التغليف والتحميل الزائد
Encapsulation and Overloading

العنوان	رقم الصفحة
مقدمة	3
1. التغليف	4
2. التحميل الزائد للباني	7
3. التحميل الزائد للمعاملات	13
4. الأنشطة المرافقة	21

الكلمات المفتاحية

التغليف، محدّدات الوصول، التحميل الزائد.

ملخص الفصل

نبيّن في هذا الفصل مفهوم التغليف الذي يسمح للمبرمج بالتحكّم بالوصول إلى أعضاء صفٍ ما من قبل المكوّنات البرمجية الأخرى، ونوضّح كيفية القيام بالتحميل الزائد لباني الصف وللمعامِلات.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- استخدام محدّدات الوصول لتغليف أعضاء الصف
- القيام بالتحميل الزائد لباني الصف
- القيام بالتحميل الزائد للمعامِلات

مقدمة

تقوم البرمجة الإجرائية على توافر العديد من الإجرائيات التي تستدعي بعضها البعض وتتشارك الكثير من المعطيات ويمكن لأي منها تعديل هذه المعطيات. ويناسب هذا الأسلوب من البرمجة التطبيقات الصغيرة نسبياً من حيث عدد أسطر الرمّاز المصدري، ويصبح غير عمليّ من أجل التطبيقات المتوسطة والكبيرة. فإجراء أي تغيير في مثل هذه التطبيقات على قيم المعطيات المتشاركة يحتاج إلى الكثير من التأني والحذر، ويتطلب التفكير بكافة الإجرائيات التي تتشارك هذه المعطيات. وتم حل هذه المسألة في البرمجة غرضية التوجه من خلال تجميع المعطيات الخاصة بغرض ما مع الطرائق التي تُطبَق عليها ضمن بنية واحدة. ولم يقتصر الأمر على ذلك، بل أصبح بالإمكان التحكم بالوصول إلى معطيات كل غرض من خلال التغليف Encapsulation. وبما أن البرمجة غرضية التوجه تدعم بشدة إعادة استخدام الرمّاز البرمجي، فمن الطبيعي أن يستخدم غرض من صفٍ ما أعضاء تابعة لصف آخر، ومن الطبيعي أن يختلف هذا الاستخدام من غرض لآخر، ومن الطبيعي أيضاً أن يكون لدينا عدة أشكال لباني الصف. وهذا ما سُمي بالتحميل الزائد للباني Constructor Overloading.

1. التغليف

في البرمجة غرضية التوجّه، يُقصد بالتغليف منع وصول صف إلى أعضاء صف آخر، حيث يمكن التحكّم بصلاحيات الوصول إلى أعضاء كل صف من خلال استخدام محدّد الوصول Access Specifier المناسب مع كلّ منهم.

نوضّح في الجدول الآتي الكلمات المستخدمة للتعبير عن محدّدات الوصول مرفقة بشرحٍ موجزٍ عنها:

محدّد الوصول	توصيف
Private	الوصول إلى العضو مقتصر على الأعضاء الموجودة داخل صفه
Protected	الوصول إلى العضو مقتصر على الأعضاء الموجودة داخل صفه وعلى أعضاء الصفوف الوارثة من ذلك الصف
Internal	الوصول إلى العضو مقتصر على المكوّنات الموجودة ضمن نفس التجمّع Assembly الذي يضمّ صفه
Public	الوصول إلى العضو غير محدود بالنسبة لجميع أعضاء الصفوف التي يمكنها الوصول إلى صفه

وعند عدم التصريح عن محدّد الوصول إلى الأعضاء، يكون محدّد الوصول private. أمّا بالنسبة للصف ككلّ، فعند عدم التصريح عن محدّد الوصول إليه يكون internal، أي يمكن الوصول إليه من ضمن التجمّع الخاص به. والتجمّع هو وحدة برمجية (ذات اللاحقة .exe أو .dll) تنتج عن ترجمة الرمّاز المصدري وتكون بلغة مايكروسوفت الوسيطة MSIL.

مثال:

في الرمّاز الآتي، نوضّح كيفية استخدام محدّدي الوصول private وpublic.

```
using System;
namespace Encapsulation
{
    class Rectangle
    {
        //member variables
        double length;
        double width;
        // member method
        double GetArea()
```

```
        {
            return length * width;
        }

    }//end class Rectangle
    class RectangleTest
    {
        static void Main()
        {
            Rectangle rec = new Rectangle();
            // accessing rec members
            rec.length = 5.0;
            rec.width = 4.0;
            Console.WriteLine("Area: {0}", rec.GetArea());
            Console.ReadLine();
        }// end Main
    }// end class RectangleTest
}// end namespace Encapsulation
```

وعند تنفيذ الرمّاز، نحصل على رسائل الخطأ التالية:

```
'Rectangle.length' is inaccessible due to its protection level
'Rectangle.width' is inaccessible due to its protection level
'Rectangle.GetArea()' is inaccessible due to its protection level
```

والتي تفيد بأنّه لا يمكن الوصول إلى العناصر length وwidth وGetArea الموجودة ضمن الصف Rectangle وذلك لأنّ محدّدات الوصول إلى هذه الأعضاء من النمط private ولا يمكن الوصول إليها من خارج الصف المعرَّفة ضمنه.

إذا جعلنا محدّد الوصول إلى هذه الأعضاء public:

```
class Rectangle
{
    //member variables
    double length;
    double width;
    // member method
```

```
    double GetArea()
    {
        return length * width;
    }
```

وأعدنا تنفيذ الرمّاز السابق، حُلّت هذه المسألة، وأصبح الوصول إلى أعضاء المستطيل متاحاً، ونحصل على النتيجة الآتية:

```
Area: 20
```

2. التحميل الزائد للباني

يتمّ استدعاء باني صف ما عند إنشاء غرض منه، ويقوم الباني بتنفيذ تعليمات خاصّة بتهيئة الغرض. وقد يحتوي الصف على عدّة بناة يحملون جميعهم نفس الاسم ويختلفون في عدد وسائط الدخل و(أو) أنماط معطياتها، وبحيث يكون لكل بانٍ منهم طريقته الخاصّة لتهيئة الغرض، ويُطلق على هذه الحالة التحميل الزائد للباني.

مثال:

وفي الرمّاز الآتي، نوضّح كيفية استخدام التحميل الزائد لباني الصف Time:

```
// Time.cs
// Time class declaration with overloaded constructors.
using System;
namespace ConstructorOverloading
{
    class Time
    {
        private int hour; // 0 - 23
        private int minute; // 0 - 59
        private int second; // 0 - 59
                            // constructor with zero argument
        public Time()
        {
            SetTime(0, 0, 0); // invoke SetTime to validate time 00:00:00
        } // end Time default constructor
          //constructor with one argument
        public Time(int h)
        {
            SetTime(h, 0, 0); // invoke SetTime to validate time
        } // end constructor with one argument
          //constructor with two arguments
        public Time(int h, int m)
        {
            SetTime(h, m, 0); // invoke SetTime to validate time
        } // end constructor with two arguments
          //constructor with three arguments
        public Time(int h, int m, int s)
```

```
        {
            SetTime(h, m, s); // invoke SetTime to validate time
        } // end constructor with three arguments
        // Time constructor: another Time object supplied as an argument
        public Time(Time 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
                    Console.WriteLine("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
                    Console.WriteLine("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
                    Console.WriteLine("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 string ToStandardString()
        {
            return string.Format("{0}:{1:D2}:{2:D2} {3}",
               ((Hour == 0 || Hour == 12) ? 12 : Hour % 12),
                Minute, Second, (Hour < 12 ? "AM" : "PM"));
        } // end method ToStandardString
    } // end class Time
    public class TimeTest
    {
        public static void Main()
        {
            Time t1 = new Time(); // 00:00:00
            Time t2 = new Time(2); // 02:00:00
            Time t3 = new Time(21, 34); // 21:34:00
```

```
            Time t4 = new Time(12, 25, 42); // 12:25:42
            Time t5 = new Time(t4); // 12:25:42
            Time 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.ToStandardString());
            // 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.ToStandardString());
            // 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.ToStandardString());
            // 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.ToStandardString());
            // 12:25:42 PM
            Console.WriteLine("t5: Time object t4 specified");
            Console.WriteLine(" {0}", t5.ToUniversalString());
            // 12:25:42
            Console.WriteLine(" {0}", t5.ToStandardString());
            // 12:25:42 PM
            // attempt to initialize t6 with invalid values
            t6 = new Time(27, 74, 99); // invalid values become 00:00:00
            Console.WriteLine(" {0}", t6.ToUniversalString());
            Console.ReadKey();
        } // end Main
    } // end class TimeTest
}// namespace ConstructorOverloading
```

تمّ تعريف خمسة بُناة:

1. الباني الافتراضي Time() الذي لا يقبل وسائط دخل ويسند القيمة (0) كقيمة افتراضية لكلّ من الخصائص Hour وMinute وSecond من خلال استدعاء الطريقة SetTime.
2. الباني Time(int h) الذي يقبل وسيط دخل واحد يتمّ إسناده إلى الخاصّية Hour، ويسند القيمة (0) كقيمة افتراضية لكلّ من الخاصّيتين Minute وSecond من خلال استدعاء الطريقة SetTime.
3. الباني Time(int h, int m) الذي يقبل وسيطي دخل يتمّ إسناد قيمتيهما إلى الخاصّية Hour والخاصّية Minute على الترتيب، ويسند صفراً إلى الخاصّية Second من خلال استدعاء الطريقة SetTime.
4. الباني Time(int h, int m, int s) الذي يقبل ثلاثة وسائط دخل يتمّ إسنادها إلى الخصائص Hour وMinute وSecond على الترتيب من خلال استدعاء الطريقة SetTime.
5. الباني Time(Timetime):this(time.Hour, time.Minute,time.Second){} الذي يقبل كوسيط دخل غرضاً من الصف Time، ويستخدم الكلمة المفتاحية this لاستدعاء الباني الذي يأخذ ثلاثة وسائط دخل. ويتمّ نسخ قيم الخصائص Hour وMinute وSecond للغرض المُدخَل إلى مقابِلاتها في الغرض المُراد إنشاؤه بواسطة هذا الباني، ولذلك يُسمّى أحياناً بالباني الناسخ.

وبعد تنفيذ الرمّاز السابق، نحصل على الخرج الآتي:

```
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: Time object t4 specified
12:25:42
 12:25:42 PM

Hour  27  Hour must be 0-23
```

```
Minute  74  Minute must be 0-59
Second  99  Second must be 0-59
00:00:00
```

نلاحظ أنّه تمَّ الاستدلال على الباني المطلوب تنفيذه في كلّ مرّة من خلال وسائط الدخل التي يتمّ تمريرها إلى الباني. قم باستبدال البناة الأربعة الأُول بالباني الآتي:

```
// constructor can be called with zero, one, two or three arguments
public Time(int h = 0, int m = 0, int s = 0)
{
    SetTime(h, m, s); // invoke SetTime to validate time
} // end constructor
```

ثمّ أعد تنفيذ البرنامج من دون تغيير محتوى الطريقة Main، فستحصل على نفس الخرج السابق.

ملاحظة: عندما يتمّ تعريف بانٍ واحد أو أكثر من قبل المبرمج، لا يُسمح باستخدام الباني الافتراضي (الذي لا يمتلك وسائط دخل) من دون تعريفه.

3. التحميل الزائد للمعاملات

من المعروف أنّ معامل الجمع (+) يُستخدم للقيام بعملية جمع لقيم عددية، ويمكن استخدامه أيضاً لتجميع سلاسل محرفية مع بعضها. وفي لغة C#، يمكن إعادة استخدام بعض المعاملات بحيث تُطبَّق على أغراض منتسخة من صفوف مُحدَثة وتقوم بوظائف يحدّدها المبرمج. وبذلك تكتسب هذه المعاملات المقدرة على القيام بوظائف إضافية، وهذا ما يُسمّى بالتحميل الزائد للمعاملات Operator Overloading. وعند استدعاء أي من المعاملات، يتمّ استدعاء الطريقة الموافقة له. وتكون الطرائق الموافقة للمعاملات عامة ساكنة public static، وأسماؤها تبدأ بالكلمة operator، فالطريقة الموافقة لمعامل الجمع مثلاً هي operator+.

مثال1:

نوضّح في هذا المثال، كيفية القيام بتحميل زائد لمعاملات الجمع (+) والطرح (−) والجداء (*) بحيث يمكن استخدامها مع أغراض من الصف ComplexNumber المعبِّر عن الأعداد العقدية. وتمّ استخدام العلاقات الرياضية الآتية:

```
(X1 + Y1 i) + (X2 + Y2 i) = (X1 + X2) + (Y1 + Y2) i
(X1 + Y1 i) - (X2 + Y2 i) = (X1 - X2) + (Y1 - Y2) i
(X1 + Y1 i) * (X2 + Y2 i) = (X1*X2 – Y1*Y2) + (X1*Y2 + X2*Y1) i
```

حيث: (X1 + Y1 i) يمثّل عدداً عقدياً جزؤه الحقيقي Real هو X1، وجزؤه التخيّلي Imaginary هو Y1، و(X2 + Y2 i) يمثّل عدداً عقدياً جزؤه الحقيقي X2، وجزؤه التخيّلي Y2.

```
// ComplexNumbers.cs
// Class that overloads operators for adding, subtracting
// and multiplying complex numbers.
using System;
namespace ComplexNumbers
{
    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 r, double i)
        {
            Real = r; Imaginary = i;
        } // end constructor
```

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

        // overloading the addition operator
        public static ComplexNumber operator +(ComplexNumber x,
                                               ComplexNumber y)
        {

            return new ComplexNumber(x.Real + y.Real, x.Imaginary +
                 y.Imaginary);
        } // end operator +

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

        // overloading 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

    // class ComplexTest
    public class ComplexTest
    {

        public static void Main()
        {
```

```
            // 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);
            Console.WriteLine("x =  " + x.PrintComplexNumber());

            // 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);
            Console.WriteLine("y =  " + y.PrintComplexNumber());

            // display the results of calculations with x and y
            Console.WriteLine();
            Console.WriteLine("Addition result : " +
                                (x + y).PrintComplexNumber());
            Console.WriteLine("Subtraction result : " +
                                (x - y).PrintComplexNumber());
            Console.WriteLine("Multiplication result : " +
                                (x * y).PrintComplexNumber());
            Console.ReadKey();
        } // end method Main
    } // end class ComplexTest
}// end namespace ComplexNumbers
```

وبعد التنفيذ من أجل القيم المدخلة 5+3i, 8-2i نحصل على الخرج الآتي:

```
Enter the real part of complex number x: 5
Enter the imaginary part of complex number x: 3
x =  5 + 3 i
Enter the real part of complex number y: 8
Enter the imaginary part of complex number y: -2
y =  8 - 2 i
Addition result : 13 + 1 i
Subtraction result : -3 + 5 i
Multiplication result : 46 + 14 i
```

مثال2:

وفي الرمّاز الآتي، نوضِّح كيفية التحميل الزائد لمعاملات المقارنة وكيفية التحميل الزائد لمعامل فردي أي يتعامل مع وسيط واحد فقط كالمعامل (–) الذي يبدّل إشارة الوسيط بين الموجب والسالب.

لكلّ معامل مقارنة يوجد معامل معاكس، ولذلك يجب القيام بالتحميل الزائد للمعامل ولمعاكسه إذا أردنا إجراء التحميل الزائد لأحدهما. فإذا أردنا إجراء التحميل الزائد للمعامل (>)، يجب إجراء التحميل الزائد للمعامل المعاكس (<).

وفي حال أردنا إجراء التحميل الزائد لمعامل (==)، لا يكفي إجراء التحميل الزائد للمعامل المعاكس (=!) وإنّما يجب إعادة تعريف الطريقة Object.Equals التي يرثها كلّ صف من الصف الأساس Object. ويعود السّبب في ذلك، لمنع حدوث تعارض بين وظيفة الطريقة Equals ومعامل المساواة. وكذلك الأمر بالنسبة للطريقة Object.GetHashCode المستخدمة لتحديد غرض في جدول التقطيع Hash Table، فإذا أعطت الطريقة Equals النتيجة true عند مقارنة غرضين، يجب أن تكون لهما نفس قيمة التقطيع أيضاً. ولذلك عند التحميل الزائد للطريقة Equals، يجب إجراء التحميل الزائد للطريقة GetHashCode.

وعند تطبيق المعامل (–) الأحادي على العدد العقدي (X1 + Y1 i) سينتج العدد العقدي

((-X1) + (-Y1) i)

ويتساوى عددان عقديان إذا تحقّق الشّرطان الآتيان معاً:

- الجزء الحقيقي للعدد الأول يساوي الجزء الحقيقي للعدد الثاني
- الجزء التخيّلي للعدد الأول يساوي الجزء التخيّلي للعدد الثاني

وإذا لم يتحقّق أي من الشّرطين السابقين فالعددان غير متساويين.

```
using System;
namespace OperatorOverloading
{
    class Point
    {
        public int X { get; private set; }
        public int Y { get; private set; }
        //constructor
        public Point(int x, int y)
        {
            X = x;
            Y = y;
        }// end constructor
        public static bool operator ==(Point point1, Point point2)
        {
            if (point1.X != point2.X) return false;
            if (point1.Y != point2.Y) return false;
            return true;
        } // end of operator ==

        public static bool operator !=(Point point1, Point point2)
        {
            if (!(point1 == point2)) return true;
            return false;
        }// end of operator !=

        public static Point operator -(Point p)
        {
            Point newPoint = new Point(p.X, p.Y);
            newPoint.X = -(p.X);
            newPoint.Y = -(p.Y);

            return newPoint;
        }// end of operator-

        // we use override, we will study the overriding conept later
```

```
        public override bool Equals(object obj)
        {
            Point p = (Point)obj;
            return (this == p);
        }// end Equals
        public override int GetHashCode()
        {
            //we use Tuple<T1,T2>, this subject is out of our study scope
            return Tuple.Create(X, Y).GetHashCode();

            //or we can use :
            // return (X.GetHashCode() ^ Y.GetHashCode());
            // ^ is Binary XOR Operator which
            // copies the bit if it is set in one operand but not both.
            // in this situation, the (5,2) and (2,5) rturns the same
            // hash code value
        }// end GetHashCode
    }//end class Point

    class PointTest
    {
        public static void Main()
        {
            Point point1, point2, point3;
            int x, y;
            Console.WriteLine("Enter your point1.x = ?");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter your point1.Y = ?");
            y = Convert.ToInt32(Console.ReadLine());
            point1 = new Point(x, y);
            Console.WriteLine("Enter your point2.x = ?");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter your point2.Y = ?");
            y = Convert.ToInt32(Console.ReadLine());
            point2 = new Point(x, y);
            Console.WriteLine("Enter your point3.x = ?");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter your point3.Y = ?");
            y = Convert.ToInt32(Console.ReadLine());
```

```
            point3 = new Point(x, y);
            Console.WriteLine("*************************************");

            Console.WriteLine("The point1 is ({0} ,{1})", point1.X, point1.Y);
            Console.WriteLine("The point2 is ({0} ,{1})", point1.X, point2.Y);
            Console.WriteLine("The point3 is ({0} ,{1})", point3.X, point3.Y);

            Console.WriteLine("*************************************");



            if (point1 == point2)
                System.Console.WriteLine("point1 and point2 are at the same
coordinates.");
            else
                System.Console.WriteLine("point1 and point2 are not at the same
coordinates.");
            //if (point1 == point3)
            if (point1.Equals(point3))
                System.Console.WriteLine("point1 and point3 are at the same
coordinates.");
            else
                System.Console.WriteLine("point1 and point3 are not at the same
coordinates.");

            //Testing hash code values
            Console.WriteLine("\n\n");
            Console.WriteLine("point1 hash code is " + point1.GetHashCode());
            Console.WriteLine("point2 hash code is " + point2.GetHashCode());
            Console.WriteLine("point3 hash code is " + point3.GetHashCode());

            Console.WriteLine("\n\nThe point1 is ({0} ,{1})", point1.X, point1.Y);
            point1 = -point1;
            Console.WriteLine("The new point1 (-point1) is ({0} ,{1})", point1.X,
point1.Y);

            Console.ReadKey();
        }// end Main
    }//end PointTest
}//end namespace OperatorOverloading
```

بعد تنفيذ البرنامج، نحصل على الخرج الآتي:

```
Enter your point1.x? =
5
Enter your point1.Y? =
6
Enter your point2.x? =
2
Enter your point2.Y? =
3
Enter your point3.x? =
5
Enter your point3.Y? =
6
**************************************
The point1 is(5, 6)
The point2 is(5, 3)
The point3 is(5, 6)
**************************************
point1 and point2 are not at the same  coordinates.
point1 and point3 are at the same coordinates.

point1 hash code is 163
point2 hash code is 65
point3 hash code is 163

The point1 is(5, 6)
The new point1 (-point1) is (-5, -6)
```

إن إعادة استخدام الرمّاز المصدري في البرمجة غرضية التوجه فرضت مفهوم التغليف، وأدت إلى ضرورة القيام بالتحميل الزائد للبناة والمعاملات والطرائق. واقتضت استخدام صفٍ لغرض من صف آخر وهو ما يُعرف بالتركيب والذي سيكون موضوع الفصل القادم.

الأنشطة المرافقة

التمرين الأول: صف الأعداد العقدية ComplexNumber

لقد تعلّمت كيفية إجراء التحميل الزائد لمعاملات الجمع (+) والطرح (−) والجداء (*) من أجل استخدامها مع أغراض من الصف `ComplexNumber` المعبِّر عن الأعداد العقدية.

1. قم بإضافة التحميل الزائد لمعامل القسمة (/) إلى الصف المذكور علماً أنّ عملية قسمة عددين عقدين موضّحة فيما يأتي:

$$\frac{p+iq}{r+is}=\frac{pr+qs}{r^2+s^2}+i\frac{qr+ps}{r^2+s^2}$$

2. إذا عرّفنا عملية مقارنة بين عددين عقديين على النحو التالي:
يكون عدد عقدي أول أكبر أو يساوي (>=) عدداً عقدياً ثانياً إذا كان مطال العدد الأول أكبر أو يساوي مطال العدد الثاني.
قم بإضافة التحميل الزائد للمعامل (>=) إلى الصف `ComplexNumber`.
3. اكتب صفاً بلغة C# وسمِّهِ `ComplexNumberTester`، وضمّنه الطريقة `Main` واستخدمه لاختبار الطرائق التي أضفتها.

التمرين الثاني: صف عدّاد المسافة DistanceCounter

نعرِّف عدّاد المسافة بأنّه غرض يُستخدم لتخزين مسافة ما، ويسمح بتحويل المسافة من متر إلى كيلو متر وبالعكس، كما يسمح بالمقارنة بين عدّادين من حيث المسافة.

1. اكتب صفاً بلغة C# وسمِّهِ `DistanceCounter` للدلالة على عدّاد مسافة، بحيث يحتوي الصف على الأعضاء التالية:
   - الخاصّية `Meter` والخاصّية `Kilometer`.
   - الباني الافتراضي الذي يمنح قيماً صفرية للحقلين السابقين.
   - بانٍ يقبل قيماً ابتدائية للحقلين السابقين كوسيطي دخل له.
   - الطريقة `PrintInfo` التي تسمح بكتابة عدّاد المسافة وفقاً للصيغة `KM _ _ : M _ _`.
   - الطريقة `GetDistance` التي تأخذ عدداً يمثّل المسافة مقدّرة بالمتر كوسيط دخل لها وتقوم بتحويله إلى صيغة عدّاد المسافة `KM _ _ : M _ _` (مثال `5400 Meter => KM 5 : M 400`).
2. قم بإضافة التحميل الزائد للمعامل (−) بحيث يسمح بحساب الفرق بين عدّادي مسافة.
3. قم بإضافة التحميل الزائد للمعامل (>) بحيث يسمح بالمقارنة بين عدّادي مسافة.
4. اكتب صفاً بلغة C# وسمِّهِ `DistanceCounterTester` وضمّنه الطريقة `Main` واستخدمه لاختبار طرائق الصف `DistanceCounter` التي قمت بتعريفها.

التمرين الثالث:

اقرأ بتمعّن الرمّاز الآتي، واكتب الخرج الناتج عن تنفيذه

```
using System;
namespace Test
{
    class Sample
    {
        int a; int b;

        public Sample(int a)
        {
            this.a = a; b = 0;
        }

        public void PrintInfo()
        {
            Console.WriteLine("a = {0} , b = {1} ", a, b);
        }

    }// end class Sample
    class program
    {
        public static void Main()
        {
            Sample s1 = new Sample(5);
            Sample s2 = new Sample();
            s1.PrintInfo();
            s2.PrintInfo();
        }// end Main
    }// end class program Sample
}//end Test
```

المراجع

1. https://docs.microsoft.com/en-us/dotnet/csharp/
2. Dan Clark: Beginning C# Object-Oriented Programming, Berkeley, CA, Apress, 2013.