## 一句话概括：

**可以生产多种产品的超级工厂（简单工厂模式只能上产一种产品）**。

## 补充介绍：

抽象工厂模式（Abstract Factory Pattern）的核心是一个超级工厂，这个超级工厂可以创建很多不同的产品，每种产品又都是简单工厂模式生产出来的，可以说它是简单工厂的工厂。

超级工厂不需要知道具体产品的类型，只需要知道每种产品的基类类型。

如果说简单工厂可以生产某种产品，那么抽象工厂是可以生产一个产品族。

比如工厂A可以上产套套，工厂B可以生产充气娃娃，工厂C生产sex衣物… 那么使用抽象工厂模式可以在抽象超级工厂接口里面定义三个方法：

TaoTao getTaoTao();

WaWa getWaWa();

YiWu getYiWu();

然后定义三个分别生产套套，娃娃和衣物的简单工厂，他们都集成抽象工厂，分别实现各自的具体产品生产方法，其他方法返回null，这样就能将一系列产品组合在一起形成一个产品族了。

## 参与角色:

1）构建抽象工厂的辅助类，根据传入的参数生成具体的工厂

2）抽象工厂类

3）具体工厂类

4）产品类

## 优点：

当一个产品族中的多个对象被设计成一起工作时，它能保证客户端始终只使用同一个产品族中的对象。

## 缺点：

产品族扩展非常困难，要增加一个系列的某一产品，既要在抽象的 Creator 里加代码，又要在具体的里面加代码。

## 使用案例或场景：

使用场景：需要生产产品族的地方

案例：QQ皮肤，一键换肤可以换掉很多很多图标的外观或主题，还有Windows更换主题

## 示例程序

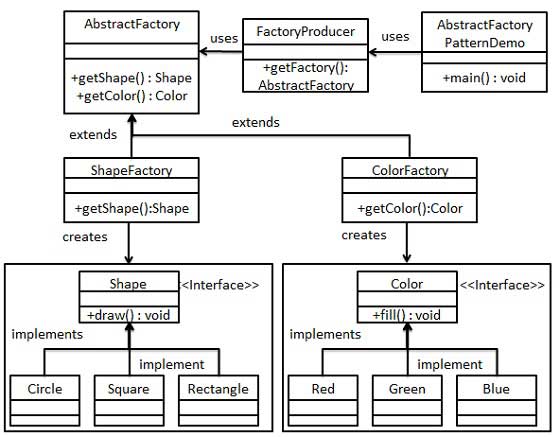
需要源码的朋友可以前往github下载：

<https://github.com/aharddreamer/chendong/tree/master/design-patterns/demo-code/design-patterns>

**程序简介**：

我们将创建 Shape 和 Color 接口和实现这些接口的实体类。下一步是创建抽象工厂类 AbstractFactory。接着定义工厂类 ShapeFactory 和 ColorFactory，这两个工厂类都是扩展了 AbstractFactory。然后创建一个工厂创造器/生成器类 FactoryProducer。

AbstractFactoryPatternDemo，我们的演示类使用 FactoryProducer 来获取 AbstractFactory 对象。它将向 AbstractFactory 传递形状信息 Shape（CIRCLE / RECTANGLE / SQUARE），以便获取它所需对象的类型。同时它还向 AbstractFactory 传递颜色信息 Color（RED / GREEN / BLUE），以便获取它所需对象的类型。



**代码：**

步骤 1

为形状创建一个接口。

Shape.java

public interface Shape {

void draw();

}

步骤 2

创建实现接口的实体类。

Rectangle.java

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.");

}

}

步骤 3

为颜色创建一个接口。

Color.java

public interface Color {

void fill();

}

步骤4

创建实现接口的实体类。

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.");

}

}

步骤 5

为 Color 和 Shape 对象创建抽象类来获取工厂。

AbstractFactory.java

public abstract class AbstractFactory {

public abstract Color getColor(String color);

public abstract Shape getShape(String shape) ;

}

步骤 6

创建扩展了 AbstractFactory 的工厂类，基于给定的信息生成实体类的对象。

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

public Color getColor(String color) {

return null;

}

}

ColorFactory.java

public class ColorFactory extends AbstractFactory {

@Override

public Shape getShape(String shapeType){

return null;

}

@Override

public 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;

}

}

步骤 7

创建一个工厂创造器/生成器类，通过传递形状或颜色信息来获取工厂。

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;

}

}

步骤 8

使用 FactoryProducer 来获取 AbstractFactory，通过传递类型信息来获取实体类的对象。

AbstractFactoryPatternDemo.java

public class AbstractFactoryPatternDemo {

public static void main(String[] args) {

//获取形状工厂

AbstractFactory shapeFactory = FactoryProducer.getFactory("SHAPE");

//获取形状为 Circle 的对象

Shape shape1 = shapeFactory.getShape("CIRCLE");

//调用 Circle 的 draw 方法

shape1.draw();

//获取形状为 Rectangle 的对象

Shape shape2 = shapeFactory.getShape("RECTANGLE");

//调用 Rectangle 的 draw 方法

shape2.draw();

//获取形状为 Square 的对象

Shape shape3 = shapeFactory.getShape("SQUARE");

//调用 Square 的 draw 方法

shape3.draw();

//获取颜色工厂

AbstractFactory colorFactory = FactoryProducer.getFactory("COLOR");

//获取颜色为 Red 的对象

Color color1 = colorFactory.getColor("RED");

//调用 Red 的 fill 方法

color1.fill();

//获取颜色为 Green 的对象

Color color2 = colorFactory.getColor("Green");

//调用 Green 的 fill 方法

color2.fill();

//获取颜色为 Blue 的对象

Color color3 = colorFactory.getColor("BLUE");

//调用 Blue 的 fill 方法

color3.fill();

}

}

步骤 9

执行程序，输出结果：

Inside Circle::draw() method.

Inside Rectangle::draw() method.

Inside Square::draw() method.

Inside Red::fill() method.

Inside Green::fill() method.

Inside Blue::fill() method.

参考：

《抽象工厂模式》菜鸟教程网站