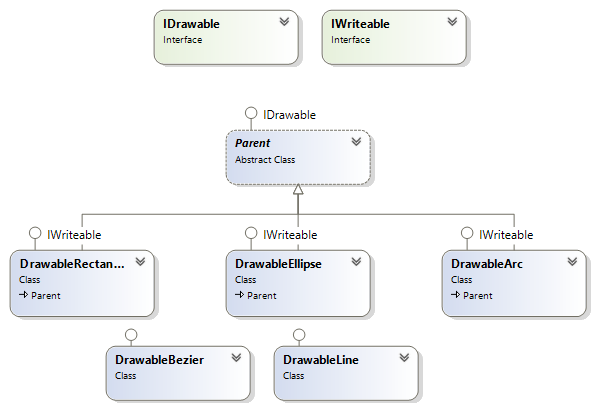
In this exercise you will be using inheritance to promote code re-use. You will also learn to use interfaces to do polymorphism.

This will be a console application that produces an image file.

# You must follow the specifications exactly (Practice only)

Because the required code statements to realise each type is very short and you have to add the same using statement in all your source code, you may choose to implement all the classes in a single code file



There are seven classes participating in this application as shown in the diagram above. Each of the seven classes is fully described below.

You will need to “google” how to declare an abstract property.

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

# The IDrawable interface

This interface comprises of only one member:

|  |
| --- |
| **IDrawable**  **Interface** |
| **Properties** |
|  |
| **Methods** |
| **Draw(Graphics g) : void** |

The names of interfaces normally start with the letter "I" and ends with "able".

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

#### Description of interface members

##### Methods:

**void Draw(Graphics g)** – This method will be defined in the implementing class.   
If you didn’t add a reference of **"System.Drawing"** library to your project, you will to do that now, also insert the following using statements: using **System.Drawing** and **System.Drawing.Imaging**

# The IWriteable interface

This interface comprises of only one member:

|  |
| --- |
| **IWriteable**  **Interface** |
| **Properties** |
|  |
| **Methods** |
| **Write(TextWriter writer) : void** |

#### Description of interface members

##### Methods:

**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 Parent class

This abstract class implements the **IDrawable** interface and it comprises of five members:

|  |
| --- |
| **Parent**  **Abstract Class**  **IDrawable** |
| **Fields** |
|  |
| **Properties** |
| **# «property setter absent» Color : Color**  **# «property setter absent» Filled : bool**  **# «property setter absent» Rectangle : Rectangle** |
| **Methods** |
| **+ «constructor» Parent(Color color, bool filled, Rectangle rectangle)**  **+ «abstract method» Draw(Graphics graphics) : void** |

#### Description of class members

##### Fields:

There are no fields

##### Properties:

All properties have protected get and the setter is absent.

**Color** – this **Color** property represents the color of this object. This property is protected with the setter absent.

**Filled** – this **bool** property indicates if this object will be filled in. This property is protected with the setter absent.

**Rectangle** – this **Rectangle** property represents the bounding rectangle of this object. This property is protected with the setter absent.

##### Constructor:

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

##### Methods:

There is one method that is abstract.

**public abstract void Draw(Graphics graphics)**

# The DrawableRectangle class

This class inherits from the Parent class and it implements the IWriteable interfaces and comprises of three members:

|  |
| --- |
| **DrawableRectangle**  **Class**  **Parent, IWriteable** |
| **Fields** |
|  |
| **Properties** |
|  |
| **Methods** |
| **+ «constructor» DrawableRectangle(Color color, bool filled, Rectangle rectangle)**  **+ Draw(Graphics g) : void**  **+ Write(TextWriter writer) : void** |

#### Description of class members

##### Fields:

There are no fields

##### Properties:

There are no properties

##### Constructor:

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

##### Methods:

There are two methods

**public override void Draw(Graphics g)** – This method overrides the Draw member of the Parent class. It checks the **Filled** property.

* 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.  
    
  **SolidBrush** brush = new **SolidBrush**(Color);  
  g.FillRectangle(brush, 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.  
    
  **Pen** pen = new **Pen**(Color);  
  g.DrawRectangle(pen, Rectangle);

**public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write the type of this object and the three properties of this object. (i.e. rectangle, color, filled and rectangle.)  
Color has a method **ToKnownColor()** that returns the name of this color

At this point, you should check your application. Copy the all the statements from the last page into your main method. Comment out the line with the classes that you did not implement as yet. You should see some output to the screen. An image file will be created as well.

# The DrawableEllipse class

This class inherits from the Parent class and comprises of two members:

|  |
| --- |
| **DrawableEllipse**  **Class**  **Parent, IWriteable** |
| **Properties** |
|  |
| **Methods** |
| **+ «constructor» DrawableEllipse(Color color, bool filled, Rectangle rectangle)**  **+ Draw(Graphics g) : void**  **+ Write(TextWriter writer) : void** |

#### Description of class members

##### Fields:

There are no fields

##### Properties:

There are no properties

##### Constructor:

**public DrawableEllipse(Color color, bool filled, Rectangle rectangle)** – This constructor passes the three parameters to its base constructor. The rectangle specifies the bounding rectangle of this ellipse.

##### Methods:

There is only one method

**public override 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.

**public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write the type of this object and the three properties of this object. (i.e. ellipse, color, filled and rectangle.)

# The DrawableLine class

This class does not have an explicit parent. This class implements the IDrawable and IWritable interfaces and comprises of six members.

I choose not to inherit from Parent class because, the filled and the rectangle properties is not applicable to a line.

|  |
| --- |
| **DrawableLine**  **Class**  **IDrawable, IWritable** |
| **Fields** |
| **- color : Color**  **- start : Point**  **- end : Point** |
| **Properties** |
|  |
| **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

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

**Start** – this field is of type **Point** and it represents the starting position of this object

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

And so is **Rectangle**

**End** – this field is of type **Point** and it represents the ending position of this object

There are no fields

##### Properties:

There are no properties

##### Constructor:

**public DrawableLine(Color color, Point start, Point end)** – This constructor assigns the three parameters to the appropriate properties.

##### Methods:

There are two methods

**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.

**public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write the type of this object and the four properties of this object. (i.e. ellipse, color, filled, start and end.).

# The DrawableBezier class

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

Again, it does not make sense to inherit from Parent class because, the filled and the rectangle properties is not applicable to this curve.

|  |
| --- |
| **DrawableBezier**  **Class**  **IDrawable, IWriteable** |
| **Fields** |
| **- color : Color**  **- start : Point**  **- first : Point**  **- second : Point**  **- end : Point** |
| **Properties** |
|  |
| **Methods** |
| **+ «constructor» DrawableBezier(Color color, Point start, Point end, Point first, Point second)**  **+ Draw(Graphics g) : void**  **+ Write(TextWriter writer) : void** |

#### Description of class members

##### Fields:

There are five private fields

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

**start** – this field is of type **Point** and it represents the starting position of this object

**first** – this field is of type **Point** and it represents the first control point of this object

**second** – this field is of type **Point** and it represents the second control point of this object

**end** – this field is of type **Point** and it represents the ending position of this object

##### Properties:

There are no properties

##### Constructor:

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

##### Methods:

There are two methods

**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 Bezier curve.

**public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write the type of this object and the five properties of this object. (i.e. Bezier curve, color, start, end, first and last.).

# The DrawableArc class

This class inherits from Parent and implements the IWriteable interfaces and comprises of six members:

|  |
| --- |
| **DrawableArc**  **Class**  **Parent, IWriteable** |
| **Fields** |
| **- start : Point**  **- end : Point** |
| **Fields** |
|  |
| **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:

There are two fields

**start** – this property is of type **float** and it represents the starting angle of this object

**end** – this property is of type **float** and it represents the ending angle of this object

##### Properties:

There are no properties

##### Constructor:

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

##### Methods:

There are two methods

**public void Draw(Graphics g)** – This method check the Filled field.   
If true, it create an appropriate brush and call **FillPie()**   
Else create an appropriate pen and calls **DrawArc()**

**public void Write(TextWriter writer)** – This method uses the **WriteLine()** method of the **TextWriter** class to write the type of this object and the four properties of this object. (i.e. arc, color, filled, start and end.)

### Test Harness

Remember in testing, you will compare the expected value to the actual value. You need to copy the following statements to your main method:

//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(215, 240), new Point(135, 280), new Point(275, 280))); //nose

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, length = 450;

using (Bitmap bmp = new Bitmap(width, length))

{

using (Graphics g = Graphics.FromImage(bmp))

{

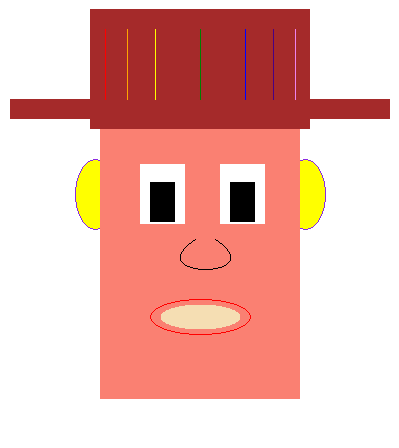
foreach (var item in face)

{

item.Draw(g);//this is polymorphic behavior

if (item is IWriteable)

{

 ((IWriteable)item).Write(System.Console.Out);

}

}

bmp.Save("face.png", ImageFormat.Png);

}

}