Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 2 Padrões de Criação - Abstract Factory

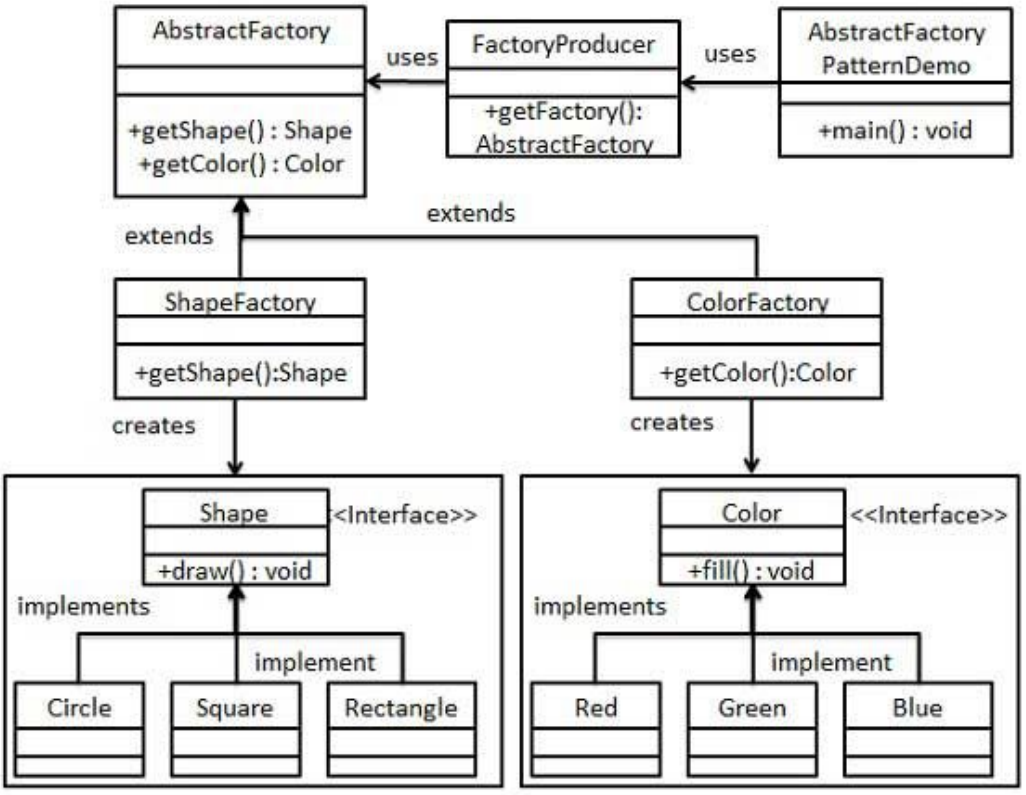
Padrões de fábrica abstratos trabalham em torno de uma super-fábrica que cria outras fábricas. Esta fábrica também é chamada como fábrica de fábricas. Esse tipo de padrão de design está em padrão de criação, pois esse padrão fornece uma das melhores maneiras de criar um objeto.

No padrão Abstract Factory, uma interface é responsável por criar uma fábrica de objetos relacionados sem especificar explicitamente suas classes. Cada fábrica gerada pode fornecer os objetos conforme o padrão Factory.

Implementação

Vamos criar interfaces Shape e Color e classes concretas implementando essas interfaces. Criamos uma classe abstrata de fábrica AbstractFactory como próximo passo. As classes de fábrica ShapeFactory e ColorFactory são definidas onde cada fábrica estende o AbstractFactory. Uma classe de criador / gerador de fábrica FactoryProducer é criada.

AbstractFactoryPatternDemo, a classe de demonstração usa FactoryProducer para obter um objeto AbstractFactory. Ele irá passar informações (CIRCLE / RECTANGLE / SQUARE para Shape) para AbstractFactory para obter o tipo de objeto que precisa. Ele também passa informações (RED / GREEN / BLUE para Color) para o AbstractFactory para obter o tipo de objeto de que precisa.



Passo 1
Crie uma interface para formas.

Shape.java

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

Passo 2
Crie classes concretas implementando a mesma interface.

Rectangle.java

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Square.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.");
   }
}
```

Passo 3
Crie uma interface para cores.

Color.java

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

Passo 4
Crie classes concretas implementando a mesma interface.

Red.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.");
   }
}
```

Passo 5
Crie uma classe abstrata para obter fábricas para objetos de cor e forma.

AbstractFactory.java

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

Passo 6
Criar classes Factory estendendo o AbstractFactory para gerar o objeto da classe concreta com base nas informações fornecidas.

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

```
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;
   }
}
```

Passo 7
Crie uma classe gerador / produtor de Fábrica para obter fábricas passando informações como Forma ou Cor

FactoryProducer.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;
   }
}
```

Passo 8
Use o FactoryProducer para obter o AbstractFactory para obter fábricas de classes concretas passando uma informação como o tipo.

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();
   }
}
```

Passo 9

Teste sua implementação!