Sofia University
Department of Mathematics and Informatics

Course : OO Programming with C#.NET

Date: November 11, 2016

Student Name:

Lab No. 6

Submit the all C# .NET files developed to solve the problems listed below. Use comments and Modified-Hungarian notation.

Problem No. 1

A. Create a project `geometry.dll` of type `Class Library` with the following classes.

Class **Point has an `array` of two integer elements** - the **x** and **y** coordinates. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get properties** for the class data members, as well as a **`ToString`**() method.

Code a **`class Rectangle`**. The class ***has*** **two private double data members**- **`length`** and **`width`**. Additionally, it ***has*** **a `Point`-** the **`left lower point`** of the **`Rectangle`**.

Let each one of the instances of **`class Rectangle`** have unique 6- digit code referred to as R_ID.

Define a **`default`** constructor, a **`general purpose`** constructor and a **`copy`** constructor.

Define properties for the data members and an **`indexer`** using characters **`('x', 'y', 'w', 'h')`** as arguments to return the values of the data members.

Define `static` methods **`Area`** ***`(Rectangle r)`*** and **`Diagonal`*`(Rectangle r)`*** to return the **`area`** and the **`diagonal`** of the rectangle, respectively.

Define a **`ToString`**() method to **`return`** the current values of the data members as a **`string`**, as well as, the **`area`** and the **`diagonal`** of the **`rectangle`** formatted with 2 digits after the **`decimal`** point.

Define a public **`delegate CompareBy`** to allow referencing methods as **`Area`** ***(*`Rectangle r`*)*** and **`Diagonal`*(*`Rectangle r`*).***

Add to **`class Rectangle`** a **`static`** method **`SortBy`** to **`sort`** (**use `LINQ`**) a **`List`** of **`Rectangle`** objects by the value returned from the method referenced by a **`CompareBy`** delegate object i.e.

```
public static IEnumerable<Rectangle> SortBy(List<Rectangle> list, CompareBy
compare)
```

B. **Create a new C# project** of type Console Application and add to its References

1. Add **an extension method** **`Perimeter()`** to **`class Rectangle`** method that returns the perimeter of the **`Rectangle`** instance.
2. Add **an extension method** **`IsSquare()`** to **`class Rectangle`** method that returns the true or false in case the **`Rectangle`** instance is a square or not a square
3. Add **an extension method** **`Move()`** to **`class Rectangle`** method that translates the **`Rectangle`** instance to another Point object provided as a parameter to the method
4. Add **an extension method** **`Scale()`** to **`class Rectangle`** method that enlarges the length and the width of the **`Rectangle`** instance by a scale factor provided as a parameter to the method

5. Write Main() method class **GeometryTest** **with the following funcitonality** :

   - create a **`List`** of 4 **`Rectangle`** objects using random values for the data members
   - Display the **`List`** sorted by the **`Area`** of the list elements
   - Display the **`List`** sorted in descending order by the **`Diagonal`** of the list element
   - Use LINQ and display groups of the **`List`** elements with **`Perimeter`** above and below 20 using the **extension method `Perimeter()`** and Lambda expressions
   - Translate all the **`Rectangle`** objects to a Point with coordinates x= 10, y= 10 and printout on the Console the data members of these objects using the ToString() method of **`class Rectangle.`** Use the **`indexer`** to access the coordinates of the Point object
   - Scale the length and the width of all the **`Rectangle`** objects by a scale factor of 2.5 and printout on the Console the data members of these objects using the ToString() method of **`class Rectangle.`** Use the **`indexer`** to access the coordinates of the Point object

Problem No. 2

Create a project of type `Class Library` with the following classes.

`class Point` has an `array` of two integer elements - the **x** and **y** coordinates. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get properties** for the **class** data members, as well as a **ToString** () method. (reuse **class Point** from **Problem** 1)

Next, write a **class Rectangle**. Design it as follows: **Rectangle** has an **array of two Points as elements- the first Point element defines the upper left corner and** the second **Point element** defines the lower right corner of the rectangle. Define a **full set of constructors** (**default**, **general purpose** and a **copy** constructor), **properties** for the class data members, as well as, a **ToString**() method (reuse the **ToString**() method defined for **class Point**). Write additionally a ***double*** **Perimeter*()*** method allowing to compute the **perimeter** of objects **Rectangle**.

Compile **Point- Rectangle** classes **as a Class Library**.

Write a Console application to **test assembly Point- Rectangle**.

Write an **extension method CircleArea** for class **Rectangle** to compute the **area** of the circle drawn around the current rectangle object

In the **Main** method create two **Points**, create a **Rectangle** by these **Points** and display the **perimeter** of the **Rectangle** object , as well as, the **coordinates** of its corners. Run the **extension method CircleArea** and **display the area** of the circle drawn around the rectangle object

Problem No. 3

It is common task to enrich the capabilities of a standard System Form control. For instance, suppose you create a **list box control** named ***myListBox*** that contains a list of strings stored in a one-dimensional array, a **private member** variable named ***myStrings***. A **list box control** contains member properties and methods in addition to its array of strings. However, it would be convenient to be able to **access the list box array with an index**, just as if the **list box** were an array. For example, such a property would permit statements like the following:

```
        string theFirstString = myListBox[0];

        string theLastString = myListBox[Length-1];
```

More general, there are times when it is desirable to access a collection within a class as though the class itself were an array. For this purpose an indexer is being used. An ***indexer*** is a C# construct that allows you to access collections contained by a class using the familiar ***[]*** syntax of arrays. An indexer is a special kind of property and includes ***get( )*** and ***set( )*** methods to specify its behavior.

You declare an indexer property within a class using the following syntax:

```
        type this [type argument]{get; set;}
```

The return type determines the type of object that will be returned by the indexer, while the type argument specifies what kind of argument will be used to index into the collection that contains the target objects. Although it is common to use integers as index values, you can index a collection on other types as well, including strings. You can even provide an indexer with multiple parameters to create a multidimensional array!

The ***this*** keyword is a reference to the object in which the indexer appears. As with a normal property, you also must define ***get( )*** and ***set( )*** methods that determine how the requested object is retrieved from or assigned to its collection.

Write a *C#.NET* class ***ListBoxTest,*** which contains a simple string ***array*** (***myStrings***), and an ***int*** counter (***ctr***),storing the current number of strings used by the **list box control** and **uses a simple indexer for accessing *myStrings*** contents. **Write also a Console application** to test the user control.

Problem No. 4

Create a class called **Rational** for performing arithmetic with fractions. Write a **Console** application to test your class by employing a menu of choices , corresponding to the tasks **a- f** given below.
Use integer variables to represent the **private** instance variables of the class- the **numerator** and the **denominator**. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form- the fraction
2/4
is equivalent to **1/2** and would be stored in the object as 1 in the **numerator** and **2** in the **denominator**.
Provide a parameterless constructor with default values in case no initializers are provided.
Provide **public** methods that perform each of the following operations (all calculation results should be stored in a reduced form):

a) Add two **Rational** numbers.

b) Subtract two **Rational** numbers.

c) Multiply two **Rational** numbers.

d) Divide two **Rational** numbers.

e) Display **Rational** numbers in the form **a/b**, where **a** is the **numerator** and b is the **denominator**.

f) Display **Rational** numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)