# الدرس الثاني عشر : أساسيات البرمجة الكائنية 2

في هذا الدرس نتابع أهم أسس البرمجة الكائنية في لغة السي شارب ، و هذا الدرس سيتناول النقاط التالية :

- استخدام كلمة this
- مِجمّع النفايات
- الأعضاء الصنف الثابتة static
- الأعضاء الثوابت const و الأعضاء القابلة للقراءة فقط readonly
- المفهرسات indexers
- تكوين مكتبات خاصة بك

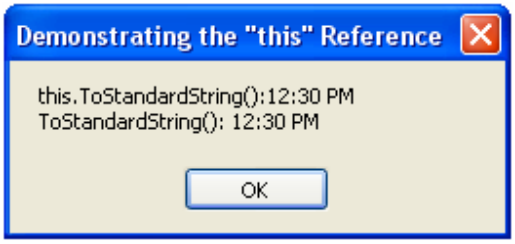
## استخدام كلمة this

في بعض الأحيان قد تريد أن تجعل أسماء معاملات إحدى الوسائل أو أسماء متغيرات محلية (خاصة بتلك الوسيلة ) في صنف ما بنفس اسم بعض المتغيرات اللحظية في ذلك الصنف ، حينئذ ستضطر إلى استخدام كلمة this ، حيث تقوم هذه الكلمة بالإشارة إلى أعضاء الصنف ، سواء كانت متغيرات أو وسائل أو خصائص ، هذا باختصار أمر هذه الكلمة ، والآن نأخذ مثال يوضح ما نقوله ، في هذا المثال سوف نكون صنف باسم الوقت كالتالي :

```
using System;
using System.Windows.Forms;

public class Time
{
        private int hour;
        private int minute;
        private int second;

        //المشيد
        public Time (int hour, int minute, int second)
        {
                this.hour = hour;
                this.minute = minute;
                this.second = second;
        }

          // هذه الوسيلة سوف تستعمل كلمة this للوصول إلى الوسائل الموجودة في الصنف
        public string BuildString ()
        {
                return "this.ToStandardString():" + this.ToStandardString()
                + "(): " + ToStandardString();
        }
```

```
        //هذه الوسيلة سوف تحول الوقت إلى نظام 12 ساعة
        public string ToStandardString()
        {
                return String.Format("{0}:{1:D2} {3}",
               ( (this.hour == 12 || this.hour == 0) ? 12 :this.hour % 12),
               this.minute, this.second, (this.hour < 12 ? "AM" : "PM" ) );

        }
}

// هنا سوف نختبر الصنف الذي كتبناه
public class ThisTest
{
        static void Main (string [] args)
        {
                Time time = new Time (12, 30, 19);

                MessageBox.Show( time.BuildString(),
                                "Demonstrating the this Reference");
        }
}
```

و هذه النتيجة :




لاحظ أننا استخدمنا كلمة this في المشيد للوصول إلى المتغيرات اللحظية في الصنف و في الوسيلة BuildString استخدمنها للوصول إلى الوسيلة ()ToStandardString ، و كذلك لاحظ أننا نستطيع أن نستغني عنها عندما نريد أن نصل إلى الوسائل ، لأنه و بكل بساطة أن مجال وصول الوسائل في صنف معين هو كل الصنف !!

## مِجمّع النفايات

قد تستغرب ما دخل مجمع النفايات بأساسيات البرمجة الكائنية ؟ و حتى تتضح لك الرؤية ؛ دعنا نراجع وظائف مجمع النفايات ، إن الوظيفة الأساسية لمجمع النفايات هو التخلص من الكائنات التي لم يعد إليها أي حاجة ، إذا هو مرتبطة بالكائنات ، هذا أولا و ثانيا أنه في أي لغة برمجية تدعم البرمجة الكائنية تشرط عند كتابة الصنف أن تكتب الهادم كما تكتب مشيد ، حيث يقوم هذا الهادم بالتخلص من كائن من الذاكرة و تحرير جميع المصادر التي كان يشغلها ( اتصال بقاعدة بيانات ، اتصال بملفات ، صور وغيرها ) ، ولغة السي شارب

ليست شاذة عن القاعدة ، ولكن بسبب وجود مجمع النفايات الذي يقوم بالتخلص بشكل آلي من الكائنات الميتة فإنه لا يوجد داعي قوي أن تكتب هادم إلا في حالة واحد هو تحرير المصادر الذي يشغلها كائن ، فكما تقوم بإيجاد و استعمال هذه المصادر بشكل يدوي ؛ فإنك يجب أن تحررها بنفسك في الهادم ، و الجدير بالذكر أنه لا يوجد وقت محدد لمجمع النفايات ليقوم بعمله ، فعندما يرى أن موارد الجهاز منخفضة سيقوم بالعمل كمهمة منفصلة عن البرنامج الأساسي ، فعندما يرى أن هناك كائن ميت سيقوم بمناداة الهادم التابع لذاك الكائن – إذا كان موجود – ثم يقوم بالتنظيف الذاكرة من ذاك الكائن.

بسبب هذه الضبابية في طريقة عمل مجمع النفايات ، يفضل أن تتعامل مع الموارد و المصادر المهمة بشكل منفصل عن مجمع النفايات ، حتى لا يحصل هدر في تلك الموارد ، وحتى تزيد من كفاءة البرنامج .

أما عن كيفية كتابة الهادم ، فبكل بساطة قم بزيادة "~" قبل اسم الصنف ، فإذا كان هناك صنف باسم Time ، وسوف يكون الهادم على هذا الشكل ~Time() ، حيث لا يوجد معاملات للهادم ، لذا لا نستطيع إعادة تعريفه، سوف نأخذ لاحقا على مثال لكيفية كتابة الهادم .

## الأعضاء الصنف الثابتة static

أخذنا في الدرس السابق نوعين من الأعضاء من حيث سماحية الوصول إليها وهي الأعضاء العامة و الأعضاء الخاصة ، والآن سوف نضيف إليها نوع ثالثا هو الأعضاء الثابتة ، و حتى يتضح دور هذا النوع لنأخذ هذا المثال ، لنفترض أنك تبرمج لعبة فيها كائنات مفترسة و هذه الكائنات تتخذ القرار بالهجوم أو الانسحاب اعتمادا على عددها ، و لنقل عندما تكون هناك خمسة كائنات ستكون في حالة هجوم ، والآن فكر في طريقة لبرمجة هذا السلوك بواسطة البرمجة الكائنية ؟ ستقول بأن الأنواع التي استعرضنها سابقا لا تفي بهذا الغرض ، فنحن نريد مؤشر موحد لجميع الكائنات ، بحيث يتحفظ بعددها ، و جميع الأنواع التي أخذنها تنشئ أعضاء غير مستقلة عن الكائنات ، فلو أنشأنا متغير بعدد الكائنات و جعلناه متغير ذو سماحية وصول عامة فإن كل كائن سوف يكون متغير خاص به !

الحل هو إنشاء نوع جديد لهذه الغاية ، و بالفعل فإن هذا النوع يسمى النوع الثابت static ، بحيث أن العضو الثابت يكون عام لجميع الكائنات ذاك الصنف ، فلو رجعنا لمثالنا لوجدنا أن كل كائن ينشئ سوف يزيد واحد على المتغير الثابت الذي يشير إلى عدد تلك الكائنات ، و كل كائن يموت سوف ينقص واحد من ذاك المتغير ، و بقية الكائنات سوف تقرأ العدد الكلي من ذاك المتغير و تتخذ القرار المناسب .

قد تسأل ألا يمكن نجعل العضو الثابت ذو سماحية وصول خاصة ؟ و الجواب نعم يمكن ذلك ؛ فالعضو الثابت شأنه شأن الأعضاء العادية يمكن تخصيص الوصول إليه عن طريق معاملات سماحية الوصول ، و يمكن الوصول إلى الأعضاء الثابتة العامة عن طريق معامل النقطة "." كما في هذا المثال (Math.PI) أما الأعضاء الثابتة الخاصة فيمكن الوصول إليها عن طريق وسائل و خصائص الصنف فقط .

تصبح الأعضاء الثابتة جاهزة للاستخدام عندما يتم تحميل الصنف في الذاكرة في وقت التنفيذ ، و تبقى حتى انتهاء البرنامج ، بطبيعة الحال الأعضاء الثابتة لا تتطلب أن يكون هناك كائن من الصنف موجود في الذاكرة ، و هنا لا بد أن تتنبه أن الأعضاء الثابتة الخاصة لا يمكن الوصول إليها مباشرة ، و حتى تمكن الوصول إليها لا بدا أن تنشئ وسيلة أو خاصية ثابتة لهذا الغرض.

و ملاحظة مهمة أن الوسائل الثابتة لا يمكن أن تصل إلى أعضاء غير ثابتين ( لحظيين ) ، ولا يمكن أن تستخدم كلمة this ؛ بسبب استقلالية الأعضاء الثابتة عن كائنات الصنف.

والآن نأتي إلى تعريف الأعضاء الثابتة ، كل المطلوب لكي تحصل على عضو ثابت أن تكتب كلمة static أمام العضو و تنتهي القضية ، ربما يخطر على بالك سؤال هل يوجد هناك مشيد ثابت ؟ دعني أفكر قليلا ، نحن نستخدم المشيد العادي في وضع القيم الابتدائية لأعضاء الصنف ، طيب إذا في ماذا سنستخدم المشيد الثابت ؟ الجواب خذ مثلا حالة أنك تريد أن تضع قيم ابتدائية للأعضاء الثابتة ، و تحتاج بعض العمليات لحساب تلك القيم الابتدائية ، ففي هذه الحالة يلزمك أن تستخدم المشيد الثابت لحساب تلك القيم ، و تتم مناداة هذا المشيد قبل أن تستخدم أي عضو ثابت و قبل أن ينشئ أي كائن من ذاك الصنف ، و يتميز المشيد الثابتة بأنه لا يمكن الوصول إليه و لا يأخذ أي متغيرات.

و هذا يقودنا إلى ذكر النوع الثالث من المشيدات ، و هو المشيد الخاص ، و تبرز قيمة هذا المشيد في حالة أنه يوجد صنف يحوي على أعضاء ثابتة فقط ، فلذا نحن نريد أن نمنع تخليق كائنات من هذا الصنف ، ففي هذه الحالة سنلجأ نضع مشيد خاص ، بحيث لا يسمح بالوصول إليه ، تذكر أنك إذا لم تكتب مشيد فإن المترجم سوف ينشئ مشيد افتراضي .

والآن سنأخذ مثال يبين كيفية إنشاء الأعضاء الثابتة و استخدامها ، بالإضافة إلى كيفية عمل الهادم ، و المثال سوف ينشئ صنف باسم الموظف Employee يحوي على ومتغيرات للاسم الأول و الأخير بالإضافة إلى عدد الموظفين .

```
using System;

public class Employee
{
	private string firstName;
	private string lastName;
	private static int count;

	//المشيد سوف يقوم بإضافة واحد لعداد الموظفين كلما تمنت مناداته
	public Employee (string fName, string lName)
	{
		firstName = fName;
		lastName = lName;

		++count;
		Console.WriteLine ("Employee object constructor:" +
			firstName + " " + lastName + "; conut = " + Count);
	}

	//الهادم سوف يقوم بإنقاص واحد من العداد الموظفين كلما تمت مناداته
	~Employee()
	{
		--count;
		Console.WriteLine( " Employee object destructor: " +
			firstName + " " + lastName + "; conut = " + Count);
	}

	//خاصية للاسم الأول
	public string FirstName
	{
		get
		{
			return firstName;
		}
	}

	//خاصية للاسم الآخير
	public string LastName
	{
		get
		{
			return lastName;
		}
	}

	//خاصية ثابتة للعداد
	public static int Count
	{
		get
		{
			return count;
		}
	}
}
```

```
using System;
class StaticTest
{
      static void Main (string [] args)
      {
            Console.WriteLine("Employee before instantiation: " +
                              Employee.Count + "\n" );

            //هنا سوف نقوم بإنشاء كائنين من صنف الموظف
            Employee employee1 = new Employee("Ahmed", "Said");
            Employee employee2 = new Employee("Zayed", "Saif");

            Console.WriteLine("after instantiation: " +
                   "Employee.Conut = " + Employee.Count + "\n" );

            // سوف نعرض الموظفين
            Console.WriteLine( "Employee 1: " + employee1.FirstName +
                  " " + employee1.LastName + "\nEmployee 2: " +
                 employee2.FirstName +" " + employee2.LastName +"\n" );

            //  هنا سوف نزيل أي مرجع للكائنين
            // مما يجعلهما قابلين للإزالة بواسطة مجمع النفايات
            employee1 = null;
            employee2 = null;

            // سوف نجبر مجمع النفايات على العمل
            System.GC.Collect();

            // ننتظر حتى ينهي مجمع النفايات عمله
            System.GC.WaitForPendingFinalizers();

            Console.WriteLine( "\nEmployee after garbage collection: "
                     + Employee.Count);
      }
}
```

و النتيجة هي :

```
Employee before instantiation: 0

Employee object constructor:Ahmed Said; conut = 1
Employee object constructor:Zayed Saif; conut = 2

Employee after instantiation: Employee.Conut = 2

Employee 1: Ahmed Said
Employee 2: Zayed Saif

 Employee object destructor: Zayed Saif; conut = 1
 Employee object destructor: Ahmed Said; conut = 0

Employee after garbage collection: 0
```

في هذا المثال استخدمنا الوسيلة GC.**Collect**() لإجبار مجمع النفايات على العمل ، و ينبغي الحذر عند استعمال هذه الوسيلة جيد عند كتابة البرامج الاحترافية ، حيث أنها قد تبطئ من سرعة عمل البرنامج ، ولكن أخذنها لغرض الدراسة ، و الأمر الثاني أننا استخدمنا الوسيلة GC.**WaitForPendingFinalizers**() حتى نتأكد من أن مجمع النفايات قام بمهمته ، فكما أشرنا سابقا أن مجمع النفايات يعمل كمهمة مستقلة عن البرنامج .

و ينبغي التنبيه أن مجمع النفايات قد لا يجمع كل الكائنات الميتة و قد يجمعها بترتيب يختلف عن الذي حصلت أنا عليه.

## الأعضاء الثوابت const و الأعضاء القابلة للقراءة فقط readonly

تتيح لغة السي شارب للمبرمجين أن ينشئوا ثوابت ( أي لا تتغير قيمتها إطلاقا) ، وهي تعطيهم الحرية في أن يختاروا متى يريدون أن يضعوا لها قيم ابتدائية ، و حتى ندخل في عمق هذه الثوابت فإن هناك نوعين منها النوع الأول هو الثوابت و تنشئ بأن يكتب كلمة const أمامها ، و هذه الثوابت لا تتغير ، و يجب أن توضع قيمتها مباشرة عند تعريفها ، أو بمعنى أدق يجب أن تكون قيمتها موجودة وقت الترجمة ، و ربما استنتجت أن الأعضاء الثوابت ما هي إلا أعضاء ثابتة static بشكل ضمني .

أما النوع الثاني فهي الأعضاء القابلة للقراءة فقط readonly ، و هذه تعطيك حرية أكبر متى سوف تقوم بوضع قيمتها ، فبالإضافة إلى إمكانية وضع قيمتها عند الإعلان عنها ، فإنك تستطيع أن تضع قيمتها في المشيد ، و متى تم وضع قيمة لها فإنها لا تتغير إطلاقًا ، و يجب ملاحظة أنك إذا قررت أن تجعل الأعضاء القابلة للقراءة أعضاء ثابتة ولم تضع قيمتها عند الإعلان عنها فإنك يجب أن تستخدم المشيد الثابت في وضع قيمها وليس المشيد اللحظي.

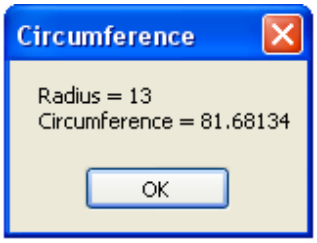
و لا تنسى أن هذه الثوابت يمكن أن تجعلها خاصة أو عامة !

والآن سوف نأخذ مثال يطبق ما قلناه ، في هذا المثال سوف نكتب صنف يقوم بحساب قيمة محيط الدائرة ، و هو يساوي طول نصف القطر مضروب في الثابت PI مضروب في اثنان.

```
using System;
using System.Windows.Forms;

public class  Constants
{
      public const double PI = 3.14159;
```

```
        // سوف ننشئ ثابت قابل للقراءة ولكن سوف نضع قيمته عندما ننشئ كائن من هذا الصنف
        public readonly int radius;

        public Constants (int radiusValue)
        {
                radius = radiusValue;
        }

  }

public class UsingConstAndReadonly
{
        static void Main ()
        {
                // سوف نضع قيمة عشوائية لطول نصف القطر
                Random random = new Random();

                Constants constantValue = new Constants(random.Next(1,20));

                MessageBox.Show("Radius = " + constantValue.radius +
                        "= " + 2 * Constants.PI * constantValue.radius,
                    "Circumference");
        }
}
```

و النتيجة هي :




الآن أترك لك مهمة أن تختبر هل صحيح ما قلنا أم لا ، فحاول أن تغير قيمة PI و قيمة radius من خلال وسيلة Main .

## المفهرسات indexers

هل مللت من المصفوفة العادية و تريد أن تنشئ مصفوفة على مزاجك ؟ مثلا بدلا عن أن يكون المفهرس رقما لما لا يكون اسما ؟ و ما رأيك بأن ترجع قيم رقمية بدلا عن قيم حرفية في مصفوفة حرفية ؟ كل هذه المميزات تقدمها لك السي شارب فيما يسمى المفهرسات ، حيث تستطيع أن تفهرس المعلومات التي يحتويها الكائن على نفس النسق الذي يوجد في المصفوفات.

```
معامل سماحية الوصول **نوع الرجع** this [نوع المفهرس 1 name1, نوع المفهرس2 name2, …]
{
        get
        {
                // استعمل أسماء المفهرسات لكي ترجع البيانات
        }
        set
        {
                // استعمل أسماء المفهرسات لكي تضع البيانات
        }
}
```

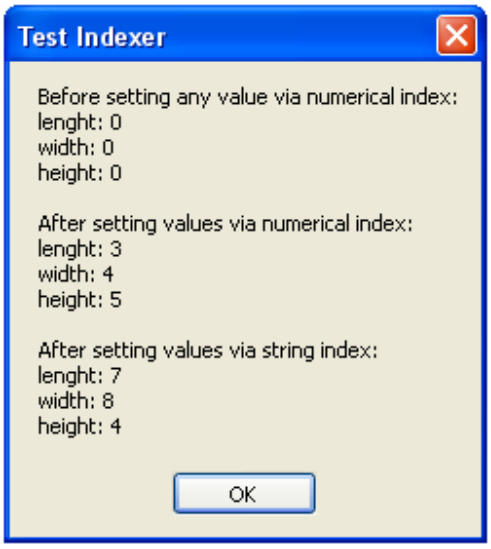
فكما تلاحظ بأننا نستطيع أن نكون مصفوفات تكون لنا السلطة في تحديد عدد المفهرسات و أنواعها ، و في نوعية الرجع لهذه المصفوفة ، و حتى يثبت مفهوم المفهرسات دعنا نأخذ مثال عليها ، سوف ننشئ صنف باسم صندوق Box ، يحتوى على ثلاث متغيرات وهي الطول و العرض والارتفاع ، و استعملنا المفهرسات حتى نستطيع أن نصل إلى هذه المتغيرات على هيئة المصفوفات .

```
using System;
using System.Windows.Forms;

public class Box
{
        // سوف ننشئ مصفوفتين أحدهما للأبعاد و الأخرى للأسماء
        private string[] names ={ "length", "width", "height"};
        private double [] dimensions = new double [3];

        // المشيد
        public Box(double length, double width, double height)
        {
                dimensions[0] = length;
                dimensions[1] = width;
                dimensions[2] = height;
        }

        //هنا سوف ننشئ مفهرس رقمي
        public double this [int index ]
        {
                get
                {
                        // سنختبر إذا كان المفهرس أكبر من الصفر و أصغر من طول مصفوفة الأبعاد
                        return (index < 0 || index >= dimensions.Length ) ?
                         -1 : dimensions [index];
                }
```

```
            set
            {
                  // سنختبر إذا كان المفهرس أكبر من الصفر و أصغر من طول مصفوفة الأبعاد
                  if (index >= 0 && index < dimensions.Length)
                        dimensions[ index ] = value;
            }
      }

      // هنا سوف ننشئ مفهرس حرفي
      public double this [string name]
      {
            get
            {
                  //   سننشئ متغير محلي
                  int i = 0;

                  // سنبحث عن اسم في مصفوفة الأسماء يتطابق مع المفهرس
                  while (i < names.Length && name.ToLower()
                        != names[i])
                        i++;
                  return (i == names.Length ) ? -1 : dimensions[i];
            }

            set
            {
                  //سننشئ متغير محلي
                  int i = 0;

                  while (i < names.Length && name.ToLower()
                         != names[i])
                        i++;

                  if (i != names.Length )
                        dimensions[i] = value;

            }
      }
}

// هنا سوف نكتب تطبيق سيختبر صنف الصندوق
class TestIndexers
{
      static void Main ()
      {
            string output;

            Box box = new Box (0.0, 0.0, 0.0);

            // أولا سنختبر المفهرس الرقمي

            // سنعرض القيم الابتدائية

            output ="Before setting any value via numerical index: : "
                + box[0] +": " + box[1] + ": " +box[2];
```

```
            // سنضع القيم عن طريق المفهرس الرقمي

            box[0] = 3;
            box[1] = 4;
            box[2] = 5;

            //بعد وضع القيم

            output +="setting values via numerical index: : " + box[0]
                   +": " + box[1] + ": " + box[2];

            // ثانيا سنختبر المفهرس الحرفي

            // سنضع قيم جديدة عن طريق المفهرس الحرفي
            box["length"] = 7;
            box["width"] = 8;
            box["height"] = 4;

            // سنعرض القيم الجديدة عن طريق المفهرس الحرفي
            output +="setting values via string index: : " +
             box["length"] +": " + box["width"] + ": " + box["height"];
            MessageBox.Show(output,"Test Indexer");
        }
}
```

و النتيجة هي




و كالعادة أريد منك أن تتسلى قليلا مع هذه المثال و تضع قيم خاطئة و ترى النتيجة .

## تكوين مكتبات خاصة بك

من المعلوم في صناعة البرمجيات أنه لا بد من تنظيم الشفرات التي كتبت حتى يسهل الاستفادة منها لاحقا ، و يتم ذلك عن طريق تجميع الشفرات المتشابه في مكتبات

برمجية ، و عند الحاجة إليها يتم استدعائها ، على سبيل المثال مكتبة Windows.Forms ، فهذه المكتبة تمت كتابتها مرة واحدة ، و كلما دعت الحاجة إليها قمنا باستيرادها و استعملها في برامجنا ، تتيح مترجمات لغة السي الشارب أن تكون مكتباتك المستقلة ، إذا كنت تستعمل الفيجوال استديوا أو شارب دفلوب فكل ما عليك هو إنشاء مشروع جديد من نوع Class Library و بعد ذلك قم بكتابة الأصناف و الوسائل التي ترغب في جعلها في المكتبة الجديدة ، ولا تنسى أن تستعمل اسم فضاء خاص بك كما شرحنا في الدرس الماضي ، و من ثم قم بترجمتها .

والآن عندما تريد استعمالها قم أولا باستيرادها كما تفعل مع مكتبة Windows.Forms و استعملها كأي مكتبة ! ، أما الذي يستعملون السطر الأوامر في ترجمة برامجهم فليراجعوا ملفات المساعدة ليجدوا خيار ترجمة الملف كمكتبة .

## تمرين بسيط للمراجعة

قم بكتابة صنف جديد للأرقام المركبة يقوم ببعض العمليات الحسابية ( الجمع و الطرح) ، حيث أن الأرقام المركبة تتكون من قسمين القسم الحقيقي و القسم التخيلي على هذا الشكل :

القسم الحقيقي + القسم التخيلي * i

و تتم عملية الجمع بين كعددين مركبين بجمع كل القسم الحقيقي مع القسم الحقيقي و القسم التخيلي مع القسم التخيلي ، و كذلك عملية الطرح .

زايد السيد
منتدى الإبداع الإسلامي