链滴

GOF 设计模式小白教程之抽象工厂

作者：valarchie

原文链接：https://ld246.com/article/1566911409185

来源网站：链滴

抽象工厂

定义：

是一种为访问类提供一个创建一组相关或相互依赖对象的接口，且访问类无须指定所要产品的具体类就能得到同族的不同等级的产品的模式结构。

通俗解释：

有一个很爱吃汉堡可乐的顾客，他每次去肯德基或者麦当劳的时候，只需要跟服务员说我要吃汉堡薯条可乐即可。这里的顾客就是访问类，快餐店就是抽象工厂，肯德基和麦当劳是工厂实现类。汉堡、可乐是抽象产品族。肯德基生产具体产品奥尔良鸡腿堡和百事可乐。麦当劳生产具体产品麦辣鸡腿堡和可口可乐。

代码：

抽象工厂类：提供获取汉堡和可乐的方法

```
public interface AbstractFactory {
    Burger getBurger();
    Cola getCola();
}
```

具体工厂类：肯德基和麦当劳

```
public class KentuckyFactory implements AbstractFactory{
    @Override
    public Burger getBurger() {
        return new KentuckyBurger();
    }
    @Override
    public Cola getCola() {
        return new PepsiCola();
    }
}

public class McDonaldFactory implements AbstractFactory{
    @Override
    public Burger getBurger() {
        return new McDonaldBurger();
    }

    @Override
    public Cola getCola() {
        return new CokeCola();
    }
}
```

抽象产品族：汉堡和可乐

```
public interface Burger {
    void eat();
}

public interface Cola {
    void drink();
}
```

具体产品族：奥尔良鸡腿堡百事可乐 和 麦辣鸡腿堡和可口可乐

```
public class KentuckyBurger implements Burger{
    @Override
    public void eat() {
        System.out.println("吃奥尔良鸡腿堡！");
    }
}

public class PepsiCola implements Cola{

    @Override
    public void drink() {
        System.out.println("喝百事可乐！");
    }
}

public class McDonaldBurger implements Burger {
    @Override
    public void eat() {
        System.out.println("吃麦辣鸡腿堡！");
    }
}
public class CokeCola implements Cola{

    @Override
    public void drink() {
        System.out.println("喝可口可乐！");
    }

}
```

测试：这样一来我们就可以只用eatBurgerAndCola()这个方法来处理不同的快餐店不同的汉堡和可乐。

```
public class TestAbstractFactory {

    public static void main(String[] args) {

        eatBurgerAndCola(new KentuckyFactory());
        eatBurgerAndCola(new McDonaldFactory());

    }
```

```
    public static void eatBurgerAndCola(AbstractFactory factory) {
        Burger burger = factory.getBurger();
        burger.eat();
        Cola cola = factory.getCola();
        cola.drink();
    }

}
```

解析：

当增加一个新的产品族时不需要修改原代码，满足开闭原则。

但是当产品族中需要增加一个新的产品时，所有的工厂类都需要进行修改。