Programming II

Assignment #4 – Testing, Inheritance, Polymorphism, Abstract Classes and Interfaces

Due Date: **Sunday 14th Apr. 2019 - at the start of the class – 8:30am**

Marks/Weight: **30 / 10%**

Purpose: The purpose of this Assignment is to:

- Practice the use of testing, inheritance, polymorphism, abstract classes and interfaces in C#

References: - Text book **"Visual C# 2017, Intro to Object Oriented Programming"**, Chapter 10
- Links available under **Week 7, 8, 9** and **10** on eCentennial;

Instructions: Be sure to read the following general instructions carefully:
This assignment should be completed in groups of 3 students. Submit the project **through e-Centennial, Assessments / Assignment**. You must name your Visual Studio solution according to the following rule: *GroupCode_COMP123_AssignmentNumber*
For Example: **sec006-6_COMP123_04**

Submit your assignment in a **zip file** that is named according to the following rule:
GroupCode_COMP123_AssignmentNumber.zip
Example: **sec006-6_COMP123_04.zip**

Apply the naming conventions for variables, methods, classes, and namespaces:
- *variable names, parameters and fields* – use camelCasing
- *classes, namespaces, methods, properties, enumerations* – use PascalCasing
- *constants:* SNAKE_UPPERCASE
- *interfaces:* start with uppercase "I" and the next letter should be uppercase too. Then continue as PascalCasing
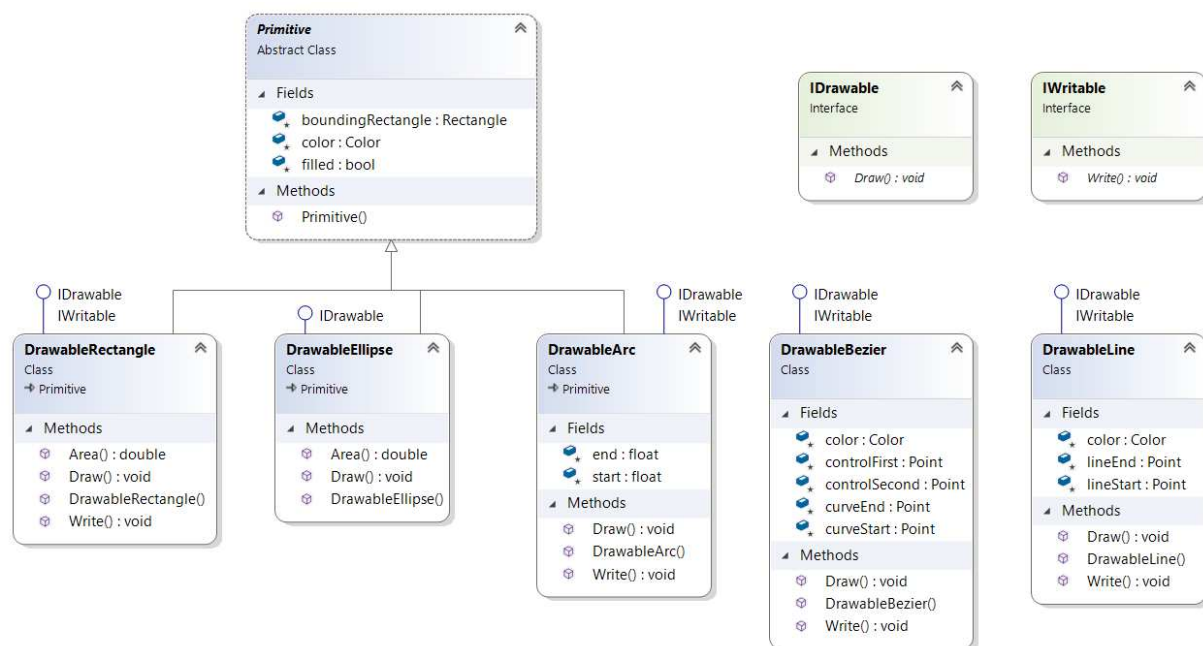
Exercise:

In this exercise you will be using inheritance to promote code re-use.

UML Diagram

You may choose to implement all the classes and interfaces from the diagram below in a single code file.

Note: You must follow the specifications exactly.

There are six classes and two interfaces participating in this application are shown in the diagram above. Each of them is fully described below. A bigger version of this diagram is also available on eCentennial and is called **"Assignment_04_Testing_Inheritance-uml.pdf"**

You will need to add a reference of the **"System.Drawing"** library to your project

Goal of the Exercise: Using the classes defined below and the test code in the end of this document, you should generate an image file showing this 2D face:
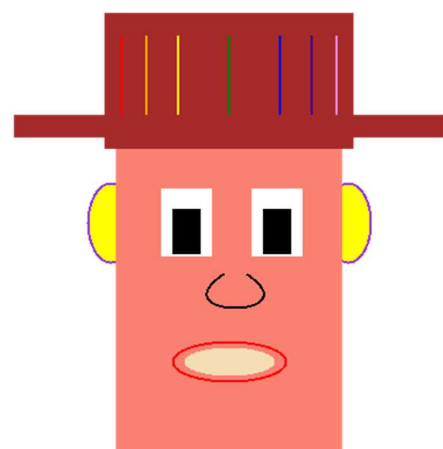
The IDrawable interface

This interface comprises of only one member:

IDrawable
Interface
Properties
Methods
Draw(Graphics g) : void

Description of interface members

Methods:

0.5 marks

void Draw(Graphics g) – This method will be defined in the implementing class.
You will have to add a reference to the **"System.Drawing"** library and insert the following using statements: using **System.Drawing** and **System.Drawing.Imaging**

> The names of interfaces normally start with the letter "I".
>
> Interfaces do not have fields or constructors, may contain properties, indexers and methods.
>
> They contain just methods without bodies i.e. abstract.
>
> The methods are always public; it is illegal to specify any accessibility specifiers e.g. public, protected or private or the abstract modifier

The IWritable interface

This interface comprises of only one member:

IWritable
Interface
Properties
Methods
Write(TextWriter writer) : void

Description of interface members

Methods:

0.5 marks

void Write(TextWriter writer) – This method will be defined in the implementing class. You will need to add a using statement for the **TextWriter** class

The Primitive class

This abstract class comprises of four members:

Primitive
Abstract Class
Fields
color : Color # filled : bool # boundingRectangle : Rectangle
Methods
+ <<constructor>> Primitive(Color color, bool filled, Rectangle rectangle)

Description of class members

Fields:

All the fields are protected

0.5 marks

color – this **Color** represents the color of this object. This field is protected.

0.5 marks

filled – this **bool** indicates if this object will be filled in. This field is protected.

0.5 marks

boundingRectangle – this **Rectangle** represents the bounding rectangle of this object. This field is protected.

Properties:

There are no properties.

Constructor:

1 mark

Primitive(Color color, bool filled, Rectangle rectangle) – This constructor takes three parameters and assigns it to the appropriate fields.

Methods:

There are no methods.

The DrawableRectangle class

This class inherits from the Primitive class and it implements the IDrawable and IWritable interfaces and comprises of three members:

DrawableRectangle
Class
➡ **Primitive, IDrawable, IWritable**
Properties
Methods
+ <<constructor>> DrawableRectangle(Color color, bool filled, Rectangle rectangle) + Draw(Graphics g) : void + Write(TextWriter writer) : void + Area() : double

Description of class members

Fields:
There are no fields

Properties:
There are no properties

Constructor:

1 mark

DrawableRectangle(Color color, bool filled, Rectangle rectangle) – This constructor passes the three parameters to its base constructor.

Methods:
There are two methods

1 mark

public void Draw(Graphics g) – This method check the filled field.
If true, it creates a **SolidBrush** object with the appropriate color and then uses the **FillRectangle()** method of the **Graphics** class to draw a solid rectangle.
Otherwise, it creates a **Pen** object with the appropriate color and then uses the **DrawRectangle()** method of the **Graphics** class to draw the outline of a rectangle.

1 mark

public void Write(TextWriter writer) – This method uses the **WriteLine()** method of the **TextWriter** class to write the three fields of this object.

1 mark

public double Area() – This method uses Height and Width of the **boundingRectangle** to calculate the area.

The DrawableEllipse class

This class inherits from the Primitive class and it implements the IDrawable interface and comprises of two members:

DrawableEllipse
Class
➡ Primitive, IDrawable
Properties
Methods
+ <<constructor>> DrawableEllipse(string name, bool filled, Rectangle rectangle) + Draw(Graphics g) : void + Area() : double

Description of class members

Fields:
There are no fields

Properties:
There are no properties

Constructor:

1 mark

DrawableEllipse(Color color, bool filled, Rectangle rectangle) – This constructor passes the three parameters to its base constructor.

Methods:
There is only one method

1 mark

public void Draw(Graphics g) – This method check the filled field.
If true, it creates a **SolidBrush** object with the appropriate color and then uses the **FillEllipse()** method of the **Graphics** class to draw a solid ellipse.
Otherwise, it creates a **Pen** object with the appropriate color and then uses the **DrawEllipse()** method of the **Graphics** class to draw the outline of a ellipse.

1 mark

public double Area() – This method uses Height and Width of the **boundingRectangle** to calculate the area of the Ellipsis – You need to find our how to calculate the area of an Ellipsis.

The DrawableLine class

This class implements the IDrawable and IWritable interfaces and comprises of six members:

```
DrawableLine
Class
➡ IDrawable, IWritable
Fields
# color : Color
# lineStart: Point
# lineEnd: Point

Methods
+ <<constructor>> DrawableLine(Color color, Point start, Point end)
+ Draw(Graphics g) : void
+ Write(TextWriter writer) : void
```

Description of class members

Fields:
There are three fields

0.5 marks   **color** – this field is of type **Color** and it represents the color of this object

0.5 marks   **lineStart** – this field is of type **Point** and it represents the starting position of this object

0.5 marks   **lineEnd** – this field is of type **Point** and it represents the ending position of this object

> Did you know that the type **Point** is actually a **struct** instead of a **class**?
>
> And so is **Rectangle**

Properties:
There are no properties

Constructor:
1 mark   **DrawableLine(Color color, Point start, Point end)** – This constructor assigns the three parameters to the appropriate fields.

Methods:
There are two methods

1 mark   **public void Draw(Graphics g)** – This method creates a **Pen** object with the appropriate color and then uses the **DrawLine()** method of the **Graphics** class to draw a line.

1 mark   **public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write the three fields of this object.

The DrawableBezier class

This class implements the IDrawable and IWritable interfaces and comprises of six members:

```
DrawableBezier
Class
➡ IDrawable, IWritable
Fields
# color : Color
# curveStart : Point
# controlFirst : Point
# controlSecond : Point
# curveEnd : Point

Methods
+ <<constructor>> DrawableBezier(Color color, Point start, Point first,
Point second Point end)
+ Draw(Graphics g) : void
+ Write(TextWriter writer) : void
```

Description of class members

Fields:

(0.5 marks) **color** – this field is of type **Color** and it represents the color of this object

(0.5 marks) **curveStart** – this field is of type **Point** and it represents the starting position of this object

(0.5 marks) **controlFirst** – this field is of type **Point** and it represents the first control point of this object

(0.5 marks) **controlSecond** – this field is of type **Point** and it represents the second control point of this object

(0.5 marks) **curveEnd** – this field is of type **Point** and it represents the ending position of this object

Constructor:

(1 mark) **DrawableBezier(Color color, Point start, Point first, Point second, Point end)** – This constructor assigns the five parameters to the appropriate fields.

Methods:
There are two methods

(1 mark) **public void Draw(Graphics g)** – This method creates a **Pen** object with the appropriate color and then uses the **DrawBezier()** method of the **Graphics** class to draw a line

(1 mark) **public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write all of the fields of this object.

The DrawableArc class

This class implements the IDrawable and IWritable interfaces and comprises of six members:

DrawableArc
Class
➡ Primitive, IDrawable, IWritable
Fields
start : float
end : float
Methods
+ <<constructor>> DrawableArc(Color color, bool filled, Rectangle rectangle, float start, float end)
+ Draw(Graphics g) : void
+ Write(TextWriter writer) : void

Description of class members

Fields:

(0.5 marks) **start** – this field is of type **float** and it represents the starting angle of this object

(0.5 marks) **end** – this field is of type **float** and it represents the ending angle of this object

Constructor:

(0.5 marks) **DrawableArc(Color color, bool filled, Rectangle rectangle, float start, float end)** – This constructor passes to the base constructor with the first three parameters and assigns the last two to the appropriate fields.

Methods:
There are two methods

(1 mark) **public void Draw(Graphics g)** – This method creates a **Pen** object with the appropriate color and then uses the **DrawArc()** method of the **Graphics** class to draw an arc

(1 mark) **public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write all of the fields of this object.

Test Project

Add a test project to the Visual Studio Solution using the Test Driven Development steps that you have studied and create 2 test methods.

(2 marks) 1 – One to test the **Area()** method of **DrawableRectangle** class;

(2 marks) 2 – Another one to test the **Area()** method of **DrawableEllipse** class;

Make sure that both tests pass.

How to test your application?

Add the following code to the **Main()** method of your program to see it in action and ensure it is creating the face with a hat below:

Note: For easy of copying, this code is also provided as a text file under eCentennial called **"Assignment_04_Testing_Inheritance-main-code.txt"** – You can just copy and paste from there.

```
//interfaces are types.
List<IDrawable> face = new List<IDrawable>();

face.Add(new DrawableEllipse(Color.Yellow, true, new Rectangle(75, 160, 40, 70)));      //left ear
face.Add(new DrawableEllipse(Color.BlueViolet, false, new Rectangle(75, 160, 40, 70))); //left ear
face.Add(new DrawableEllipse(Color.Yellow, true, new Rectangle(285, 160, 40, 70)));     //right ear
face.Add(new DrawableEllipse(Color.BlueViolet, false, new Rectangle(285, 160, 40, 70)));//right ear
face.Add(new DrawableRectangle(Color.Salmon, true, new Rectangle(100, 100, 200, 300))); //face
face.Add(new DrawableRectangle(Color.White, true, new Rectangle(140, 165, 45, 60)));    //right eye
face.Add(new DrawableRectangle(Color.White, true, new Rectangle(220, 165, 45, 60)));    //left eye
face.Add(new DrawableRectangle(Color.Black, true, new Rectangle(150, 183, 25, 40)));    //right pupil
face.Add(new DrawableRectangle(Color.Black, true, new Rectangle(230, 183, 25, 40)));    //left pupil
face.Add(new DrawableRectangle(Color.Brown, true, new Rectangle(90, 10, 220, 120)));    //hat top
face.Add(new DrawableRectangle(Color.Brown, true, new Rectangle(10, 100, 380, 20)));    //hat rim

face.Add(new DrawableBezier(Color.Black, new Point(195, 240), new Point(135, 280), new Point(275, 280),
new Point(215, 240)));

face.Add(new DrawableEllipse(Color.Red, false, new Rectangle(150, 300, 100, 35)));      //lips
face.Add(new DrawableEllipse(Color.Wheat, true, new Rectangle(160, 305, 80, 25)));      //mouth
face.Add(new DrawableLine(Color.Red, new Point(105, 30), new Point(105, 100)));         //lines
face.Add(new DrawableLine(Color.Orange, new Point(127, 30), new Point(127, 100)));      //lines
face.Add(new DrawableLine(Color.Yellow, new Point(155, 30), new Point(155, 100)));      //lines
face.Add(new DrawableLine(Color.Green, new Point(200, 30), new Point(200, 100)));       //lines
face.Add(new DrawableLine(Color.Blue, new Point(245, 30), new Point(245, 100)));        //lines
face.Add(new DrawableLine(Color.Indigo, new Point(273, 30), new Point(273, 100)));      //lines
face.Add(new DrawableLine(Color.Violet, new Point(295, 30), new Point(295, 100)));      //lines

int width = 400;
int length = 450;
```

```
Bitmap bitmap = new Bitmap(width, length);
Graphics graphic = Graphics.FromImage(bitmap);

foreach (var item in face)
{
    item.Draw(graphic);
    if (item is IWritable)
    {
        ((IWritable)item).Write(System.Console.Out);
    }
}

graphic.Dispose();

bitmap.Save("face.png", ImageFormat.Png);
bitmap.Dispose();

Console.ReadKey();
```

2 marks