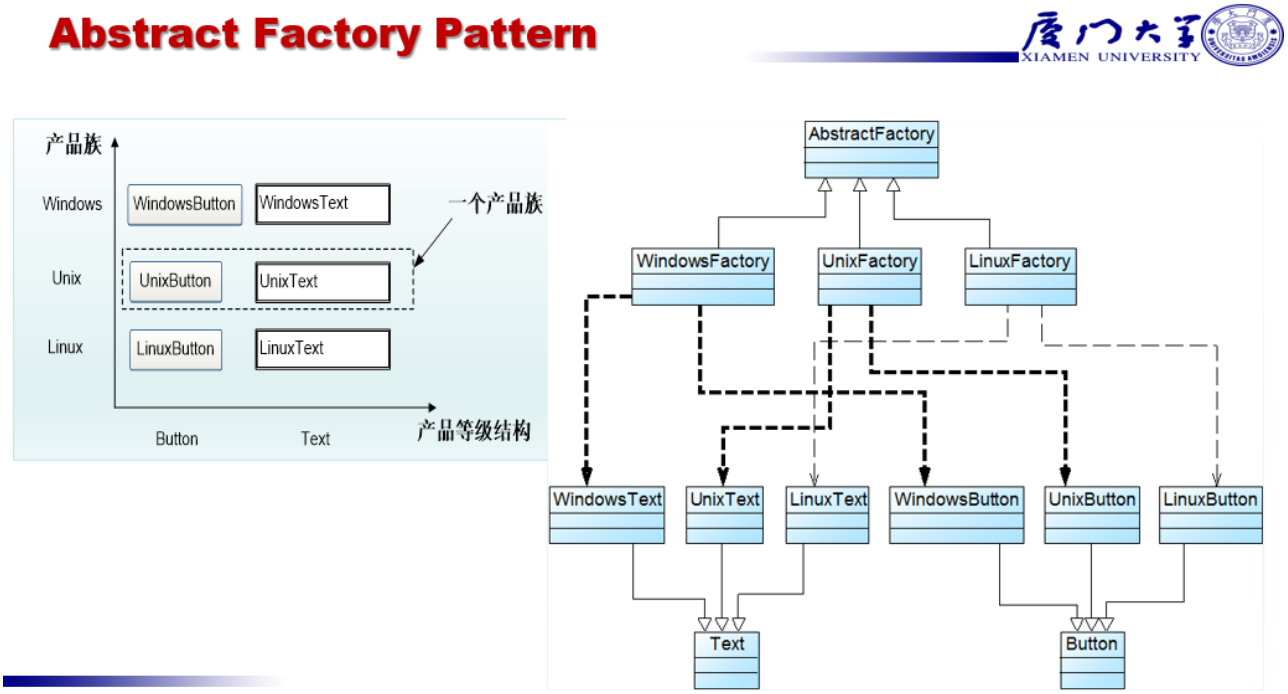
软件体系结构 作业13

22920212204392 黄勖

1　阅读Abstract Factory的例子的代码，举例说明使用Abstract Factory模式的其他应用。

1.1　阅读Abstract Factory的例子的代码

抽象工厂模式提供一个创建一系列相关或相互依赖对象的接口，而无需指定他们具体的类。其中AbstractFactory声明一个创建抽象产品对象的操作接口。ConcreteFactory实现创建具体产品对象的操作。AbstractProduct为一类产品对象声明一个接口。ConcreteProduct定义一个将被相应的具体工厂创建的产品对象；实现AbstractProduct的接口。



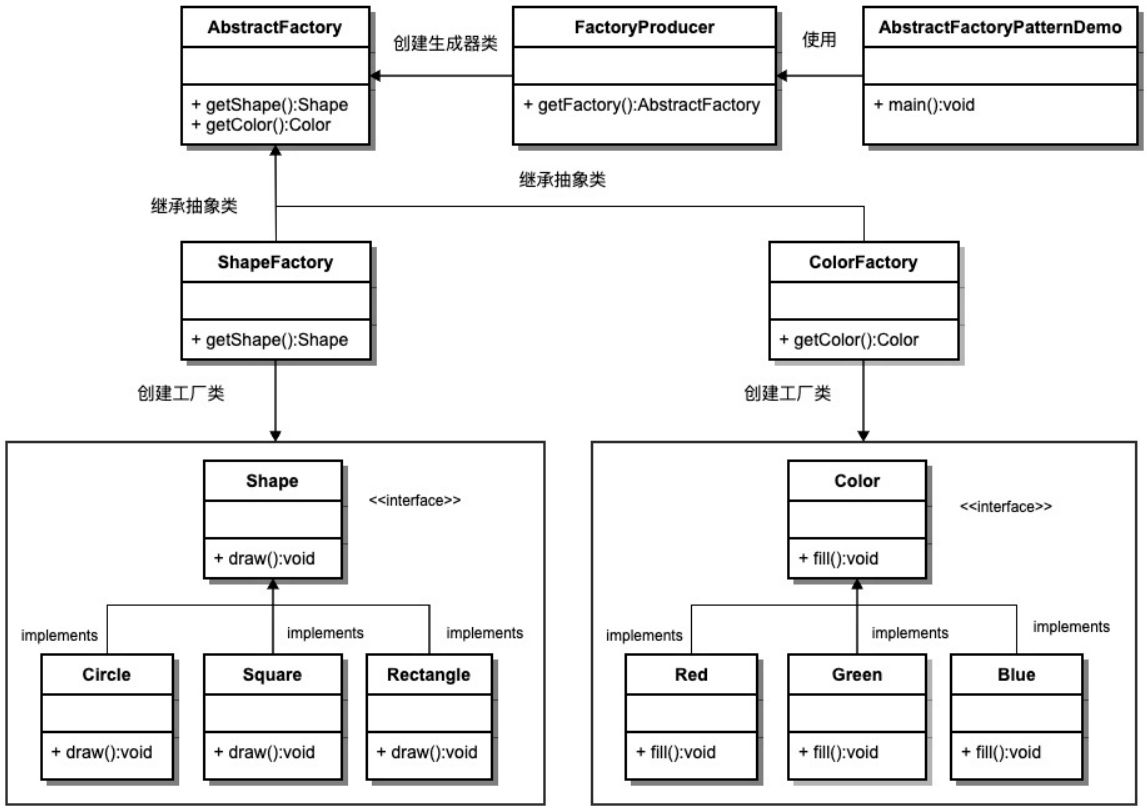
而在实例代码中，便是将一个层次结构的相关网站链接做成了HTML文件。程序中包括三个包：Factory包（Factory、Item、Link类）包含抽象工厂、零件和产品。Listfactory包（ListFactory、ListLink、ListPage、ListTray类）包含List的具体工厂、零件和产品。TableFactory包（TableFactory、TableLink、TablePage、TableTray类）包含Table的具体工厂、零件和产品。实现生产一组包含Link、Page、Tray的产品。

Example:

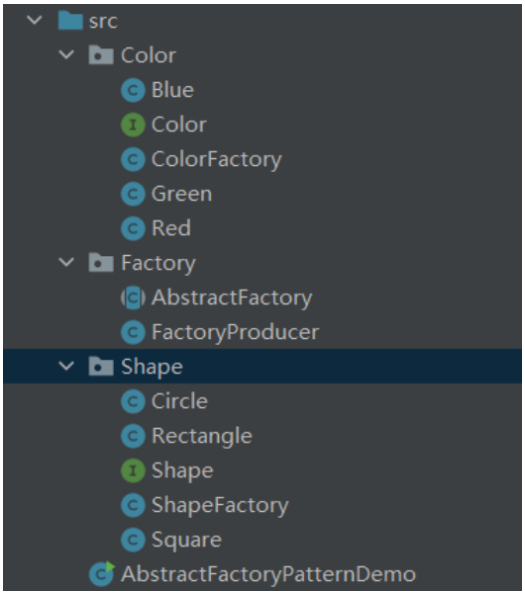
- 将一个层次结构的相关网站链接做成HTML文件
- 程序中包括三个包
  - Factory包：含抽象工厂、零件和产品的包
  - 未命名的包：含Main类的包
  - Listfactory包：含具体工厂、零件和产品的包
- Source Code
  - Factory.java Item.java Link.java
  - Page.java Tray.java
  - ListFactory.java ListLink.java
  - ListPage.java ListTray.java
  - Main.java

1.2 举例说明使用Abstract Factory模式的其他应用

下面用另一个例子说明Abstract Factory模式的应用。在例子中，创建了Shape和Color接口和实现这些接口的实体类。以及抽象工厂类 AbstractFactory、具体工厂类ShapeFactory和ColorFactory，然后创建一个工厂创造器/生成器类FactoryProducer。Demo类使用FactoryProducer来获取AbstractFactory对象。它将向AbstractFactory传递形状信息Shape和颜色信息Color，以便获取它所需对象的类型。



代码结构如下：



首先是为形状创建一个接口，然后再创建实现接口的三个类Rectangle、Circle、Shape。

AbstractFactory.java | Shape.java | ShapeFactory.java

```
1  package Shape;
2  public interface Shape {
       3 usages  3 implementations
3      void draw();
4  }
```

AbstractFactory.java | Shape.java | Circle.java | ShapeFactory.java | ColorFactory.java | Fact

```
1  package Shape;
2  public class Circle implements Shape {
       3 usages
3      @Override
4      public void draw() { System.out.println("Inside Shape.Circle::draw() method."); }
7  }
```

AbstractFactory.java | Shape.java | Circle.java | Rectangle.java | ShapeFactory.java

```
1  package Shape;
2  public class Rectangle implements Shape {
       3 usages
3      @Override
4      public void draw() {
5          System.out.println("Inside Shape.Rectangle::draw() method.");
6      }
7  }
```

AbstractFactory.java | Shape.java | Circle.java | Rectangle.java | Square.java

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

与形状类似，为颜色创建一个接口，然后创建实现接口的实体类Red、Green、Blue。

ColorFactory.java | Blue.java | Color.java

```java
package Color;
public interface Color {
    3 usages  3 implementations
    void fill();
}
```

ColorFactory.java | Blue.java | Color.java | Red.java | FactoryProducer.java | AbstractFactoryPatternDemo.java

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

ColorFactory.java | Blue.java | Color.java | Red.java | Green.java | FactoryProducer.java | AbstractFactoryPatternDemo.java

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

ColorFactory.java | Blue.java | Color.java | Red.java | Green.java | FactoryProducer.java

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

之后给Color和Shape对象创建抽象工厂AbstractFactory，再创建扩展AbstractFactory的具体工厂ColorFactory和AbstractFactory，基于给定的信息生成实体类的对象。

ColorFactory.java | Blue.java | AbstractFactory.java | Color.java

```java
package Factory;
import Color.Color;
import Shape.Shape;
8 usages  2 inheritors
public abstract class AbstractFactory {
    3 usages  2 implementations
    public abstract Color getColor(String color);
    3 usages  2 implementations
    public abstract Shape getShape(String shape);
}
```

ColorFactory.java | Blue.java | Color.java | Red.java | Green.java | FactoryProdu

```java
import Factory.AbstractFactory;
import Shape.Shape;
2 usages
public class ColorFactory extends AbstractFactory {
    3 usages
    @Override
    public Shape getShape(String shapeType){
        return null;
    }
    3 usages
    @Override
    public Color getColor(String color) {
        if(color == null){
            return null;
        }
        if(color.equalsIgnoreCase( anotherString: "RED")){
            return new Red();
        } else if(color.equalsIgnoreCase( anotherString: "GREEN")){
            return new Green();
        } else if(color.equalsIgnoreCase( anotherString: "BLUE")){
            return new Blue();
        }
        return null;
    }
}
```

```java
import Factory.AbstractFactory;
import Color.Color;
2 usages
public class ShapeFactory extends AbstractFactory {
    3 usages
    @Override
    public Shape getShape(String shapeType){
        if(shapeType == null){
            return null;
        }
        if(shapeType.equalsIgnoreCase( anotherString: "CIRCLE")){
            return new Circle();
        } else if(shapeType.equalsIgnoreCase( anotherString: "RECTANGLE")){
            return new Rectangle();
        } else if(shapeType.equalsIgnoreCase( anotherString: "SQUARE")){
            return new Square();
        }
        return null;
    }
    3 usages
    @Override
    public Color getColor(String color) { return null; }
}
```

最后，创建一个工厂创造器FactoryProducer，通过传递形状或颜色来获取工厂。

```java
package Factory;
import Color.ColorFactory;
import Shape.ShapeFactory;
3 usages
public class FactoryProducer {
    2 usages
    public static AbstractFactory getFactory(String choice){
        if(choice.equalsIgnoreCase( anotherString: "SHAPE")){
            return new ShapeFactory();
        } else if(choice.equalsIgnoreCase( anotherString: "COLOR")){
            return new ColorFactory();
        }
        return null;
    }
}
```

在Demo类中测试，使用抽象工厂创造不同类型的产品，首先通过调用FactoryProducer的getFactory方法获取工厂对象 shapeFactory/colorFactory。

使用工厂获取具体的对象，如 shapeFactory.getShape("CIRCLE") 获取一个圆形对象，调用具体对象的 draw() 方法或fill() 方法进行绘制。

```java
import Color.Color;
import Factory.AbstractFactory;
import Factory.FactoryProducer;
import Shape.Shape;

public class AbstractFactoryPatternDemo {
    public static void main(String[] args) {
        //获取形状工厂
        AbstractFactory shapeFactory = FactoryProducer.getFactory( choice: "SHAPE");
        //获取形状为 Shape.Circle 的对象
        Shape shape1 = shapeFactory.getShape("CIRCLE");
        //调用 Shape.Circle 的 draw 方法
        shape1.draw();
        //获取形状为 Shape.Rectangle 的对象
        Shape shape2 = shapeFactory.getShape("RECTANGLE");
        //调用 Shape.Rectangle 的 draw 方法
        shape2.draw();
        //获取形状为 Shape.Square 的对象
        Shape shape3 = shapeFactory.getShape("SQUARE");
        //调用 Shape.Square 的 draw 方法
        shape3.draw();

        //获取颜色工厂
        AbstractFactory colorFactory = FactoryProducer.getFactory( choice: "COLOR");
        //获取颜色为 Color.Color.Red 的对象
        Color color1 = colorFactory.getColor("RED");
        //调用 Color.Color.Red 的 fill 方法
        color1.fill();
        //获取颜色为 Color.Color.Green 的对象
        Color color2 = colorFactory.getColor("GREEN");
        //调用 Color.Color.Green 的 fill 方法
        color2.fill();
        //获取颜色为 Color.Color.Blue 的对象
        Color color3 = colorFactory.getColor("BLUE");
        //调用 Color.Color.Blue 的 fill 方法
        color3.fill();
    }
}
```

输出结果如下，可以看到调用了Shape和Color这两组对象的方法进行绘制，符合要求。

```
Run: AbstractFactoryPatternDemo
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Pr
Inside Shape.Circle::draw() method.
Inside Shape.Rectangle::draw() method.
Inside Shape.Square::draw() method.
Inside Color.Color.Red::fill() method.
Inside Color.Color.Green::fill() method.
Inside Color.Color.Blue::fill() method.

Process finished with exit code 0
```