Sofia University
Department of Mathematics and Informatics

Course : **OO Programming with C#.NET**
Date: **11/12 2020**
Student Name:

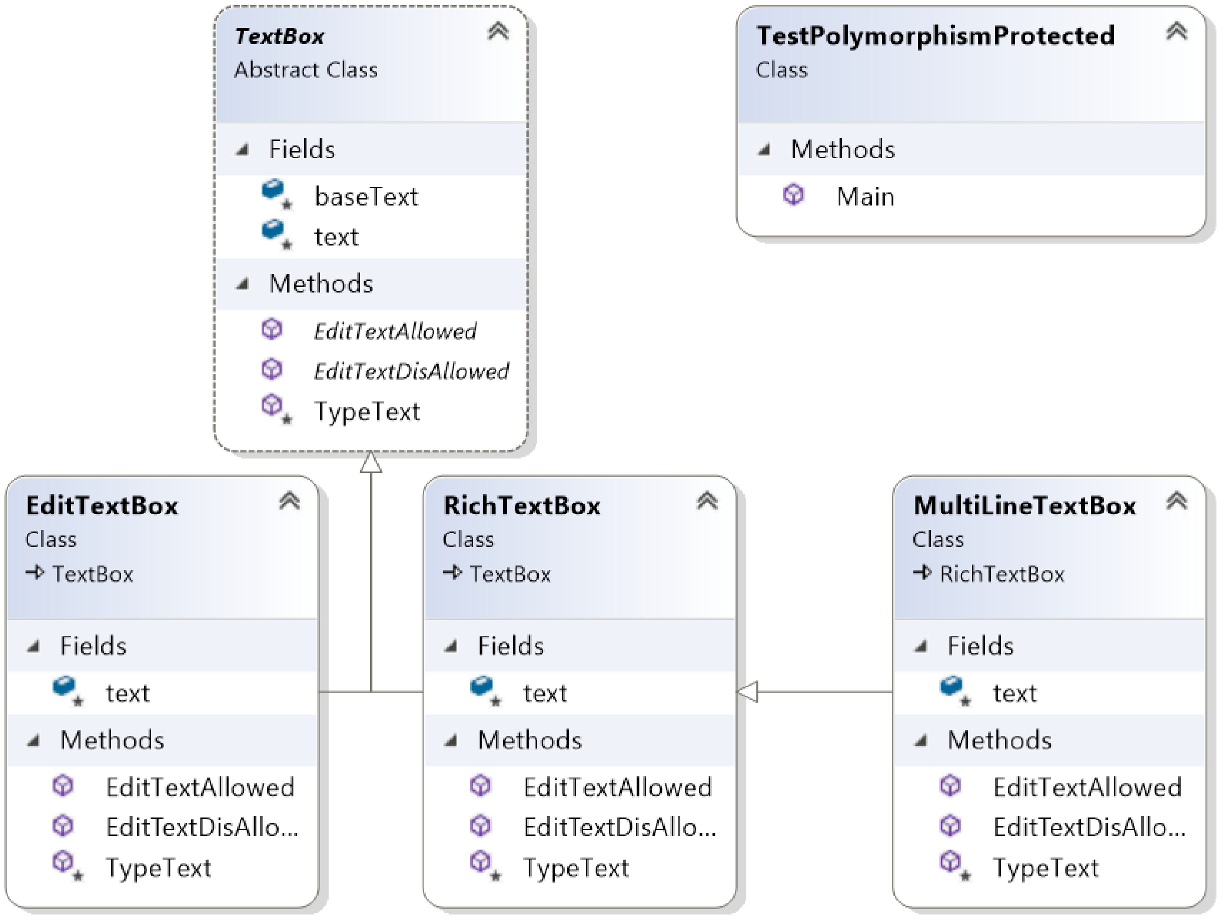
Домашно No. 5

Problem No. 1

According to C# reference documentation *"A protected member is accessible within its class and by derived class instances"*.

Now let's investigate how this definition affects polymorphism involving protected methods of classes in inheritance relationships.

Create the following inheritance hierarchy:



Datamember **`text`** is of type **`string`** and it is **`protected`** in classes **`TextBox`**, **`EditTextBox`**, **`RichTextBox`** and **`MultlineTextBox.`** Use the keyword **`new`** to hide the inherited data member in the classes derived from the **abstract** class **TextBox** . Initialize **text** to the **string** $"**{(GetType())}:Type text**".

Data member **baseText** is of type **string** and it is **protected** in class **TextBox**. Initialize **baseText** to the **string** $"**{(GetType())}:Type baseText**".

Add **protected void** method **TypeText**() to **class TextBox** and **override** it in the derived classes. Each of the methods overriding **TypeText**() prints on standard output the current value of data member **text**.

Add **public abstract void** methods **EditTextAllowed**() and **EditTextDisAllowed** () to **class TextBox** and **override** them in the derived classes

Each of the overridden versions of method **EditTextAllowed**()

- Executes the version of in method **TypeText**() in the direct base class
- Prints on standard output the current value of datamember **text** in the direct base class. (**baseText** is protected and it is OK**)**
- Prints on standard output the current value of datamember **baseText** in the direct base class. (**baseText** is protected and it is OK**)**

Each of the overridden versions of method **EditTextDisAllowed** ()

- Upcasts an instance of the current class to the base **class TextBox**
- Attempt to execute method **TypeText**() via the upcasted instance results in compiler error. Polymorphism is impossible in this case although **TypeText**() is overridden in the derived class, **explain why**!
- Attempt to assign a new value to data member **text** via the upcasted instance results in compiler error. **text** is **protected** and it is still disallowed, **explain why**!
- Attempt to assign a new value to data member **baseText** via the upcasted instance results in compiler error**. baseText** is **protected** and it is still disallowed, **explain why**!

Add class **TestPolymorphismProtected.** Add an array of instances of classes **EditTextBox**, **RichTextBox** and **MultlineTextBox** in the **public static void Main()** method of this class. Write a loop **to execute polymorphically method TypeText**() of the array elements. Notice, you get compiler error, **explain why**.

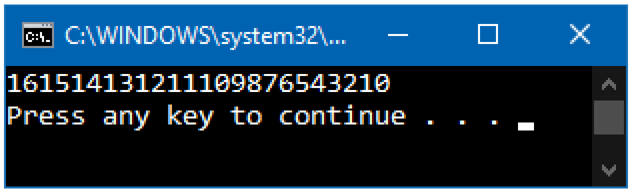
Problem No. 2a

Create **`interface IEnumerator`** with methods

```
bool MoveNext();
object Current { get; }
void Reset();
```

Write an implementation of **`IEnumerator`** in class **`CountDown`** allowing to use the **`interface`** methods in a while loop for the purpose of printing the sequence of numbers from 16 to 0. This class represents a default implementation of the interface methods(*without explicit qualification of the interface name in the method implementation*).

Test the implementation of the **`interface`** methods



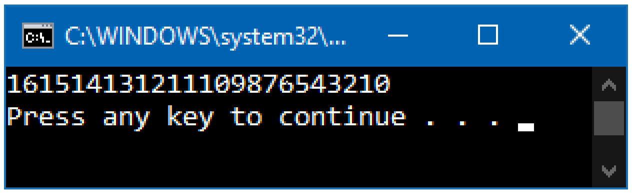
C:\WINDOWS\system32\...

```
16151413121110987654321O
Press any key to continue . . .
```

Problem No. 2b (C# 8)

Provide a default implementation of **`IEnumerator`** methods by embedding the implementation of **`IEnumerator`** in class **`CountDown`** from Problem 2a inside **`interface IEnumerator.`**

Test the default implementation of the **`interface`** methods



C:\WINDOWS\system32\...

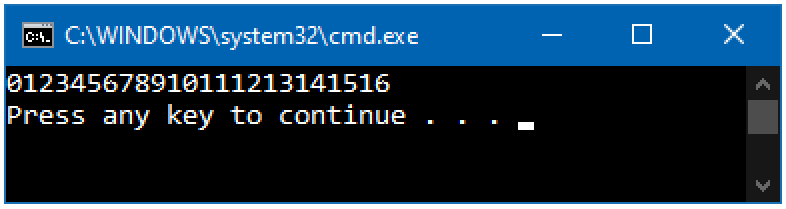
```
16151413121110987654321O
Press any key to continue . . .
```

Problem No. 2c (C# 8)

Provide the same implementation for each of the **`IEnumerator`** methods in class **`CountDown`** as in 7b but make these methods **`virtual`**. Inherit **`IEnumerator.CountDown`** in class **`CountDownWithOverride.`**

Override the default methods implementation in **`IEnumerator`** done in class **`CountDown`** with versions allowing to printing the sequence of numbers from 0 to 16

Test the default implementation of the interface methods in class **`CountDownWithDefaults`**

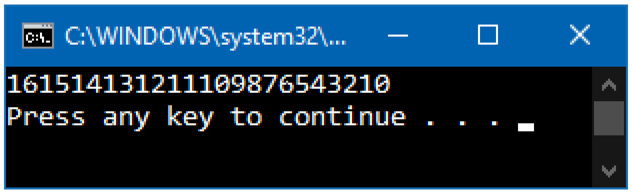


```
012345678910111213141516
Press any key to continue . . .
```

Problem No. 2d (C# 8)

Repeat tasks in Problem 2b where the implementation of **`IEnumerator`** methods in the embedded class **`CountDown`** in inside **`interface IEnumerator`** is done with explicit qualification of the interface name.

Test the implementation of the **`interface`** methods

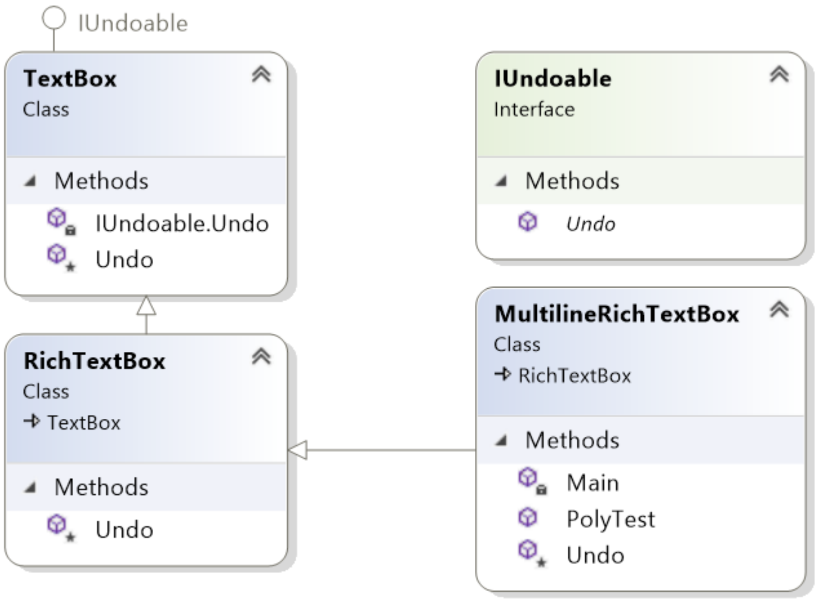


```
161514131211109876543210
Press any key to continue . . .
```

Problem No. 3

Even with explicit member implementation, interface reimplementation is problematic for a couple of reasons:

- The subclass has no way to call the base class method.
- The base class author may not anticipate that a method will be reimplemented and may not allow for the potential consequences.



A better option, however, is to design a base class such that reimplementation will never be required.

Create the artefacts in the above UML class diagram, where:

- Method Undo() in IUndoable is void and takes no arguments

- method Undo() is implemented in class TextBox as protected and overriden in classes, RichTextBox and MultilineRichTextBox. Each one of the implementations of this method prints on the console text $"{(GetType())}.Undo"

- interface IUndoable is implemented with explicit name qualification in class TextBox by calling the protected method Undo().

Write method void PolyTest() in class MultilineRichTextBox to test the polymorphic behavior of the implementation of method Undo() with explicit interface name qualification:

- Create instances of classes RichTextBox and MultilineRichTextBox
- Upcast them to IUndoable and execute method Undo()

Test the execution of method PolyTest() and explain the result.