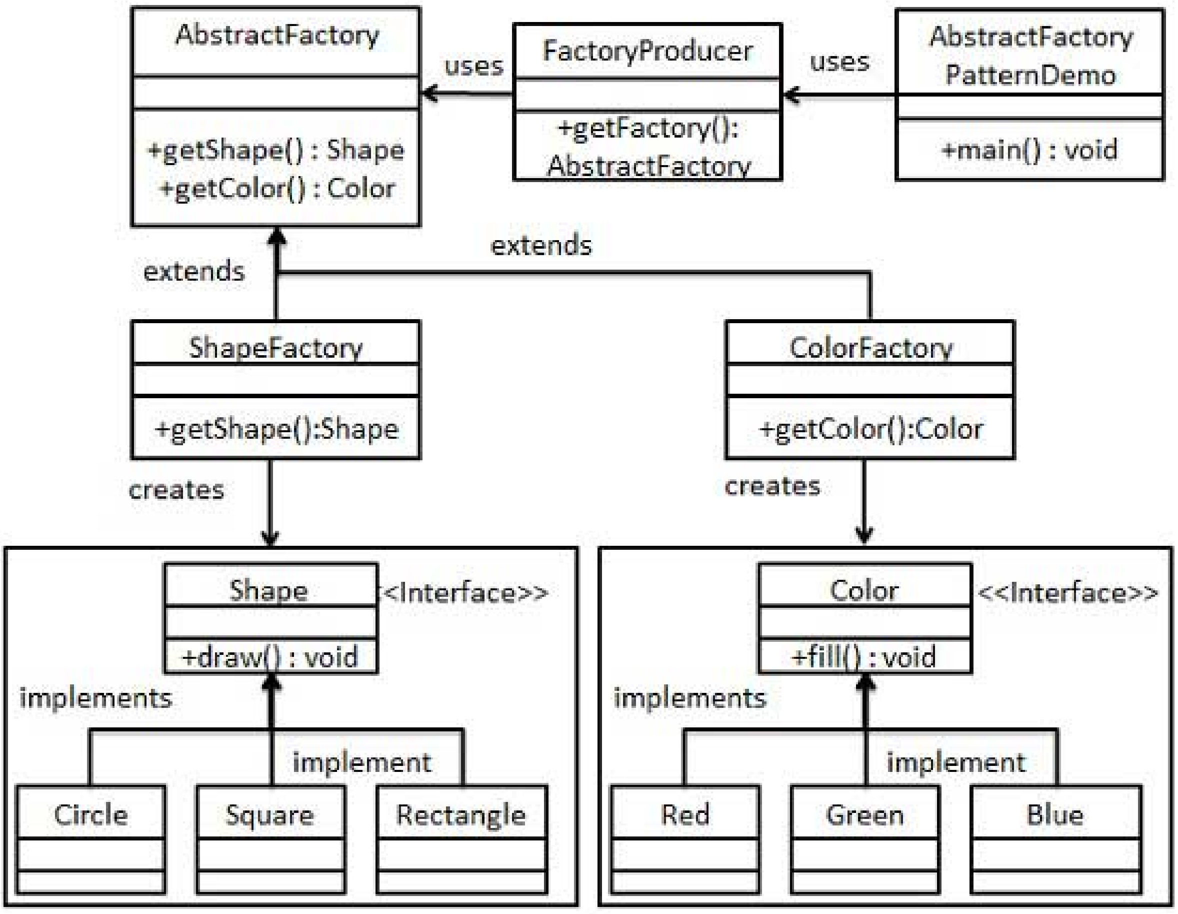
抽象工厂模式

抽象工厂模式使用一个超级工厂来创建其它工厂。这个工厂也被称作工厂的工厂。这个设计模式属于创建型模式，提供了一种创建一个对象的最佳实践。

在抽象工厂模式中，有一个接口用来创建相关对象的工厂而不用显式指定它们的类。

实现

我们准备创建一个 `Shape` 和 `Color` 接口和实现这些接口的实现类。然后，我们创建一个抽象工厂类 `AbstractFactory` 。



第一步

创建一个 `Shape` 接口：
Shape.java

```java
public interface Shape {
   void draw();
}
```

第二步

创建实现类
Rectangle.java

```
public class Square implements Shape {
   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Circle.java

```
public class Circle implements Shape {
   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

第三步

Color.java

```
public interface Color {
   void fill();
}
```

第四步

Read.java

```
public class Red implements Color {
   @Override
   public void fill() {
      System.out.println("Inside Red::fill() method.");
   }
}
```

Green.java

```
public class Green implements Color {
   @Override
   public void fill() {
      System.out.println("Inside Green::fill() method.");
   }
}
```

Blue.java

```
public class Blue implements Color {
   @Override
   public void fill() {
      System.out.println("Inside Blue::fill() method.");
   }
}
```

第五步

AbstractFactory.java

```
public abstract class AbstractFactory {
   abstract Color getColor(String color);
   abstract Shape getShape(String shape) ;
}
```

第六步

ShapeFactory.java

```
public class ShapeFactory extends AbstractFactory {

   @Override
   public Shape getShape(String shapeType){

      if(shapeType == null){
         return null;
      }

      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      }else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

      }else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }

      return null;
   }

   @Override
   Color getColor(String color) {
      return null;
   }
}
```

ColorFactory.java

```java
public class ColorFactory extends AbstractFactory {

    @Override
    public Shape getShape(String shapeType){
        return null;
    }

    @Override
    Color getColor(String color) {

        if(color == null){
            return null;
        }

        if(color.equalsIgnoreCase("RED")){
            return new Red();

        }else if(color.equalsIgnoreCase("GREEN")){
            return new Green();

        }else if(color.equalsIgnoreCase("BLUE")){
            return new Blue();
        }

        return null;
    }
}
```

第七步

创建一个工厂生成器/生产者

FactoryProducer.java

```java
public class FactoryProducer {
    public static AbstractFactory getFactory(String choice){

        if(choice.equalsIgnoreCase("SHAPE")){
            return new ShapeFactory();

        }else if(choice.equalsIgnoreCase("COLOR")){
            return new ColorFactory();
        }

        return null;
    }
}
```

第八步

AbstractFactoryPatternDemo.java

```
public class AbstractFactoryPatternDemo {
   public static void main(String[] args) {

      //get shape factory
      AbstractFactory shapeFactory = 
FactoryProducer.getFactory("SHAPE");

      //get an object of Shape Circle
      Shape shape1 = shapeFactory.getShape("CIRCLE");

      //call draw method of Shape Circle
      shape1.draw();

      //get an object of Shape Rectangle
      Shape shape2 = shapeFactory.getShape("RECTANGLE");

      //call draw method of Shape Rectangle
      shape2.draw();

      //get an object of Shape Square
      Shape shape3 = shapeFactory.getShape("SQUARE");

      //call draw method of Shape Square
      shape3.draw();

      //get color factory
      AbstractFactory colorFactory = 
FactoryProducer.getFactory("COLOR");

      //get an object of Color Red
      Color color1 = colorFactory.getColor("RED");

      //call fill method of Red
      color1.fill();

      //get an object of Color Green
      Color color2 = colorFactory.getColor("Green");

      //call fill method of Green
      color2.fill();

      //get an object of Color Blue
      Color color3 = colorFactory.getColor("BLUE");

      //call fill method of Color Blue
      color3.fill();
   }
}
```

第九步

验证输出

```
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
Inside Red::fill() method.
Inside Green::fill() method.
Inside Blue::fill() method.
```