الصفوف والأغراض
Classes and Objects

العنوان	رقم الصفحة
مقدمة	3
1. الصفوف والأغراض في#C	4
2. الحقول والخصائص	6
3. الباني والهادم	9
4. أنماط القيمة وأنماط المرجع	11
5. مثال تعليمي	13
6. الأنشطة المرافقة	19

الكلمات المفتاحية

الصف، الغرض، الحقول، الخصائص، الطرائق، محددات الوصول، الباني، الهادم، أنماط القيمة، أنماط المرجع.

ملخص الفصل

نبيّن في هذا الفصل أساسيات البرمجة غرضية التوجّه، فنعرض كيفية التصريح عن الصف وتضمينه حقولاً وخصائص وطرائق وبناة وهادم. كما نوضّح كيفية استخدام الصف كنمط مرجع جديد لإنشاء أغراضٍ منه.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- المفاهيم الأساسية للبرمجة غرضية التوجّه
- التصريح عن الصفوف
- إنشاء الأغراض من الصفوف
- الحقول والخصائص والطرائق
- الباني والهادم
- الفرق بين أنماط القيمة وأنماط المرجع

مقدمة

تركّز البرمجة الإجرائية Procedural Programming على استخدام الإجرائيات Procedures (أو التوابع Functions)، حيث تقوم كل إجرائية بمعالجة معطيات تُمرَّر إليها كوسائط دخل، ثم تقدِّم القيم الناتجة عن المعالجة إلى الإجرائية التي قامت باستدعائها. وعلى الرغم من أنّ البرمجة الإجرائية قلّلت من تكرار الترميز Code Duplication، إلا أنّها محدودة من ناحية أنماط وبنى المعطيات التي تتعامل معها. وتمّ تجاوز هذا الأمر في البرمجة غرضية التوجّه Object Oriented Programming (OOP) التي سمحت بإنشاء أغراضٍ ذات أنماط معطيات جديدة، وتحتوي الأغراض على إجرائيات (سنسمّيها من الآن فصاعداً طرائق Methods) وعلى حقول المعطيات التي تستخدمها هذه الإجرائيات جنباً لجنب.

وتمتلك البرمجة غرضية التوجّه OOP ميزات إضافية مقارنة مع البرمجة الإجرائية، نذكر منها:

- توفِّر OOP بنياناً أوضح وأكثر تنظيماً للرمّاز المصدري Source Code.
- تسمح OOP بإعادة استخدام الكتل البرمجية بسهولة مما يقلّل من عدد أسطر الرمّاز المصدري ويوفّر في الزمن اللازم لإنجازه.
- تجعل OOP تعديل الرمّاز المصدري أسهل مقارنة مع البرمجة الإجرائية.

يمكن تلخيص المفاهيم الأساسية التي تقوم عليها البرمجة غرضية التوجّه بما يأتي:

- **التغليف Encapsulation**: حيث يتمّ إخفاء تفاصيل بعض عناصر المكونات البرمجية عن المكونات الأخرى التي لا تحتاجها ويتمّ توفير واجهات تخاطب مناسبة بين المكونات البرمجية المختلفة.
- **الوراثة Inheritance**: يسمح هذا المبدأ بإعادة استخدام الرمّاز المصدري ويعزّز من سهولة قراءته.
- **التجريد Abstraction**: يتيح هذا المبدأ للمطوّر إمكانية التركيز على الأشياء الهامة أولاً وإهمال بقية التفاصيل ليتمّ التعامل معها عند اللزوم.
- **تعدُّد الاشكال Polymorphism**: يسمح هذا المبدأ بأن يكون للطريقة الواحدة أكثر من شكلٍ للاستخدام (أي أكثر من تنجيز Implementation) وكل شكلٍ يُعبِّر عن سلوكٍ مختلف، ويتم تخصيص السلوك المطلوب وفقاً لاحتياجات البنية التي تحوي الطريقة.

1. الصفوف والأغراض في C#

يتكوّن البرنامج في لغة C# من صف أو أكثر Class، وهو بنية معطيات توصف مجموعة من الأشياء التي تجمعها خصائص وسلوكيات مشتركة، وتُسمى بالأغراض Objects.

يحوي كل صف، بالإضافة إلى حقول المعطيات Fields الخاصة به، مجموعة من الطرائق والخصائص Properties التي تحدّد سلوكه وتسمح بالوصول إلى معطياته ومعالجتها، ويُطلق على مكوناته اسم أعضاء الصف Class Members.

إنشاء صف:

- لإنشاء صف نستخدم الكلمة المفتاحية class متبوعةً باسم الصف الذي يكون بمثابة معرِّف Identifier له.
- تُستخدَم الأقواس { } للدلالة على جسم الصف الذي يحتوي على أعضاء الصف من حقول وخصائص وطرائق وغيرها.
- يُسبَق كل عضو من أعضاء الصف بمحدد وصول Access Specifier للدلالة على سماحيات الوصول إليه:
  - المحدد public يُشير إلى أنّ العضو عام ويمكن الوصول إليه من دون قيود.
  - المحدّد private يُشير إلى أنّ العضو خاص ولا يمكن الوصول إليه إلا من قبل الأعضاء المنتمية لنفس الصف المعرّف ضمنه، ويمكن التصريح عن محدد وصول للصف ككل.

مثال:

يحتوي الصف Student على حقلٍ اسمه name للدلالة على اسم الطالب، وعلى خاصيةٍ اسمها Age للدلالة على عمره، وعلى الطريقة PrintInfo لطباعة معلومات الطالب. ويتمّ التصريح عن الصف Student كما يأتي:

```
public class Student {
     public  string name;
     public int Age { set; get; }
     public void PrintInfo(){
        Console.WriteLine("The name is: {0} , the age is: {1}", name, Age);
        }// end PrintInfo
      }// end class Student
```

إنشاء غرض:

يُمثِّل كلّ غرضٍ منتسخ Instance من الصف، ويتمّ إنشاؤه باستخدام الكلمة المفتاحية new متبوعة باستدعاء لطريقة خاصة تحمل نفس اسم الصف وتُدعى باني الصف Class Constructor.

مثال:

نوضّح كيفية التصريح عن الغرض obj1 المعبِّر عن طالبٍ اسمه MyName، وعمره 24، ويمكنه استخدام الطريقة PrintInfo لطباعة اسمه وعمره على الشاشة:

```
Student obj1 = new Student();
obj1.Age = 24;
obj1.name = "MyName";
obj1.PrintInfo();
```

2. الحقول والخصائص

1.2. الحقول

هي متغيّرات يتمّ تعريفها في الصف لتشكّل أحد أعضائه، وعند إنشاء غرضٍ من الصف ستكون له نسخته الخاصة من قيم حقول الصف. واصطلاحاً، يجب أن يبدأ اسم الحقل بحرفٍ صغيرٍ، وأن يبقى محدّد الوصول للحقل private.

2.2. الخصائص

للوصول إلى الحقول من خارج الصف، نستخدم الخصائص التي تشبه الحقول من حيث الاستخدام، وتشبه أيضاً الطرائق من حيث البنية، لكن ليس لها وسائط دخل. وتمتلك الخصائص موصلاتٍ Accessors تحتوي على عمليات يتمّ تنفيذها عند قراءة قيمها أو إسناد قيم جديدة لها. واصطلاحاً، يجب أن يبدأ اسم الخاصية Property بحرفٍ كبيرٍ، وأن يكون محدّد الوصول إليها public.

يكون للخاصية عادة موصلين:

- get: يتمّ استخدامه عند طلب قراءة قيمة الحقل الموافق للخاصية.
- set: يتمّ استخدامه عند طلب إسناد قيمة إلى الحقل الموافق للخاصية، وعندها تُستخدم الكلمة value للدلالة على القيمة.

مثال:

نوضّح من خلال التعليمات الآتية كيفية التصريح عن الخاصية Name الموافقة للحقل name ضمن الصف Student:

```
private string name; // field name
public string Name { // property Name
     set { name = value;}
     get { return name; }
       }// end Name
```

وعندما تكون الخاصية بسيطة بحيث يحوي كل موصل على تعليمة واحدة كما في حالتنا هذه، يمكن الاستعاضة عن التصريح السابق للخاصية Name بالتصريح الآتي:

```
    private string name; // field name
    public string Name { // property Name
           get => name;
           set => name = value;
             }// end Name
```

وعند إجراء هذا التغيير على الصف Student، يجب الاستعاضة عن الحقل name المستخدم خارج الصف بالخاصية Name.

3.2. الخصائص التلقائية

عوضاً عن التصريح عن حقلٍ وخاصيةٍ معاً، يمكن استخدام الخصائص التلقائية كما في حالة الخاصية Age:

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

وفي مرحلة ترجمة الرمّاز المصدري، سيقوم المترجم تلقائياً بإنشاء حقلٍ موافقٍ للخاصية وإنشاء الموصلات get وset اللازمين.

وفي معظم الحالات، يتمّ استخدام الخصائص التلقائية Auto-Implemented Properties، ويقتصر استخدام الحقول على تعريف الثوابت أو عند الحاجة لتعريف حقلٍ ساكنٍ أو عند وجود شروط على القيمة الواجب إسنادها إلى الحقل.

وليس بالضرورة أن يكون للموصلين نفس سماحيات الوصول، فيمكن جعل تعديل قيمة الحقل name الموافق للخاصية Name مقتصراً على الأعضاء الموجودة داخل الصف ويتمّ ذلك كما يأتي:

```
public int Age { private  set; get; }
```

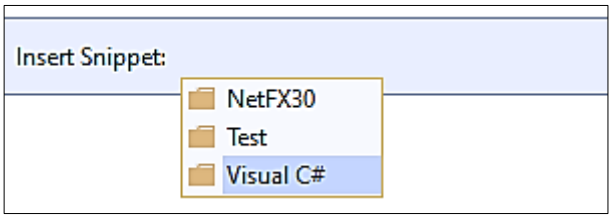
وتجدر الإشارة إلى أنّه في حال كان محدّد الوصول إلى الخاصية private، لا يُسمَح بأن يكون محدّد الوصول إلى أي من موصليها public.

وفي بعض الأحيان، يمكن أن تقتصر الخاصية على الموصل get فقط (أي تصبح للقراءة فقط)، وفي هذه الحالة يجب أن تقوم تعليمات أخرى داخل الصف بإسناد قيمة للحقل الموافق، وعادة ما يتمّ ذلك ضمن باني الصف.

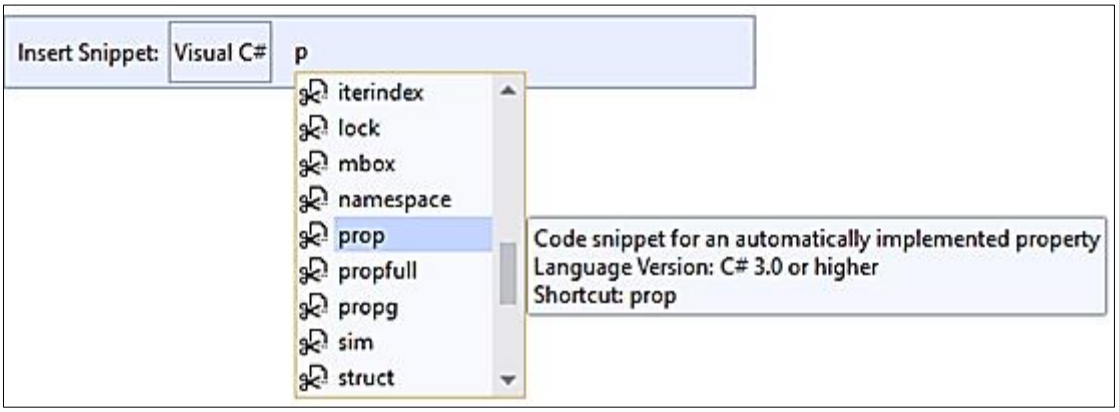
يمكن إنشاء الخصائص التلقائية باستخدام قصاصات الكود الجاهزة كما هو موضح في المثال:

يمكن استخدام قصاصات الكود الجاهزة Code Snippets لإنشاء الخصائص التلقائية كما يلي:

1. بالضغط على Ctrl+k ثم Ctrl+x تظهر النافذة:



2. نختار Visual C#، ثم نختار prop فتظهر النافذة الآتية:



3. نضغط على Enter فتتمّ كتابة الخاصية:

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

4. وأخيراً، نقوم بإجراء بعض التعديلات للحصول على الخاصية المطلوبة.

3. الباني والهادم

الباني:

عند إنشاء أي منتَسخ من الصف، يتمّ استدعاء أحد بناة الصف الذي يتيح للمبرمِج إمكانية إسناد قيم افتراضية لأعضاء الصف التي تحتاج لتهيئة. والباني هو طريقة خاصة تحمل نفس اسم الصف ولا تعيد أي قيمة ولا يجب استخدام كلمة void في توقيعها، وغالباً ما يكون محدّد الوصول إليها public.

وفي حال عدم التصريح عن أي بانٍ ضمن الصف، يقوم المترجم باستدعاء الباني الافتراضي وهو بانٍ لا يمتلك وسائط دخل وإنّما يقوم بتهيئة الأعضاء بالقيم الافتراضية الموافقة لأنماط معطياتها، فيتمّ إسناد القيمة (0) إلى المتغيّرات العددية والقيمة false للمتغيّرات المنطقية والقيمة null لمتغيّرات المرجع. وفي مثالنا السابق، الباني الافتراضي Student المستخدم لإنشاء الغرض obj1 يُسنِد القيمة null إلى الحقل name.

ويمكن القيام بتحميلٍ زائد Overloading للباني، أي يمكن للصف أن يكون له أكثر من بانٍ واحدٍ، وعندها يجب أن تختلف البناة بعدد وسائط الدخل أو أنماط معطياتها. وفي حال تمّ تعريف بانٍ واحد على الأقل، لا يقوم المترجم باستدعاء بانٍ افتراضي للصف، وفي حال الحاجة إلى بانٍ افتراضي (من دون وسائط دخل) يجب تعريفه.

الهادم:

في لغة C#، يمكن للمبرمج أن يرتاح من الاهتمام بإدارة الذاكرة لأنّ مجمّع النفايات Garbage Collector يقوم بهذه المهمة. فيعمل مجمّع النفايات على تحديد الأغراض التي لا مرجع لها (انتهى البرنامج من استخدامها) واستعادة الذاكرة المخصّصة لها لاستخدامها من قبل أغراض أخرى. أما بالنسبة للموارد الأخرى كقواعد البيانات واتصالات الشبكة والملفّات، فيجب على المبرمج أن يضمن تحريرها عند الانتهاء من استخدامها. ويمكن أن يتمّ ذلك من خلال تعريف هادم Destructor (يُسمّى أحياناً Finalizer) للقيام بتحرير الموارد المحجوزة، حيث يقوم مجمع النفايات تلقائياً باستدعاء الهادم قبل استعادة الذاكرة المخصّصة للغرض.

والهادم هو طريقة خاصة تحمل نفس اسم الصف وتُسبَق بالمحرف (~)، ويتمّ تنفيذ التعليمات الواردة ضمنها في اللحظة التي تسبق مباشرة تدمير الغرض. وليس للهادم وسطاء دخل وبالتالي لا يمكن القيام بتحميل زائد له، وليس للهادم محدّد وصول، ولا يحتوي الصف إلا على هادم واحد. ونوضّح في المثال الآتي كيفية التصريح عن هادم الصف Student:

```
// destructor
        ~Student(){
            Console.WriteLine("The object will be destroyed ");
        }// end destructor
```

ولا يمكن للمبرمج التنبؤ بالتوقيت الذي سيقوم فيه مجمّع النفايات باستدعاء الهادم، فيلجأ بعض المبرمجين إلى إرغام مجمّع النفايات على القيام بذلك من خلال استدعاء الطريقة GC.Collect، ولكن هذا التصرف غير مرغوب لأنّه يؤثّر كثيراً على الأداء.

4. أنماط القيمة وأنماط المرجع

تُصنَّف الأنماط في لغة C# إلى أنماط قيمة Value Data Type وأنماط مرجع Reference Data Type.

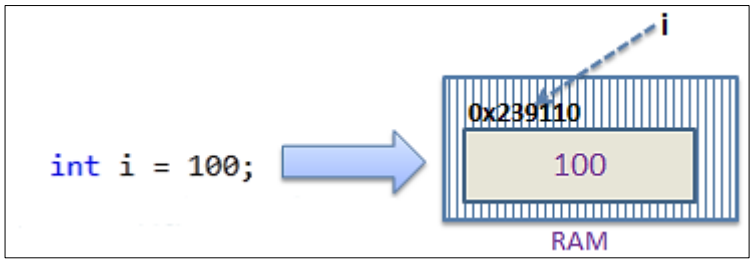
أنماط القيمة:

هي أنماط بسيطة مثل int وdouble وfloat، ويضمّ المتغيّر من أحد هذه الأنماط قيمة معيّنة في نفس المكان المخصّص له في الذاكرة.

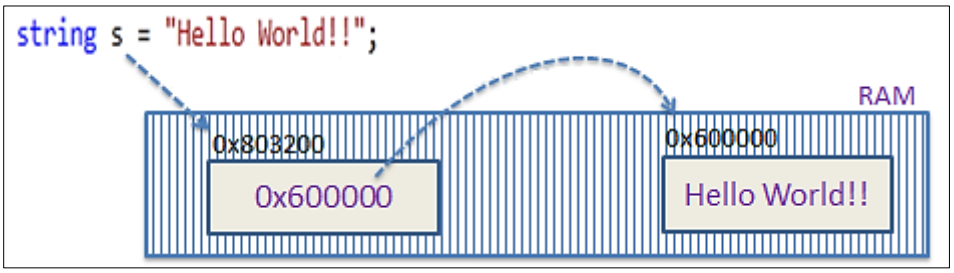
مثال:

لتعريف المتغيّر i من النمط int نكتب: int i;

وعند التنفيذ، سيتمّ منح المتغيّر عنواناً، لنفرض أنّ العنوان هو 0x239110.



ولإسناد قيمة إليه نكتب: i=100; فيتمّ تخزين القيمة 100 في الذاكرة في المكان الذي عنوانه 0x239110 ويتمّ توضيح ذلك في الصورة [1] المرفقة:

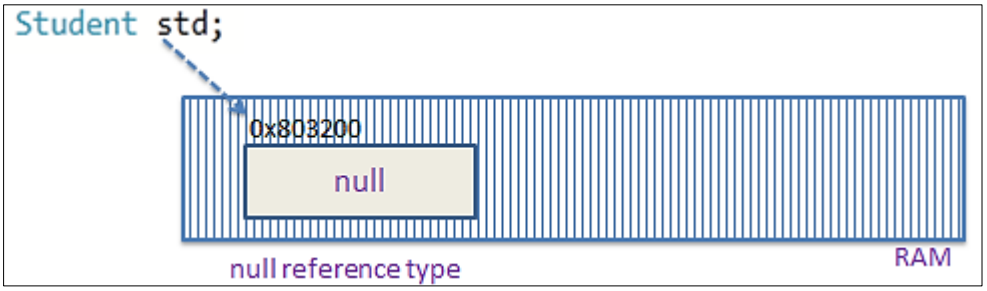


أنماط المرجع:

وهي عادة ما تكون مركّبة مثل السلاسل المحرفية والمصفوفات ويضمّ المتغيّر من أحد هذه الأنماط عنواناً لمكان في الذاكرة يتمّ فيه تخزين معطيات المتغيّر. فعلى سبيل المثال، في الصورة [1] الآتية:

نلاحظ أنه تمّ تعريف المتغيّر s من النمط string وإسناد القيمة Hello World!! له، وتمّ تخصيص مكان في الذاكرة للمتغيّر s عنوانه 0x803200، ويحتوي هذا المكان بدوره على عنوان لمكان آخر في الذاكرة 0x600000 والذي سيحوي قيمة المتغيّر.

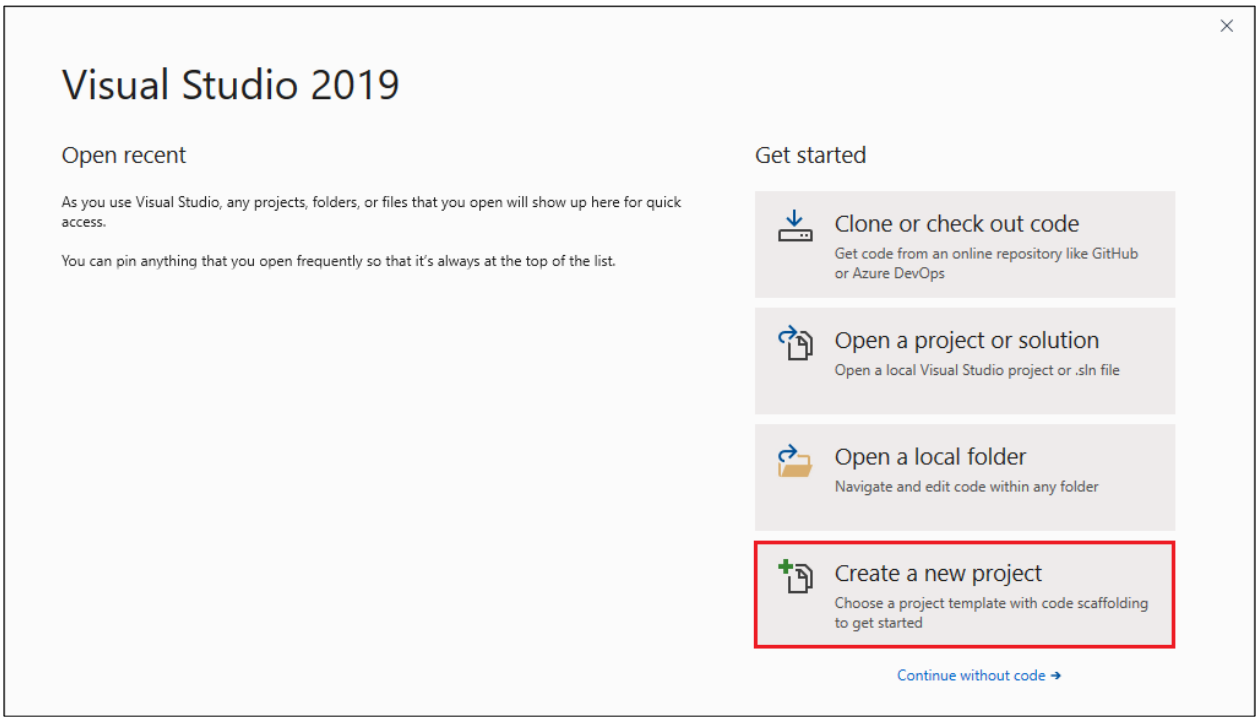
وعند تعريف أي صف، نكون قد عرفنا نمط معطيات مرجعي جديد، ويجب التذكير بأنّ القيمة الافتراضية لمتغيّر من نمط مرجعي هي null. فعلى سبيل المثال [1]، عند التصريح عن المتحوّل std من النمط Student، ومن دون استخدام بانٍ لإنشاء غرض، تكون القيمة المخزنة فيه null.



5. مثال تعليمي

نوضّح فيما يأتي كيفية إنشاء الصف Student باستخدام بيئة العمل Visual Studio 2019 التي تسمح بإنشاء التطبيقات بأسلوب بسيط وسريع:

1. بداية، نشغّل Visual Studio 2019، ثم نختار إنشاء مشروع جديد "Create a new project".

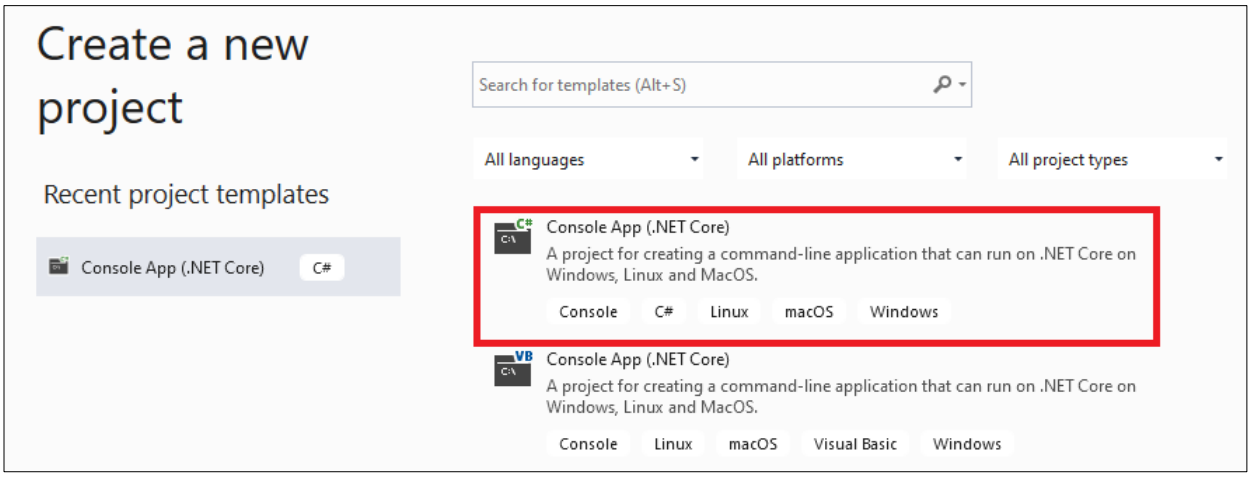


ويمكن إنشاء مشروع جديد من الواجهة الرئيسية لبيئة العمل من خلال اختيار ما يأتي:

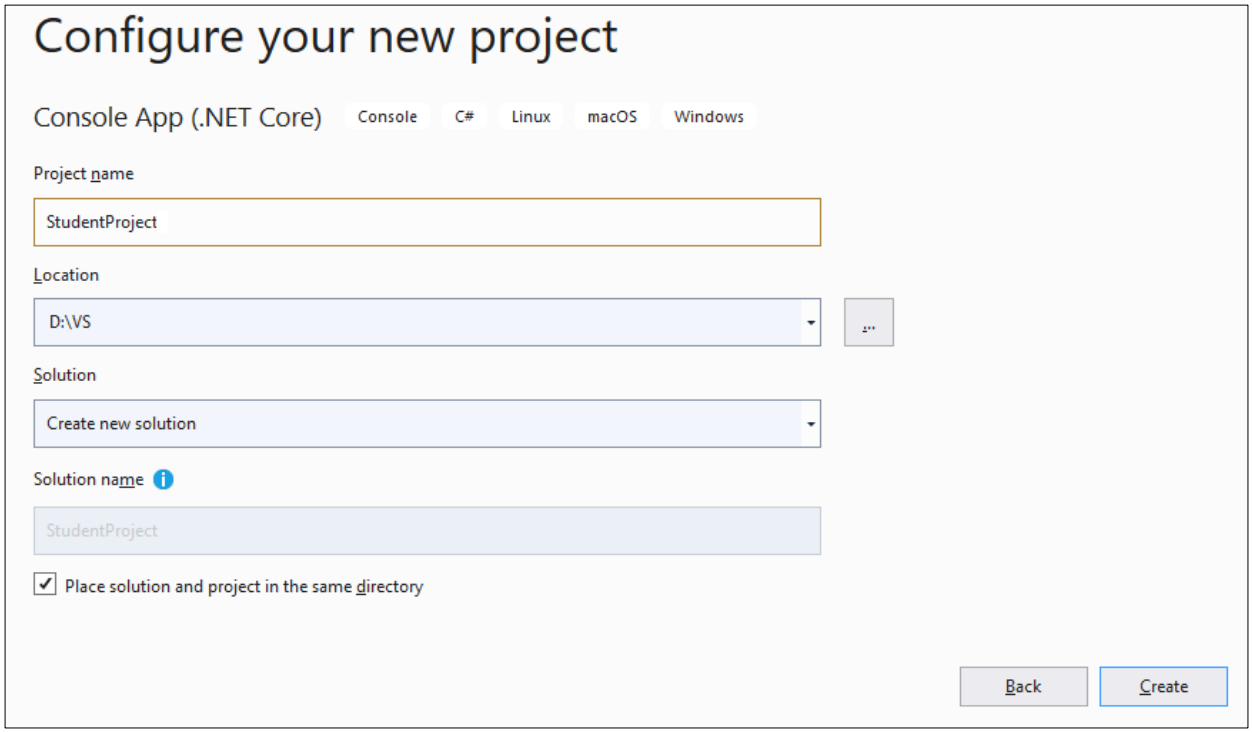
File → New → Project

أو من خلال الضغط على المحارف Ctrl+ Shift+ N معاً.

2. نختار نوع القالب Console App (.NET Core).

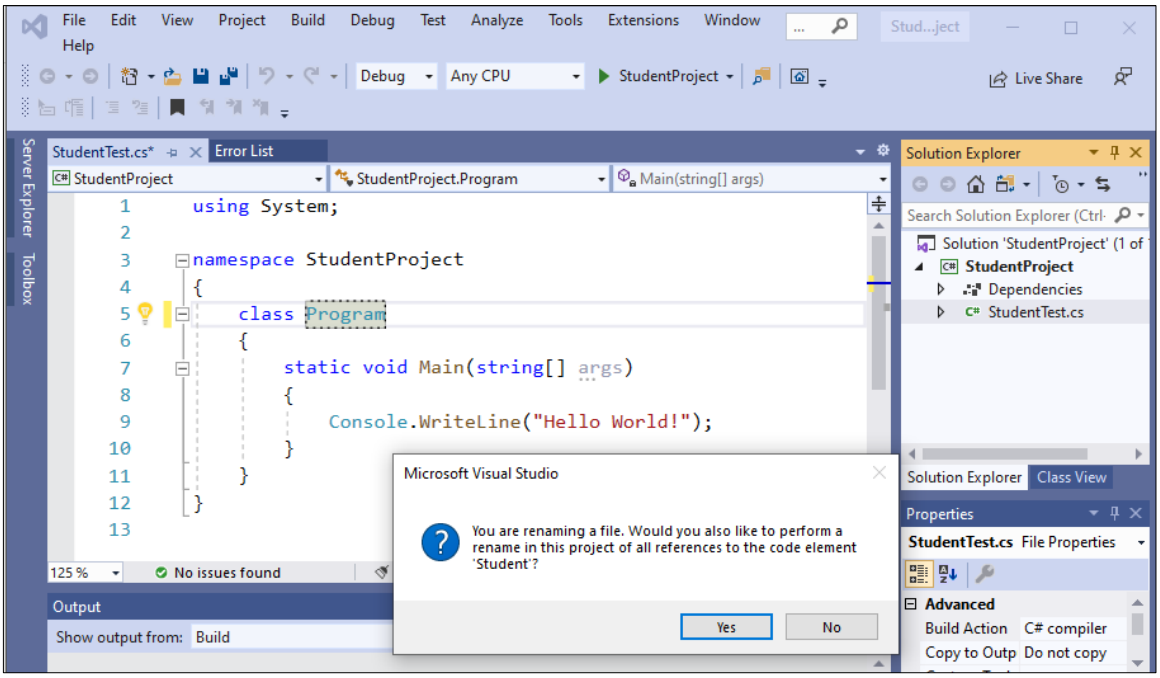


3. ندخل اسم المشروع StudentProject وندخل مكان تخزينه D:\VS، ثم نختار Create.

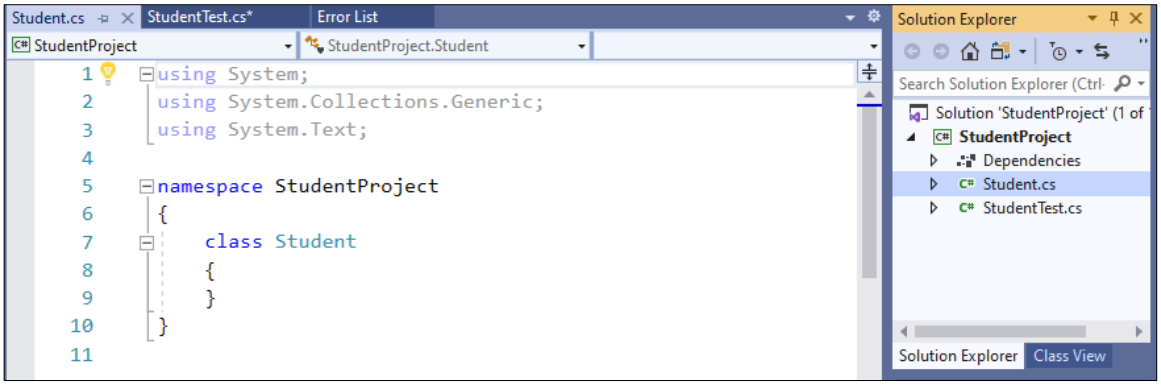


سيتمّ إنشاء مشروع جديد باسم StudentProject حاوٍ على الملفّ النصّي program.cs الذي بدوره يحتوي على الصف program.

4. نستبدل اسم الملفّ النصّي بالاسم StudentTest.cs، فتظهر رسالة للسؤال عن الرغبة في تغيير جميع المراجع الدالة على program. فنختار Yes للموافقة على المقترح، فيتمّ تغيير اسم الصف إلى StudentTest. وضمن الصف، توجد الطريقة Main التي تُعتبَر نقطة البدء عند تنفيذ البرنامج.



5. لإضافة صف جديد، نقوم بالنقر بالزر اليميني على أيقونة المشروع StudentProject، فتظهر نافذة نختار منها add، فتظهر نافذة جديدة، نختار منها Class.



نُدخل اسم الصف الجديد Student ثم نضغط add، فتتمّ إضافة ملفّ جديد باسم Student.cs وبداخله الصف Student.

6. نعدِّل محتوى الملف Student.cs ليصبح كما يأتي:

```
using System;
 namespace StudentProject {
     class Student  {
        // name is a field
        private string name;

        // Name is a property
        public string Name {
              get => name;
            private set => name = value;
            // it is private, it means we cannot
            // change name from outside the class
             }// end Name

        // Age is an auto_implemented property,
        // the compiler will create the field age
        public int Age { set; get; }

        //constructor with two input parameters
        public Student(string stdName, int stdAge) {
            Name = stdName;
            Age = stdAge;
        }// end constructor

       // PrintInfo is a method for printing Name and Age
        public void PrintInfo( ) {
            Console.WriteLine("The student name is: {0}", Name);
            Console.WriteLine("The student age is: {0}", Age);
        }// end method PrintInfo
     }// end class Student
 }//end namespace StudentProject
```

7. لاحظ أنّه قمنا بتعريف بانٍ للصف Student يأخذ وسيطي دخل، الأول من النمط string ويُسند قيمته إلى الخاصية Name، والثاني من النمط int ويُسند قيمته إلى الخاصية Age.
8. نكتب في الملفّ StudentTest.cs رمّاز الصف StudentTest الذي يُستخدَم لاختبار الصف وأعضائه.

```
using System;
namespace StudentProject {
    class StudentTest   {
        static void Main( ) {
             //creating the object obj1 using constructor
            // with two input parameters
            Student obj1 = new Student("Salee", 18);

            //creating the object std using constructor
            //with one input parameter
            Student std = new Student("Hany", 20);
            //try to correct the name spelling
            // obj1.Name = "Saly";
            // Error CS0272: The property or indexer 'Student.Name' cannot be
            //  used in this context because the set accessor is inaccessible

            // print object information
            obj1.PrintInfo();
            std.PrintInfo();

            // waiting for a key press to prevent the screen
            // from closing quickly
            Console.ReadKey();
         }// end Main
    }// end  class StudentTest
}// end namespace StudentProject
```

9. لاحظ أننا استخدمنا الباني الذي تمّ تعريفه لإنشاء الغرض obj1 من الصف Student مع القيم "Salee",18 كوسيطي دخل واستخدمناه مرّة ثانية لإنشاء الغرض std مع القيم "Hany ",20 كوسيطي دخل. كما قمنا بتغيير الاسم Salee إلى Saly في الغرض الأول، واستخدمنا الطريقة PrintInfo مع كل من الغرضين لطباعة معلومات كل غرض.

لقد حاولنا تنفيذ البرنامج، فظهرت رسالة الخطأ الآتية:

```
The property or indexer 'Student.Name' cannot be used in this
context because the set accessor is inaccessible
```

وتمّت الإشارة إلى التعليمة obj1.Name="Saly"; كمصدر للخطأ. ويعود السّبب في ذلك، إلى أنّ محدّد الوصول إلى الخاصية Name في حالة الإسناد هو private، أي لا يمكن إسناد قيمة للحقل name إلا من قبل أعضاء صفه، وهذا ما يُفسّر تمكّن الباني من إسناد قيمة للحقل عن طريق الخاصية Name.
وبعد إلغاء التعليمة السابقة وتنفيذ الرمّاز السابق، نحصل على الخرج الآتي:

```
The student name is: Salee
The student age is: 18
The student name is: Hany
The student age is: 20
```

لاحِظ أنه في بداية كل ملف استخدمنا الكلمة المفتاحية using متبوعة بفضاء الأسماء System، كما وضعنا كلا الصفين Student وStudentTest ضمن فضاء الأسماء StudentProject باستخدام الكلمة المفتاحية namespace. وقبل نهاية الطريقة Main، استدعينا الطريقة ReadKey باستخدام اسم الصف الساكن Console الموجود ضمن فضاء الأسماء System. وتفيد الطريقة ReadKey بمنع إغلاق الشاشة التي سيظهر عليها الخرج إلا بعد الضغط على أحد مفاتيح لوحة المفاتيح. ولمعرفة المزيد حول الأعضاء الساكنة وفضاءات الأسماء، يجب الاطلاع على الفصل القادم.

الأنشطة المرافقة

التمرين الأول: صف المستطيل Rectangle

1. قم بإنشاء الصف Rectangle، وضمنه الحقل length الدال على طوله والحقل width الدال على عرضه.
2. قم بإنشاء الخاصيتين العامتين Length وWidth، والتحقّق من كون القيم المُراد إسنادها للحقلين موجبة وإلا فيتمّ إسناد القيمة 0).
3. قم بتعريف بانٍ للصف يأخذ قيمتين كوسيطي دخل ويقوم بإسنادهما إلى الخاصيتين Length وWidth.
4. أضف إلى الصف الطريقة Perimeter التي تُعيد في خرجها محيط المستطيل والطريقة Area التي تُعيد مساحة المستطيل.
5. قم باختبار الصف السابق من خلال إنشاء عدة أغراض منه واستخدام الخصائص والطرائق المذكورة.

التمرين الثاني: صف الدائرة Circle

1. قم بإنشاء الصف Circle، وضمنه الحقل radius الدال على نصف القطر.
2. قم بإنشاء الخاصية العامة Radius، والتحقّق من كون القيمة المسندة إليها موجبة وإلا فيتمّ إسناد القيمة (0).
3. قم بتعريف بانٍ للصف يأخذ قيمة كوسيط دخل ويقوم بإسنادها إلى الخاصية Radius.
4. أضف إلى الصف الطريقة Circumference التي تُعيد في خرجها محيط الدائرة والطريقة Area التي تُعيد مساحة الدائرة.
5. قم باختبار الصف السابق من خلال إنشاء عدة أغراض منه واستخدام الخصائص والطرائق المذكورة.

التمرين الثالث: صف الزبون Customer

1. قم بإنشاء الصف Customer وضمنه الخصائص:
   - رقم الزبون Id
   - الاسم Name
   - الرصيد Balance
2. قم بتعريف بانٍ للصف يأخذ ثلاثة وسطاء دخل ويقوم بإسنادها إلى خصائص الصف.
3. أضف إلى الصف الطريقة PrintInfo التي تطبع معلومات الزبون: الرقم والاسم والرصيد.
4. قم باختبار الصف السابق من خلال إنشاء عدة أغراض منه واستخدام الخصائص والطرائق المذكورة.

المراجع

1. “Value Type and Reference Type” https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type.
2. https://docs.microsoft.com/en-us/dotnet/csharp/.
3. Dan Clark: “Beginning C# Object-Oriented Programming”, Berkeley, CA, Apress, 2013.
4. "التصميم والبرمجة غرضية التوجّه"، الدكتور سامي خيمي، الإجازة في تقانة المعلومات، من منشورات الجامعة الافتراضية السورية، الجمهورية العربية السورية، 2018.