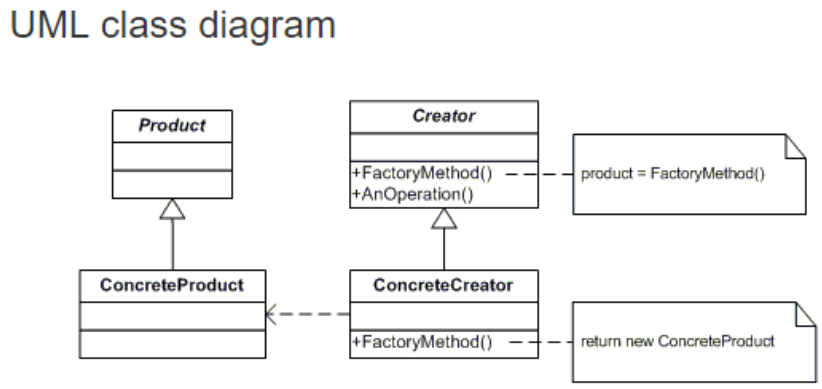
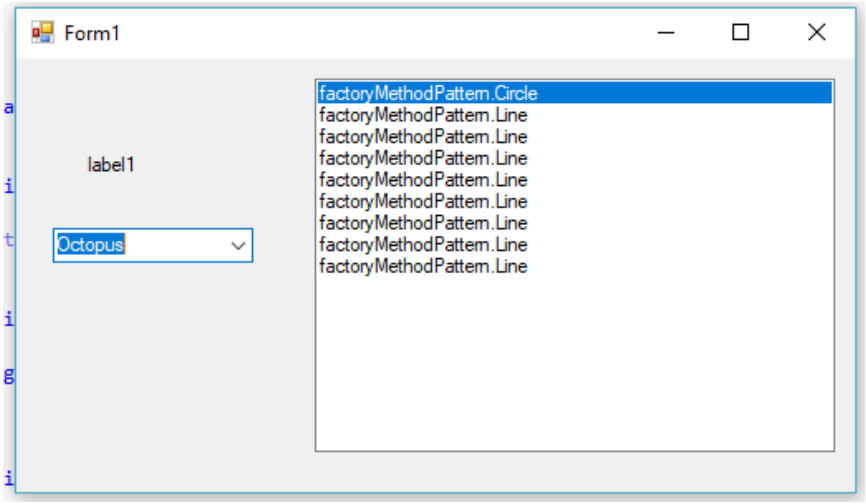
Factory Method Pattern

Introduction:

For this assignment I was supposed to create a demo that implemented the Factory Method design pattern which is described as : *Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses*. (dofactory.com)





Narrative:

For this demo, I created an application that "draws" shapes. There's a combobox that allows you to select what you would like to draw, and in a listbox the application tells you all the components you would need to make that drawing. For example, one of the options is an octopus, so in the listbox a circle and 8 lines are displayed. It doesn't actually draw anything though, it's more just to demonstrate what components would make up the drawing.

```
abstract class Creator

    {
        private List<Product> shapes = new
List<Product>();


        public Creator()
        {
            this.CreateShapes();
        }

        public List<Product> Shapes
        {
            get { return shapes; }
        }


        public abstract void CreateShapes();
    }
```

```
    class Triangle : Product { }

    class Circle : Product { }

    class Square : Product { }

    class Rectangle : Product { }

    class Line : Product { }
```

```
abstract class Product
    {

    }
```

Conclusion: I think that this design pattern could be very useful, especially if you're writing an application that has a lot of components that are made up of other components. It took me a while to understand the actual purpose of the factory method, but after a while of just messing around with the code I kind of had an 'ah-ha!' moment where it finally clicked.

```
class ConcreteCreatorOctopus : Creator
    {
        public override void CreateShapes()
        {
            Shapes.Add(new Circle());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
        }
    }
    class ConcreteCreatorPerson : Creator
    {
        public override void CreateShapes()
        {
            Shapes.Add(new Circle());
            Shapes.Add(new Triangle());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
            Shapes.Add(new Line());
        }
    }
    class ConcreteCreatorHouse : Creator
    {
        public override void CreateShapes()
        {
            Shapes.Add(new Square());
            Shapes.Add(new Square());
            Shapes.Add(new Square());
            Shapes.Add(new Rectangle());
            Shapes.Add(new Circle());
        }
    }
```