Abstract Factory Pattern

	Name
Nguyễn Thăng Long	LongNT8
Nguyễn Tiến Tú	TuNT7
Nguyễn Việt Hoàng	HoangNV2
Quách Hữu Trung Anh	AnhQHT
Nguyễn Huy Hoàng	HoangNH11

1. Tổng quan

Abstract Factory Design Pattern?

Abstract Factory là sự mở rộng của tính chất đa hình trong lập trình hướng đối tượng. Mục tiêu hướng đến là có thể tạo ra các đối tượng mà chưa biết trước chính xác kiểu dữ liệu của chúng.

1. Tổng quan

Khi nào chúng ta nên sử dụng Abstract Factory Pattern?

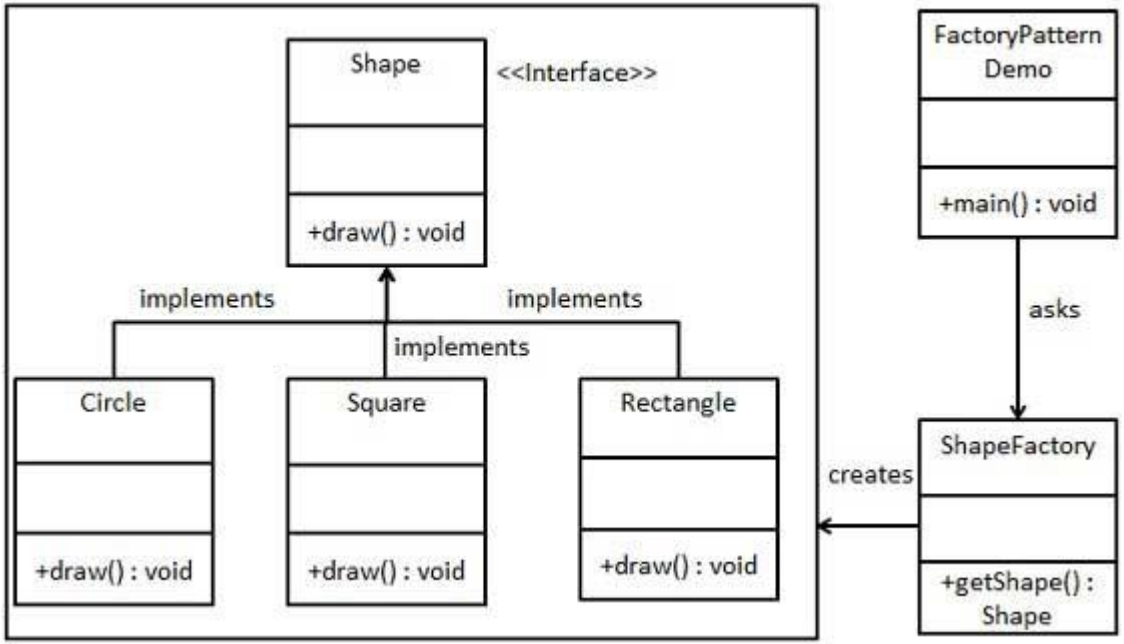
- Tạo ra đối tượng mà không cần biết chính xác kiểu dữ liệu.
- Giúp mã nguồn của chúng ta trở nên dễ dàng bảo trì nếu có sự thay đổi.

2. Factory Design Pattern

Pattern này được sử dụng khi hệ thống của bạn có nhiều đối tượng với thuộc tính, hành vi tương tự nhau.

Factory Pattern giống như một nhà máy sản sinh các đối tượng tương tự nhau này cho bạn.

Ví dụ


Shape
<<Interface>>
+draw() : void
implements
implements
implements
Circle
+draw() : void
Square
+draw() : void
Rectangle
+draw() : void
FactoryPattern Demo
+main() : void
asks
creates
ShapeFactory
+getShape() : Shape

Step 1: Tạo interface Shape

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

Step 2: Tạo các lớp thực thi interface Shape

Rectangle.java

```
public class Rectangle implements Shape {

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

Step 2: Tạo các lớp thực thi interface Shape

Square.java

```
public class Square implements Shape {

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

Step 2: Tạo các lớp thực thi interface Shape

Circle.java

```
public class Circle implements Shape {

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

Step 3: Viết lớp ShapeFactory

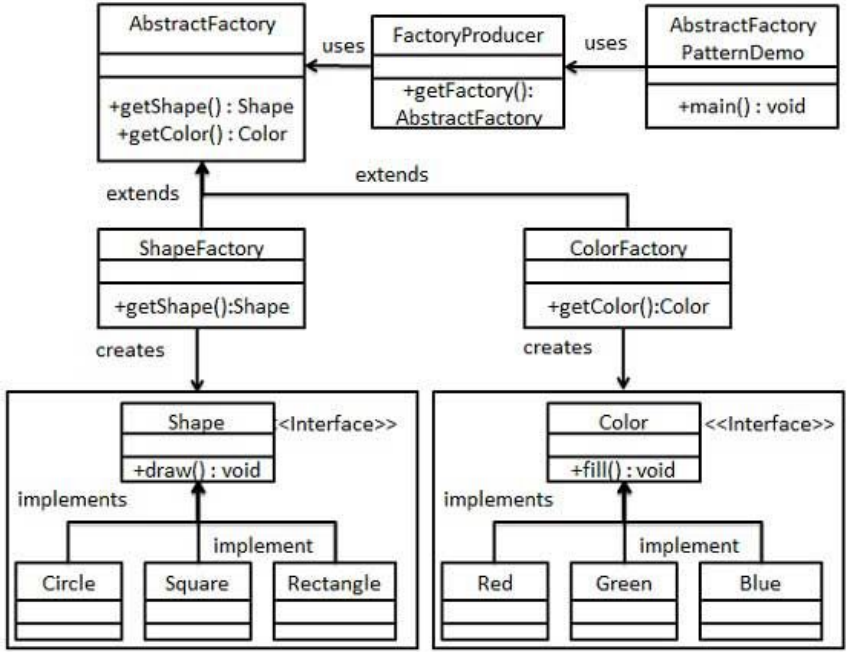
ShapeFactory.java

```
public class ShapeFactory {

   //use getShape method to get object of type shape
   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;
   }
}
```

3. Abstract Factory Pattern

Design pattern này cung cấp một cách hỗ trợ việc quản lý và tạo ra các đối tượng cùng nhóm. Như tên của pattern này thì nó giống là một nhà máy sản sinh ra các đối tượng.

Ví dụ


AbstractFactory
+getShape() : Shape
+getColor() : Color
uses
FactoryProducer
+getFactory(): AbstractFactory
uses
AbstractFactory PatternDemo
+main() : void
extends
extends
ShapeFactory
+getShape():Shape
ColorFactory
+getColor():Color
creates
creates
Shape
<<Interface>>
+draw() : void
implements
implement
Circle
Square
Rectangle
Color
<<Interface>>
+fill() : void
implements
implement
Red
Green
Blue

Step 1: Tạo interface Shape.java

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

Step 2: Tạo các lớp thực thi interface Shape

Rectangle.java

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

Step 2: Tạo các lớp thực thi interface Shape

Square.java

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

Step 2: Tạo các lớp thực thi interface Shape

Circle.java

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

Step 3: Tạo interface Color.java

Color.java

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

Step 4: Tạo các lớp thực thi interface Color

Red.java

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

Step 4: Tạo các lớp thực thi interface Color

Green.java

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

Step 4: Tạo các lớp thực thi interface Color

Blue.java

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

Step 5: Xây dựng lớp abstract khởi tạo các đối tượng

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

Step 6: Triển khai lớp abstract đã xây dựng ở bước 5

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;
   }
}
```

Step 6: Triển khai lớp abstract đã xây dựng ở bước 5

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;
   }
}
```

Step 7: Tạo một Factory generator class để tạo ra Factory dựa trên tham số truyền vào

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;
   }
}
```

Step 7: Tạo một Factory generator class để tạo ra Factory dựa trên tham số truyền vào

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;
   }
}
```

Step 8: Sử dụng Factory generator class để lấy ra factory của các lớp thực thi dựa trên tham số truyền vào

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");
```

Step 8: Sử dụng Factory generator class để lấy ra factory của các lớp thực thi dựa trên tham số truyền vào

```
//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();
  }
}
```

Q&A