Shapes Factory Example

[**Shapes Factory Example Visual Studio Project**](https://drive.google.com/open?id=1zRYpwS6sptpWF2iwfoJVdv2EB7rd_f54)

**Stage 1** Create an interface

|  |
| --- |
| interface Shapes  {  void set(Color c, params int[] list);  void draw(Graphics g);  double calcArea();  double calcPerimeter();  } |

**Stage 2** Create an abstract class that implements the interface and provides concrete methods of what it can and leaves the rest to be overridden.

|  |
| --- |
| abstract class Shape:Shapes  {  protected Color colour; //shape's colour  protected int x, y;  public Shape()  {  colour = Color.Red;  x = y = 100;  }    public Shape(Color colour, int x, int y)  {  this.colour = colour; //shape's colour  this.x = x; //its x pos  this.y = y; //its y pos  //can't provide anything else as "shape" is too general  }  //the three methods below are from the Shapes interface  //here we are passing on the obligation to implement them to the derived classes by declaring them as abstract  public abstract void draw(Graphics g);  public abstract double calcArea();  public abstract double calcPerimeter();  //set is declared as virtual so it can be overridden by a more specific child version  //but is here so it can be called by that child version to do the generic stuff  //note the use of the [param keyword](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/params) to provide a variable parameter list to cope with some shapes having more setup information than others (in Java it is called varargs and uses the … notation  public virtual void set(Color colour, params int[] list)  {  this.colour = colour;  this.x = list[0];  this.y = list[1];  }  public override string ToString()  {  return base.ToString()+" "+this.x+","+this.y+" : ";  }  } |

**Stage 3** Implement derived classes

|  |
| --- |
| class Rectangle:Shape  {  int width, height;  public Rectangle():base()  {  width = 100;  height = 100;  }  public Rectangle(Color colour, int x, int y, int width, int height) : base(colour, x, y)  {  this.width = width; //the only thing that is different from shape  this.height = height;  }  public override void set(Color colour, params int[] list)  {  //list[0] is x, list[1] is y, list[2] is width, list[3] is height  base.set(colour, list[0], list[1]);  this.width = list[2];  this.height = list[3];  }  public void draw(Graphics g)  {  Pen p = new Pen(Color.Black,2);  SolidBrush b = new SolidBrush(colour);  g.FillRectangle(b, x, y, width, height);  g.DrawRectangle(p, x, y, width, height);  }  public double calcArea()  {  return width \* height;  }  public double calcPerimeter()  {  return 2 \* width + 2 \* height;  }  } |

**Stage 4** Make a Factory

|  |
| --- |
| class ShapeFactory  {  public Shape getShape(String shapeType)  {  shapeType = shapeType.ToUpper().Trim(); //you could argue that you want a specific word string to create an object but I'm allowing any case combination      if (shapeType.Equals("CIRCLE"))  {  return new Circle();  }  else if (shapeType.Equals("RECTANGLE"))  {  return new Rectangle();  }  else if (shapeType.Equals("SQUARE"))  {  return new Square();  }  else  {  //if we get here then what has been passed in is unknown so throw an appropriate exception  System.ArgumentException argEx = new System.ArgumentException("Factory error: "+shapeType+" does not exist");  throw argEx;  }    }  } |

**Stage 5** Create your objects

|  |
| --- |
| public Form1()  {  InitializeComponent();  ShapeFactory factory = new ShapeFactory();  Circle c = factory.getShape(“circle”);  try  {  shapes.Add(factory.getShape("circle"));  shapes.Add(factory.getShape("triangle"));  shapes.Add(factory.getShape("rectangle"));    }  catch (ArgumentException e)  {  Console.WriteLine("Invalid shape: " + e);    }      //add some random shapes  Random rand = new Random(77887);  for (int i=0; i<150; i++)  {  int x = rand.Next(Size.Width);  int y = rand.Next(Size.Height);  int size = rand.Next(250);  int red = rand.Next(255);  int green = rand.Next(255);  int blue = rand.Next(255);  Color newColour = Color.FromArgb(128, red, green, blue); //128 is semi transparent  int shape = rand.Next(2);  Shape s;  switch(shape)  {  case 0 :  s = factory.getShape("circle");  s.set(newColour, x, y, size);  shapes.Add(s);// new Circle(newColour, x, y, size));  break;  case 1:  s = factory.getShape("rectangle");  s.set(newColour, x, y, size, size);  shapes.Add(s);  break;    }  }    } |
|  |

Iterator Pattern

**Stage 1** provide an interface for an iterator and an interface for a container class

|  |
| --- |
| public interface Iterator  {  public boolean hasNext();  public Object next(); } |

|  |
| --- |
| public interface Container  {  public Iterator getIterator(); } |

**Stage 2** Create concrete class implementing the *Container* interface. This class has inner class *NameIterator* implementing the *Iterator* interface.

|  |
| --- |
| public class NameRepository implements Container {  public String names[] = {"Robert" , "John" ,"Julie" , "Lora"};   public override Iterator getIterator() {  return new NameIterator();  }   private class NameIterator implements Iterator {   int index;   public override boolean hasNext() {    if(index < names.length){  return true;  }  return false;  }     public Object next() {  if(this.hasNext()){  return names[index++];  }  return null;  }   } } |

**Stage 3** Use the *NameRepository* to get iterator and print names

|  |
| --- |
| public class IteratorPatternDemo {    public static void main(String[] args) {  NameRepository namesRepository = new NameRepository();   for(Iterator iter = namesRepository.getIterator(); iter.hasNext();){  String name = (String)iter.next();  System.out.println("Name : " + name);  }   } } |

(Note this file uses the Code Pretty add on for Chrome to display syntax colouring. You won’t see it coloured on your computer if you don’t have it installed.)