Sofia University
Department of Mathematics and Informatics

Course : OO Programming C#.NET
Date: November 12, 2019
Student Name:

Lab No. 7

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

Problem No. 1

Write a class **Point**. It **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. Write an indexer to access the point coordinates with 1 and 2 accordingly for the **x** and **y** coordinate.

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 Diagonal()*** method allowing to compute the diagonal of objects ***Rectangle***. Write a Console application to test the inheritance hierarchy **Point**- **Rectangle** (create two **Points**, create a Rectangle by these Points and display the diagonal of the **Rectangle** object , as well as, the coordinates of its corners.

Create a Class Library containing both class **Point and** class **Rectangle.**

Create a new Console application making use of the Class library containing both class **Point and** class **Rectangle to test these classes.**

Problem No. 2

Create and build a **Class Library** project with the following classes:

1. Write a class **Point**. It has an array of two integer data 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 member, as well as a **ToString**() method.
2. Next, write a class **Rectangle**. Design it as follows: **Rectangle is a Point,** which defines **the upper left corner and has a Point** that 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 Area()*** method allowing to compute the area of the ***Rectangle*** object.

3. Finally, write a class **Parallelepiped**. Design it as follows: **Parallelepiped is a Rectangle,** which defines **the base side of the Parallelepiped and has height** that defines the **height** of the **Parallelepiped** . 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* **Rectangle**). Write override ***double Area()*** method allowing to compute the total area of the **Parallelepiped** object

Create a new Project with the **Console application** template, where **add a reference** to the previously built Class **Library**:

- **Add an extension method Volume() to** class **Parallelepiped, allowing to compute its volume.**
- **Add an extension method Perimeter() to** class **Rectangle, allowing to compute its perimeter.**

Using the Main() method test the inheritance hierarchy **Point- Rectangle** (create two Points, create a Rectangle by these Points, create a **Parallelepiped using the** Rectangle and display the area of the Rectangle and the **Parallelepiped** objects. **Printout the results of the execution of the extension methods** of thus created objects of type Rectangle and **Parallelepiped**

Problem No. 3

You normally use an ***int*** to hold an integer value. Internally an ***int*** stores its value as a sequence of ***32*** bits, where each ***bit*** can be either ***0*** or ***1***. Most of the time you don't care about this internal binary representation; you just use an ***int*** type as a bucket to hold an ***integer*** value. In other words, occasionally a program might use an ***int*** because it holds ***32*** bits and not because it can represent an ***integer***.

We'd like to use an ***int*** not as an ***int*** but as an ***array*** of ***32 bits***. Therefore, the best way to solve this problem is to use an ***int*** as if it were an array of ***32 bits***! In other words, if bits is an ***int***, what we'd like to be able to write to access the bit at ***index 6*** is:

```
bits[6]
```

And, for example, set the bit at ***index 6*** to ***true***, we'd like to be able to write:

```
bits[6] = true
```

Write a ***class IntBits*** (or a ***struct***) that has an ***indexer***, allowing you the following operations

```
int adapted = 63;
IntBits bits = new IntBits(adapted);
Bool peek = bits[6]; // retrieve bool at index 6
bits[0] = true; // set the bit at index 0 to true
bits[31] = false; // set the bit at index
```

Problem No. 4

Write **a WPF application,** which **computes** the **one's and the two's complement** of a given decimal number. The **number is input as a decimal number in a text box**, **the one's and the two's complement are initiated by means of user defined buttons** and the **result (displayed in binary digits) of each computation** should be displayed in separate textboxes (with disabled editing property). For validation purposes, **there should be a textbox** displaying the sum of the obtained **one's and the two's compliment of the given number.**

Problem No. 5

Code a **`class Rectangle`** a project of type Class Library in .Net Framework. The class ***has*** **two private double data members**- length and width. Additionally, it ***has*** **two double data members**- the x and y coordinates of the left lower point of the Rectangle (*reuse the Point class from Problem 1*). **Define** a default constructor, a general purpose constructor and a copy constructor. **Define** member function ***Area()*** and ***Perimeter()*** to return the area and the perimeter of the rectangle, respectively. **Define** a ***ToString***() method to return the current values of the data members as a string. **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. Build the Class Library project.

Write a WPF application with a Reference to the above build Class library. Making use of the GUI of the WPF application create two objects of class Rectangle and find which one has greater area and perimeter. Allow the user to input the data members of the Rectangle objects using graphical user interface. Display the result of comparison on the application window.

Problem No. 6

Code a declaration for a Point class. A Point object ***has*** **two private data members**- the x and y coordinates of the Point. **Define** a **default** constructor, a **general purpose** constructor and a **copy** constructor. Define **properties** for the coordinates of a Point object and a **ToString**() method.

Problem No. 7

A **point** can be defined by using polar coordinates (r,θ), where r is the distance of the point from the origin. If we imagine a straight line drawn through the origin and the point, this line will form an angle with the x- axis, and this is θ. Conversion from polar coordinates can be effected with the formula

$x = r \bullet \cos\theta$

$y = r \bullet \sin\theta$

Define a class **Rpoint**, where the points are defined using polar coordinates **and a member function** that computes the distance between two **Rpoints** (r_1,θ_1) and (r_2,θ_2) using the formula**:**

$$d = \sqrt{(r_1\cos\theta_1 - r_2\cos\theta_2)^2 + (r_1\sin\theta_1 - r_2\sin\theta_2)^2}$$

Problem No. 8

Construct a class **Passenger** that will define a passenger on a **flight**. Each passenger can **have a maximum of ten departures** during a flight. A passenger **object must therefore know this** (contain references to). There **should be a constructor** that initializes a passenger with appropriate flight departures. Here you **should check that times and flights correspond**, so that a person will not take a flight that leaves before the previous flight has arrived. There should **also be a method with which you should replace one flight by another**. Check here, too, that times and flights correspond. **If a flight is delayed, a passenger may need to change the flight that follows**. **Write a method** that checks whether this is necessary for a particular passenger. To solve this program task, you must **add two methods which make it possible to examine the departure and the arrival** of a flight.

Submit the source/ executables of the C# program used to test this class **Passenger, as well as, the** class **Passenger**

Problem No. 9

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. 10

The term used in **coding theory** to describe the combined message unit, comprising the **useful data bits** and **the additional check bits** is ***codeword***.

Definition. *The minimum number of bit positions in which two valid codewords differ is known as the **Hamming distance** of the code.*

For example, consider a coding scheme that has seven data bits and a single parity bit per codeword. Assuming **even parity** is being used, consecutive **codewords** in this scheme are as follows:

```
0000000  0
0000001  1
0000010  1
0000011  0
```

We can see from this list that **such a scheme has a Hamming distance** of 2, as each valid codeword differs in at least two bit positions. This means that it does not detect 2-bit errors since the resulting (corrupted) bit pattern will be a different but valid codeword.

Write a C#.NET Windows application allowing you to **enter** in two separate text boxes decimal numbers representing **7- bit binary numbers** (the useful bits). Next to each such textbox **display the parity bit** (use even parity) for the respective binary number. C**reate the respective codewords** for each of the given binary numbers and **compute the Hamming distance** between the two codewords. **Display the obtained result** in a Label control on the Windows form.
Hint: Use bitwise shift.

Problem No. 11

Write a program that **reverses** the order of the bits in an integer value. The program inputs an integer value and calls a method ***`reverseBits`***() to print the reverse of the bits. Print the value of the integer before its bits have been reversed and after that. Consider both an **iterative** and **recursive** solution.