# 实验二：工厂模式

## 实验内容

简单工厂方法模式：利用简单工厂方法模式创建pad, phone, watch的对象，并使用这些对象娱乐。

工厂模式：利用工厂模式创建pad, phone, watch的对象，并使用这些对象娱乐。

抽象工厂模式：利用抽象工厂模式创建华为、小米、苹果的pad, phone, watch的对象，并使用这些对象娱乐。

参考代码下载：<https://gitee.com/liangsan/sa-factory.git> ，抽象工厂模式的代码部分还没有完善，请注意更新。

类名、包名、项目名等请参考，可以适当调整，但不要随意乱命名。

格式等参照默认的规范，细节这里就不在仔细描述了。

## 简单工厂方法模式

### UML设计



图1 简单工厂方法类图

### 核心代码

FactoryClient.java核心代码

public class FactoryClient {

private static void testSimpleFactory() {

// 简单工厂模式

ConsumerElectronics pad = cn.edu.scau.sec.tangxuexi.simpleFactoryMethod.factory.ConsumerElectronicsFactory.getConsumerElectronics("pad");

pad.recreation();

ConsumerElectronics phone = cn.edu.scau.sec.tangxuexi.simpleFactoryMethod.factory.ConsumerElectronicsFactory.getConsumerElectronics("phone");

phone.recreation();

ConsumerElectronics watch = cn.edu.scau.sec.tangxuexi.simpleFactoryMethod.factory.ConsumerElectronicsFactory.getConsumerElectronics("watch");

watch.recreation();

}}

ConsumerElectronicsFactory.java核心代码

public static ConsumerElectronics getConsumerElectronics(String consumerElectronicsType) {

ConsumerElectronics consumerElectronics = null;

switch(consumerElectronicsType) {

case "pad": consumerElectronics = new Pad();break;

case "phone": consumerElectronics = new Phone();break;

case "watch": consumerElectronics = new Watch();break;

}

return consumerElectronics;

}

ConsumerElectronics.java核心代码

void recreation();

Pad.java核心代码

public void recreation() {

System.out.println("使用平板娱乐");

}

Phone.java核心代码

public void recreation() {

System.out.println("使用手机娱乐");

}

Watch.java核心代码

public void recreation() {

System.out.println("使用手表娱乐");

}

### 实现效果

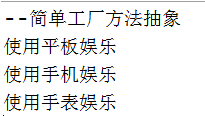


图2 简单工厂方法实现效果图

## 工厂模式

### UML设计



图3 工厂模式类图

### 核心代码

FactoryClient.java核心代码

private static void testFactoryMethod() {

// 工厂模式

ConsumerElectronicsFactory padFactory = new PadFactory();

ConsumerElectronics pad = padFactory.getConsumerElectronics();

pad.recreation();

ConsumerElectronicsFactory phoneFactory = new PhoneFactory();

ConsumerElectronics phone = phoneFactory.getConsumerElectronics();

phone.recreation();

ConsumerElectronicsFactory watchFactory = new WatchFactory();

ConsumerElectronics watch = watchFactory.getConsumerElectronics();

watch.recreation();

}

ConsumerElectronicsFactory.java核心代码

public abstract class ConsumerElectronicsFactory {

public abstract ConsumerElectronics getConsumerElectronics();

}

PadFactory.java核心代码

public ConsumerElectronics getConsumerElectronics() {

return new Pad();

};

PhoneFactory.java核心代码

public ConsumerElectronics getConsumerElectronics() {

return new Phone();

};

WatchFactory.java核心代码

public ConsumerElectronics getConsumerElectronics() {

return new Watch();

};

### 实现效果

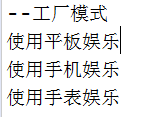


图4 工厂模式实现效果图

## 抽象工厂模式

### UML设计



图5 抽象工厂模式类图

### 核心代码

FactoryClient.java核心代码

private static void testAbstractFactory() {

// 抽象工厂

AbstractFactory huaweiFactory = AbstractFactory.getConsumerElectronicsFactory("huawei");

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics huaweiPad = huaweiFactory.getPad();

huaweiPad.recreation();

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics huaweiPhone = huaweiFactory.getPhone();

huaweiPhone.recreation();

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics huaweiWatch = huaweiFactory.getWatch();

huaweiWatch.recreation();

AbstractFactory appleFactory = AbstractFactory.getConsumerElectronicsFactory("apple");

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics applePad = appleFactory.getPad();

applePad.recreation();

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics applePhone = appleFactory.getPhone();

applePhone.recreation();

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics appleWatch = appleFactory.getWatch();

appleWatch.recreation();

AbstractFactory miFactory = AbstractFactory.getConsumerElectronicsFactory("mi");

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics miPad = miFactory.getPad();

miPad.recreation();

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics miPhone = miFactory.getPhone();

miPhone.recreation();

cn.edu.scau.sec.tangxuexi.abstractFactory.abstractClass.ConsumerElectronics miWatch = miFactory.getWatch();

miWatch.recreation();

}

AbstractFactory.java核心代码

public static AbstractFactory getConsumerElectronicsFactory(String consumerElectronicsBrand) {

AbstractFactory factory = null;

switch(consumerElectronicsBrand) {

case "huawei": factory = new HuaweiFactory();break;

case "apple": factory = new AppleFactory();break;

case "mi": factory = new MiFactory();break;

}

return factory;

}

public abstract ConsumerElectronics getPad();

public abstract ConsumerElectronics getPhone();

public abstract ConsumerElectronics getWatch();

AppleFactory.java核心代码

public ConsumerElectronics getPad(){

return new ApplePad();

}

public ConsumerElectronics getPhone(){

return new ApplePhone();

}

public ConsumerElectronics getWatch(){

return new AppleWatch();

}

HuaweiFactory.java核心代码

public ConsumerElectronics getPad() {

return new HuaweiPad();

}

public ConsumerElectronics getPhone() {

return new HuaweiPhone();

}

public ConsumerElectronics getWatch() {

return new HuaweiWatch();

}

MiFactory.java核心代码

public ConsumerElectronics getPad() {

return new MiPad();

}

public ConsumerElectronics getPhone() {

return new MiPhone();

}

public ConsumerElectronics getWatch() {

return new MiWatch();

}

ConsumerElectronics.java核心代码

abstract public void recreation();

Pad.java核心代码

protected String productName = "平板";

Phone.java核心代码

protected String productName = "手机";

Watch.java核心代码

protected String productName = "手表";

ApplePad.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

ApplePhone.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

AppleWatch.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

HuaweiPad.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

HuaweiPhone.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

HuaweiWatch.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

MiPad.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

MiPhone.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

MiWatch.java核心代码

public void recreation() {

System.out.println(brandName + productName);

}

Apple.java核心代码

static final String brandName = "苹果";

AppleBrand.java核心代码

public static final String name = "苹果";

Huawei.java核心代码

static final String brandName = "华为";

HuaweiBrand.java核心代码

public static final String name = "华为";

Mi.java核心代码

static final String brandName = "小米";

MiBrand.java核心代码

public static final String name = "小米";

### 实现效果

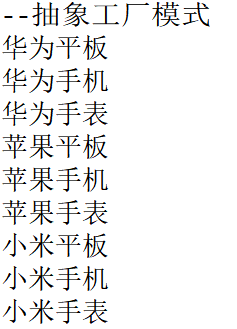


图6 抽象工厂模式实现效果图

## 实验体会

通过这次实验，我学习到了三种工厂模式的使用场景以及使用的方法。虽然这次的实验总体来说比较抽象，但是其实仔细想想，落实到现实中其实也是非常有用的一些方法。例如抽象工厂模式，看似创建的包，创建的类很多，但是其实在现实生产中，反而是最有用的一个方法。