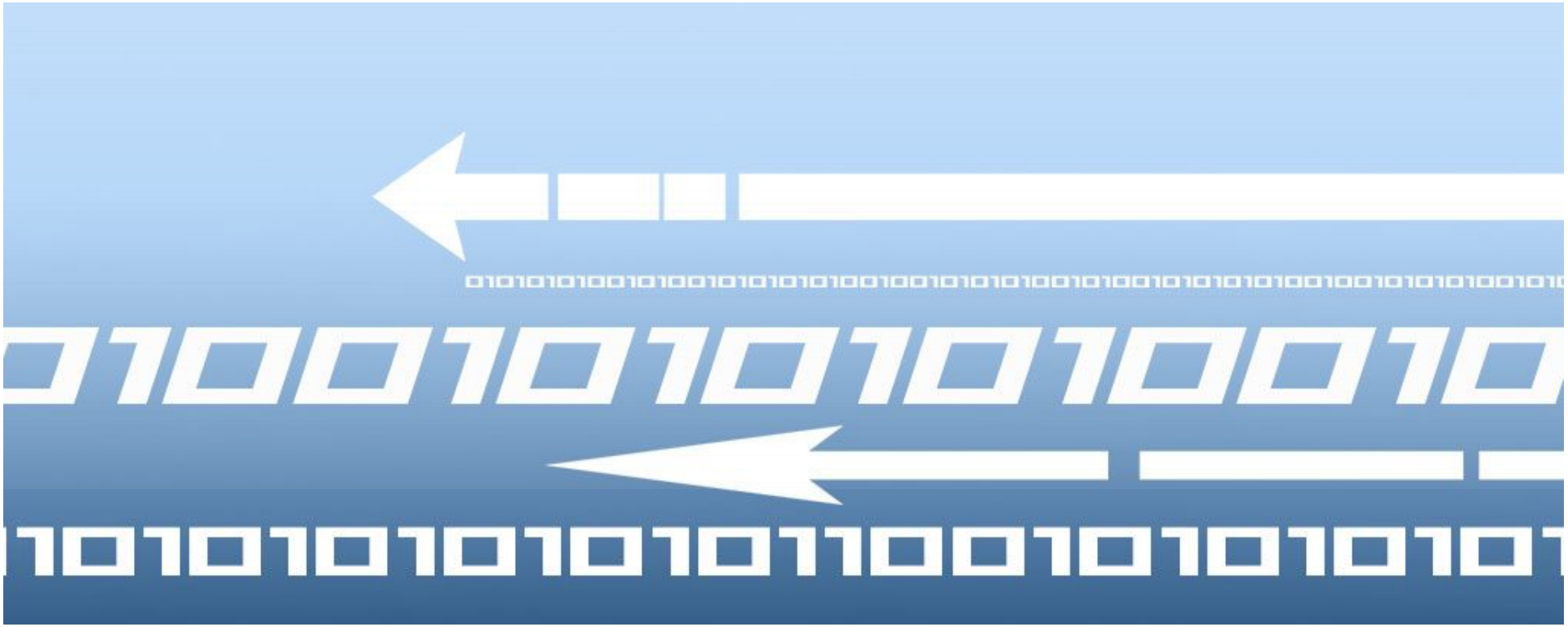


简单工厂模式

包子

2014-07

1. 创建型模式
2. 简单工厂模式概述
3. 简单工厂模式的结构与实现
4. 简单工厂模式的应用实例
5. 简单工厂模式的模式扩展

初开垫毫弓

- ✓ 创建型模式(Creational Pattern)关注对象的创建过程
- ✓ 创建型模式对类的实例化过程进行了抽象，能够将软件模块中对象的创建和对象的使用分离，对用户隐藏了类的实例的创建细节
- ✓ 创建型模式描述如何将对象的创建和使用分离，让用户在使用对象时无须关心对象的创建细节，从而降低系统的耦合度，让设计方案更易于修改和扩展

初开垫毫弓

- ✓ 创建型模式关注点
  - 创建什么(What)
  - 由谁创建(Who)
  - 何时创建(When)

创建对象

使用对象

初开垫毫弓

✓ 创建型模式概述

想吃苹果!?

初开垫毫弓

✓ 创建型模式概述

获取苹果的两种方式

自己种苹果树

去超市买

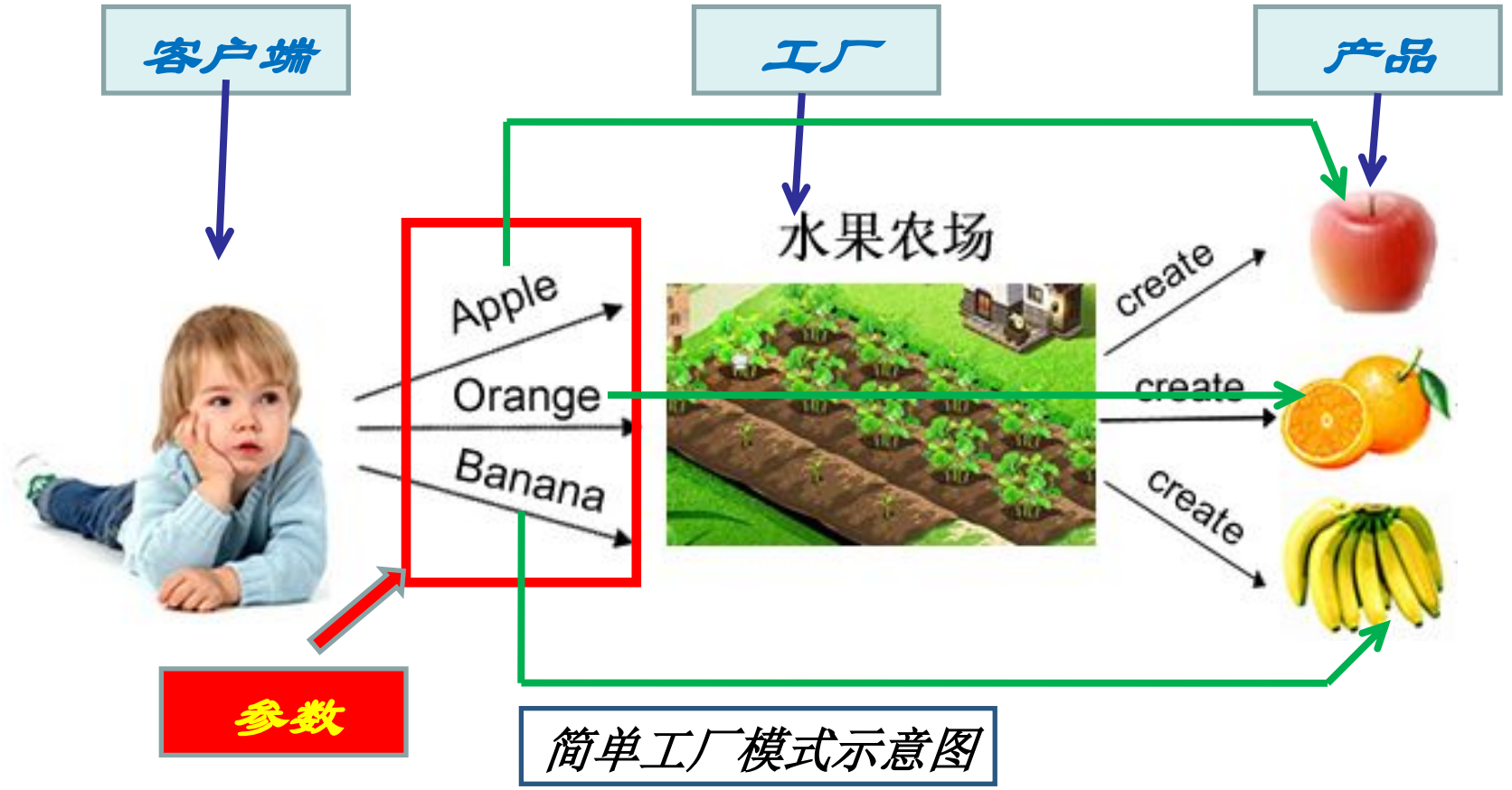
初开垫毫弓

模式名称	定 义	学习难度	使用频率
简单工厂模式 (Simple Factory Pattern)	定义一个工厂类，它可以根据参数的不同返回不同类的实例，被创建的实例通常都具有共同的父类。	★★☆☆☆	★★★☆☆
工厂方法模式 (Factory Method Pattern)	定义一个用于创建对象的接口，但是让子类决定将哪一个类实例化。工厂方法模式让一个类的实例化延迟到其子类。	★★☆☆☆	★★★★★
抽象工厂模式 (Abstract Factory Pattern)	提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。	★★★★☆	★★★★★
建造者模式 (Builder Pattern)	将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。	★★★★☆	★★☆☆☆
原型模式 (Prototype Pattern)	使用原型实例指定待创建对象的类型，并且通过复制这个原型来创建新的对象。	★★★☆☆	★★★☆☆
单例模式 (Singleton Pattern)	确保一个类只有一个实例，并提供一个全局访问点来访问这个唯一实例。	★☆☆☆☆	★★★★☆

天群

1. 创建型模式
2. 简单工厂模式概述
3. 简单工厂模式的结构与实现
4. 简单工厂模式的应用实例
5. 简单工厂模式的模式扩展

练卖左厄毫弓比释



简单工厂模式示意图

练卖左厄毫弓

✓ 模式动机

- 考虑一个简单的软件应用场景，一个软件系统可以提供多个外观不同的按钮（如圆形按钮、矩形按钮、菱形按钮等），这些按钮都源自同一个基类，不过在继承基类后不同的子类修改了部分属性从而使得它们可以呈现不同的外观，如果我们希望在使用这些按钮时，不需要知道这些具体按钮类的名字，只需要知道表示该按钮类的一个参数，并提供一个调用方便的方法，把该参数传入方法即可返回一个相应的按钮对象，此时，就可以使用简单工厂模式。

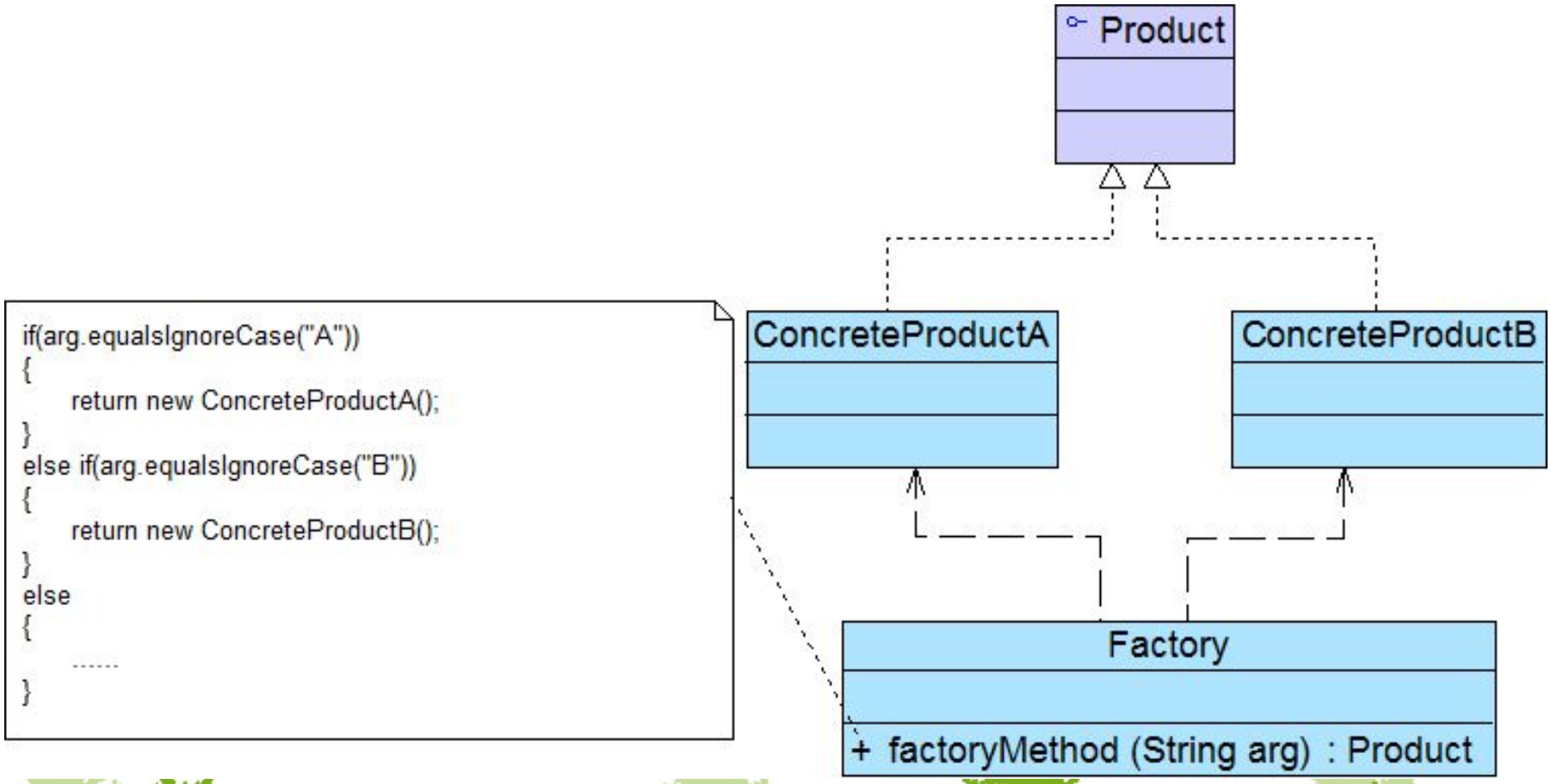
练卖左厄毫弓

✓模式定义

- 简单工厂模式(Simple Factory Pattern)：又称为静态工厂方法(Static Factory Method)模式，它属于类创建型模式。在简单工厂模式中，可以根据参数的不同返回不同类的实例。简单工厂模式专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类。

练卖左厄毫弓

✓模式结构


Product
ConcreteProductA
ConcreteProductB
Factory
+ factoryMethod (String arg) : Product
if(arg.equalsIgnoreCase("A"))
{
return new ConcreteProductA();
}
else if(arg.equalsIgnoreCase("B"))
{
return new ConcreteProductB();
}
else
{
.....
}

练卖左厄毫弓

✓ 模式结构

- 简单工厂模式包含如下角色：

  Factory：工厂角色

  Product：抽象产品角色

  ConcreteProduct：具体产品角色

1. 创建型模式
2. 简单工厂模式概述
3. 简单工厂模式的结构与实现
4. 简单工厂模式的应用实例
5. 简单工厂模式的模式扩展

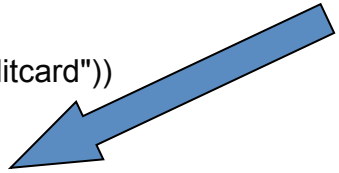
练卖左厄毫弓

✓模式分析

- 分析如下代码：

```
public void pay(String type)
{
    if(type.equalsIgnoreCase("cash"))
    {
        //现金支付处理代码
    }
    else if(type.equalsIgnoreCase("creditcard"))
    {
        //信用卡支付处理代码
    }
    else if(type.equalsIgnoreCase("voucher"))
    {
        //代金券支付处理代码
    }
    else
    {
        ……
    }
}
```

代码复杂，难以维护



练卖左厄毫弓

✓模式分析

- 重构后的代码：

抽象支付类

```
public abstract class AbstractPay
{
   public abstract void pay();
}
```

具体支付类

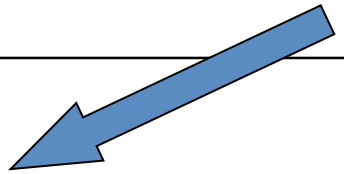
```
public class CashPay extends AbstractPay
{
    public void pay()
    {
        //现金支付处理代码
    }
}
```

练卖左厄毫弓

✓ 模式分析

- 重构后的代码：

支付工厂



```
public class PayMethodFactory
{
    public static AbstractPay getPayMethod(String type)
    {
        if(type.equalsIgnoreCase("cash"))
        {
            return new CashPay();       //根据参数创建具体产品
         }
        else if(type.equalsIgnoreCase("creditcard"))
        {
            return new CreditcardPay();   //根据参数创建具体产品
        }
        ……
    }
}
```

练卖左厄毫弓

✓ 模式分析

- 将对象的创建和对象本身业务处理分离可以降低系统的耦合度，使得两者修改起来都相对容易。
- 在调用工厂类的工厂方法时，由于工厂方法是静态方法，使用起来很方便，可通过类名直接调用，而且只需要传入一个简单的参数即可，在实际开发中，还可以在调用时将所传入的参数保存在XML等格式的配置文件中，修改参数时无须修改任何Java源代码。
- 简单工厂模式最大的问题在于工厂类的职责相对过重，增加新的产品需要修改工厂类的判断逻辑，这一点与开闭原则是相违背的。
- 简单工厂模式的要点在于：当你需要什么，只需要传入一个正确的参数，就可以获取你所需要的对象，而无须知道其创建细节。

✓ 模式实例与解析

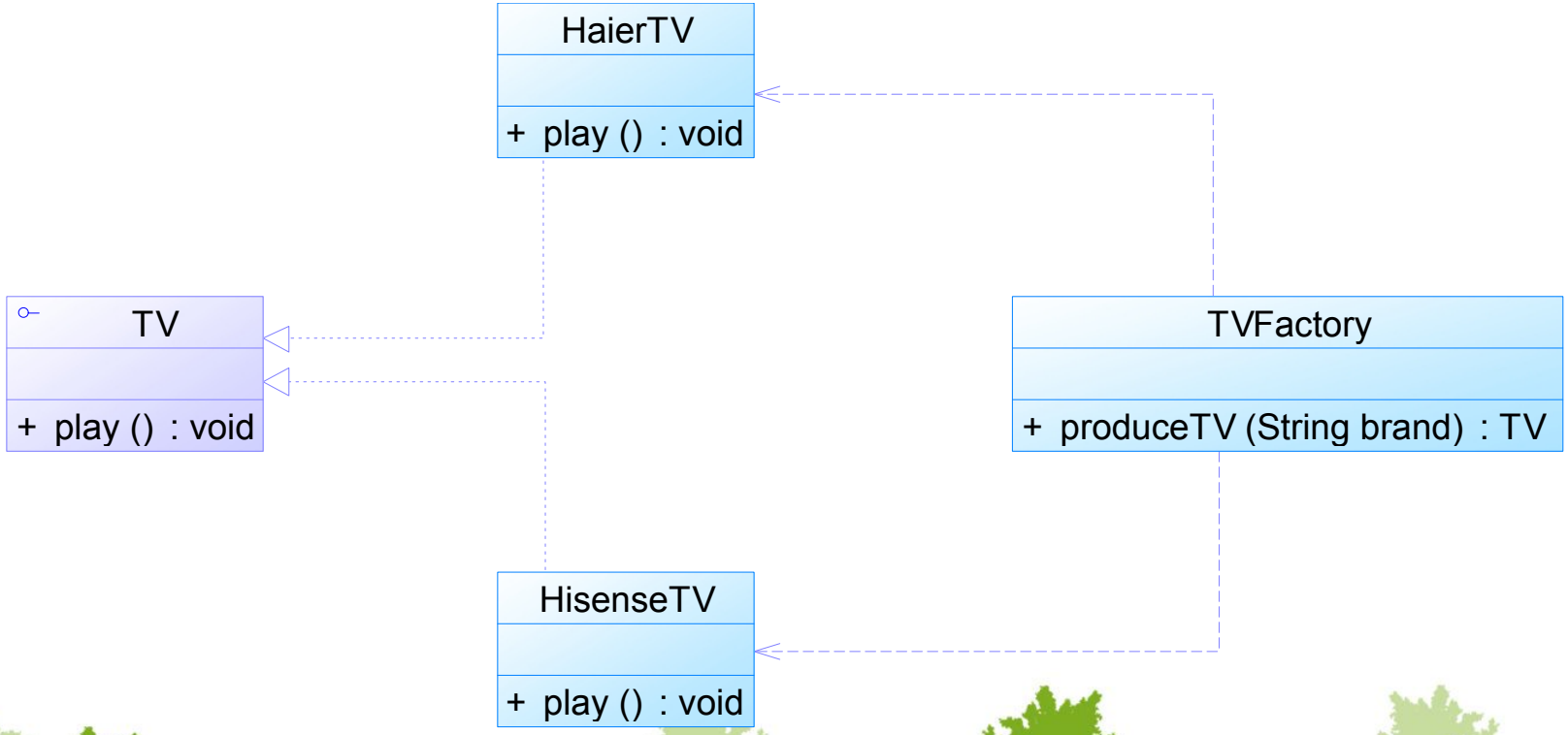
实例一：简单电视机工厂

某电视机厂专为各知名电视机品牌代工生产各类电视机，当需要海尔牌电视机时只需要在调用该工厂的工厂方法时传入参数“Haier”，需要海信电视机时只需要传入参数“Hisense”，工厂可以根据传入的不同参数返回不同品牌的电视机。现使用简单工厂模式来模拟该电视机工厂的生产过程。

1. 创建型模式
2. 简单工厂模式概述
3. 简单工厂模式的结构与实现
4. 简单工厂模式的应用实例
5. 简单工厂模式的模式扩展

✓ 模式实例与解析

实例一：简单电视机工厂



练卖左厄毫弓

✓ 模式实例与解析

- 实例一：简单电视机工厂

  参考代码(Chapter 04 Simple Factory\sample01)

演示.....

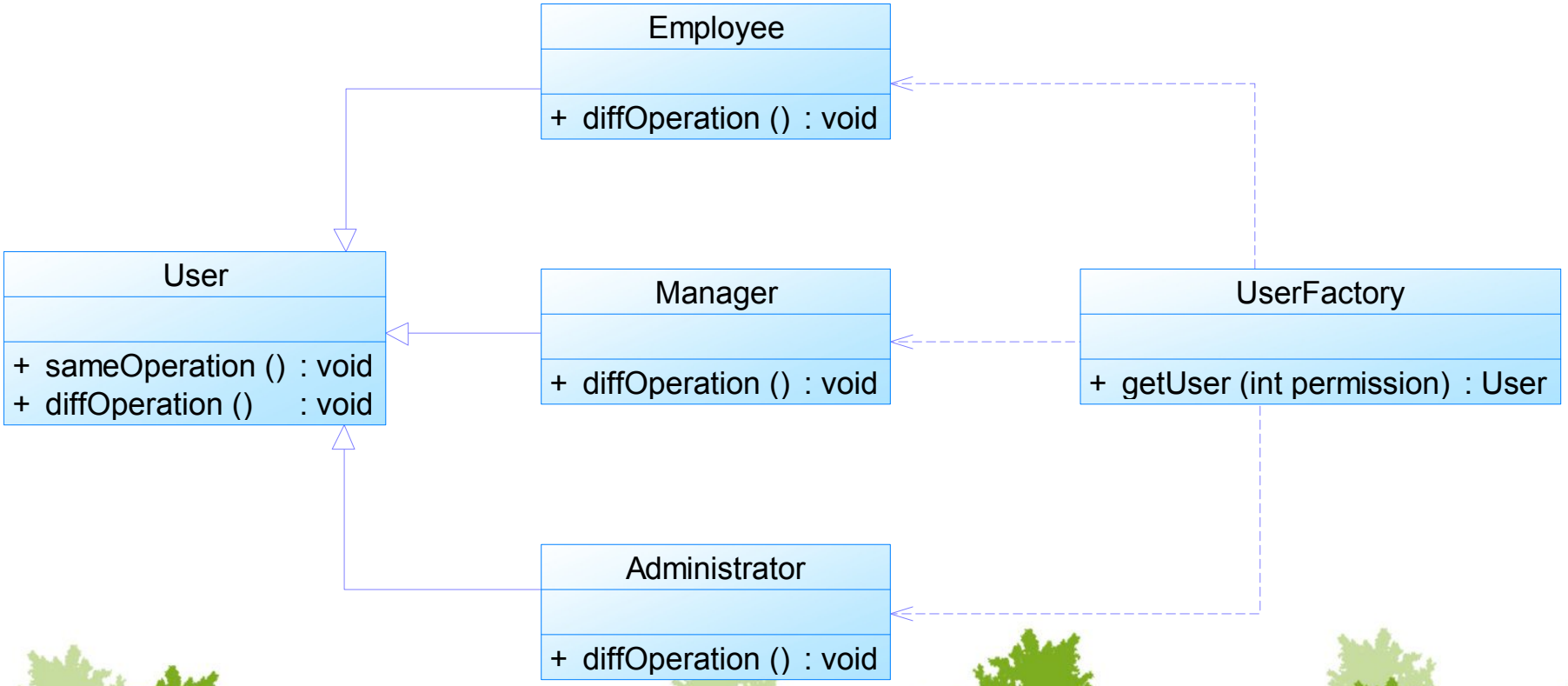
✓ 模式实例与解析

实例二：权限管理

在某OA系统中，系统根据对比用户在登录时输入的账号和密码以及在数据库中存储的账号和密码是否一致来进行身份验证，如果验证通过，则取出存储在数据库中的用户权限等级（以整数形式存储），根据不同的权限等级创建不同等级的用户对象，不同等级的用户对象拥有不同的操作权限。现使用简单工厂模式来设计该权限管理模块。

✓模式实例与解析

□实例二：权限管理



练卖左厄毫弓

✓模式实例与解析

- 实例二：权限管理

  参考代码(Chapter 04 Simple Factory\sample02)

演示.....

1. 创建型模式
2. 简单工厂模式概述
3. 简单工厂模式的结构与实现
4. 简单工厂模式的应用实例
5. 简单工厂模式的模式扩展

练卖左厄毫弓碍伙腊玉专釜看皆墙

✓模式优点

- 实现了对象创建和使用的分离
- 客户端无须知道所创建的具体产品类的类名，只需要知道具体产品类所对应的参数即可
- 通过引入配置文件，可以在不修改任何客户端代码的情况下更换和增加新的具体产品类，在一定程度上提高了系统的灵活性

练卖左厄毫弓碍伙腊玉专釜看皆墙

✓ 模式缺点

- 工厂类集中了所有产品的创建逻辑，职责过重，一旦不能正常工作，整个系统都要受到影响
- 增加系统中类的个数（引入了新的工厂类），增加了系统的复杂度和理解难度
- 系统扩展困难，一旦添加新产品不得不修改工厂逻辑
- 由于使用了静态工厂方法，造成工厂角色无法形成基于继承的等级结构，工厂类不能得到很好地扩展

✓模式适用环境

- 在以下情况下可以使用简单工厂模式：

  工厂类负责创建的对象比较少：由于创建的对象较少，不会造成工厂方法中的业务逻辑太过复杂。

  客户端只知道传入工厂类的参数，对于如何创建对象不关心：客户端既不需要关心创建细节，甚至连类名都不需要记住，只需要知道类型所对应的参数。

初开寻踩专例看寻踩

- ✓ JAVA语言创建对象的几种方式
  - 使用new关键字直接创建对象
  - 通过反射机制创建对象
  - 通过克隆方法创建对象
  - 通过工厂类创建对象

练卖左厄毫弓

- ✓ **模式应用**
  - (1) 在JDK类库中广泛使用了简单工厂模式，如工具类java.text.DateFormat，它用于格式化一个本地日期或者时间。

```
public final static DateFormat getDateInstance();
public final static DateFormat getDateInstance(int style);
public final static DateFormat getDateInstance(int style,Locale
    locale);
```

练卖左厄毫弓

- ✓ **模式应用**
  - (2) Java加密技术
  - 参考代码：**DESEncrypt.java**

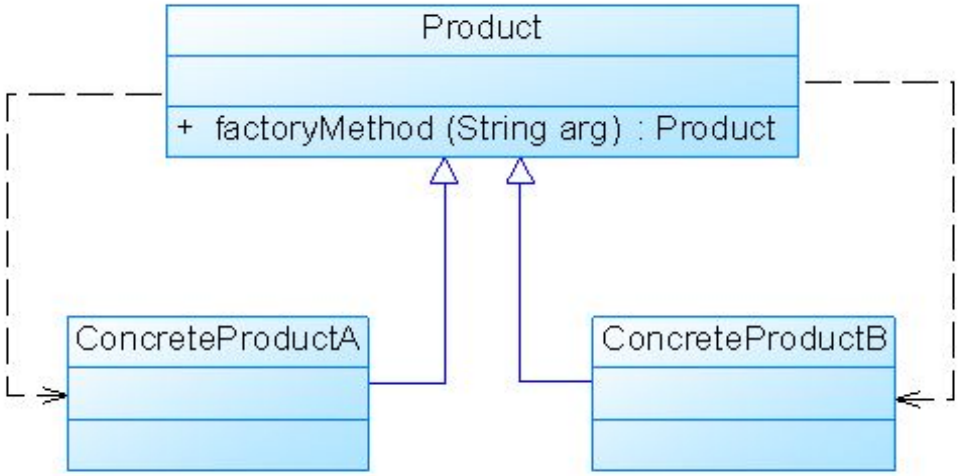
```
//获取不同加密算法的密钥生成器
KeyGenerator keyGen=KeyGenerator.getInstance("DESede");
```

```
//创建密码器
Cipher cp=Cipher.getInstance("DESede");
```

✓ 模式扩展

简单工厂模式的简化：

在有些情况下工厂类可以由抽象产品角色扮演，一个抽象产品类同时也是子类的工厂，也就是说把静态工厂方法写到抽象产品类中。



初开寻踩专例看寻踩

✓何时不需要工厂？

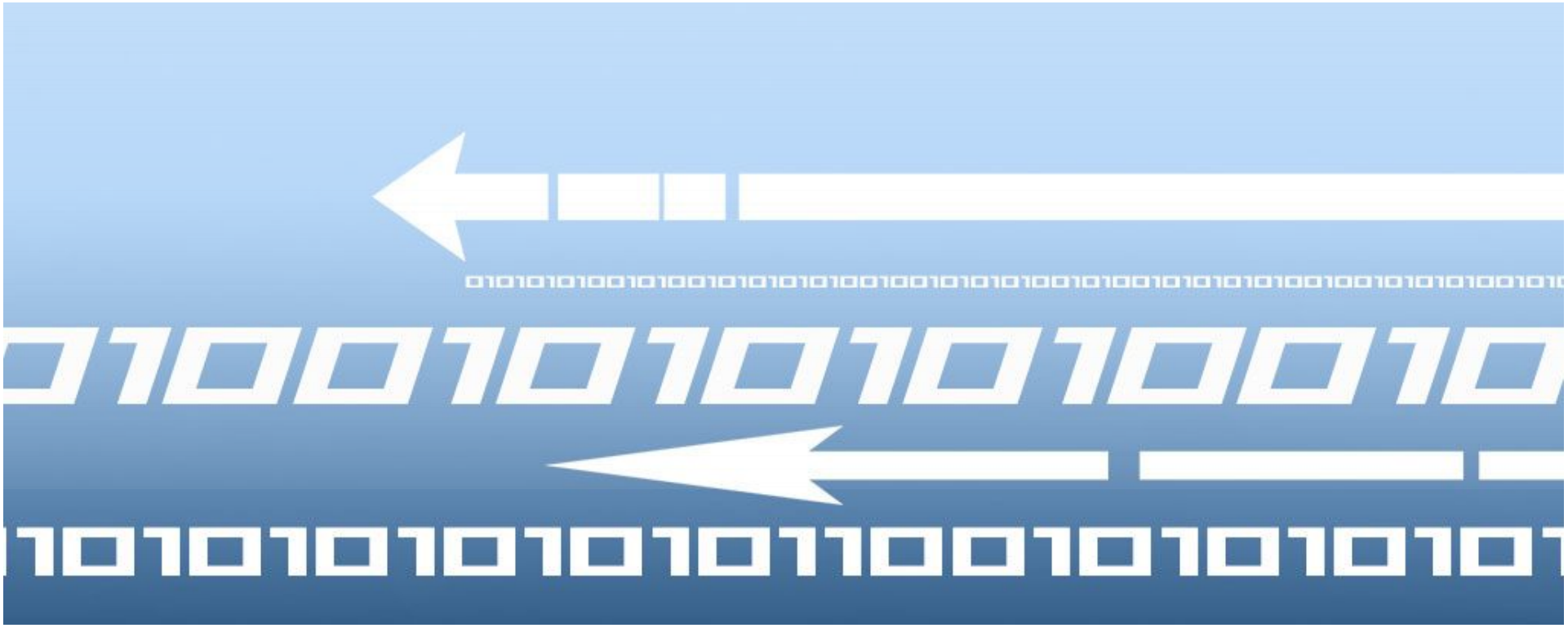
- 无须为系统中的每一个类都配备一个工厂类
- 如果一个类很简单，而且不存在太多变化，其构造过程也很简单，此时就无须为其提供工厂类，直接在使用之前实例化即可
- 否则会导致工厂泛滥，增加系统的复杂度
- 例如：string类

柏繁少聋

- ✓ 创建型模式对类的实例化过程进行了抽象，能够将对象的创建与对象的使用过程分离。
- ✓ 简单工厂模式又称为静态工厂方法模式，它属于类创建型模式。在简单工厂模式中，可以根据参数的不同返回不同类的实例。简单工厂模式专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类。
- ✓ 简单工厂模式包含三个角色：工厂角色负责实现创建所有实例的内部逻辑；抽象产品角色是所创建的所有对象的父类，负责描述所有实例所共有的公共接口；具体产品角色是创建目标，所有创建的对象都充当这个角色的某个具体类的实例。

柏繁少聋

- ✓ 简单工厂模式的要点在于：当你需要什么，只需要传入一个正确的参数，就可以获取你所需要的对象，而无须知道其创建细节。
- ✓ 简单工厂模式最大的优点在于实现对象的创建和对象的使用分离，将对象的创建交给专门的工厂类负责，但是其最大的缺点在于工厂类不够灵活，增加新的具体产品需要修改工厂类的判断逻辑代码，而且产品较多时，工厂方法代码将会非常复杂。
- ✓ 简单工厂模式适用情况包括：工厂类负责创建的对象比较少；客户端只知道传入工厂类的参数，对于如何创建对象不关心。



感谢观赏

QQ：253692170

13755115139@139.com