض سيارة، قرض منزلي، رهن عقاري CarLoan, HomeImprovementLoan, MortgageLoan

- بما أن كل غرض من صف مشتق هو بنفس الوقت غرض من الصف الأساسي. وبما أن كل صف يُمكن أن يكون له أكثر من صف مشتق فإن مجموعة الأغراض التي تُمثّل الصف الأساسي تكون عادةً أكثر من مجموعة الأغراض التي تُمثل أي صف مشتق. فمثلاً يُمثّل الصف عربة Vehicle جميع العربات من الصفوف سيارة Car، شاحنة Truck، سفينة Boat. بينما يُمثّل الصف سيارة Car مجموعة أصغر وأكثر تحديداً من العربات.
- تكوّن علاقات الوراثة بنية هرمية (شجرية).
- تُبين الهرمية التالية علاقات الوراثة في الكادر البشري لجامعة. حيث يتفرع عن الصف الأساسي شخص في المجموعة CommunityMember الصفوف المشتقة: موظف Employee، طالب Student، متخرج سابق Alumnus.

- يتفرع عن صف الموظف Employee الصف كلية Faculty والذي يُمثّل العاملين في الكلية. والصف كادر Staff والذي يُمثل بقية العاملين.
- ينقسم العاملون في الكلية إلى معلمين Teacher وإداريين Administrator.



- تُبين الهرمية التالية علاقات الوراثة في مسألة أشكال هندسية:
- يتفرع عن الصف الأساسي شكل Shape الصفين: الأشكال ثنائية البعد TwoDimensionalShape والأشكال ثلاثية الأبعاد ThreeDimensionalShape.
- يحوي صف الأشكال ثنائية البعد: الدائرة Circle والمربع Square والمثلث Triangle.
- بينما يضم صف الأشكال ثلاثية البعد: الكرة Sphere والمكعب Cube والرباعي السطوح Tetrahedron.



أعضاء الصف المحمية Protected Members

- ناقشنا سابقاً الفرق بين مُحدّد الوصول عام public والذي يعني أنه يُمكن الوصول لعضو الصف من خارج الصف. ومحدّد الوصول خاص private والذي يعني أنه لا يُمكن الوصول إلى العضو إلا ضمن الصف نفسه ولا يُمكن الوصول له من الصفوف المشتقة من الصف.
- يوفر مُحدّد الوصول محمي protected إمكانية الوصول لعضو صف في الصف نفسه وفي جميع الصفوف المشتقة من هذا الصف.
- تبقى مُحدّدات الصف نفسها عند وراثة الأعضاء. بمعنى أنه عندما يرث صف عضو عام public من صف آخر فإن هذا العضو يكون أيضاً عام public في الصف المشتق. وهكذا.

2- الصفوف الأساسية والصفوف المشتقة (2)

- سنقوم بشرح مفاهيم الوراثة من خلال المثال التعليمي التالي: ليكن لدينا في نظام محاسبة الموظفين في شركة نوعين من الموظفين. النوع الأول يتمّ محاسبته وفق نسبة معينة من مبيعاته. أما النوع الثاني من الموظفين فله معاش ثابت إضافة إلى نسبة من مبيعاته.

إنشاء واستخدام الصف CommissionEmployee

- نقوم في الصف CommissionEmployee بتعريف خمسة حقول خاصة:

```
private string firstName;//الاسم الأول
private string lastName;//الاسم الأخير
private string socialSecurityNumber;//رقم التأمينات الاجتماعية
private decimal grossSales; // المبيعات الأسبوعية
private decimal commissionRate;//  نسبة العمولة
```

- نقوم في باني الصف بإسناد قيم لهذه الحقول.
- نُصرح عن ثلاثة خصائص عامة للقراءة فقط (لا نستخدم set) للاسم الأول والأخير ورقم التأمينات.
- نُصرح عن الخاصية العامة GrossSales والتي تسمح بقراءة المبيعات الأسبوعية أو إسناد قيمة لها بعد التحقق من أنها أكبر من الصفر.

- نُصرح عن الخاصية العامة CommissionRate والتي تسمح بقراءة نسبة العمولة أو إسناد قيمة لها بعد التحقق من أنها بين الصفر والواحد.
- تقوم الطريقة العامة Earnings() بحساب استحقاق الموظف (ناتج جداء المبيعات الأسبوعية بنسبة العمولة).
- تقوم الطريقة العامة ToString() بركوب الطريقة ToString() المعرّفة على الصف الأساسي object لإرجاع سلسلة نصية تُظهر بيانات الموظف.

```
//  CommissionEmployee.cs
// CommissionEmployee class represents a commission employee.
using System;
public class CommissionEmployee : object
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee( string first, string last, string ssn,
      decimal sales, decimal rate )
   {
      // implicit call to object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      GrossSales = sales; // validate gross sales via property
      CommissionRate = rate; // validate commission rate via property
   } // end five-parameter CommissionEmployee constructor

   // read-only property that gets commission employee's first name
   public string FirstName
   {
      get
      {
         return firstName;
      } // end get
   } // end property FirstName

   // read-only property that gets commission employee's last name
   public string LastName
   {
      get
      {
         return lastName;
      } // end get
   } // end property LastName
   // read-only property that gets
   // commission employee's social security number
   public string SocialSecurityNumber
   {
      get
      {
         return socialSecurityNumber;
```

```
      } // end get
   } // end property SocialSecurityNumber
   // property that gets and sets commission employee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
      set
      {
         if ( value >= 0 )
            grossSales = value;
         else
            throw new ArgumentOutOfRangeException(
               "GrossSales", value, "GrossSales must be >= 0" );
      } // end set
   } // end property GrossSales
   // property that gets and sets commission employee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate

   // calculate commission employee's pay
   public decimal Earnings()
   {
      return commissionRate * grossSales;
   } // end method Earnings

   // return string representation of CommissionEmployee object
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}",
         "commission employee", FirstName, LastName,
         "social security number", SocialSecurityNumber,
         "gross sales", GrossSales, "commission rate", CommissionRate );
   } // end method ToString
} // end class CommissionEmployee
```

- نستخدم فيما يلي الصف السابق:

```
//  CommissionEmployeeTest.cs
// Testing class CommissionEmployee.
using System;
public class CommissionEmployeeTest
{
   public static void Main( string[] args )
```

```
   {
      // instantiate CommissionEmployee object
      CommissionEmployee employee = new CommissionEmployee( "Sue",
         "Jones", "222-22-2222", 10000.00M, .06M );
      // display commission employee data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}", employee.FirstName );
      Console.WriteLine( "Last name is {0}", employee.LastName );
      Console.WriteLine( "Social security number is {0}",
         employee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}", employee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         employee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}", employee.Earnings() );

      employee.GrossSales = 5000.00M; // set gross sales
      employee.CommissionRate = .1M; // set commission rate

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString", employee );
      Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
   } // end Main
} // end class CommissionEmployeeTest
```

- يكون ناتج التنفيذ:

```
Employee information obtained by properties and methods:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales are $10,000.00
Commission rate is 0.06
Earnings are $600.00

Updated employee information obtained by ToString:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: $5,000.00
commission rate: 0.10
earnings: $500.00
Press any key to continue . . .
```

إنشاء الصف BasePlusCommissionEmployee بدون استخدام الوراثة

- نقوم في الصف BasePlusCommissionEmployee بتعريف ستة حقول خاصة (لاحظ أن الحقول الخمسة الأولى مماثلة لتلك الموجودة في الصف السابق CommissionEmployee):

```
private string firstName;//الاسم الأول
private string lastName;//الاسم الأخير
private string socialSecurityNumber;//رقم التأمينات الاجتماعية
private decimal grossSales; // المبيعات الأسبوعية
private decimal commissionRate;// نسبة العمولة
private decimal baseSalary; // المعاش القاعدي
```

- نقوم في باني الصف بإسناد قيم لهذه الحقول.
- نُصرح عن ثلاثة خصائص عامة للقراءة فقط (لا نستخدم set) للاسم الأول والأخير ورقم التأمينات.
- نُصرح عن الخاصية العامة GrossSales والتي تسمح بقراءة المبيعات الأسبوعية أو إسناد قيمة لها بعد التحقق من أنها أكبر من الصفر.
- نُصرح عن الخاصية العامة CommissionRate والتي تسمح بقراءة نسبة العمولة أو إسناد قيمة لها بعد التحقق من أنها بين الصفر والواحد.
- نُصرح عن الخاصية العامة BaseSalary والتي تسمح بقراءة المعاش القاعدي أو إسناد قيمة له بعد التحقق من أنها أكبر من الصفر.
- تقوم الطريقة العامة Earnings() بحساب استحقاق الموظف (المعاش القاعدي + ناتج جداء المبيعات الأسبوعية بنسبة العمولة).
- تقوم الطريقة العامة ToString() بركوب الطريقة ToString() المعرّفة على الصف الأساسي object لإرجاع سلسلة نصية تُظهر بيانات الموظف.

```
//  BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee class represents an employee that receives
// a base salary in addition to a commission.
using System;

public class BasePlusCommissionEmployee
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage
```

```
   private decimal baseSalary; // base salary per week

   // six-parameter constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
   {
      // implicit call to object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      GrossSales = sales; // validate gross sales via property
      CommissionRate = rate; // validate commission rate via property
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor

   // read-only property that gets
   // BasePlusCommissionEmployee's first name
   public string FirstName
   {
      get
      {
         return firstName;
      } // end get
   } // end property FirstName

   // read-only property that gets
   // BasePlusCommissionEmployee's last name
   public string LastName
   {
      get
      {
         return lastName;
      } // end get
   } // end property LastName

   // read-only property that gets
   // BasePlusCommissionEmployee's social security number
   public string SocialSecurityNumber
   {
      get
      {
         return socialSecurityNumber;
      } // end get
   } // end property SocialSecurityNumber

   // property that gets and sets
   // BasePlusCommissionEmployee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
      set
      {
         if ( value >= 0 )
            grossSales = value;
         else
            throw new ArgumentOutOfRangeException(
               "GrossSales", value, "GrossSales must be >= 0" );
      } // end set
   } // end property GrossSales
```

```
   // property that gets and sets
   // BasePlusCommissionEmployee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate

   // property that gets and sets
   // BasePlusCommissionEmployee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
      } // end set
   } // end property BaseSalary

   // calculate earnings
   public decimal Earnings()
   {
      return baseSalary + ( commissionRate * grossSales );
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}\n{9}: {10:C}",
         "base-salaried commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate,
         "base salary", baseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

- نستخدم فيما يلي الصف السابق:

```
//  BasePlusCommissionEmployeeTest.cs
// Testing class BasePlusCommissionEmployee.
using System;
```

```
public class BasePlusCommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );
      // display BasePlusCommissionEmployee's data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}", employee.FirstName );
      Console.WriteLine( "Last name is {0}", employee.LastName );
      Console.WriteLine( "Social security number is {0}",
         employee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}", employee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         employee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}", employee.Earnings() );
      Console.WriteLine( "Base salary is {0:C}", employee.BaseSalary );
      employee.BaseSalary = 1000.00M; // set base salary

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString", employee );
      Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
   } // end Main
} // end class BasePlusCommissionEmployeeTest
```

- يكون ناتج التنفيذ:

```
Employee information obtained by properties and methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales are $5,000.00
Commission rate is 0.04
Earnings are $500.00
Base salary is $300.00

Updated employee information obtained by ToString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00
earnings: $1,200.00
Press any key to continue . . .
```

العلاقات بين الصفوف الأساسية والصفوف المشتقة

- سنقوم بشرح مفاهيم الوراثة من خلال المثال التعليمي التالي: ليكن لدينا في نظام محاسبة الموظفين في شركة نوعين من الموظفين. النوع الأول يتمّ محاسبته وفق نسبة معينة من مبيعاته. أما النوع الثاني من الموظفين فله معاش ثابت إضافة إلى نسبة من مبيعاته.
- سوف نقوم بعرض الحالات التعليمية عبر الأمثلة الخمسة التالية:

1-في المثال الأول، نقوم بإنشاء الصف موظف بالعمولة CommissionEmployee والذي يرث من الصف غرض `object`. نُصرّح داخل هذا الصف عن الحقول الخاصة `private` التالية: الاسم الأول، الاسم الأخير، رقم التأمين الاجتماعي، نسبة العمولة، المبيعات الكلية للموظف.

2-نقوم في المثال الثاني بالتصريح عن الصف قاعدة مع عمولة BasePlusCommissionEmployee والذي يرث أيضاً من الصف غرض `object`. ونُصرّح أيضاً عن الحقول الخاصة التالية: الاسم الأول، الاسم الأخير، رقم التأمين الاجتماعي، نسبة العمولة، المبيعات الكلية للموظف. (سنرى طبعاً أنه من الأفضل توريث هذا الصف من الصف السابق).

3-نقوم في المثال الثالث بالتصريح عن الصف BasePlusCommissionEmployee بأنه مشتق من الصف CommissionEmployee. (الموظف من النوع BasePlusCommissionEmployee هو موظف من النوع CommissionEmployee إضافة إلى أن له معاش قاعدي). سنرى في هذا المثال أنه يجب التصريح بشكل افتراضي `virtual` في الصف الأساسي عن أي طريقة سنقوم بالركوب فوقها `override` في الصف المشتق. سنحاول في الصف المشتق الوصول إلى الخصائص الخاصة `private` في الصف الأساسي مما سيُنتج خطأ.

4-نقوم في المثال الرابع بالتصريح عن أعضاء الصف الأساسي بأنها محمية `protected` وسنرى إذاً أنه يُمكن الوصول إليها في الصف المشتق.

5-نقوم في المثال الخامس بتقديم أفضل الممارسات البرمجية وذلك بإرجاع أعضاء الصف الأساسي إلى خاصة `private` (كما يجب أن تكون من وجهة نظر هندسة البرمجيات). ثم نستخدم في الصف المشتق BasePlusCommissionEmployee الطرق العامة التي يوفرها

الصف الأساسي CommissionEmployee للتعامل من خلالها مع الأعضاء المحمية للصف الأساسي.

3- الصفوف الأساسية والصفوف المشتقة (3)

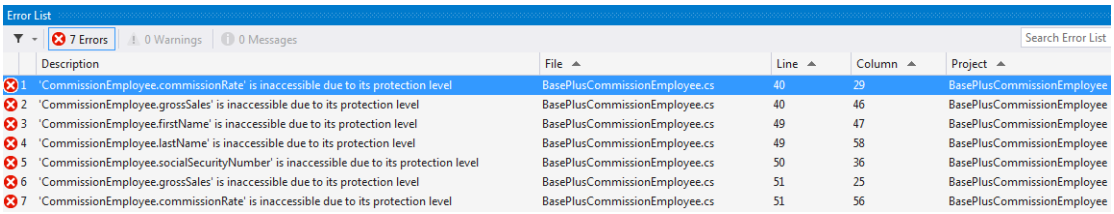
التصريح عن الوراثة بين الصف CommissionEmployee والصف BasePlusCommissionEmployee

- نقوم فيما يلي بالتصريح عن الصف BasePlusCommissionEmployee كصف يرث من الصف CommissionEmployee.
- نقوم فقط بالتصريح عن الحقل baseSalary (إذ أن الحقول الأخرى ستكون موروثة من الصف الأساسي CommissionEmployee).
- نقوم في باني الصف المشتق باستدعاء باني الصف الأساسي عن طريق الكلمة المفتاحية base.

```
// BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee inherits from class CommissionEmployee.
using System;
public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor

   // property that gets and sets
   // BasePlusCommissionEmployee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
```

```
      } // end set
   } // end property BaseSalary
   // calculate earnings
   public override decimal Earnings()
   {
      // not allowed: commissionRate and grossSales private in base class
      return baseSalary + ( commissionRate * grossSales );
   } // end method Earnings
   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      // not allowed: attempts to access private base class members
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}\n{9}: {10:C}",
         "base-salaried commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate,
         "base salary", baseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

- سيُعطي المترجم قائمة الأخطاء التالية نتيجة محاولة الوصول في الصف المشتق إلى الحقول الخاصة `private` في الصف الأساسي:



التصريح عن الوراثة بين الصف CommissionEmployee والصف BasePlusCommissionEmployee مع استخدام الحقول المحمية protected

- نقوم فيما يلي بتعديل الصف `CommissionEmployee` وبحيث نجعل جميع الحقول محمية `protected` عوضاً عن خاصة `private`.

```
// CommissionEmployee.cs
// CommissionEmployee with protected instance variables.
using System;
public class CommissionEmployee : object
{
   protected string firstName;
   protected string lastName;
   protected string socialSecurityNumber;
   protected decimal grossSales; // gross weekly sales
   protected decimal commissionRate; // commission percentage
   // five-parameter constructor
   public CommissionEmployee( string first, string last, string ssn,
```

```
      decimal sales, decimal rate )
   {
      // implicit call to object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      GrossSales = sales; // validate gross sales via property
      CommissionRate = rate; // validate commission rate via property
   } // end five-parameter CommissionEmployee constructor
   // read-only property that gets commission employee's first name
   public string FirstName
   {
      get
      {
         return firstName;
      } // end get
   } // end property FirstName

   // read-only property that gets commission employee's last name
   public string LastName
   {
      get
      {
         return lastName;
      } // end get
   } // end property LastName
   // read-only property that gets
   // commission employee's social security number
   public string SocialSecurityNumber
   {
      get
      {
         return socialSecurityNumber;
      } // end get
   } // end property SocialSecurityNumber
   // property that gets and sets commission employee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
      set
      {
         if ( value >= 0 )
            grossSales = value;
         else
            throw new ArgumentOutOfRangeException(
               "GrossSales", value, "GrossSales must be >= 0" );
      } // end set
   } // end property GrossSales
   // property that gets and sets commission employee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
```

```
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate
   // calculate commission employee's pay
   public virtual decimal Earnings()
   {
      return commissionRate * grossSales;
   } // end method Earnings
   // return string representation of CommissionEmployee object
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}",
         "commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate );
   } // end method ToString
} // end class CommissionEmployee
```

- نقوم بالتصريح عن الصف المشتق `BasePlusCommissionEmployee` واستخدام الحقول المحمية المُعرّفة في الصف الأساسي:

```
//  BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee inherits from CommissionEmployee and has
// access to CommissionEmployee's protected members.
using System;
public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor

   // property that gets and sets
   // BasePlusCommissionEmployee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
      } // end set
   } // end property BaseSalary
```

```
   // calculate earnings
   public override decimal Earnings()
   {
      return baseSalary + ( commissionRate * grossSales );
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}\n{9}: {10:C}",
         "base-salaried commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate,
         "base salary", baseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

- نستخدم فيما يلي الصف السابق:

```
//   BasePlusCommissionEmployeeTest.cs
// Testing class BasePlusCommissionEmployee.
using System;
public class BasePlusCommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee basePlusCommissionEmployee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );

      // display BasePlusCommissionEmployee's data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}",
         basePlusCommissionEmployee.FirstName );
      Console.WriteLine( "Last name is {0}",
         basePlusCommissionEmployee.LastName );
      Console.WriteLine( "Social security number is {0}",
         basePlusCommissionEmployee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}",
         basePlusCommissionEmployee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         basePlusCommissionEmployee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}",
         basePlusCommissionEmployee.Earnings() );
      Console.WriteLine( "Base salary is {0:C}",
         basePlusCommissionEmployee.BaseSalary );

      basePlusCommissionEmployee.BaseSalary = 1000.00M; // set base salary

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString",
         basePlusCommissionEmployee );
      Console.WriteLine( "earnings: {0:C}",
         basePlusCommissionEmployee.Earnings() );
   } // end Main
} // end class BasePlusCommissionEmployeeTest
```

- يكون ناتج التنفيذ:

```
Employee information obtained by properties and methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales are $5,000.00
Commission rate is 0.04
Earnings are $500.00
Base salary is $300.00

Updated employee information obtained by ToString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00
earnings: $1,200.00
Press any key to continue . . .
```

التصريح عن الوراثة بين الصف CommissionEmployee والصف BasePlusCommissionEmployee مع استخدام الحقول الخاصة private

- نقوم فيما يلي بتعديل الصف `CommissionEmployee` وبحيث نُعيد جميع الحقول خاصة `private` (كما يُفترض أن تكون).
- تسمح الخصائص العامة `public` المعرفة في الصف بالتعامل مع هذه الحقول.
- لاحظ أننا نستخدم هذه الخصائص في جميع طرق الصف (`Earnings()` و `ToString()`) عوضاً عن الحقول بهدف أن تكون الخصائص فقط هي واجهة التعامل مع الحقول.
- لو قمنا بتعديل أسماء الحقول الخاصة في الصف `CommissionEmployee`، سنقوم أيضاً بتعديلات في الخصائص فقط لهذا الصف. ولا يتوجب أي تعديلات في الصفوف المشتقة.

من الممارسات الجيدة من وجهة نظر هندسة البرمجيات أن تكون التعديلات محدودة ضمن الصف الواحد.

```
//   CommissionEmployee.cs
// CommissionEmployee class represents a commission employee.
using System;
public class CommissionEmployee
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee( string first, string last, string ssn,
      decimal sales, decimal rate )
   {
      // implicit call to object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      GrossSales = sales; // validate gross sales via property
      CommissionRate = rate; // validate commission rate via property
   } // end five-parameter CommissionEmployee constructor
   // read-only property that gets commission employee's first name
   public string FirstName
   {
      get
      {
         return firstName;
      } // end get
   } // end property FirstName

   // read-only property that gets commission employee's last name
   public string LastName
   {
      get
      {
         return lastName;
      } // end get
   } // end property LastName
   // read-only property that gets
   // commission employee's social security number
   public string SocialSecurityNumber
   {
      get
      {
         return socialSecurityNumber;
      } // end get
   } // end property SocialSecurityNumber

   // property that gets and sets commission employee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
```

```
      set
      {
         if ( value >= 0 )
            grossSales = value;
         else
            throw new ArgumentOutOfRangeException(
               "GrossSales", value, "GrossSales must be >= 0" );
      } // end set
   } // end property GrossSales
   // property that gets and sets commission employee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate
   // calculate commission employee's pay
   public virtual decimal Earnings()
   {
      return CommissionRate * GrossSales;
   } // end method Earnings

   // return string representation of CommissionEmployee object
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}",
         "commission employee", FirstName, LastName,
         "social security number", SocialSecurityNumber,
         "gross sales", GrossSales, "commission rate", CommissionRate );
   } // end method ToString
} // end class CommissionEmployee
```

- نُعرّف الصف المشتق `BasePlusCommissionEmployee` من الصف السابق `CommissionEmployee`.
- لاحظ أننا أيضاً نستخدم في طرق الصف خصائص الصف.
- لاحظ أننا في الطريقة `Earnings()` للصف نستدعي الطريقة `base.Earnings()` للصف الأساسي (ونجمع معها المعاش القاعدي). أيضاً هذا الأسلوب من الممارسات الجيدة من منظور هندسة البرمجيات حيث يُخفف من إعادة كتابة نفس الكود ويُسهّل مسائل صيانة الكود.

```
//  BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee inherits from CommissionEmployee and has
// access to CommissionEmployee's private data via
// its public properties.
using System;
public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week
   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor
   // property that gets and sets
   // BasePlusCommissionEmployee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
      } // end set
   } // end property BaseSalary
   // calculate earnings
   public override decimal Earnings()
   {
      return BaseSalary + base.Earnings();
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      return string.Format( "base-salaried {0}\nbase salary: {1:C}",
         base.ToString(), BaseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

- نستخدم فيما يلي الصف:

```
// BasePlusCommissionEmployeeTest.cs
// Testing class BasePlusCommissionEmployee.
using System;
public class BasePlusCommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );
```

```
      // display BasePlusCommissionEmployee's data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}", employee.FirstName );
      Console.WriteLine( "Last name is {0}", employee.LastName );
      Console.WriteLine( "Social security number is {0}",
         employee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}", employee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         employee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}", employee.Earnings() );
      Console.WriteLine( "Base salary is {0:C}", employee.BaseSalary );

      employee.BaseSalary = 1000.00M; // set base salary

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString", employee );
      Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
   } // end Main
} // end class BasePlusCommissionEmployeeTest
```

- يكون ناتج التنفيذ:

```
Employee information obtained by properties and methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales are $5,000.00
Commission rate is 0.04
Earnings are $500.00
Base salary is $300.00

Updated employee information obtained by ToString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00
earnings: $1,200.00
Press any key to continue . . .
```

4- اقتراحات وتمارين

تمرين 1: صف الدائرة Circle وصف الاسطوانة Cylinder

قم باستخدام صف الدائرة Circle الذي قُمت بإنشائه سابقاً:

- قم بالتصريح عن الصف الاسطوانة Cylinder والذي يرث من الصف الدائرة Circle.
- يكون للاسطوانة خاصية إضافية هي الإرتفاع Height ويجب أن تكون موجبة.
- يكون للاسطوانة باني بدون معاملات لإعطاء القيمة 0 لنصف القطر والارتفاع.
- يكون للاسطوانة باني نُمرر له نصف القطر والارتفاع.
- أضف باني للاسطوانه نمرر له دائرة والارتفاع.
- أضف باني نسخ له معامل دخل من النمط Cylinder.
- ترث الاسطوانة الطريقة محيط الدائرة Circumference() (محيط الاسطوانة هو نفس محيط الدائرة).
- قم بركوب طريقة الدائرة Area() لحساب مساحة الاسطوانة.
- أضف طريقة جديدة لحساب Volume() حجم الاسطوانة.
- أضف الطريقة ToString() للصف لتقوم بإظهار معلومات الاسطوانة.
- قم باختبار الصف السابق مع عدة كائنات منه.

تمرين 2: صف المستطيل Rectangle وصف متوازي المستطيلات Cuboid

قم باستخدام صف المستطيل Rectangle الذي قُمت بإنشائه في فصل سابق:

- قم بالتصريح عن الصف متوازي المستطيلات Cuboid والذي يرث من الصف المستطيل Rectangle.
- يكون لمتوازي المستطيلات خاصية إضافية هي الإرتفاع Height ويجب أن تكون موجبة.
- يكون لمتوازي المستطيلات باني بدون معاملات لإعطاء القيمة 0 لجميع الأبعاد.

- يكون لمتوازي المستطيلات باني نُمرر له الأبعاد الثلاثة.
- أضف باني لمتوازي المستطيلات نمرر له مستطيل والارتفاع.
- أضف باني نسخ له معامل دخل من النمط Rectangle.
- يرث متوازي المستطيلات الطريقة محيط المستطيل Perimeter() (محيط متوازي المستطيلات هو نفس محيط المستطيل).
- قم بركوب طريقة المستطيل Area() لحساب مساحة متوازي المستطيلات.
- أضف طريقة جديدة لحساب Volume() حجم متوازي المستطيلات.
- أضف الطريقة ToString() للصف لتقوم بإظهار معلومات متوازي المستطيلات.
- قم باختبار الصف السابق مع عدة كائنات منه.

الفصل الخامس

تعدد الأشكال Polymorphism.

عنوان الموضوع:

تعدد الأشكال Polymorphism.

الكلمات المفتاحية:

Polymorphism ، Abstract Classes، Sealed Classes.

ملخص:

نستعرض في هذا الفصل استخدام مبدأ تعدد الأشكال في البرمجة غرضية التوجه. كما نعرض استخدام الصفوف والطرق المجردة والعقيمة.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- تعدد الأشكال Polymorphism.
- الصفوف والطرق المجردة Abstract Classes.
- الصفوف والطرق العقيمة Sealed methods and classes.

المخطط:

تعدد الأشكال Polymorphism

- 4 وحدات (Learning Objects)

1- تعدد الأشكال Polymorphism

- يسمح تعدد الأشكال بشكل عام من معالجة أغراض من صفوف مختلفة إلا أنها جميعاً مورثة من صف أساسي واحد وذلك بنفس الطرق. يُمكن، كما سنرى، أن تكون طريقة معالجة نفس الطريقة مختلفة وذلك حسب الصف الذي ينتمي الغرض إليه.

مثال على تعدد الأشكال

- يسمح المترجم بإسناد غرض من صف مشتق إلى غرض مُعرّف من الصف الأساسي بينما لا يسمح بالعكس. بالطبع هذا السلوك متوقع لأن كل غرض من صف مشتق هو بنفس الوقت غرض من الصف الأساسي.
- عند استدعاء طريقة على الغرض من الصف الأساسي والذي تمّ إسناده بغرض من صف مشتق، سيتمّ تنفيذ الطريقة المُعرّفة في الصف المشتق.
- يُبين المثال التالي هذا المبدأ: بالعودة إلى مثال الصف الأساسي CommissionEmployee والصف المشتق BasePlusCommissionEmployee. ليكن لدينا الاستخدام التالي:
- نقوم أولاً بإنشاء الغرض commissionEmployee من الصف CommissionEmployee، والغرض basePlusCommissionEmployee من الصف BasePlusCommissionEmployee.
- نقوم باستدعاء الطريقة ToString() على الغرض commissionEmployee مما يؤدي إلى استدعاء الطريقة ToString() المُعرّفة في الصف CommissionEmployee.
- نقوم باستدعاء الطريقة ToString() على الغرض basePlusCommissionEmployee مما يؤدي إلى استدعاء الطريقة ToString() المُعرّفة في الصف BasePlusCommissionEmployee.
- نقوم بعدها بالتصريح عن الغرض commissionEmployee2 من نمط الصف الأساسي CommissionEmployee وإسناد غرض من الصف المشتق BasePlusCommissionEmployee له:

```
CommissionEmployee commissionEmployee2 = basePlusCommissionEmployee;
```

- نستدعي الطريقة ToString() على الغرض commissionEmployee2. يؤدي هذا الاستدعاء إلى تنفيذ الطريقة ToString() المُعرّفة في الصف المشتق BasePlusCommissionEmployee.
- نستدعي الطريقة Earnings() على الغرض commissionEmployee2. يؤدي هذا الاستدعاء إلى تنفيذ الطريقة Earnings() المُعرّفة في الصف المشتق BasePlusCommissionEmployee.

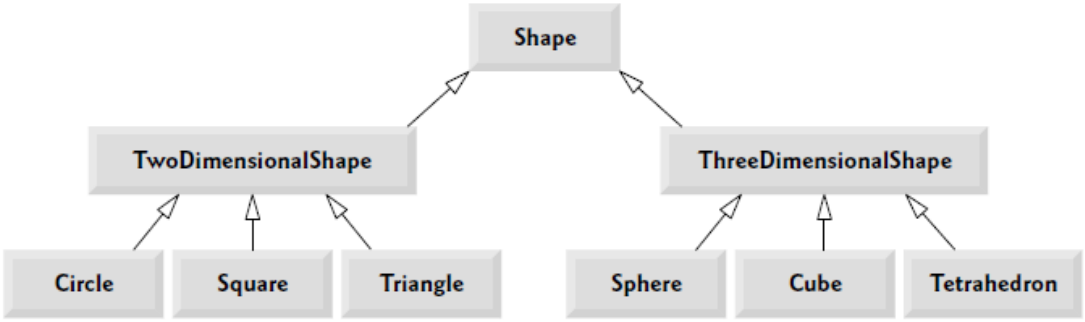
```
//  PolymorphismTest.cs
// Assigning base class and derived class references to base class and
// derived class variables.
using System;
public class PolymorphismTest
{
   public static void Main( string[] args )
   {
      // assign base class reference to base class variable
      CommissionEmployee commissionEmployee = new CommissionEmployee(
         "Sue", "Jones", "222-22-2222", 10000.00M, .06M );

      // assign derived class reference to derived class variable
      BasePlusCommissionEmployee basePlusCommissionEmployee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );

      // invoke ToString and Earnings on base class object
      // using base class variable
      Console.WriteLine( "{0} {1}:\n\n{2}\n{3}: {4:C}\n",
         "Call CommissionEmployee's ToString and Earnings methods",
         "with base class reference to base class object",
         commissionEmployee.ToString(),
         "earnings", commissionEmployee.Earnings() );

      // invoke ToString and Earnings on derived class object
      // using derived class variable
      Console.WriteLine( "{0} {1}:\n\n{2}\n{3}: {4:C}\n",
         "Call BasePlusCommissionEmployee's ToString and Earnings",
         "methods with derived class reference to derived class object",
         basePlusCommissionEmployee.ToString(),
         "earnings", basePlusCommissionEmployee.Earnings() );

      // invoke ToString and Earnings on derived class object
      // using base class variable
      CommissionEmployee commissionEmployee2 = basePlusCommissionEmployee;
      Console.WriteLine( "{0} {1}:\n\n{2}\n{3}: {4:C}",
         "Call BasePlusCommissionEmployee's ToString and Earnings",
         "methods with base class reference to derived class object",
         commissionEmployee2.ToString(), "earnings",
         commissionEmployee2.Earnings() );
   } // end Main
} // end class PolymorphismTest
```

- يكون ناتج التنفيذ:

```
Call CommissionEmployee's ToString and Earnings methods with base class reference to base class object:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: $10,000.00
commission rate: 0.06
earnings: $600.00

Call BasePlusCommissionEmployee's ToString and Earnings methods with derived class reference to derived class object:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $300.00
earnings: $500.00

Call BasePlusCommissionEmployee's ToString and Earnings methods with base class reference to derived class object:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $300.00
earnings: $500.00
Press any key to continue . . .
```

2- الصفوف المجردة Abstract Classes

- يكون من المفيد في الكثير من الحالات التصريح عن صفوف لن نقوم بإنشاء (new) أغراض منها. يُمكن، بالطبع، التصريح عن أغراض منها.
- يُدعى الصف في هذه الحالة بالصف المجرد abstract.
- يُستخدم الصف المجرد عادةً لتجميع الأعضاء المشتركة في هرمية في صف أساسي واحد.
- نقوم مثلاً في هرمية الأشكال الهندسية بتعريف الصف المجرد Shape، والتصريح عن كل الأعضاء المشتركة للصفوف المشتقة فيه.



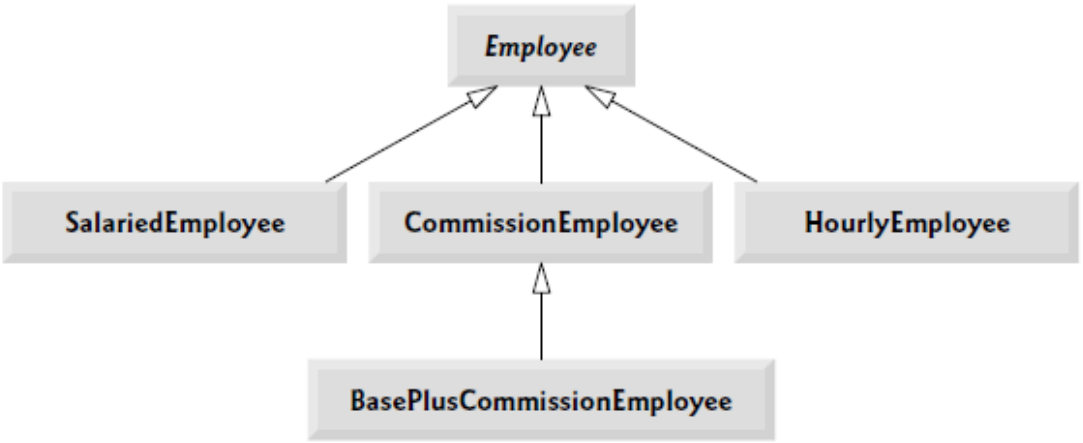
- يُمكن في الصف المجرد كتابة طرق مع الكود اللازم لها. كما يُمكن التصريح عن طرق مجردة abstract وبحيث يتم كتابة توقيع الطريقة فقط.
- سيكون من الواجب على كل صف مشتق من الصف الأساسي المجرد ركوب الطريقة المجردة (override) وكتابة الكود اللازم لها.

3- مثال تعليمي

دراسة حالة: نظام دفع رواتب الموظفين في شركة باستخدام تعدد الأشكال

- تقوم شركة بدفع الرواتب أسبوعياً لموظفيها. تضم هذه الشركة أربعة أنواع من الموظفين:
- الموظفون بمعاش ثابت Salaried Employees: يكون لهم معاش ثابت أسبوعياً.
- الموظفون بالساعة Hourly Employees: يُدفع لهم بالساعة. كما يكون لهم تعويض عمل إضافي حيث يُدفع لهم أجر ساعة ونصف عن كل ساعة عمل فوق 40 ساعة أسبوعياً.
- الموظفون بالعمولة Commission Employees: يتقاضون عمولة على مبيعاتهم.

- الموظفون بمعاش وعمولة Base Plus Commission Employees: يكون لهم معاش ثابت وعمولة على مبيعاتهم. كما أن الشركة قررت منحهم مكافاة عبارة عن 10% من معاشهم.



- يُبين الجدول التالي الصفوف اللازمة وكيفية حساب مبلغ كل موظف حسب صفه Earnings، والسلسلة النصية التي يجب أن تُعيدها الطريقة ToString لكل صف:

	Earnings	ToString
Employee	abstract	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* 40 * wage + (hours - 40) * wage * 1.5	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage* hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales* commission rate: *commissionRate*
BasePlus-Commission-Employee	(commissionRate * grossSales) + baseSalary	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales* commission rate: *commissionRate* base salary: *baseSalary*

- نبدأ أولاً بكتابة الصف المجرد Employee والذي نُصرح فيه عن الخصائص المشتركة لجميع الموظفين: الاسم الأول FirstName، الاسم الأخير LastName، رقم التأمينات الاجتماعية SocialSecurityNumber.
- كما نُصرّح عن باني الصف والذي له ثلاثة معاملات توافق الخصائص السابقة.
- نُصرّح أيضاً عن الطريقة ToString() لإعادة سلسلة نصية تُظهر الخصائص السابقة.
- نُصرّح عن الطريقة المجردة Earnings(). سيتوجب على كل صف مشتق من هذا الصف المجرد ركوب (override) هذه الطريقة وكتابة الكود اللازم لها.

```
// Employee.cs
// Employee abstract base class.
public abstract class Employee
{
   // read-only property that gets employee's first name
   public string FirstName { get; private set; }

   // read-only property that gets employee's last name
   public string LastName { get; private set; }

   // read-only property that gets employee's social security number
   public string SocialSecurityNumber { get; private set; }

   // three-parameter constructor
   public Employee( string first, string last, string ssn )
   {
      FirstName = first;
      LastName = last;
      SocialSecurityNumber = ssn;
   } // end three-parameter Employee constructor

   // return string representation of Employee object, using properties
   public override string ToString()
   {
      return string.Format( "{0} {1}\nsocial security number: {2}",
         FirstName, LastName, SocialSecurityNumber );
   } // end method ToString

   // abstract method overridden by derived classes
   public abstract decimal Earnings(); // no implementation here
} // end abstract class Employee
```

- يكون الصف SalariedEmployee مشتق من الصف السابق.
- يكون لهذا الصف الخاصية الإضافية SalariedEmployee المستخدمة للمعاش.
- نقوم بركوب الطريقة ToString() مع استخدام الطريقة base.ToString() من الصف الأساسي.

- نقوم في هذا الصف بركوب override الطريقة المجردة Earnings() المُصرح عنها في الصف المجرد الأساسي.

```
//  SalariedEmployee.cs
// SalariedEmployee class that extends Employee.
using System;
public class SalariedEmployee : Employee
{
   private decimal weeklySalary;

   // four-parameter constructor
   public SalariedEmployee( string first, string last, string ssn,
      decimal salary ) : base( first, last, ssn )
   {
      WeeklySalary = salary; // validate salary via property
   } // end four-parameter SalariedEmployee constructor

   // property that gets and sets salaried employee's salary
   public decimal WeeklySalary
   {
      get
      {
         return weeklySalary;
      } // end get
      set
      {
         if ( value >= 0 ) // validation
            weeklySalary = value;
         else
            throw new ArgumentOutOfRangeException( "WeeklySalary",
               value, "WeeklySalary must be >= 0" );
      } // end set
   } // end property WeeklySalary

   // calculate earnings; override abstract method Earnings in Employee
   public override decimal Earnings()
   {
      return WeeklySalary;
   } // end method Earnings

   // return string representation of SalariedEmployee object
   public override string ToString()
   {
      return string.Format( "salaried employee: {0}\n{1}: {2:C}",
         base.ToString(), "weekly salary", WeeklySalary );
   } // end method ToString
} // end class SalariedEmployee
```

- يكون الصف HourlyEmployee مشتق أيضاً من الصف السابق.
- يكون لهذا الصف الخاصيتين wage و hours لأجرة الساعة وعدد الساعات.
- نقوم بركوب الطريقة ToString() مع استخدام الطريقة base.ToString() من الصف الأساسي.

- نقوم في هذا الصف أيضاً بركوب override الطريقة المجردة Earnings() المُصرح عنها في الصف المجرد الأساسي.

```
// HourlyEmployee.cs
// HourlyEmployee class that extends Employee.
using System;

public class HourlyEmployee : Employee
{
   private decimal wage; // wage per hour
   private decimal hours; // hours worked for the week

   // five-parameter constructor
   public HourlyEmployee( string first, string last, string ssn,
      decimal hourlyWage, decimal hoursWorked )
      : base( first, last, ssn )
   {
      Wage = hourlyWage; // validate hourly wage via property
      Hours = hoursWorked; // validate hours worked via property
   } // end five-parameter HourlyEmployee constructor

   // property that gets and sets hourly employee's wage
   public decimal Wage
   {
      get
      {
         return wage;
      } // end get
      set
      {
         if ( value >= 0 ) // validation
            wage = value;
         else
            throw new ArgumentOutOfRangeException( "Wage",
               value, "Wage must be >= 0" );
      } // end set
   } // end property Wage

   // property that gets and sets hourly employee's hours
   public decimal Hours
   {
      get
      {
         return hours;
      } // end get
      set
      {
         if ( value >= 0 && value <= 168 ) // validation
            hours = value;
         else
            throw new ArgumentOutOfRangeException( "Hours",
               value, "Hours must be >= 0 and <= 168" );
      } // end set
   } // end property Hours

   // calculate earnings; override Employee's abstract method Earnings
   public override decimal Earnings()
   {
      if ( Hours <= 40 ) // no overtime
         return Wage * Hours;
```

```
      else
         return ( 40 * Wage ) + ( ( Hours - 40 ) * Wage * 1.5M );
   } // end method Earnings

   // return string representation of HourlyEmployee object
   public override string ToString()
   {
      return string.Format(
         "hourly employee: {0}\n{1}: {2:C}; {3}: {4:F2}",
         base.ToString(), "hourly wage", Wage, "hours worked", Hours );
   } // end method ToString
} // end class HourlyEmployee
```

- يكون الصف CommissionEmployee مشتق أيضاً من الصف السابق.
- يكون لهذا الصف الخاصيتين grossSales و commissionRate لمجموع المبيعات ونسبة العمولة.
- نقوم بركوب الطريقة ToString() مع استخدام الطريقة base.ToString() من الصف الأساسي.
- نقوم في هذا الصف أيضاً بركوب override الطريقة المجردة Earnings() المُصرح عنها في الصف المجرد الأساسي.

```
// CommissionEmployee.cs
// CommissionEmployee class that extends Employee.
using System;

public class CommissionEmployee : Employee
{
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee( string first, string last, string ssn,
      decimal sales, decimal rate ) : base( first, last, ssn )
   {
      GrossSales = sales; // validate gross sales via property
      CommissionRate = rate; // validate commission rate via property
   } // end five-parameter CommissionEmployee constructor

   // property that gets and sets commission employee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
      set
      {
         if ( value >= 0 )
            grossSales = value;
         else
            throw new ArgumentOutOfRangeException(
               "GrossSales", value, "GrossSales must be >= 0" );
      } // end set
```

```
   } // end property GrossSales

   // property that gets and sets commission employee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate

   // calculate earnings; override abstract method Earnings in Employee
   public override decimal Earnings()
   {
      return CommissionRate * GrossSales;
   } // end method Earnings

   // return string representation of CommissionEmployee object
   public override string ToString()
   {
      return string.Format( "{0}: {1}\n{2}: {3:C}\n{4}: {5:F2}",
         "commission employee", base.ToString(),
         "gross sales", GrossSales, "commission rate", CommissionRate );
   } // end method ToString
} // end class CommissionEmployee
```

- يكون الصف BasePlusCommissionEmployee مشتق من الصف السابق CommissionEmployee.
- يكون لهذا الصف الخاصية baseSalary للمعاش.
- نقوم بركوب الطريقة ToString() مع استخدام الطريقة base.ToString() من الصف الأساسي.
- نقوم في هذا الصف أيضاً بركوب override الطريقة Earnings() المُصرح عنها في الصف الأساسي مع استخدام الطريقة الأساسية base.Earnings().

```
//  BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee class that extends CommissionEmployee.
using System;

public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
```

```
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor

   // property that gets and sets
   // base-salaried commission employee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
      } // end set
   } // end property BaseSalary

   // calculate earnings; override method Earnings in CommissionEmployee
   public override decimal Earnings()
   {
      return BaseSalary + base.Earnings();
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee object
   public override string ToString()
   {
      return string.Format( "base-salaried {0}; base salary: {1:C}",
         base.ToString(), BaseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

- نقوم في المثال التالي بإنشاء أربعة أغراض: غرض من كل صف من الصفوف السابقة.
- نقوم بعدها بالتصريح عن مصفوفة من أربعة أغراض من الصف الأساسي المجرد `Employee`.
- نقوم بإسناد الأغراض الأربعة السابقة إلى عناصر المصفوفة.
- نقوم بعدها بالدوران على عناصر المصفوفة لطباعة معلومات كل غرض. سيتم استدعاء الطريقة `ToString()` الموافقة لصف الغرض.
- كما يتم أيضاً استدعاء الطريقة `Earnings()` على كل عنصر من عناصر المصفوفة. سيتم أيضاً تنفيذ الطريقة `Earnings()` الموافقة لصف العنصر.
- يُمكن اختبار انتماء غرض لصف معين باستخدام الكلمة المفتاحية `is`.

- نقوم في هذا المثال باختبار انتماء عنصر من المصفوفة إلى الصف BasePlusCommissionEmployee. عند تحقق هذا الاختبار، يتم إسناد هذا العنصر (من النمط Employee) إلى غرض من النمط BasePlusCommissionEmployee وذلك كي نتمكن من الوصول إلى الخاصية BaseSalary للصف BasePlusCommissionEmployee.
- لاحظ أنه لإسناد غرض من نمط أساسي إلى غرض من نمط مشتق يجب إجراء عملية قصر casting للغرض.

```
// PayrollSystemTest.cs
// Employee hierarchy test application.
using System;

public class PayrollSystemTest
{
   public static void Main( string[] args )
   {
      // create derived class objects
      SalariedEmployee salariedEmployee =
         new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00M );
      HourlyEmployee hourlyEmployee =
         new HourlyEmployee( "Karen", "Price",
         "222-22-2222", 16.75M, 40.0M );
      CommissionEmployee commissionEmployee =
         new CommissionEmployee( "Sue", "Jones",
         "333-33-3333", 10000.00M, .06M );
      BasePlusCommissionEmployee basePlusCommissionEmployee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "444-44-4444", 5000.00M, .04M, 300.00M );

      Console.WriteLine( "Employees processed individually:\n" );

      Console.WriteLine( "{0}\nearned: {1:C}\n",
         salariedEmployee, salariedEmployee.Earnings() );
      Console.WriteLine( "{0}\nearned: {1:C}\n",
         hourlyEmployee, hourlyEmployee.Earnings() );
      Console.WriteLine( "{0}\nearned: {1:C}\n",
         commissionEmployee, commissionEmployee.Earnings() );
      Console.WriteLine( "{0}\nearned: {1:C}\n",
         basePlusCommissionEmployee,
         basePlusCommissionEmployee.Earnings() );

      // create four-element Employee array
      Employee[] employees = new Employee[ 4 ];

      // initialize array with Employees of derived types
      employees[ 0 ] = salariedEmployee;
      employees[ 1 ] = hourlyEmployee;
      employees[ 2 ] = commissionEmployee;
      employees[ 3 ] = basePlusCommissionEmployee;

      Console.WriteLine( "Employees processed polymorphically:\n" );

      // generically process each element in array employees
```

```
      foreach ( Employee currentEmployee in employees )
      {
         Console.WriteLine( currentEmployee ); // invokes ToString

         // determine whether element is a BasePlusCommissionEmployee
         if ( currentEmployee is BasePlusCommissionEmployee )
         {
            // downcast Employee reference to
            // BasePlusCommissionEmployee reference
            BasePlusCommissionEmployee employee =
               ( BasePlusCommissionEmployee ) currentEmployee;

            employee.BaseSalary *= 1.10M;
            Console.WriteLine(
               "new base salary with 10% increase is: {0:C}",
               employee.BaseSalary );
         } // end if

         Console.WriteLine("earned {0:C}\n", currentEmployee.Earnings() );
      } // end foreach

      // get type name of each object in employees array
      for ( int j = 0; j < employees.Length; j++ )
         Console.WriteLine( "Employee {0} is a {1}", j,
            employees[ j ].GetType() );
   } // end Main
} // end class PayrollSystemTest
```

- يكون ناتج التنفيذ:

```
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00
commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00
```

```
commission rate: 0.04; base salary: $300.00
earned: $500.00

Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00
commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00
commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee
Press any key to continue . . .
```

4- الطرق والصفوف العقيمة Sealed Methods and Classes

- درسنا سابقاً إمكانية ركوب override طريقة مُعرّفة في صف أساسي أنها مجردة abstract أو افتراضية virtual.
- يُمكن التصريح عن طريقة أنها عقيمة sealed وبذلك لن تتمكن الصفوف المشتقة من ركوبها.
- تكون كل الطرق المصرح عنها بأنها خاصة private عقيمة. وبذلك لا يُمكن ركوبها. يُمكن لصف مشتق تعريف طريقة لها نفس توقيع طريقة خاصة في الصف الأساسي حين الحاجة.
- تكون الطرق الساكنة static أيضاً عقيمة حيث لا يمكن ركوب طريقة ساكنة.
- يساعد التصريح عن طريقة بأنها عقيمة sealed المترجم في تحسين الكود الناتج حيث بما أنه يعرف بأن الطرق العقيمة لن يتم ركوبها، فسيقوم باستبدال الاستدعاءات إلى هذه الطرق بكودها.
- في حال التصريح عن صف بأنه عقيم sealed. لا يمكن أن يكون هذا الصف صف أساسي لصفوف مشتقة.

5- اقتراحات وتمارين

تمرين 1: Payroll

- قم بتعديل نظام الدفع كما يلي:
- أضف تاريخ الميلاد birthDate إلى صف الموظف Employee.
- قم بإنشاء مصفوفة من الصف Employee ومن ثم أسند لعناصرها موظفين من مختلف الصفوف المشتقة من صف الموظف Employee.
- قم في حلقة بالدوران على عناصر المصفوفة لحساب المبلغ المترتب لكل موظف مع إضافة مكافآة قيمتها 100 لكل موظف يصدف تاريخ ميلاده ضمن الشهر الحالي.

تمرين 2: Payroll

- قم بتعديل نظام الدفع كما يلي:
- أضف الصف PieceWorker المشتق من الصف Employee لتمثيل الموظفين الذين يعملون على القطعة. يحوي هذا الصف عدد القطع pieces وأجرة القطعة wage.
- قم بكتابة الطريقة ()Earning لهذا الصف الجديد.
- قم بإنشاء مصفوفة من الصف Employee وضع في عناصرها موظفين من مختلف الصفوف المشتقة من صف الموظف Employee. ثم احسب مجموع المبالغ المترتبة لهم.

الفصل السادس
الواجهات Interfaces.

عنوان الموضوع:

الواجهات Interfaces.

الكلمات المفتاحية:

الواجهات Interfaces.

ملخص:

نتعرض في هذا الفصل لآليات استخدام الواجهات.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- استخدام الواجهات Interfaces.

المخطط:

الواجهات Interfaces

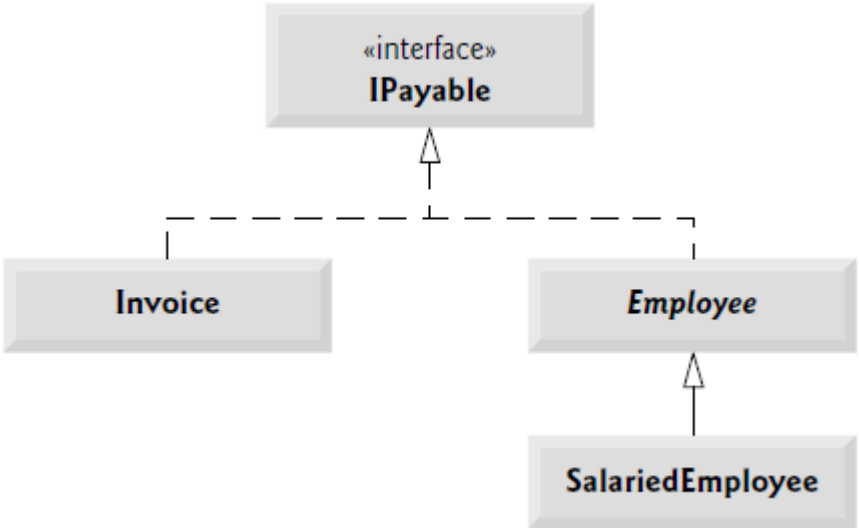
- 1 وحدة (Learning Objects)

1- الواجهات Interfaces

- يُتيح استخدام الواجهة توصيف مجموعة من الخصائص والطرق التي تتوفر في جميع الأغراض من هذه الواجهة.
- تحوي الواجهة حصراً أعضاء مجردة abstract.
- تُستخدم الواجهة بشكل عام لتكون الصف الأساسي لمجموعة من الصفوف المختلفة عن بعضها بشكل عام إلا أن لها بعض الأعضاء المشتركة والتي تختلف في تنفيذها من صف لصف آخر.
- لتوضيح مفهوم الواجهة، سنقوم فيما يلي بتعريف الواجهة `IPayable` والتي تُعبّر عن أي شيء له مبلغ مستحق للدفع. يكون لهذه الواجهة الطريقة المجردة `GetPaymentAmount()` لتحديد المبلغ الواجب دفعه.

```
// IPayable.cs
// IPayable interface declaration.
public interface IPayable
{
   decimal GetPaymentAmount(); // calculate payment; no implementation
} // end interface IPayable
```

- يُمكن أن تكون هذه الواجهة الأساس لصف مثل صف الفاتورة (الذي يحوي المبلغ الواجب دفعه) وصف مثل صف الموظف (والذي يحوي المبلغ الواجب دفعه للموظف):



صف الفاتورة Invoice

- يكون صف الفاتورة Invoice مشتق من الواجهة IPayable.
- يحوي صف الفاتورة مجموعة من الخصائص: الكمية Quantity، سعر الوحدة PricePerItem، رقم القطعة PartNumber، وصف القطعة PartDescription.
- يقوم الصف Invoice بكتابة الطريقة المطلوبة GetPaymentAmount() من الواجهة IPayable.

```
// Invoice.cs
// Invoice class implements IPayable.
using System;

public class Invoice : IPayable
{
   private int quantity;
   private decimal pricePerItem;

   // property that gets and sets the part number on the invoice
   public string PartNumber { get; set; }

   // property that gets and sets the part description on the invoice
   public string PartDescription { get; set; }

   // four-parameter constructor
   public Invoice( string part, string description, int count,
      decimal price )
   {
      PartNumber = part;
      PartDescription = description;
      Quantity = count; // validate quantity via property
      PricePerItem = price; // validate price per item via property
   } // end four-parameter Invoice constructor

   // property that gets and sets the quantity on the invoice
   public int Quantity
   {
      get
      {
         return quantity;
      } // end get
      set
      {
         if ( value >= 0 ) // validate quantity
            quantity = value;
         else
            throw new ArgumentOutOfRangeException( "Quantity",
               value, "Quantity must be >= 0" );
      } // end set
   } // end property Quantity

   // property that gets and sets the price per item
   public decimal PricePerItem
   {
      get
      {
         return pricePerItem;
```

```
      } // end get
      set
      {
         if ( value >= 0 ) // validate price
            pricePerItem = value;
         else
            throw new ArgumentOutOfRangeException( "PricePerItem",
               value, "PricePerItem must be >= 0" );
      } // end set
   } // end property PricePerItem

   // return string representation of Invoice object
   public override string ToString()
   {
      return string.Format(
         "{0}: \n{1}: {2} ({3}) \n{4}: {5} \n{6}: {7:C}",
         "invoice", "part number", PartNumber, PartDescription,
         "quantity", Quantity, "price per item", PricePerItem );
   } // end method ToString

   // method required to carry out contract with interface IPayable
   public decimal GetPaymentAmount()
   {
      return Quantity * PricePerItem; // calculate total cost
   } // end method GetPaymentAmount
} // end class Invoice
```

صف الموظف Employee

- يكون صف الموظف المجرد `Employee` مشتق من الواجهة `IPayable`.
- يحوي صف الموظف المجرد مجموعة الخصائص المشتركة للموظفين: الاسم الأول، الاسم الأخير، رقم التأمينات الاجتماعية.
- لا يقوم الصف المجرد في حالتنا بكتابة كود الطريقة المطلوبة `GetPaymentAmount()` من الواجهة `IPayable`. يقبل المترجم في هذه الحالة بالتصريح فقط أن هذه الطريقة مجردة `abstract`.

```
// Employee.cs
// Employee abstract base class.
public abstract class Employee : IPayable
{
   // read-only property that gets employee's first name
   public string FirstName { get; private set; }

   // read-only property that gets employee's last name
   public string LastName { get; private set; }

   // read-only property that gets employee's social security number
   public string SocialSecurityNumber { get; private set; }

   // three-parameter constructor
   public Employee( string first, string last, string ssn )
   {
      FirstName = first;
      LastName = last;
      SocialSecurityNumber = ssn;
```

```
   } // end three-parameter Employee constructor

   // return string representation of Employee object
   public override string ToString()
   {
      return string.Format( "{0} {1}\nsocial security number: {2}",
         FirstName, LastName, SocialSecurityNumber );
   } // end method ToString

   // Note: We do not implement IPayable method GetPaymentAmount here so
   // this class must be declared abstract to avoid a compilation error.
   public abstract decimal GetPaymentAmount();
} // end abstract class Employee
```

صف الموظف بمعاش SalariedEmployee

- يكون صف الموظف بمعاش SalariedEmployee مشتق من الصف المجرد Employee.
- يحوي هذا الصف خاصية المعاش WeeklySalary.
- نقوم في هذا الصف بركوب override الطريقة المجردة GetPaymentAmount() المصرح عنها في الصف المجرد الأساسي وكتابة كودها.

```
// SalariedEmployee.cs
// SalariedEmployee class that extends Employee.
using System;

public class SalariedEmployee : Employee
{
   private decimal weeklySalary;

   // four-parameter constructor
   public SalariedEmployee( string first, string last, string ssn,
      decimal salary ) : base( first, last, ssn )
   {
      WeeklySalary = salary; // validate salary via property
   } // end four-parameter SalariedEmployee constructor

   // property that gets and sets salaried employee's salary
   public decimal WeeklySalary
   {
      get
      {
         return weeklySalary;
      } // end get
      set
      {
         if ( value >= 0 ) // validation
            weeklySalary = value;
         else
            throw new ArgumentOutOfRangeException( "WeeklySalary",
               value, "WeeklySalary must be >= 0" );
      } // end set
   } // end property WeeklySalary

   // calculate earnings; implement interface IPayable method
```

```
   // that was abstract in base class Employee
   public override decimal GetPaymentAmount()
   {
      return WeeklySalary;
   } // end method GetPaymentAmount

   // return string representation of SalariedEmployee object
   public override string ToString()
   {
      return string.Format( "salaried employee: {0}\n{1}: {2:C}",
         base.ToString(), "weekly salary", WeeklySalary );
   } // end method ToString
} // end class SalariedEmployee
```

- نقوم فيما يلي باستخدام الواجهة والصفوف السابقة:
- نقوم بالتصريح عن مصفوفة من أربعة عناصر من نمط الواجهة IPayable.
- نقوم بإسناد غرضين من الصف Invoice لأول عنصرين من المصفوفة.
- نقوم بإسناد غرضين من الصف SalariedEmployee للعنصر الثالث والرابع من المصفوفة.
- نقوم بعدها بالدوران على عناصر المصفوفة واستدعاء الطريقة GetPaymentAmount() على كل من هذه العناصر.
- بالطبع، تؤدي ميزة تعدد الأشكال إلى تنفيذ الطريقة GetPaymentAmount() المُعرّفة في الصف Invoice عندما يكون العنصر الموافق من هذا الصف. وتنفيذ الطريقة GetPaymentAmount() المُعرّفة في الصف SalariedEmployee عندما يكون العنصر الموافق من الصف SalariedEmployee.

```
//  PayableInterfaceTest.cs
// Tests interface IPayable with disparate classes.
using System;

public class PayableInterfaceTest
{
   public static void Main( string[] args )
   {
      // create four-element IPayable array
      IPayable[] payableObjects = new IPayable[ 4 ];

      // populate array with objects that implement IPayable
      payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00M );
      payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95M );
      payableObjects[ 2 ] = new SalariedEmployee( "John", "Smith",
         "111-11-1111", 800.00M );
      payableObjects[ 3 ] = new SalariedEmployee( "Lisa", "Barnes",
         "888-88-8888", 1200.00M );

      Console.WriteLine(
         "Invoices and Employees processed polymorphically:\n" );
```

```
      // generically process each element in array payableObjects
      foreach ( var currentPayable in payableObjects )
      {
         // output currentPayable and its appropriate payment amount
         Console.WriteLine( "{0}\npayment due: {1:C}\n",
            currentPayable, currentPayable.GetPaymentAmount() );
      } // end foreach
   } // end Main
} // end class PayableInterfaceTest
```

- يكون ناتج التنفيذ:

```
Invoices and Employees processed polymorphically:

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
payment due: $800.00

salaried employee: Lisa Barnes
social security number: 888-88-8888
weekly salary: $1,200.00
payment due: $1,200.00

Press any key to continue . . .
```

2- اقتراحات وتمارين

تمرين:

- قم بتعريف الواجهة ICarbonFootprint والتي تحوي الطريقة المجردة GetCarbonFootprint لحساب معدل الكربون المنبعث.
- تُستخدم هذه الواجهة مع صفوف مختلفة تتميز أغراضها بأنها تقوم بتلويث الهواء بغاز الكربون.
- استخدم هذه الواجهة لكل من الصفوف بناء Building، سيارة Car، سفينة Boat.
- قم بكتابة الطريقة GetCarbonFootprint لكل من هذه الصفوف (حسابات بسيطة على بعض الخصائص مثل معدل استهلاك الوقود).

الفصل السابع
التحميل الزائد للعمليات Operator Overloading.

عنوان الموضوع:

التحميل الزائد للعمليات Operator Overloading.

الكلمات المفتاحية:

التحميل الزائد للعمليات Operator Overloading.

ملخص:

نعرض في هذا الفصل إمكانية التصريح عن العمليات الأساسية على أغراض صف.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- كيفية تعريف العمليات الأساسية على أغراض الصفوف.

المخطط:

التحميل الزائد للعمليات Operator Overloading

- 2 وحدة (Learning Objects)

1- التحميل الزائد للعمليات (1) Operator Overloading

- يُمكن استخدام العمليات الأساسية التي توفرها لغة C# (مثل +، -، *، /) للتعامل بين أغراض صف. ندعو هذه التقانة بالتحميل الزائد للعمليات.
- تسمح هذه التقانة بالتعامل في كثير من الحالات بشكل "طبيعي" بين أغراض الصف.
- نقوم مثلاً فيما يلي بتعريف صف الأعداد العقدية ComplexNumber ومن ثم تعريف العمليات الأساسية على الأعداد العقدية (الجمع، الطرح، الضرب).
- يحوي صف الأعداد العقدية الخاصية Real لتمثيل القسم الحقيقي للعدد العقدي، والخاصية Imaginary لتمثيل القسم التخيلي للعدد العقدي.
- يكون التصريح عن عملية الجمع مثلاً بكتابة الطريقة الساكنة operator+.

```
// ComplexNumber.cs
// Class that overloads operators for adding, subtracting
// and multiplying complex numbers.
using System;

public class ComplexNumber
{
   // read-only property that gets the real component
   public double Real { get; private set; }

   // read-only property that gets the imaginary component
   public double Imaginary { get; private set; }

   // constructor
   public ComplexNumber( double a, double b )
   {
      Real = a;
      Imaginary = b;
   } // end constructor

   // return string representation of ComplexNumber
   public override string ToString()
   {
      return string.Format( "({0} {1} {2}i)",
         Real, ( Imaginary < 0 ? "-" : "+" ), Math.Abs( Imaginary ) );
   } // end method ToString

   // overload the addition operator
   public static ComplexNumber operator+ (ComplexNumber x, ComplexNumber y )
   {
      return new ComplexNumber( x.Real + y.Real, x.Imaginary + y.Imaginary );
   } // end operator +

   // overload the subtraction operator
   public static ComplexNumber operator- (ComplexNumber x, ComplexNumber y )
   {
      return new ComplexNumber( x.Real - y.Real, x.Imaginary - y.Imaginary );
   } // end operator -

   // overload the multiplication operator
   public static ComplexNumber operator* (ComplexNumber x, ComplexNumber y )
```

```
   {
      return new ComplexNumber(x.Real * y.Real - x.Imaginary * y.Imaginary,
         x.Real * y.Imaginary + y.Real * x.Imaginary );
   } // end operator *
} // end class ComplexNumber
```

- نقوم فيما يلي بالتصريح عن العددين العقديين x و y. ثم نقوم بتنفيذ عمليات الجمع والطرح والضرب عليهما وإظهار النتائج:

```
// ComplexTest.cs
// Overloading operators for complex numbers.
using System;

public class ComplexTest
{
   public static void Main( string[] args )
   {
      // declare two variables to store complex numbers
      // to be entered by user
      ComplexNumber x, y;

      // prompt the user to enter the first complex number
      Console.Write( "Enter the real part of complex number x: " );
      double realPart = Convert.ToDouble( Console.ReadLine() );
      Console.Write("Enter the imaginary part of complex number x: " );
      double imaginaryPart = Convert.ToDouble( Console.ReadLine() );
      x = new ComplexNumber( realPart, imaginaryPart );

      // prompt the user to enter the second complex number
      Console.Write( "\nEnter the real part of complex number y: " );
      realPart = Convert.ToDouble( Console.ReadLine() );
      Console.Write("Enter the imaginary part of complex number y: " );
      imaginaryPart = Convert.ToDouble( Console.ReadLine() );
      y = new ComplexNumber( realPart, imaginaryPart );

      // display the results of calculations with x and y
      Console.WriteLine();
      Console.WriteLine( "{0} + {1} = {2}", x, y, x + y );
      Console.WriteLine( "{0} - {1} = {2}", x, y, x - y );
      Console.WriteLine( "{0} * {1} = {2}", x, y, x * y );
   } // end method Main
} // end class ComplexTest
```

- يكون ناتج التنفيذ:

```
Enter the real part of complex number x: 2
Enter the imaginary part of complex number x: 4

Enter the real part of complex number y: 4
Enter the imaginary part of complex number y: -2

(2 + 4i) + (4 - 2i) = (6 + 2i)
(2 + 4i) - (4 - 2i) = (-2 + 6i)
(2 + 4i) * (4 - 2i) = (16 + 12i)
Press any key to continue . . .
```

2- التحميل الزائد للعمليات (2) Operator Overloading

- يحوي الصف `Fraction` الحقلين البسط `numerator` والمقام `denominator`.
- يكون للصف باني نُمرر له البسط والمقام. وباني آخر نُمرر له البسط فقط فيقوم بإسناد القيمة 1 إلى المقام.
- نقوم بالتحميل الزائد لعملية الجمع +.
- نُصرح عن التحويل الضمني (`implicit`) بين العدد الطبيعي والكسر (التصريح ضمني لأن كل عدد طبيعي هو كسر مقامه 1).
- نُصرح عن التحويل الظاهر (`explicit`) بين الكسر والعدد الطبيعي (نُعيد حاصل قسمة البسط على المقام).
- نقوم بالتحميل الزائد لعملية اختبار المساواة == ولاختبار عدم التساوي=!.

```
using System;
public class Fraction
{
private int numerator;
private int denominator;

public Fraction(int numerator, int denominator)
	{
		this.numerator=numerator;
		this.denominator=denominator;
	}
public Fraction(int wholeNumber)
	{
		numerator = wholeNumber;
		denominator = 1;
	}
public static Fraction operator +(Fraction lhs, Fraction rhs)
```

```
    {
        if (lhs.denominator == rhs.denominator)
        {
            return new Fraction(lhs.numerator + rhs.numerator,
                        lhs.denominator);
        }

        // simplistic solution for unlike fractions
        // 1/2 + 3/4 == (1*4) + (3*2) / (2*4) == 10/8
        int firstProduct = lhs.numerator * rhs.denominator;
        int secondProduct = rhs.numerator * lhs.denominator;
        return new Fraction(
            firstProduct + secondProduct,
            lhs.denominator * rhs.denominator
            );

    }
public static implicit operator Fraction(int theInt)
       {
       return new Fraction(theInt);
       }

public static explicit operator int(Fraction theFraction)
       {
       return theFraction.numerator / theFraction.denominator;
       }
public static bool operator==(Fraction lhs, Fraction rhs)
       {
              if (lhs.denominator == rhs.denominator &&
                                    lhs.numerator == rhs.numerator)
              {
                     return true;
              }
              // code here to handle unlike fractions
              return false;
       }

public static bool operator !=(Fraction lhs, Fraction rhs)
       {
              return !(lhs==rhs);
       }
public override bool Equals(object o)
       {
              if (! (o is Fraction) )
              {
                     return false;
              }
              return this == (Fraction) o;
       }

  public override string ToString()
  {
    string s = numerator.ToString() + "/" +
        denominator.ToString();
    return s;
  }
}
```

- نقوم فيما يلي باستخدام الصف السابق:

```
using System;
```

```
public class FractionTest
{
   public static void Main( string[] args )
   {
       Fraction f1 = new Fraction(3, 4);
       Console.WriteLine("f1: {0}", f1.ToString());
       Fraction f2 = new Fraction(2, 4);
       Console.WriteLine("f2: {0}", f2.ToString());

       Fraction f3 = f1 + f2;
       Console.WriteLine("f1 + f2 = f3: {0}", f3.ToString());
       Fraction f4 = f3 + 5;
       Console.WriteLine("f3 + 5 = f4: {0}", f4.ToString());
       Fraction f5 = new Fraction(2, 4);
       if (f5 == f2)
       {
           Console.WriteLine("F5: {0} == F2: {1}",
               f5.ToString(),
               f2.ToString());
       }
       int i = (int)f5;
       Console.WriteLine("i={0}", i);
   } // end method Main
}
```

- تكون نتيجة التنفيذ:

```
f1: 3/4
f2: 2/4
f1 + f2 = f3: 5/4
f3 + 5 = f4: 25/4
F5: 2/4 == F2: 2/4
i=0
Press any key to continue . . .
```

3- اقتراحات وتمارين

تمرين:

- قم بتعديل صف الكسور `Fraction` السابق لإضافة جميع العمليات عليه:

-, *, /, >, <, >=, <=

- قم باستخدام الصف السابق في أمثلة مختلفة واختبر النتائج.

الفصل الثامن

الاستثناءات Exceptions

عنوان الموضوع:

الاستثناءات Exceptions.

الكلمات المفتاحية:

الاستثناءات، التقاط ومعالجة الاستثناءات، تعريف صف استثناء مخصص.

ملخص:

نستعرض في هذا الفصل كيفية معالجة الاستثناءات التي يُمكن أن تظهر خلال تنفيذ البرامج. كما نتعرض إلى آلية التصريح عن صفوف استثناءات جديدة مخصصة.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- معالجة الاستثناءات.
- تعريف صفوف الاستثناءات المخصصة

المخطط:

الاستثناءات

- 2 وحدة (Learning Objects)

مقدمة

- تُطلق الاستثناءات عند حصول مشكلة غير متوقعة أثناء تنفيذ البرنامج. تسمح معالجة الاستثناءات بكتابة تطبيقات تستمر في عملها بعد حدوث الاستثناءات مما يجعل هذه التطبيقات أكثر مرونة.
- يُبين المثال التالي السلوك الافتراضي في حال عدم التقاط ومعالجة الاستثناءات.
- يُمكن في المثال التالي حصول استثناء من النوع قسمة على صفر في حال قيام المستخدم بإدخال قيمة الصفر للعدد المقسوم عليه:

```
// DivideByZeroNoExceptionHandling.cs
// Integer division without exception handling.
using System;

class DivideByZeroNoExceptionHandling
{
   static void Main()
   {
      // get numerator
      Console.Write( "Please enter an integer numerator: " );
      int numerator = Convert.ToInt32( Console.ReadLine() );

      // get denominator
      Console.Write( "Please enter an integer denominator: " );
      int denominator = Convert.ToInt32( Console.ReadLine() );

      // divide the two integers, then display the result
      int result = numerator / denominator;
      Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
         numerator, denominator, result );
   } // end Main
} // end class DivideByZeroNoExceptionHandling
```

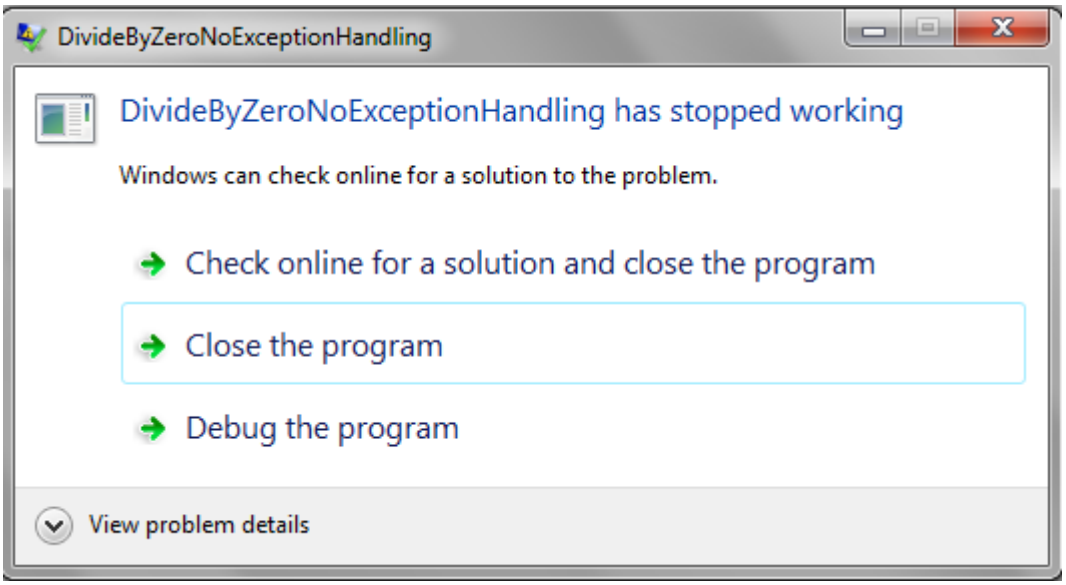
- يُعطي تنفيذ البرنامج مثلاً:

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

- أما في حال قيام المستخدم بإدخال قيمة مساوية للصفر للعدد المقسوم عليه، فسيتم إظهار الاستثناء:

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
   at DivideByZeroNoExceptionHandling.Main() in
e:\5.OOP\Chapter_8\DivideByZeroN
oExceptionHandling\DivideByZeroNoExceptionHandling\DivideByZeroNoException
Handli
ng.cs:line 18
```

- كما يتوقف تنفيذ البرنامج وتظهر النافذة التالية:



1- التقاط الاستثناءات

- يُبين المثال التالي التقاط استثناء القسمة على صفر DivideByZeroException واستثناء التنسيق FormatException.

- عند حصول أي مشكلة أثناء التنفيذ في كتلة `try`، يتم انتقال التحكم إلى أحد الكتلتين `catch` وذلك حسب نوع الاستثناء المُنطلق.
- يتم في الكتلة `catch` في المثال التالي إظهار رسالة الخطأ الموافقة للمستخدم. ويتم متابعة التنفيذ وبدون توقف البرنامج:

```
// DivideByZeroExceptionHandling.cs
// FormatException and DivideByZeroException handlers.
using System;

class DivideByZeroExceptionHandling
{
   static void Main( string[] args )
   {
      bool continueLoop = true; // determines whether to keep looping

      do
      {
         // retrieve user input and calculate quotient
         try
         {
            // Convert.ToInt32 generates FormatException
            // if argument cannot be converted to an integer
            Console.Write( "Enter an integer numerator: " );
            int numerator = Convert.ToInt32( Console.ReadLine() );
            Console.Write( "Enter an integer denominator: " );
            int denominator = Convert.ToInt32( Console.ReadLine() );

            // division generates DivideByZeroException
            // if denominator is 0
            int result = numerator / denominator;

            // display result
            Console.WriteLine( "\nResult: {0} / {1} = {2}",
               numerator, denominator, result );
            continueLoop = false;
         } // end try
         catch ( FormatException formatException )
         {
            Console.WriteLine( "\n" + formatException.Message );
            Console.WriteLine(
               "You must enter two integers. Please try again.\n" );
         } // end catch
         catch ( DivideByZeroException divideByZeroException )
         {
            Console.WriteLine( "\n" + divideByZeroException.Message );
            Console.WriteLine(
               "Zero is an invalid denominator. Please try again.\n" );
         } // end catch
      } while ( continueLoop ); // end do...while
   } // end Main
} // end class DivideByZeroExceptionHandling
```

- يُمكن أن يكون التنفيذ بدون استثناءات:

```
Enter an integer numerator: 100
Enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

- في حال قيام المستخدم بإدخال قيمة الصفر للمقسوم عليه، سيتم إطلاق استثناء القسمة على صفر `DivideByZeroException` ومعالجته في الكتلة `catch` الموافقة:

```
Enter an integer numerator: 100
Enter an integer denominator: 0

Attempted to divide by zero.
Zero is an invalid denominator. Please try again.

Enter an integer numerator: 100
Enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

- في حال قيام المستخدم مثلاً بإدخال قيمة نصية عوضاً عن قيمة رقمية للمقسوم عليه، سيتم إطلاق استثناء التنسيق `FormatException` ومعالجته في الكتلة `catch` الموافقة:

```
Enter an integer numerator: 100
Enter an integer denominator: Hello

Input string was not in a correct format.
You must enter two integers. Please try again.

Enter an integer numerator: 100
Enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

صفوف الاستثناءات المخصصة

- يُمكن في الكثير من الحالات استخدام الاستثناءات المُعرّفة مسبقاً. كما يُمكن التصريح عن صفوف استثناءات جديدة مخصصة User-defined exception classes.
- يجب أن تُشتق هذه الصفوف من الصف `Exception` من فضاء الأسماء System.
- نقوم في المثال التالي بتعريف صف الاستثناء المخصص `NegativeNumberException` لاستخدامه لإظهار استثناء قيمة سالبة حين الحاجة:

```
// NegativeNumberException.cs
// NegativeNumberException represents exceptions caused by
// illegal operations performed on negative numbers.
using System;

class NegativeNumberException : Exception
{
   // default constructor
   public NegativeNumberException()
      : base( "Illegal operation for a negative number" )
   {
      // empty body
   } // end default constructor

   // constructor for customizing error message
   public NegativeNumberException( string messageValue ) : base( messageValue )
   {
      // empty body
   } // end one-argument constructor

   // constructor for customizing the exception's error
   // message and specifying the InnerException object
   public NegativeNumberException( string messageValue, Exception inner )
      : base( messageValue, inner )
   {
      // empty body
   } // end two-argument constructor
} // end class NegativeNumberException
```

- نستخدم في المثال التالي صف الاستثناء السابق:

```
// SquareRootTest.cs
// Demonstrating a user-defined exception class.
using System;

class SquareRootTest
{
   static void Main( string[] args )
   {
      bool continueLoop = true;

      do
      {
         // catch any NegativeNumberException thrown
         try
         {
            Console.Write( "Enter a value to calculate the square root of: " );
            double inputValue = Convert.ToDouble( Console.ReadLine() );
            double result = SquareRoot( inputValue );

            Console.WriteLine( "The square root of {0} is {1:F6}\n",
               inputValue, result );
            continueLoop = false;
         } // end try
         catch ( FormatException formatException )
         {
            Console.WriteLine( "\n" + formatException.Message );
            Console.WriteLine( "Please enter a double value.\n" );
         } // end catch
         catch ( NegativeNumberException negativeNumberException )
         {
            Console.WriteLine( "\n" + negativeNumberException.Message );
```

```
            Console.WriteLine( "Please enter a non-negative value.\n" );
         } // end catch
      } while ( continueLoop );
   } // end Main

   // computes square root of parameter; throws
   // NegativeNumberException if parameter is negative
   public static double SquareRoot( double value )
   {
      // if negative operand, throw NegativeNumberException
      if ( value < 0 )
         throw new NegativeNumberException(
            "Square root of negative number not permitted" );
      else
         return Math.Sqrt( value ); // compute square root
   } // end method SquareRoot
} // end class SquareRootTest
```

- يكون تنفيذ البرنامج السابق عند إدخال قيمة موجبة:

```
Enter a value to calculate the square root of: 30
The square root of 30 is 5.477226

Press any key to continue . . .
```

- أما عند إدخال سلسلة نصية، فيتم التقاط استثناء التنسيق الخاطئ `FormatException`:

```
Enter a value to calculate the square root of: Hello

Input string was not in a correct format.
Please enter a double value.

Enter a value to calculate the square root of: 25
The square root of 25 is 5.000000

Press any key to continue . . .
```

- وعند إدخال قيمة سالبة، سيتم التقاط استثناء القيمة السالبة `NegativeNumberException`:

```
Enter a value to calculate the square root of: -2

Square root of negative number not permitted
Please enter a non-negative value.

Enter a value to calculate the square root of: 2
The square root of 2 is 1.414214

Press any key to continue . . .
```

2- صفوف الاستثناءات المخصصة User-Defined Exception Classes

- يُمكن في الكثير من الحالات استخدام الاستثناءات المُعرّفة مسبقاً. كما يُمكن التصريح عن صفوف استثناءات جديدة مخصصة User-defined exception classes.
- يجب أن تُشتق هذه الصفوف من الصف `Exception` من فضاء الأسماء System.
- نقوم في المثال التالي بتعريف صف الاستثناء المخصص `NegativeNumberException` لاستخدامه لإظهار استثناء قيمة سالبة حين الحاجة:

```
// NegativeNumberException.cs
// NegativeNumberException represents exceptions caused by
// illegal operations performed on negative numbers.
using System;

class NegativeNumberException : Exception
{
   // default constructor
   public NegativeNumberException()
      : base( "Illegal operation for a negative number" )
   {
      // empty body
   } // end default constructor

   // constructor for customizing error message
   public NegativeNumberException( string messageValue ) : base( messageValue )
   {
      // empty body
   } // end one-argument constructor

   // constructor for customizing the exception's error
   // message and specifying the InnerException object
   public NegativeNumberException( string messageValue, Exception inner )
      : base( messageValue, inner )
   {
      // empty body
   } // end two-argument constructor
} // end class NegativeNumberException
```

- نستخدم في المثال التالي صف الاستثناء السابق:

```
// SquareRootTest.cs
// Demonstrating a user-defined exception class.
using System;

class SquareRootTest
{
   static void Main( string[] args )
   {
      bool continueLoop = true;

      do
      {
         // catch any NegativeNumberException thrown
         try
         {
```

```
            Console.Write( "Enter a value to calculate the square root of: " );
            double inputValue = Convert.ToDouble( Console.ReadLine() );
            double result = SquareRoot( inputValue );

            Console.WriteLine( "The square root of {0} is {1:F6}\n",
               inputValue, result );
            continueLoop = false;
         } // end try
         catch ( FormatException formatException )
         {
            Console.WriteLine( "\n" + formatException.Message );
            Console.WriteLine( "Please enter a double value.\n" );
         } // end catch
         catch ( NegativeNumberException negativeNumberException )
         {
            Console.WriteLine( "\n" + negativeNumberException.Message );
            Console.WriteLine( "Please enter a non-negative value.\n" );
         } // end catch
      } while ( continueLoop );
   } // end Main

   // computes square root of parameter; throws
   // NegativeNumberException if parameter is negative
   public static double SquareRoot( double value )
   {
      // if negative operand, throw NegativeNumberException
      if ( value < 0 )
         throw new NegativeNumberException(
            "Square root of negative number not permitted" );
      else
         return Math.Sqrt( value ); // compute square root
   } // end method SquareRoot
} // end class SquareRootTest
```

- يكون تنفيذ البرنامج السابق عند إدخال قيمة موجبة:

```
Enter a value to calculate the square root of: 30
The square root of 30 is 5.477226

Press any key to continue . . .
```

- أما عند إدخال سلسلة نصية، فيتم التقاط استثناء التنسيق الخاطئ `FormatException`:

```
Enter a value to calculate the square root of: Hello

Input string was not in a correct format.
Please enter a double value.

Enter a value to calculate the square root of: 25
The square root of 25 is 5.000000

Press any key to continue . . .
```

- وعند إدخال قيمة سالبة، سيتم التقاط استثناء القيمة السالبة `NegativeNumberException`:

```
Enter a value to calculate the square root of: -2

Square root of negative number not permitted
Please enter a non-negative value.

Enter a value to calculate the square root of: 2
The square root of 2 is 1.414214

Press any key to continue . . .
```

3- اقتراحات وتمارين

تمرين:

- قم بكتابة برنامج يقوم من خلاله المستخدم بإدخال عدد الكيلومترات المقطوعة وعدد الليترات المصروفة ليحسب البرنامج الاستهلاك بالكيلو متر الواحد.
- يجب معالجة كل القيم المدخلة غير المقبولة من قبل المستخدم: قيم غير رقمية، قيم سالبة، القسمة على صفر.

الفصل التاسع
المُفهرس Indexer

عنوان الموضوع:

المُفهرس Indexer.

الكلمات المفتاحية:

المُفهرس Indexer.

ملخص:

نستعرض في هذا الفصل إنشاء واستخدام مُفهرس الصف.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- إنشاء مُفهرس.
- استخدام المُفهرس.

المخطط:

المُفهرس Indexer

- 2 وحدة (Learning Objects)

1- المُفهرس Indexer

- يسمح المُفهرس بالتعامل مع مجموعة (collection) في الصف كما لو كان الصف نفسه مصفوفة.
- يسمح المُفهرس إذاً بالتعامل مع مجموعة في الصف باستخدام الأقواس المتوسطة [ ] التي نستخدمها مع المصفوفات.
- يكون المُفهرس عبارة عن خاصية في الصف ولها الموصلين get و set.
- نقوم في المثال التعليمي التالي بالتصريح عن الصف صندوق القائمة ListBox.
- يُستخدم الغرض من هذا الصف لتخزين سلاسل نصية (بدون تحديد عددها مسبقاً) ومن ثم الوصول إليها عن طريق فهرسها.

```
using System;
namespace SquareRootTest
{
    public class ListBox
    {
        private string[] strings;
        private int ctr = 0;

        // initialize the list box with strings
        public ListBox(params string[] initialStrings)
        {
            // allocate space for the strings
            strings = new String[256];

            // copy the strings passed in to the constructor
            foreach (string s in initialStrings)
            {
                strings[ctr++] = s;
            }
        }
        // add a single string to the end of the list box
        public void Add(string theString)
        {
            if (ctr >= strings.Length)
            {
                Console.WriteLine("List Overflow !");
            }
            else
                strings[ctr++] = theString;
        }

        // allow array-like access
        public string this[int index]
        {
            get
            {
                if (index < 0 || index >= strings.Length)
                {
                    Console.WriteLine("Bad List Index !");
                }
                return strings[index];
```

```
            }
            set
            {
                // add only through the add method
                if (index < 0 || index >= ctr)
                {
                    Console.WriteLine("Bad List Index !");
                }
                else
                    strings[index] = value;
            }
        }
        // publish how many strings you hold
        public int GetNumEntries()
        { return ctr; }
    }
}
```

- لاحظ استخدام الشكل `this[int index]` للتصريح عن المُفهرس.
- نستخدم فيما يلي الصف السابق:

```
// IndexerTest.cs
using System;
namespace SquareRootTest
{
    class IndexerTest
    {
        static void Main(string[] args)
        {
            // create a new list box and initialize
            ListBox lbt = new ListBox("Hello", "World");

            // add a few strings
            lbt.Add("Who");
            lbt.Add("Is");
            lbt.Add("John");
            lbt.Add("Galt");

            // test the access
            string subst = "Universe";
            lbt[1] = subst;

            // access all the strings
            for (int i = 0; i < lbt.GetNumEntries(); i++)
            {
                Console.WriteLine("lbt[{0}]: {1}", i, lbt[i]);
            }

        } // end Main
    }
}
```

- يكون ناتج التنفيذ:

```
lbt[0]: Hello
lbt[1]: Universe
lbt[2]: Who
lbt[3]: Is
lbt[4]: John
lbt[5]: Galt
Press any key to continue . . .
```

2- مثال تعليمي

- بالعودة إلى مثال نظام دفع رواتب الموظفين في فصل سابق. نُضيف الصف شركة Company ونُصرح فيه عن مُفهرس:

```
using System;

    public class Company
    {
        private Employee[] AE;
        private int Max;

        public Company(int Max)
        {
            this.Max = Max;
            AE = new Employee[Max];

        }

        public Employee this[int index]
        {
            get
            {
                if (index < 0 || index >= Max)
                {
                    throw new IndexOutOfRangeException();
                }
                return AE[index];
            }
            set
            {
                if (index < 0 || index >= Max)
                {
                    throw new IndexOutOfRangeException();
                }
                AE[index] = value;
            }
        }
        public override string ToString()
        {
            string s = "";
            for (int i = 0; i < this.Max; i++)
                s = s + AE[i].ToString() + "\n";
            return s;
        }
    }
```

- نستخدم الصف السابق كما يلي:

```
// PayrollSystemTest.cs
using System;

public class PayrollSystemTest
{
   public static void Main( string[] args )
   {
      // create derived class objects
      SalariedEmployee salariedEmployee =
         new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00M );
      HourlyEmployee hourlyEmployee =
         new HourlyEmployee( "Karen", "Price",
         "222-22-2222", 16.75M, 40.0M );
      CommissionEmployee commissionEmployee =
         new CommissionEmployee( "Sue", "Jones",
         "333-33-3333", 10000.00M, .06M );
      BasePlusCommissionEmployee basePlusCommissionEmployee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "444-44-4444", 5000.00M, .04M, 300.00M );
      try
      {
          // create four-element Employee array
          Company C = new Company(4);

          // initialize array with Employees of derived types
          C[0] = salariedEmployee;
          C[1] = hourlyEmployee;
          C[2] = commissionEmployee;
          C[3] = basePlusCommissionEmployee;

          Console.WriteLine("Company Info:\n {0} ", C);
      }
      catch (Exception x)
      {
          Console.WriteLine(x.Message);
      }
   } // end Main
} // end class PayrollSystemTest
```

- يكون ناتج التنفيذ:

```
Company Info:
 salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00
```

```
commission rate: 0.06
base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00
commission rate: 0.04; base salary: $300.00

Press any key to continue . . .
```

3- اقتراحات وتمارين

تمرين:

- قم بإضافة الطريقة SortEmpBySal إلى الصف Company السابق لترتيب مصفوفة الموظفين تصاعدياً وفق معاشهم.

الفصل العاشر
بنى المعطيات

عنوان الموضوع:

بنى المعطيات.

الكلمات المفتاحية:

struct، Boxing and Unboxing، Self-Referential Classes.

ملخص:

نستعرض في هذا الفصل بعض المفاهيم الأساسية في بنى المعطيات.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- استخدام البنى struct.
- التحويل بين أنماط القيمة وأنماط المرجع.
- الصفوف مع مرجع لنفسها.

المخطط:

بنى المعطيات

- 3 وحدات (Learning Objects)

1- struct

- تُستخدم البنى struct عادةً لتمثيل أغراض "خفيفة" مثل النقطة Point واللون Color.
- مع أنه يُمكن التصريح عن صف النقطة Point، إلا أنه يُمكن في بعض الحالات أن يكون استخدام بنية أكثر فاعلية. فلو قمنا بالتصريح عن مصفوفة من 1000 غرض من الصف Point سيكون هنالك ذاكرة إضافية لتخزين المراجع لهذه الأغراض.
- تكون البنى أنماط قيمة Value Types بينما تكون الصفوف أنماط مرجع Reference Types.
- لا يُمكن التصريح في البنية عن باني ليس له معاملات.
- يُمكن إعطاء قيم ابتدائية لحقول البنية إما في باني له معاملات أو من خلال الوصول إلى الحقول بعد التصريح عن غرض من البنية.
- يُمكن عدم استخدام الكلمة المفتاحية new. يجب في هذه الحالة إعطاء قيم ابتدائية للحقول.
- كما هو الحال في الصفوف، يتم إعطاء قيم ابتدائية للحقول (0 للحقول الرقمية) في حال استخدام new مع الباني الافتراضي بدون معاملات.
- نقوم في المثال التالي بتعريف البنية CoOrds لتمثيل نقطة:

```
public struct CoOrds
    {
        public int x, y;

        public CoOrds(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }
```

- نستخدم في المثال التالي البنية السابقة:

```
public static void Main( string[] args )
 {
     // Initialize:
     CoOrds coords1 = new CoOrds();
     CoOrds coords2 = new CoOrds(10, 10);
     CoOrds coords3;
     coords3.x = 5;
     coords3.y = 6;

     // Display results:
     Console.Write("CoOrds 1: ");
     Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);

     Console.Write("CoOrds 2: ");
     Console.WriteLine("x = {0}, y = {1}", coords2.x, coords2.y);
```

```
        Console.Write("CoOrds 3: ");
        Console.WriteLine("x = {0}, y = {1}", coords3.x, coords3.y);

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();

    } // end Main
```

- يكون ناتج التنفيذ:

```
CoOrds 1: x = 0, y = 0
CoOrds 2: x = 10, y = 10
CoOrds 3: x = 5, y = 6
Press any key to exit.
```

2- بنى الأنماط البسيطة Simple Types structs

- يكون لكل نمط بسيط من أنماط لغة C# بنية (struct) موافقة له مُعرّفة في فضاء الأسماء System والتي تقوم بتعريف هذا النمط البسيط.
- تُدعى هذه البنى:

Boolean, Byte, SByte, Char, Decimal, Double, Single, Int16, UInt16 Int32, UInt32, Int64 and UInt64.

- تكون الأنماط المُعرّفة باستخدام الكلمة المفتاحية `struct` أنماط قيمة value type.
- يُمكن تعريف متغير من نمط بسيط إما باستخدام اسم البنية أو باستخدام المرادف المُعرّف له. مثلاً يُمكن استخدام اسم البنية Int32 أو المرادف لها int لتعريف متغير صحيح.
- تكون الطرق المرتبطة مع نمط بسيط مُعرّفة في البنية الموافقة. مثلاً تكون الطريقة Parse التي تقوم بتحويل سلسلة نصية string إلى عدد صحيح int موجودة في البنية Int32.

الصندقة وفك الصندقة Boxing and Unboxing

- ترث البنى والأنماط البسيطة من الصف ValueType الموجود في فضاء الأسماء System. يرث الصف ValueType من الصف object. وبهذا فإنه يُمكن إسناد أي قيمة من نمط بسيط

إلى متغير من النمط object. ندعو هذه العملية بالصنّدقة boxing conversion مما يسمح باستخدام الأنماط البسيطة في أي مكان نكون بحاجة للتعامل مع أغراض object. يتم في هذه العملية نسخ القيمة إلى غرض مما يسمح باستخدام القيمة كغرض. يُمكن القيام بهذه العملية بشكل ضمني أو بشكل صريح كما تُبين الأمثلة التالية:

```
int i = 5; // create an int value
object object1 = (object)i; // explicitly box the int value
object object2 = i; // implicitly box the int value
```

- بعد تنفيذ الكود السابق، يؤشر المتغيرين object1 وobject2 على غرضين مختلفين يحتويان قيمة المتغير i.
- يُمكن القيام بعملية فك الصندقة unboxing conversion للتحويل الصريح من مرجع كائن إلى قيمة بسيطة كما يُبين المثال التالي:

```
  int int1 = (int)object1; // explicitly unbox the int value
```

- في حال محاولة فك الصندقة عن مرجع كائن لا يؤشر إلى قيمة موافقة للنمط البسيط، سيتم ظهور الاستثناء InvalidCastException.

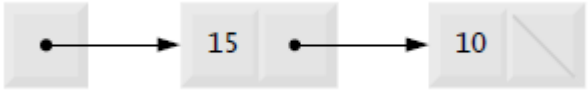
3- الصفوف مع مرجع لنفسها Self-Referential Classes

- يُمكن للصف أن يحوي عضو يقوم بالتأشير على غرض من نفس نمط الصف نفسه. ندعو هذا الصف بالصف ذو المرجع لنفسه self-referential class.
- يُعطي الصف التالي مثالاً عن صف يحوي مرجع لنفسه:

```
class Node
{
  public int Data { get; set; } // store integer data
  public Node Next { get; set; } // store reference to next Node

public Node( int dataValue )
 {
   Data = dataValue;
 } // end constructor
} // end class node
```

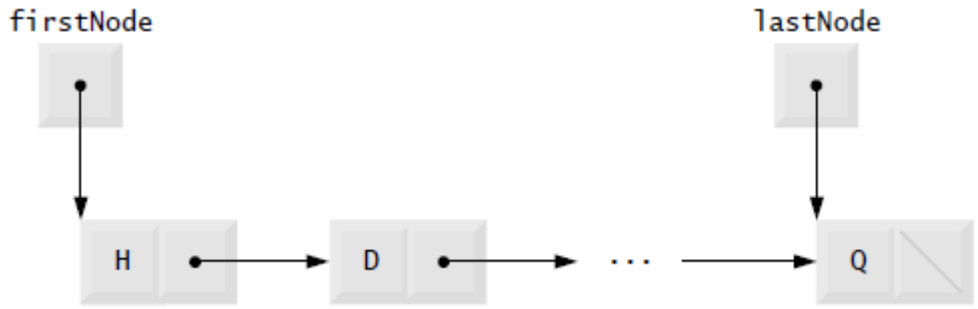
- يُمكن للأغراض أن ترتبط مع بعضها البعض مما يُشكّل بنى معطيات مفيدة مثل القائمة List، المكدّس Stack، الرتل Queue، الشجرة Tree.

- يُمثّل الشكل التالي مثلاً غرضين مرتبطين مع بعضهما مشكلين قائمة مرتبطة:



القوائم المرتبطة Linked Lists

- تتكون القائمة المرتبطة من مجموعة من الأغراض المرتبطة مع بعضها بشكل تسلسلي. ندعو عنصر القائمة بالعقدة Node.
- يتم التعامل مع القائمة عن طريق مرجع لأول عقدة فيها. كما نقوم عادةً باستخدام مرجع على آخر عقدة منها.
- نضع القيمة `null` في مرجع آخر عقدة للدلالة على انتهاء القائمة.
- يُمثّل الشكل التالي قائمة مرتبطة:



- نقوم فيما يلي بكتابة الكود اللازم للتعامل مع القوائم المرتبطة:

```
// LinkedListLibrary.cs
// ListNode, List and EmptyListException class declarations.
using System;
namespace LinkedListLibrary
{
// class to represent one node in a list
 // Self-referential Node class declaration.
 class ListNode
 {
 // automatic read-only property Data
     public object Data { get; private set; }
 // automatic property Next
     public ListNode Next { get; set; }
 // constructor to create ListNode that refers to dataValue
 // and is last node in list
 public ListNode( object dataValue )
 : this( dataValue, null )
 {
```

```
 } // end default constructor

 // constructor to create ListNode that refers to dataValue
 // and refers to next ListNode in List
 public ListNode( object dataValue, ListNode nextNode )
 {
 Data = dataValue;
 Next = nextNode;
 } // end constructor
 } // end class ListNode

// class List declaration
 public class List
 {
  private ListNode firstNode;
  private ListNode lastNode;
  private string name; // string like "list" to display
 // construct empty List with specified name
 public List( string listName )
 {
 name = listName;
 firstNode = lastNode = null;
 } // end constructor

 // construct empty List with "list" as its name
 public List()
 : this( "list" )
 {
 } // end default constructor

 // Insert object at front of List. If List is empty,
 // firstNode and lastNode will refer to same object.
 // Otherwise, firstNode refers to new node.
public void InsertAtFront( object insertItem )
 {
 if ( IsEmpty() )
     firstNode = lastNode = new ListNode( insertItem );
 else
     firstNode = new ListNode( insertItem, firstNode );
 } // end method InsertAtFront

 // Insert object at end of List. If List is empty,
 // firstNode and lastNode will refer to same object.
 // Otherwise, lastNode's Next property refers to new node.
 public void InsertAtBack( object insertItem )
 {
 if ( IsEmpty() )
     firstNode = lastNode = new ListNode( insertItem );
 else
     lastNode = lastNode.Next = new ListNode( insertItem );
 } // end method InsertAtBack

 // remove first node from List
 public object RemoveFromFront()
 {

  if ( IsEmpty() )  throw new EmptyListException( name );

  object removeItem = firstNode.Data; // retrieve data

// reset firstNode and lastNode references
 if ( firstNode == lastNode )
```

```
    firstNode = lastNode = null;
else
    firstNode = firstNode.Next;

   return removeItem; // return removed data
 } // end method RemoveFromFront

 // remove last node from List
 public object RemoveFromBack()
 {
  if ( IsEmpty() ) throw new EmptyListException( name );

  object removeItem = lastNode.Data; // retrieve data

 // reset firstNode and lastNode references
 if ( firstNode == lastNode )
   firstNode = lastNode = null;
 else
 {
  ListNode current = firstNode;

 // loop while current.Next is not lastNode
 while ( current.Next != lastNode )
   current = current.Next; // move to next node

 // current is new lastNode
     lastNode = current;
     current.Next = null;

 } // end else

 return removeItem; // return removed data
 } // end method RemoveFromBack

 // return true if List is empty
 public bool IsEmpty()
 {
   return firstNode == null;
 } // end method IsEmpty

 // output List contents
 public void Display()
 {
 if ( IsEmpty() )
 {
  Console.WriteLine( "Empty " + name );
 } // end if
 else
 {
  Console.Write( "The " + name + " is: " );

  ListNode current = firstNode;

 // output current node data while not at end of list
 while ( current != null )
 {
  Console.Write( current.Data + " " );
  current = current.Next;
 } // end while
  Console.WriteLine( "\n" );
 } // end else
 } // end method Display
```

```
} // end class List

// class EmptyListException declaration
public class EmptyListException : Exception
{
// parameterless constructor
public EmptyListException()
: base( "The list is empty" )
{
// empty constructor
} // end EmptyListException constructor

// one-parameter constructor
public EmptyListException( string name )
: base( "The " + name + " is empty" )
{
// empty constructor
} // end EmptyListException constructor

// two-parameter constructor
public EmptyListException( string exception, Exception inner )
: base( exception, inner )
{
// empty constructor
} // end EmptyListException constructor
} // end class EmptyListException
} // end namespace LinkedListLibrary
```

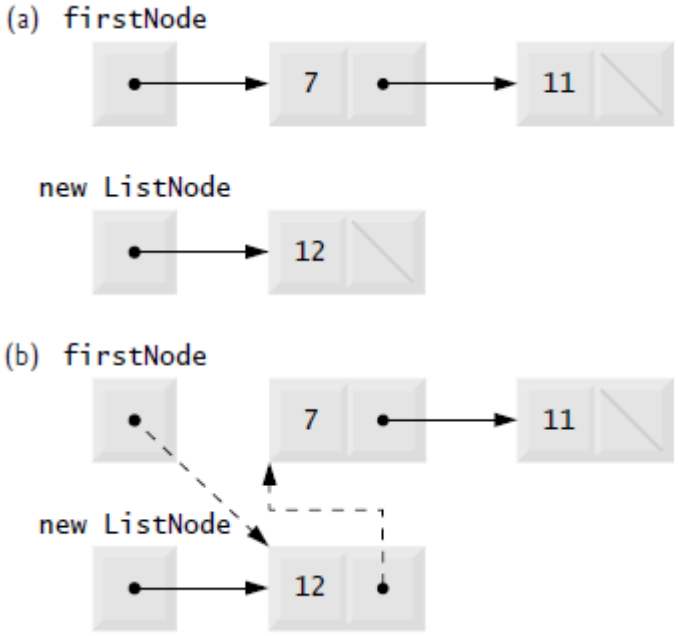
نشرح فيما يلي أهم الطرق في الكود السابق:

الطريقة InsertAtFront

تقوم هذه الطريقة بإنشاء غرض وجعل كل من مؤشر أول عنصر ومؤشر آخر عنصر يؤشران عليه إذا كانت القائمة فارغة. أما إذا كانت القائمة غير فارغة فيتم إنشاء غرض وجعله يؤشر على أول عنصر من القائمة ومن ثم إسناد مؤشر أول عنصر إلى هذا الغرض.

```
if ( IsEmpty() )
     firstNode = lastNode = new ListNode( insertItem );
 else
     firstNode = new ListNode( insertItem, firstNode );
```

يُبين الشكل التالي مثال على الإدراج في أول القائمة:

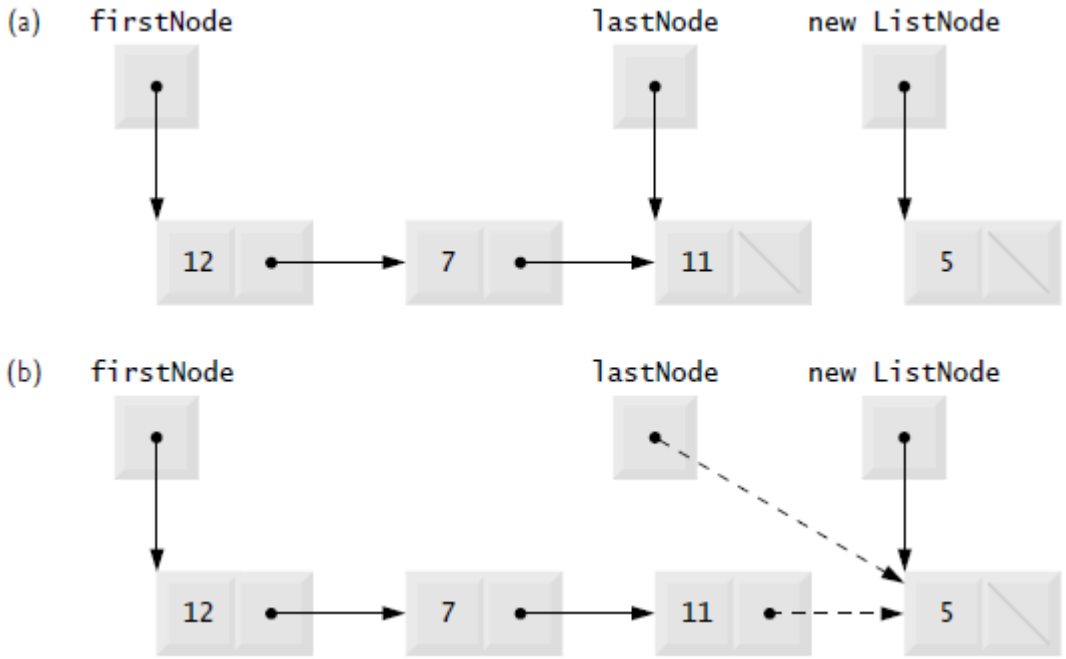


الطريقة InsertAtBack

تقوم هذه الطريقة بإنشاء غرض وجعل كل من مؤشر أول عنصر ومؤشر آخر عنصر يؤشران عليه إذا كانت القائمة فارغة. أما إذا كانت القائمة غير فارغة فيتم إنشاء غرض وجعله المؤشر التالي لآخر غرض يؤشر عليه ومن ثم جعل مؤشر آخر عنصر عليه.

```
if ( IsEmpty() )
     firstNode = lastNode = new ListNode( insertItem );
 else
     lastNode = lastNode.Next = new ListNode( insertItem );
```

يُبين الشكل التالي مثال على الإدراج في آخر القائمة:

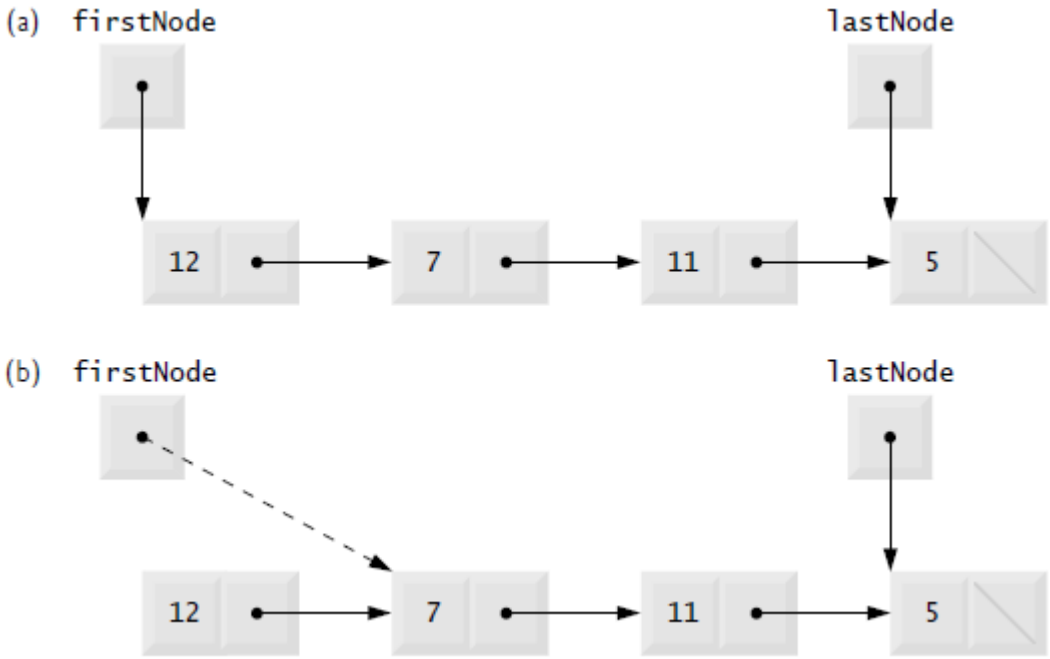


الطريقة RemoveFromFront

تقوم هذه الطريقة برفع استثناء إذا كانت القائمة فارغة. وإلا يتم حفظ قيمة أول عنصر لإعادته في نهاية الطريقة. إذا كانت الطريقة تحوي عنصر واحد، يتم وضع null في مؤشري بداية ونهاية القائمة. وإلا فيتم وضع مؤشر أول عنصر على العنصر الثاني.

```
if ( IsEmpty() )  throw new EmptyListException( name );

  object removeItem = firstNode.Data; // retrieve data

// reset firstNode and lastNode references
 if ( firstNode == lastNode )
    firstNode = lastNode = null;
else
    firstNode = firstNode.Next;

   return removeItem; // return removed data
```

يُبين الشكل التالي مثال على حذف أول عنصر من القائمة:

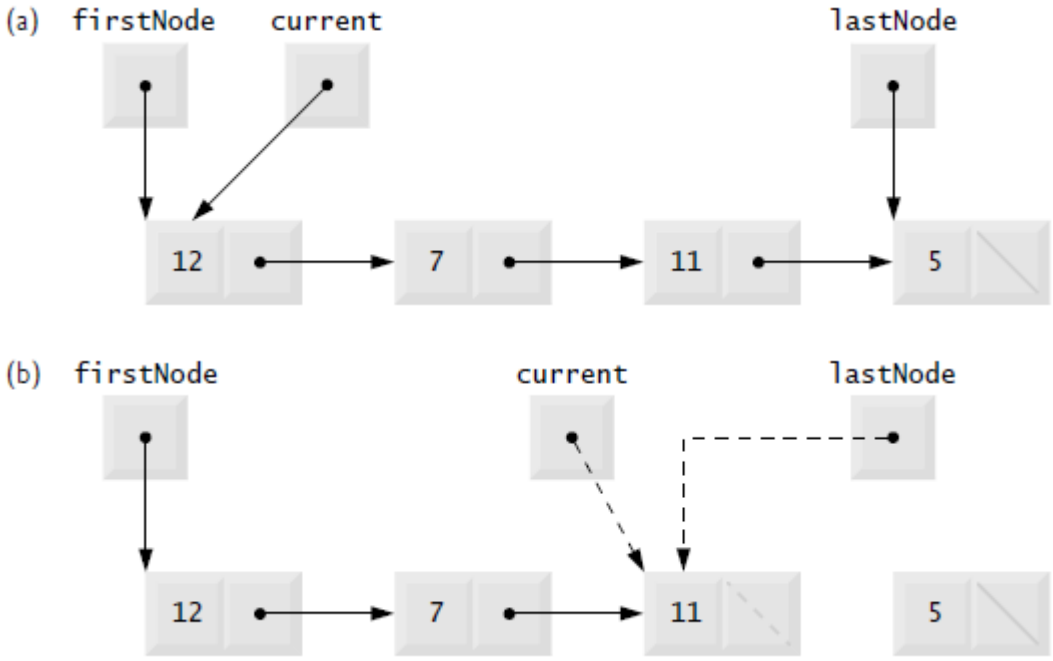


الطريقة RemoveFromBack

تقوم هذه الطريقة برفع استثناء إذا كانت القائمة فارغة. وإلا يتم حفظ قيمة آخر عنصر لإعادته في نهاية الطريقة. إذا كانت الطريقة تحوي عنصر واحد، يتم وضع null في مؤشري بداية ونهاية القائمة. وإلا فيتم الوصول إلى العنصر ما قبل الآخير من القائمة ووضع null في المؤشر التالي له ومن ثم جعل مؤشر آخر عنصر عليه.

```
 if ( IsEmpty() ) throw new EmptyListException( name );

 object removeItem = lastNode.Data; // retrieve data

// reset firstNode and lastNode references
if ( firstNode == lastNode )
  firstNode = lastNode = null;
else
{
 ListNode current = firstNode;

// loop while current.Next is not lastNode
while ( current.Next != lastNode )
  current = current.Next; // move to next node

// current is new lastNode
    lastNode = current;
    current.Next = null;

} // end else

return removeItem; // return removed data
```

يُبين الشكل التالي مثال على حذف آخر عنصر من القائمة:



نقوم فيما يلي باستخدام صف القائمة السابق:

```
using System;
using LinkedListLibrary;
// class to test List class functionality
 class ListTest
 {
 public static void Main( string[] args )
 {
 List list = new List(); // create List container
 // create data to store in List
 bool aBoolean = true;
 char aCharacter = '$';
 int anInteger = 34567;
 string aString = "hello";
 // use List insert methods
 list.InsertAtFront( aBoolean );
 list.Display();
 list.InsertAtFront( aCharacter );
 list.Display();
 list.InsertAtBack( anInteger );
 list.Display();
 list.InsertAtBack( aString );
 list.Display();
// use List remove methods
 object removedObject;
 // remove data from list and display after each removal
    try
 {
```

```
    removedObject = list.RemoveFromFront();
    Console.WriteLine( removedObject + " removed" );
    list.Display();

    removedObject = list.RemoveFromFront();
    Console.WriteLine( removedObject + " removed" );
    list.Display();

    removedObject = list.RemoveFromBack();
    Console.WriteLine( removedObject + " removed" );
    list.Display();


    removedObject = list.RemoveFromBack();
    Console.WriteLine( removedObject + " removed" );
    list.Display();

    } // end try
catch ( EmptyListException emptyListException )
{
Console.Error.WriteLine( "\n" + emptyListException );
} // end catch
} // end Main
} // end class ListTest
```

يكون ناتج التنفيذ:

```
The list is: True

The list is: $ True

The list is: $ True 34567

The list is: $ True 34567 hello

$ removed
The list is: True 34567 hello

True removed
The list is: 34567 hello

hello removed
The list is: 34567

34567 removed
Empty list
Press any key to continue . . .
```

4- اقتراحات وتمارين

تمرين:

دمج عناصر قائمتين مرتبتين Merging Ordered-List Objects

- اكتب برنامج يقوم بدمج قائمتين مرتبتين من الأعداد الطبيعية في قائمة واحدة مرتبة.

الفصل الحادي عشر

الأدوات العامة Generics

عنوان الموضوع:

الأدوات العامة Generics.

الكلمات المفتاحية:

Generic Methods، Generic Classes.

ملخص:

نعرض في هذا الفصل بشكل أساسي لاستخدام الطرق والصفوف العامة مما يسمح بكتابة الكود بطريقة مختصرة وأوضح.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- استخدام الطرق العامة.
- قيود الأنماط.
- الصفوف العامة.

المخطط:

الأدوات العامة Generics

- 3 وحدات (Learning Objects)

1- استخدام الأدوات العامة Generics

- نبدأ بالمثال البسيط التالي لعرض الحاجة للطرق العامة: ليُكن لدينا مصفوفات من أنواع مختلفة ونريد كتابة الطرق اللازمة لطباعتها.
- نضطر كما يُبين الكود التالي إلى كتابة طريقة موافقة لكل نمط مُستخدم في هذه المصفوفات:

```
//  OverloadedMethods.cs
// Using overloaded methods to display arrays of different types.
using System;

class OverloadedMethods
{
   public static void Main( string[] args )
   {
      // create arrays of int, double and char
      int[] intArray = { 1, 2, 3, 4, 5, 6 };
      double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
      char[] charArray = { 'H', 'E', 'L', 'L', 'O' };

      Console.WriteLine( "Array intArray contains:" );
      DisplayArray( intArray ); // pass an int array argument
      Console.WriteLine( "Array doubleArray contains:" );
      DisplayArray( doubleArray ); // pass a double array argument
      Console.WriteLine( "Array charArray contains:" );
      DisplayArray( charArray ); // pass a char array argument
   } // end Main

   // output int array
   private static void DisplayArray( int[] inputArray )
   {
      foreach ( int element in inputArray )
         Console.Write( element + " " );

      Console.WriteLine( "\n" );
   } // end method DisplayArray

   // output double array
   private static void DisplayArray( double[] inputArray )
   {
      foreach ( double element in inputArray )
         Console.Write( element + " " );

      Console.WriteLine( "\n" );
   } // end method DisplayArray

   // output char array
   private static void DisplayArray( char[] inputArray )
   {
      foreach ( char element in inputArray )
         Console.Write( element + " " );

      Console.WriteLine( "\n" );
   } // end method DisplayArray
} // end class OverloadedMethods
```

- تكون نتيجة التنفيذ:

```
Array intArray contains:
1 2 3 4 5 6

Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array charArray contains:
H E L L O

Press any key to continue . . .
```

كتابة الطرق العامة

- تسمح لغة C# بالتصريح عن طريقة عامة generic-method يُمكن استخدامها مع معاملات من أنماط مختلفة في كل مرة.
- نقوم في المثال التالي بالتصريح عن الطريقة العامة DisplayArray والتي يُمكن استدعائها مع أنماط مختلفة من المصفوفات:

```
//GenericMethod.cs
// Using overloaded methods to display arrays of different types.
using System;
using System.Collections.Generic;

class GenericMethod
{
   public static void Main( string[] args )
   {
      // create arrays of int, double and char
      int[] intArray = { 1, 2, 3, 4, 5, 6 };
      double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
      char[] charArray = { 'H', 'E', 'L', 'L', 'O' };

      Console.WriteLine( "Array intArray contains:" );
      DisplayArray( intArray ); // pass an int array argument
      Console.WriteLine( "Array doubleArray contains:" );
      DisplayArray( doubleArray ); // pass a double array argument
      Console.WriteLine( "Array charArray contains:" );
      DisplayArray( charArray ); // pass a char array argument
   } // end Main

   // output array of all types
   private static void DisplayArray< T >( T[] inputArray )
   {
      foreach ( T element in inputArray )
         Console.Write( element + " " );

      Console.WriteLine( "\n" );
```

```
    } // end method DisplayArray
} // end class GenericMethod
```

- يكون ناتج التنفيذ:

```
Array intArray contains:
1 2 3 4 5 6

Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array charArray contains:
H E L L O

Press any key to continue . . .
```

2- قيود الأنماط Type Constraints

- نقوم في المثال التالي بالتصريح عن الطريقة العامة Maximum التي تقوم بإعادة القيمة العظمى لمعاملاتها الثلاث.
- بما أن المعاملات أصغر < وأكبر > لا يُمكن استخدامها مع جميع الأنماط، فيجب استخدام الطريقة CompareTo المُصرح عنها في الواجهة العامة **IComparable<T>**.
- يُعيد الاستدعاء x.CompareTo(y) القيمة 0 إذا كان x يساوي y. يُعيد قيمة سالبة إذا كان x أصغر من y. وإلا، فيُعيد قيمة موجبة.
- نقوم في الطريقة Maximum بتحديد قيود النمط T بأنه من النمط: IComparable< T >
- تُصرح العبارة التالية عن هذا القيد:

```
      where T : IComparable< T >
```

- يُصبح المثال:

```
// MaximumTest.cs
// Generic method Maximum returns the largest of three objects.
using System;

class MaximumTest
{
```

```
   public static void Main( string[] args )
   {
      Console.WriteLine( "Maximum of {0}, {1} and {2} is {3}\n",
         3, 4, 5, Maximum( 3, 4, 5 ) );
      Console.WriteLine( "Maximum of {0}, {1} and {2} is {3}\n",
         6.6, 8.8, 7.7, Maximum( 6.6, 8.8, 7.7 ) );
      Console.WriteLine( "Maximum of {0}, {1} and {2} is {3}\n",
         "pear", "apple", "orange",
         Maximum( "pear", "apple", "orange" ) );
   } // end Main

   // generic function determines the
   // largest of the IComparable objects
   private static T Maximum< T >( T x, T y, T z )
      where T : IComparable< T >
   {
      T max = x; // assume x is initially the largest

      // compare y with max
      if ( y.CompareTo( max ) > 0 )
         max = y; // y is the largest so far

      // compare z with max
      if ( z.CompareTo( max ) > 0 )
         max = z; // z is the largest

      return max; // return largest object
   } // end method Maximum
} // end class MaximumTest
```

- يكون ناتج التنفيذ:

```
Maximum of 3, 4 and 5 is 5

Maximum of 6.6, 8.8 and 7.7 is 8.8

Maximum of pear, apple and orange is pear

Press any key to continue . . .
```

3- الصفوف العامة Generic Classes

- يسمح الصف العام بتوصيف صف بشكل مستقل عن أنماط البيانات المستخدمة. نقوم فيما يلي بالتصريح عن صف المكدس `Stack` والذي سيسمح لنا بإنشاء أغراض من هذا الصف مع أنماط بيانات مختلفة حسب الحاجة.

```
// Stack.cs
// Generic class Stack.
using System;

class Stack< T >
{
   private int top; // location of the top element
   private T[] elements; // array that stores stack elements

   // parameterless constructor creates a stack of the default size
   public Stack()
      : this( 10 ) // default stack size
   {
      // empty constructor; calls constructor at line 18 to perform init
   } // end stack constructor

   // constructor creates a stack of the specified number of elements
   public Stack( int stackSize )
   {
      if ( stackSize > 0 ) // validate stackSize
         elements = new T[ stackSize ]; // create stackSize elements
      else
         throw new ArgumentException( "Stack size must be positive." );

      top = -1; // stack initially empty
   } // end stack constructor

   // push element onto the stack; if unsuccessful,
   // throw FullStackException
   public void Push( T pushValue )
   {
      if ( top == elements.Length - 1 ) // stack is full
         throw new FullStackException( string.Format(
            "Stack is full, cannot push {0}", pushValue ) );

      ++top; // increment top
      elements[ top ] = pushValue; // place pushValue on stack
   } // end method Push

   // return the top element if not empty,
   // else throw EmptyStackException
   public T Pop()
   {
      if ( top == -1 ) // stack is empty
         throw new EmptyStackException( "Stack is empty, cannot pop" );

      --top; // decrement top
      return elements[ top + 1 ]; // return top value
   } // end method Pop
} // end class Stack
```

- نُصرح عن صف الاستثناء المخصص `EmptyStackException` لإظهار استثناء مكدس فارغ:

```
// EmptyStackException.cs
// EmptyStackException indicates a stack is empty.
```

```
using System;

class EmptyStackException : Exception
{
   // parameterless constructor
   public EmptyStackException() : base( "Stack is empty" )
   {
      // empty constructor
   } // end EmptyStackException constructor

   // one-parameter constructor
   public EmptyStackException( string exception ) : base( exception )
   {
      // empty constructor
   } // end EmptyStackException constructor

   // two-parameter constructor
   public EmptyStackException( string exception, Exception inner )
      : base( exception, inner )
   {
      // empty constructor
   } // end EmptyStackException constructor
} // end class EmptyStackException
```

- نُصرح عن صف الاستثناء المخصص FullStackException لإظهار استثناء مكدس مملوء:

```
// FullStackException.cs
// FullStackException indicates a stack is full.
using System;

class FullStackException : Exception
{
   // parameterless constructor
   public FullStackException() : base( "Stack is full" )
   {
      // empty constructor
   } // end FullStackException constructor

   // one-parameter constructor
   public FullStackException( string exception ) : base( exception )
   {
      // empty constructor
   } // end FullStackException constructor

   // two-parameter constructor
   public FullStackException( string exception, Exception inner )
      : base( exception, inner )
   {
      // empty constructor
   } // end FullStackException constructor
} // end class FullStackException
```

- نقوم فيما يلي باستخدام صف المكدس السابق:

```
//  StackTest.cs
// Testing generic class Stack.
using System;
```

```
class StackTest
{
   // create arrays of doubles and ints
   private static double[] doubleElements =
      new double[]{ 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
   private static int[] intElements =
      new int[]{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

   private static Stack< double > doubleStack; // stack stores doubles
   private static Stack< int > intStack; // stack stores int objects

   public static void Main( string[] args )
   {
      doubleStack = new Stack< double >( 5 ); // stack of doubles
      intStack = new Stack< int >( 10 ); // stack of ints

      TestPushDouble(); // push doubles onto doubleStack
      TestPopDouble(); // pop doubles from doubleStack
      TestPushInt(); // push ints onto intStack
      TestPopInt(); // pop ints from intStack
   } // end Main

   // test Push method with doubleStack
   private static void TestPushDouble()
   {
      // push elements onto stack
      try
      {
         Console.WriteLine( "\nPushing elements onto doubleStack" );

         // push elements onto stack
         foreach ( var element in doubleElements )
         {
            Console.Write( "{0:F1} ", element );
            doubleStack.Push( element ); // push onto doubleStack
         } // end foreach
      } // end try
      catch ( FullStackException exception )
      {
         Console.Error.WriteLine();
         Console.Error.WriteLine( "Message: " + exception.Message );
         Console.Error.WriteLine( exception.StackTrace );
      } // end catch
   } // end method TestPushDouble

   // test Pop method with doubleStack
   private static void TestPopDouble()
   {
      // pop elements from stack
      try
      {
         Console.WriteLine( "\nPopping elements from doubleStack" );

         double popValue; // store element removed from stack

         // remove all elements from stack
         while ( true )
         {
            popValue = doubleStack.Pop(); // pop from doubleStack
            Console.Write( "{0:F1} ", popValue );
         } // end while
```

```
      } // end try
      catch ( EmptyStackException exception )
      {
         Console.Error.WriteLine();
         Console.Error.WriteLine( "Message: " + exception.Message );
         Console.Error.WriteLine( exception.StackTrace );
      } // end catch
   } // end method TestPopDouble

   // test Push method with intStack
   private static void TestPushInt()
   {
      // push elements onto stack
      try
      {
         Console.WriteLine( "\nPushing elements onto intStack" );

         // push elements onto stack
         foreach ( var element in intElements )
         {
            Console.Write( "{0} ", element );
            intStack.Push( element ); // push onto intStack
         } // end foreach
      } // end try
      catch ( FullStackException exception )
      {
         Console.Error.WriteLine();
         Console.Error.WriteLine( "Message: " + exception.Message );
         Console.Error.WriteLine( exception.StackTrace );
      } // end catch
   } // end method TestPushInt

   // test Pop method with intStack
   private static void TestPopInt()
   {
      // pop elements from stack
      try
      {
         Console.WriteLine( "\nPopping elements from intStack" );

         int popValue; // store element removed from stack

         // remove all elements from stack
         while ( true )
         {
            popValue = intStack.Pop(); // pop from intStack
            Console.Write( "{0} ", popValue );
         } // end while
      } // end try
      catch ( EmptyStackException exception )
      {
         Console.Error.WriteLine();
         Console.Error.WriteLine( "Message: " + exception.Message );
         Console.Error.WriteLine( exception.StackTrace );
      } // end catch
   } // end method TestPopInt
} // end class StackTest
```

- يكون ناتج التنفيذ:

```
Pushing elements onto doubleStack
1.1 2.2 3.3 4.4 5.5 6.6
Message: Stack is full, cannot push 6.6
   at Stack`1.Push(T pushValue) in e:\5.OOP\Chapter_11\Stack\Stack\Stack.cs:line 33
   at StackTest.TestPushDouble() in e:\5.OOP\Chapter_11\Stack\Stack\StackTest.cs:line 39

Popping elements from doubleStack
5.5 4.4 3.3 2.2 1.1
Message: Stack is empty, cannot pop
   at Stack`1.Pop() in e:\5.OOP\Chapter_11\Stack\Stack\Stack.cs:line 45
   at StackTest.TestPopDouble() in e:\5.OOP\Chapter_11\Stack\Stack\StackTest.cs:line 63

Pushing elements onto intStack
1 2 3 4 5 6 7 8 9 10 11
Message: Stack is full, cannot push 11
   at Stack`1.Push(T pushValue) in e:\5.OOP\Chapter_11\Stack\Stack\Stack.cs:line 33
   at StackTest.TestPushInt() in e:\5.OOP\Chapter_11\Stack\Stack\StackTest.cs:line 87

Popping elements from intStack
10 9 8 7 6 5 4 3 2 1
Message: Stack is empty, cannot pop
   at Stack`1.Pop() in e:\5.OOP\Chapter_11\Stack\Stack\Stack.cs:line 45
   at StackTest.TestPopInt() in e:\5.OOP\Chapter_11\Stack\Stack\StackTest.cs:line 111
Press any key to continue . . .
```

4- اقتراحات وتمارين

تمرين: الطريقة العامة للبحث التسلسلي Generic Linear Search Method

- اكتب الطريقة العامة search للبحث التسلسلي في مصفوفة. تقوم الطريقة بإرجاع فهرس العنصر عند وجوده في المصفوفة وإلا تُعيد 1-.

الفصل الثاني عشر
المجموعات Collections

عنوان الموضوع:

المجموعات Collections.

الكلمات المفتاحية:

الصفوف غير العامة Nongeneric Classes، الصفوف العامة Generic Classes.

ملخص:

نعرض في هذا الصف أهم الصفوف غير العامة والصفوف العامة التي توفرها لغة C#.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- الصف ArrayList.
- الصف LinkedList.

المخطط:

المجموعات Collections

- 2 وحدة (Learning Objects)

1- الصف غير العام: مصفوفة القائمة Nongeneric Class: ArrayList

- يسمح هذا الصف بالتعامل مع مجموعة عناصر كمصفوفة من العناصر مع تمتعه بميزة ديناميكية الحجم. وبهذا فإنه يتجاوز المشكلة التقليدية لتحديد حجم المصفوفات.
- من أهم خصائص هذا الصف:

Capacity	السعة: تسمح هذه الخاصية بتحديد أو قراءة الحجم (عدد العناصر المحجوز لعناصر قائمة المصفوفة.
Count	العدد: تسمح هذه الخاصية بقراءة عدد العناصر الحالي.

- من أهم طرق هذا الصف:

Add	تُضيف غرض وتُعيد فهرسه.
Clear	حذف جميع العناصر.
Contains	تُعيد true في حال كانت مصفوفة القائمة تحوي العنصر وإلا تُعيد false.
IndexOf	تُعيد فهرس أول تواجد للعنصر في المصفوفة.
Insert	إدراج عنصر في فهرس محدّد.
Remove	إزالة أول تواجد للعنصر في المصفوفة.
RemoveAt	إزالة عنصر من فهرس محدد.
RemoveRange	إزالة عدد من العناصر اعتباراً من فهرس محدد.
Sort	ترتيب (فرز) عناصر المصفوفة.
TrimToSize	تقوم بإسناد قيمة العدد الحالي إلى خاصية السعة.

- نقوم في المثال التالي بعرض استخدام بعض الخصائص والطرق السابقة:

```
// ArrayListTest.cs
// Using class ArrayList.
using System;
using System.Collections;

public class ArrayListTest
{
   private static readonly string[] colors =
```

```
      { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
   private static readonly string[] removeColors =
      { "RED", "WHITE", "BLUE" };

   // create ArrayList, add colors to it and manipulate it
   public static void Main( string[] args )
   {
      ArrayList list = new ArrayList( 1 ); // initial capacity of 1

      // add the elements of the colors array to the ArrayList list
      foreach ( var color in colors )
         list.Add( color ); // add color to the ArrayList list

      // add elements in the removeColors array to
      // the ArrayList removeList with the ArrayList constructor
      ArrayList removeList = new ArrayList( removeColors );

      Console.WriteLine( "ArrayList: " );
      DisplayInformation( list ); // output the list

      // remove from ArrayList list the colors in removeList
      RemoveColors( list, removeList );

      Console.WriteLine( "\nArrayList after calling RemoveColors: " );
      DisplayInformation( list ); // output list contents
   } // end Main

   // displays information on the contents of an array list
   private static void DisplayInformation( ArrayList arrayList )
   {
      // iterate through array list with a foreach statement
      foreach ( var element in arrayList )
         Console.Write( "{0} ", element ); // invokes ToString

      // display the size and capacity
      Console.WriteLine( "\nSize = {0}; Capacity = {1}",
         arrayList.Count, arrayList.Capacity );

      int index = arrayList.IndexOf( "BLUE" );

      if ( index != -1 )
         Console.WriteLine( "The array list contains BLUE at index {0}.",
            index );
      else
         Console.WriteLine( "The array list does not contain BLUE." );
   } // end method DisplayInformation

   // remove colors specified in secondList from firstList
   private static void RemoveColors( ArrayList firstList, ArrayList secondList )
   {
      // iterate through second ArrayList like an array
      for ( int count = 0; count < secondList.Count; ++count )
         firstList.Remove( secondList[ count ] );
   } // end method RemoveColors
} // end class ArrayListTest
```

- يكون ناتج التنفيذ:

```
ArrayList:
MAGENTA RED WHITE BLUE CYAN
Size = 5; Capacity = 8
The array list contains BLUE at index 3.

ArrayList after calling RemoveColors:
MAGENTA CYAN
Size = 2; Capacity = 8
The array list does not contain BLUE.
Press any key to continue . . .
```

2- الصف العام: القائمة المرتبطة Generic Class: LinkedList

- يكون هذا الصف قائمة مرتبطة مضاعفة doubly linked list (يؤشر كل عنصر على العنصر التالي وعلى العنصر السابق). تحوي كل عقدة من القائمة الخاصية **Value** والخاصيتين (للقراءة فقط) التالي **Next** والسابق **Previous**.
- من أهم خصائص هذا الصف:

`First`	أول عقدة من القائمة.
`Last`	آخر عقدة من القائمة.
`Previous`	العقدة السابقة.
`Next`	العقدة التالية.
`Value`	قيمة العقدة.

- من أهم طرق هذا الصف:

`AddLast`	إضافة عنصر إلى آخر القائمة.
`Find`	إعادة العقدة التي تحوي قيمة معينة.
`Remove`	حذف عقدة.

- نقوم في هذا المثال بعرض استخدام هذا الصف:

```
// LinkedListTest.cs
// Using LinkedLists.
using System;
using System.Collections.Generic;

public class LinkedListTest
{
   private static readonly string[] colors = { "black", "yellow",
      "green", "blue", "violet", "silver" };
   private static readonly string[] colors2 = { "gold", "white",
      "brown", "blue", "gray" };

   // set up and manipulate LinkedList objects
   public static void Main( string[] args )
   {
      LinkedList< string > list1 = new LinkedList< string >();

      // add elements to first linked list
      foreach ( var color in colors )
         list1.AddLast( color );

      // add elements to second linked list via constructor
      LinkedList< string > list2 = new LinkedList< string >( colors2 );

      Concatenate( list1, list2 ); // concatenate list2 onto list1
      PrintList( list1 ); // display list1 elements

      Console.WriteLine( "\nConverting strings in list1 to uppercase\n" );
      ToUppercaseStrings( list1 ); // convert to uppercase string
      PrintList( list1 ); // display list1 elements

      Console.WriteLine( "\nDeleting strings between BLACK and BROWN\n" );
      RemoveItemsBetween( list1, "BLACK", "BROWN" );

      PrintList( list1 ); // display list1 elements
      PrintReversedList( list1 ); // display list in reverse order
   } // end Main

   // display list contents
   private static void PrintList< T >( LinkedList< T > list )
   {
      Console.WriteLine( "Linked list: " );

      foreach ( T value in list )
         Console.Write( "{0} ", value );

      Console.WriteLine();
   } // end method PrintList

   // concatenate the second list on the end of the first list
   private static void Concatenate< T >( LinkedList< T > list1,
      LinkedList< T > list2 )
   {
      // concatenate lists by copying element values
      // in order from the second list to the first list
      foreach ( T value in list2 )
         list1.AddLast( value ); // add new node
   } // end method Concatenate
```

```
   // locate string objects and convert to uppercase
   private static void ToUppercaseStrings( LinkedList< string > list )
   {
      // iterate over the list by using the nodes
      LinkedListNode< string > currentNode = list.First;

      while ( currentNode != null )
      {
         string color = currentNode.Value; // get value in node
         currentNode.Value = color.ToUpper(); // convert to uppercase

         currentNode = currentNode.Next; // get next node
      } // end while
   } // end method ToUppercaseStrings

   // delete list items between two given items
   private static void RemoveItemsBetween< T >( LinkedList< T > list,
      T startItem, T endItem )
   {
      // get the nodes corresponding to the start and end item
      LinkedListNode< T > currentNode = list.Find( startItem );
      LinkedListNode< T > endNode = list.Find( endItem );

      // remove items after the start item
      // until we find the last item or the end of the linked list
      while ( ( currentNode.Next != null ) &&
         ( currentNode.Next != endNode ) )
      {
         list.Remove( currentNode.Next ); // remove next node
      } // end while
   } // end method RemoveItemsBetween

   // display reversed list
   private static void PrintReversedList< T >( LinkedList< T > list )
   {
      Console.WriteLine( "Reversed List:" );

      // iterate over the list by using the nodes
      LinkedListNode< T > currentNode = list.Last;

      while ( currentNode != null )
      {
         Console.Write( "{0} ", currentNode.Value );
         currentNode = currentNode.Previous; // get previous node
      } // end while

      Console.WriteLine();
   } // end method PrintReversedList
} // end class LinkedListTest
```

- يكون ناتج التنفيذ:

```
Linked list:
black yellow green blue violet silver gold white brown blue gray

Converting strings in list1 to uppercase

Linked list:
```

```
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN
BLUE GRAY

Deleting strings between BLACK and BROWN

Linked list:
BLACK BROWN BLUE GRAY
Reversed List:
GRAY BLUE BROWN BLACK
Press any key to continue . . .
```

3- اقتراحات وتمارين

تمرين 1: قائمة مرتبطة بدون تكرار *LinkedList without Duplicates*

قم بكتابة برنامج يقرأ مجموعة من الأسماء ويقوم بتخزينها في قائمة مرتبطة. يجب عدم تخزين الأسماء المكررة. يجب السماح للمستخدم بالبحث عن اسم معين.

تمرين 2: عكس قائمة مرتبطة *Reversing a LinkedList*

قم بكتابة برنامج يقوم بإنشاء قائمة مرتبطة ويضع فيها 10 محارف ثم يقوم بنسخ عناصر هذه القائمة إلى قائمة أخرى بترتيب معكوس.