深圳大学
Shenzhen University

第7讲

设计模式概述&简单工厂模式&工厂方法模式

软件体系结构与设计模式

Software Architecture & Design Pattern

深圳大学计算机与软件学院

主要内容

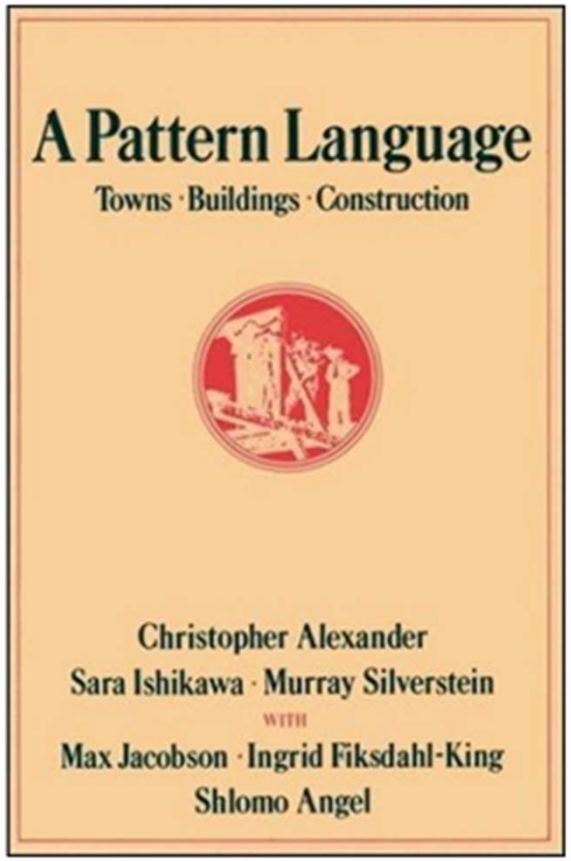
- **5.1**设计模式的诞生与发展
- **5.2**设计模式的定义与分类
- **5.3 GoF**设计模式简介
- **5.4** 简单工厂模式
- **5.5** 工厂方法模式

5.1设计模式的诞生与发展

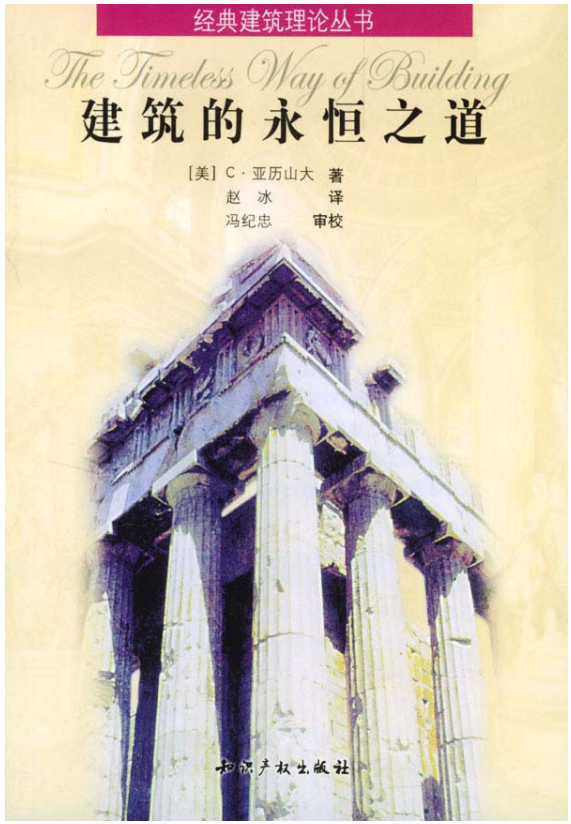
- 模式的诞生
  - 模式(Pattern)起源于建筑业而非软件业
  - 模式之父——美国加利佛尼亚大学环境结构中心研究所所长Christopher Alexander博士
  - 《A Pattern Language: Towns, Buildings, Construction》——253个建筑和城市规划模式
  - 模式
    - Context（模式可适用的前提条件）
    - Theme或Problem（在特定条件下要解决的目标问题）
    - Solution（对目标问题求解过程中各种物理关系的记述）

模式的诞生

- 模式之父——Christopher Alexander博士



Christopher Alexander

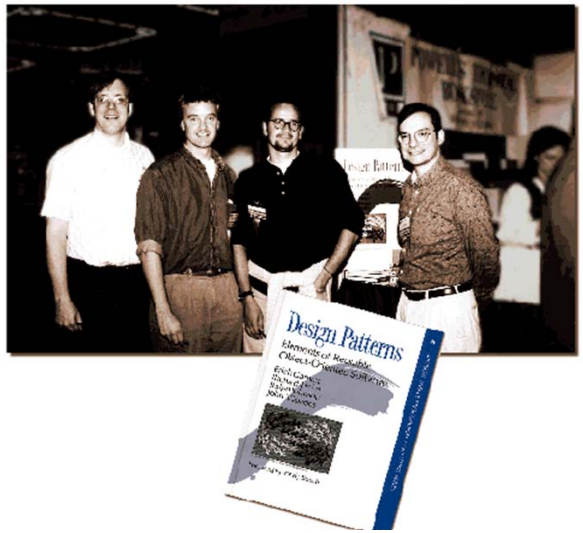


模式的定义

- Alexander给出了关于模式的经典定义：
  - 每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心，通过这种方式，人们可以无数次地重用那些已有的解决方案，无需再重复相同的工作
- 模式是在特定环境中解决问题的一种方案 (A pattern is a solution to a problem in a context)

软件模式

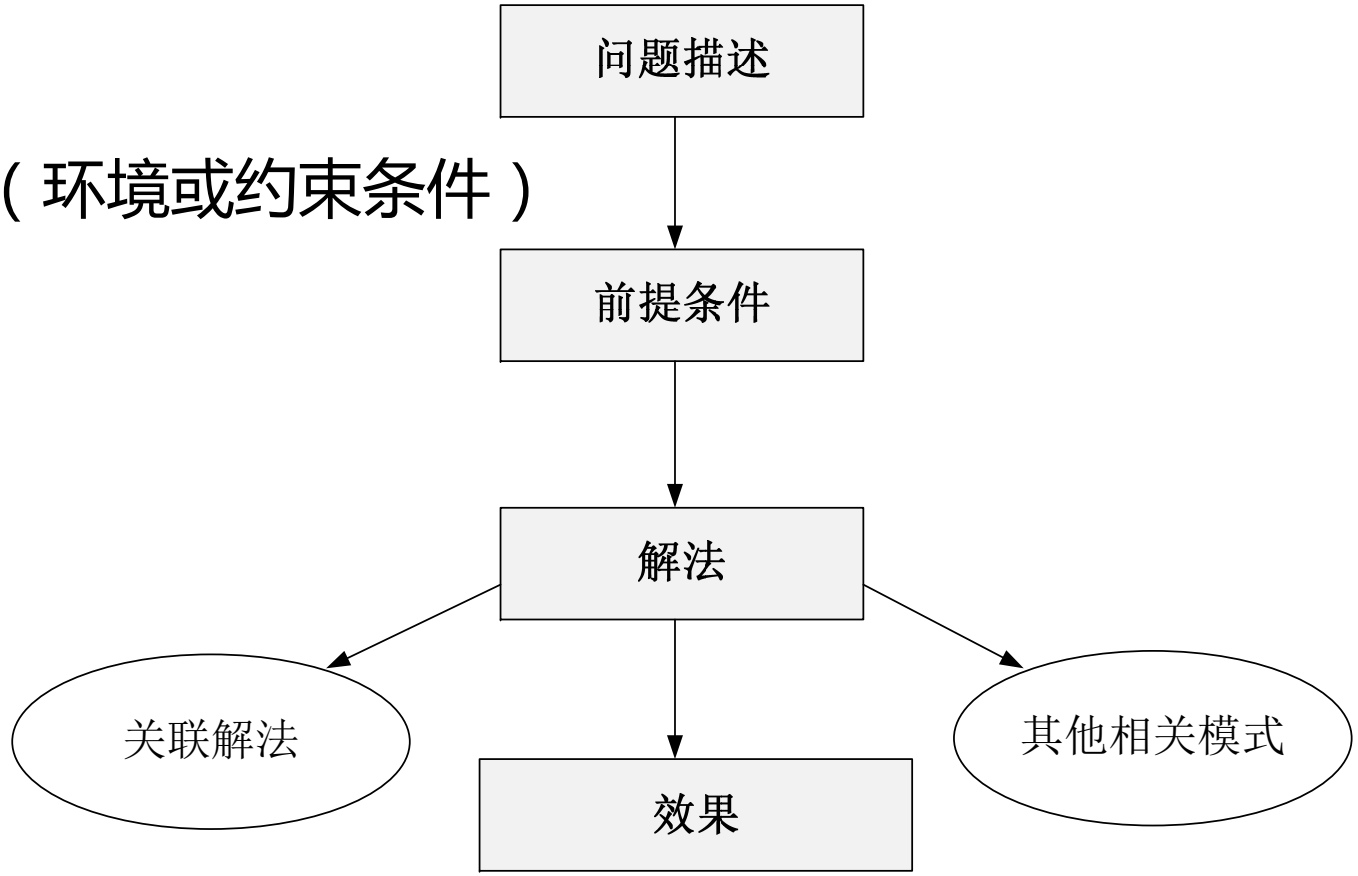
- 20世纪80年代末，软件工程界开始关注Christopher Alexander等在住宅、公共建筑与城市规划领域的这一重大突破
- “四人组(Gang of Four，GoF，分别是Erich Gamma, Richard Helm, Ralph Johnson和John Vlissides)”于1994年归纳发表了23种在软件开发中使用频率较高的设计模式，旨在用模式来统一沟通面向对象方法在分析、设计和实现间的鸿沟



Gang of Four (GoF)

软件模式

- 软件模式：在一定条件下的软件开发问题及其解法
  - 问题描述
  - 前提条件（环境或约束条件）
  - 解法
  - 效果



5.2设计模式的定义与分类

- **设计模式的定义**
  - ✓ 设计模式(Design Pattern)是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结
  - ✓ 使用设计模式是为了可重用代码、让代码更容易被他人理解、提高代码的可靠性

设计模式的基本要素

- 设计模式一般有如下几个基本要素：模式名称、问题、目的、解决方案、效果、实例代码和相关设计模式，其中的关键元素包括以下四个方面：
  - 模式名称 (Pattern name)
  - 问题 (Problem)
  - 解决方案 (Solution)
  - 效果 (Consequences)

设计模式的分类

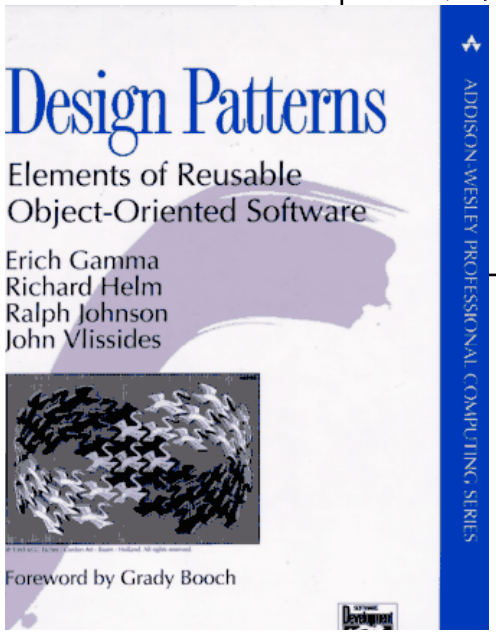
- 根据目的（模式是用来做什么的）可分为创建型(Creational)，结构型(Structural)和行为型(Behavioral)三类：
  - 创建型模式主要用于创建对象
  - 结构型模式主要用于处理类或对象的组合
  - 行为型模式主要用于描述类或对象如何交互和怎样分配职责

设计模式的分类

- 根据范围，即模式主要是处理类之间的关系还是处理对象之间的关系，可分为类模式和对象模式两种：
  - 类模式处理类和子类之间的关系，这些关系通过继承建立，在编译时刻就被确定下来，是一种静态关系
  - 对象模式处理对象间的关系，这些关系在运行时变化，更具动态性

5.3 GoF设计模式简介

范围\目的	创建型模式	结构型模式	行为型模式
类模式	工厂方法模式	（类）适配器模式	解释器模式 模板方法模式
对象模式	抽象工厂模式 建造者模式 原型模式 单例模式	（对象）适配器模式 桥接模式 组合模式 装饰模式 外观模式 享元模式 代理模式	职责链模式 命令模式 迭代器模式 中介者模式 备忘录模式 观察者模式 状态模式 策略模式 访问者模式



创建型模式（5个）

- 抽象工厂模式(Abstract Factory) ★★★★★
- 建造者模式(Builder) ★★☆☆☆
- 工厂方法模式(Factory Method) ★★★★★
- 原型模式(Prototype) ★★★☆☆
- 单例模式(Singleton) ★★★★☆

结构型模式（7个）

- 适配器模式(Adapter) ★★★★☆
- 桥接模式(Bridge) ★★★☆☆
- 组合模式(Composite) ★★★★☆
- 装饰模式(Decorator) ★★★☆☆
- 外观模式(Facade) ★★★★★
- 享元模式(Flyweight) ★☆☆☆☆
- 代理模式(Proxy) ★★★★☆

行为型模式（11个）

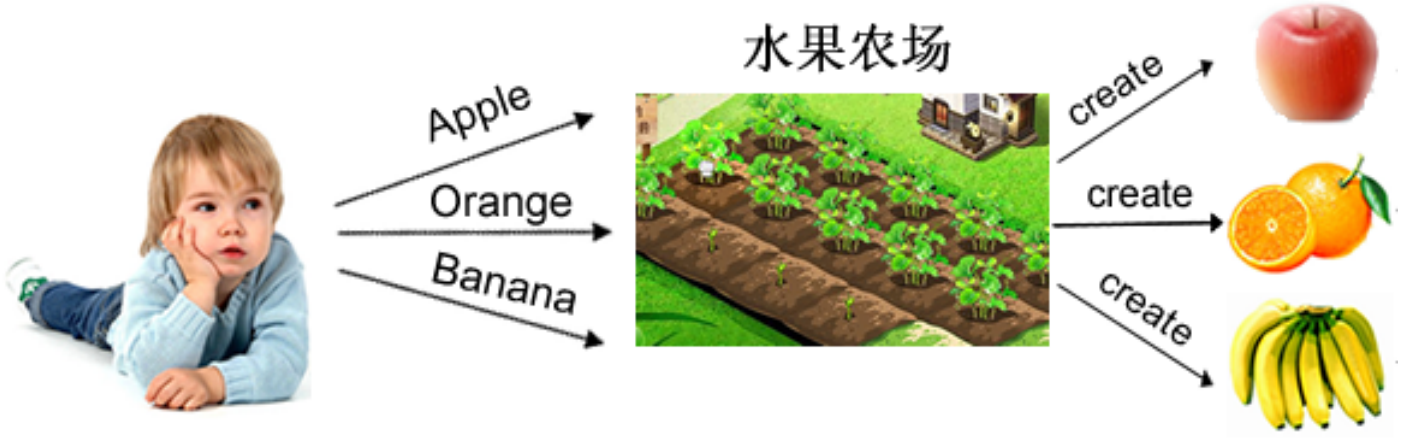
- 职责链模式(Chain of Responsibility) ★★☆☆☆
- 命令模式(Command) ★★★★☆
- 解释器模式(Interpreter) ★☆☆☆☆
- 迭代器模式(Iterator) ★★★★★
- 中介者模式(Mediator) ★★☆☆☆
- 备忘录模式(Memento) ★★☆☆☆
- 观察者模式(Observer) ★★★★★
- 状态模式(State) ★★★☆☆
- 策略模式(Strategy) ★★★★☆
- 模板方法模式(Template Method) ★★★☆☆
- 访问者模式(Visitor) ★☆☆☆☆

设计模式的优点

- 融合了众多专家的经验，并以一种标准的形式供广大开发人员所用
- 提供了一套通用的设计词汇和一种通用的语言，以方便开发人员之间进行沟通和交流，使得设计方案更加通俗易懂
- 让人们可以更加简单方便地复用成功的设计和体系结构
- 使得设计方案更加灵活，且易于修改
- 将提高软件系统的开发效率和软件质量，在一定程度上节约设计成本
- 有助于初学者更深入地理解面向对象思想，方便阅读和学习现有类库与其他系统中的源代码，还可以提高软件的设计水平和代码质量

5.4 简单工厂模式

- **简单工厂模式的模式动机**
  - 只需要知道水果的名字则可得到相应的水果



简单工厂模式的模式定义

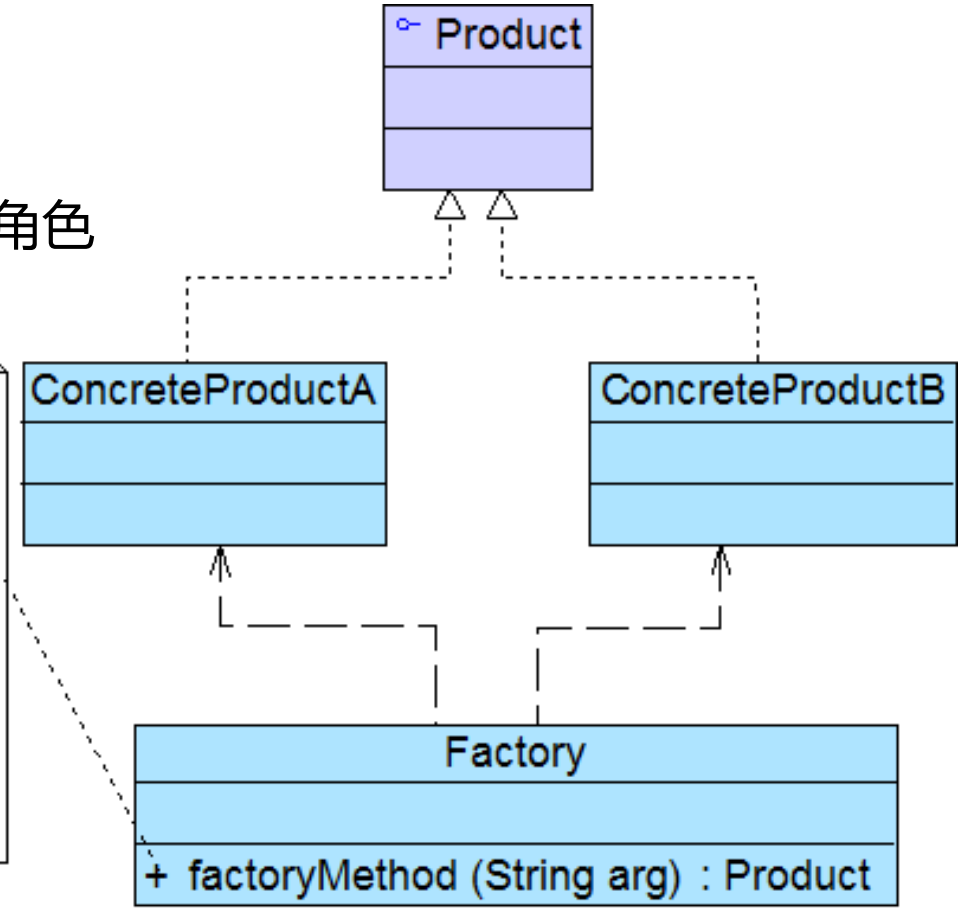
- 简单工厂模式(Simple Factory Pattern)：又称为静态工厂方法(Static Factory Method)模式，它属于类创建型模式
- 在简单工厂模式中，可以根据参数的不同返回不同类的实例
- 简单工厂模式专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类

简单工厂模式的模式结构与分析

简单工厂模式的模式结构

- 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

```
if(arg.equalsIgnoreCase("A"))
{
    return new ConcreteProductA();
}
else if(arg.equalsIgnoreCase("B"))
{
    return new ConcreteProductB();
}
else
{
    ......
}
```



简单工厂模式的模式分析

- 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，可通过工厂类类名直接调用，只需要传入一个简单的参数即可，无须知道对象的创建细节
- 可以将参数保存在XML等格式的配置文件中，修改时无须修改任何Java源代码
- 问题：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，违背开闭原则

简单工厂模式的模式实例与解析

简单工厂模式的模式实例

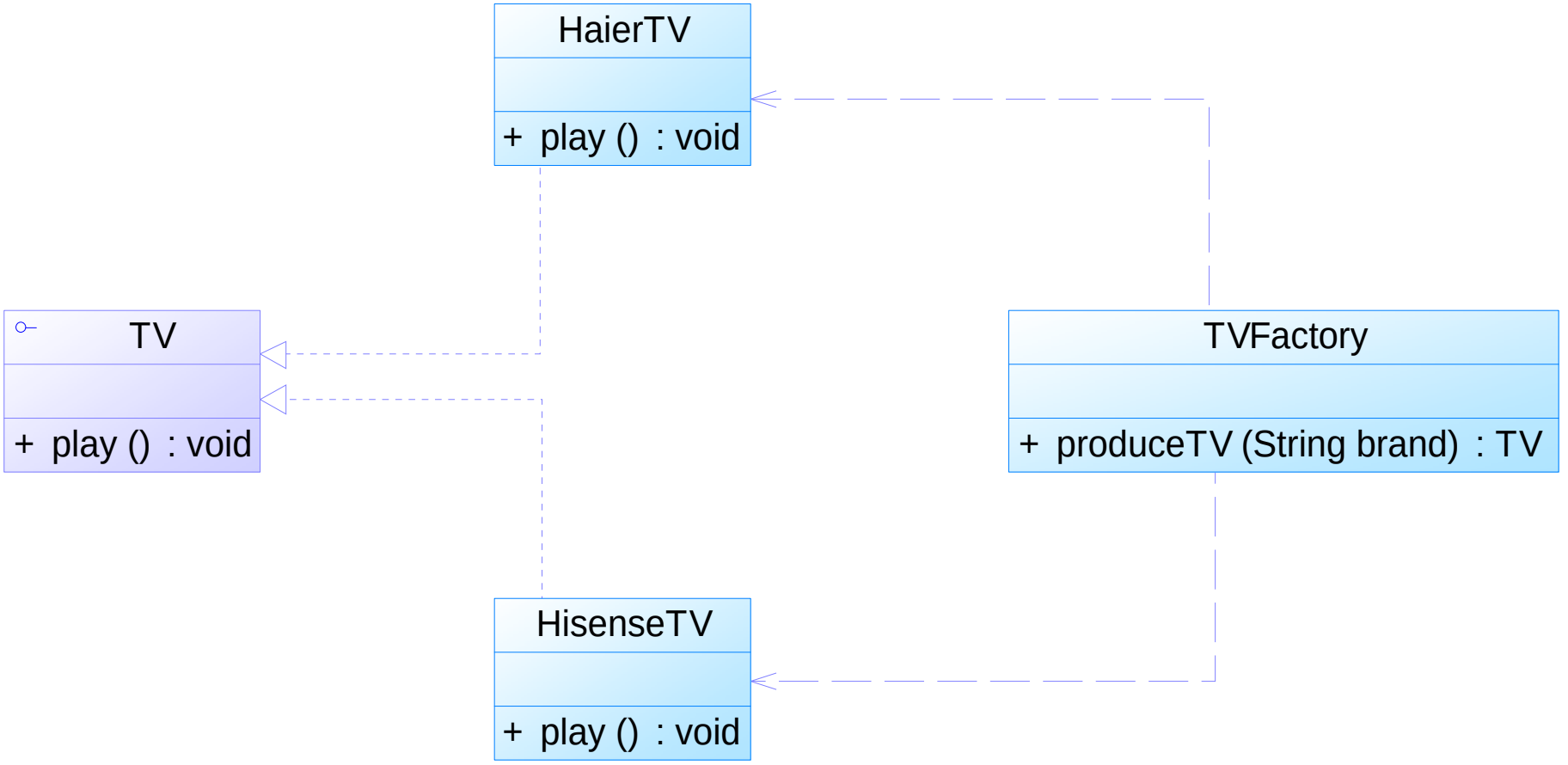
简单电视机工厂：实例说明

- 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。

简单工厂模式的模式实例与解析

简单工厂模式的模式实例

简单电视机工厂：参考类图



简单工厂模式的模式实例与解析

- **简单工厂模式的模式实例**
  - **简单电视机工厂：参考代码**
- DesignPatterns之simplefactory包

TV.java

```
3  public interface TV
4  {
5      public void play();
6  }
```

HaierTV.java

```java
package simplefactory;
import java.io.*;
public class HaierTV implements TV
{
    public void play()
    {
        System.out.println("海尔电视机播放中.....");
    }
}

```

HisenseTV.java

```java
package simplefactory;

public class HisenseTV implements TV
{
    public void play()
    {
        System.out.println("海信电视机播放中.....");
    }
}

```

TVFactory.java

```java
package simplefactory;

public class TVFactory
{
    public static TV produceTV(String brand) throws Exception
    {
        if(brand.equalsIgnoreCase("Haier"))
        {
            System.out.println("电视机工厂生产海尔电视机! ");
            return new HaierTV();
        }
        else if(brand.equalsIgnoreCase("Hisense"))
        {
            System.out.println("电视机工厂生产海信电视机! ");
            return new HisenseTV();
        }
        else
        {
            throw new Exception("对不起，暂不能生产该品牌电视机! ");
        }
    }
}
```

SimpleFactoryconfigTV.xml

```xml
<?xml version="1.0"?>
<config>
    <brandName>Haier</brandName>
</config>

```

Client.java

```java
package simplefactory;

public class Client
{
    public static void main(String args[])
    {
        try
        {
            TV tv;
            String brandName=XMLUtilTV.getBrandName();
            tv=TVFactory.produceTV(brandName);
            tv.play();
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

XMLUtilTV.java

```
import org.w3c.dom.*;
import java.io.*;
public class XMLUtilTV
{
    //该方法用于从XML配置文件中提取品牌名称，并返回该品牌名称
    public static String getBrandName()
    {
        try
        {
            //创建文档对象
            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = dFactory.newDocumentBuilder();
            Document doc;
            doc = builder.parse(new File("SimpleFactoryconfigTV.xml"));
            //获取包含品牌名称的文本节点
            NodeList nl = doc.getElementsByTagName("brandName");
            Node classNode=nl.item(0).getFirstChild();
            String brandName=classNode.getNodeValue().trim();
            return brandName;
           }
            catch(Exception e)
            {
                e.printStackTrace();
                return null;
            }
        }
}
```

简单工厂模式的模式效果与应用

- **简单工厂模式优点：**
  - 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

简单工厂模式的模式效果与应用

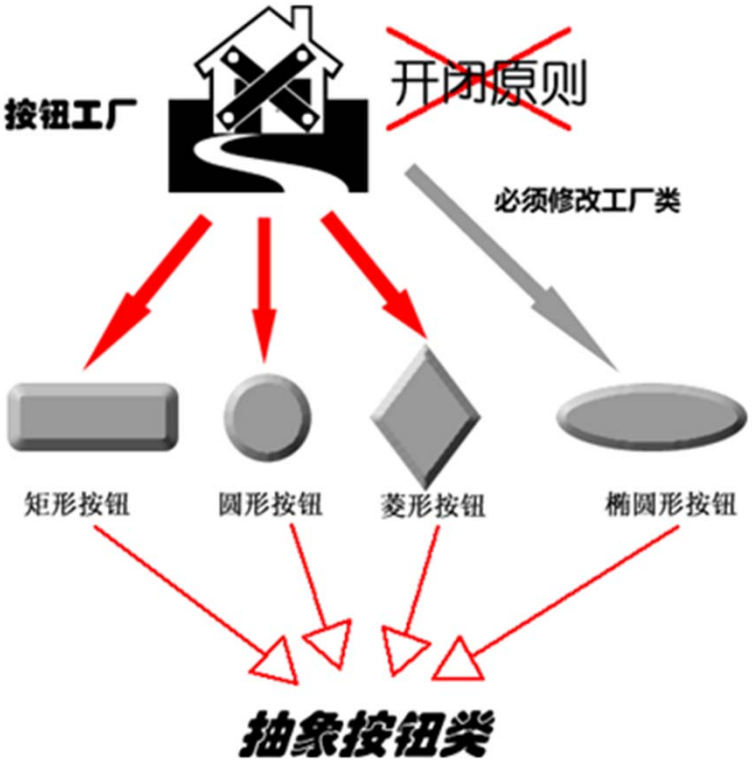
- **简单工厂模式缺点：**
  - 工厂类集中了所有产品的创建逻辑，职责过重，一旦不能正常工作，整个系统都要受到影响
  - 增加系统中类的个数（引入了新的工厂类），增加了系统的复杂度和理解难度
  - 系统扩展困难，一旦添加新产品不得不修改工厂逻辑
  - 由于使用了静态工厂方法，造成工厂角色无法形成基于继承的等级结构，工厂类不能得到很好地扩展

简单工厂模式的模式效果与应用

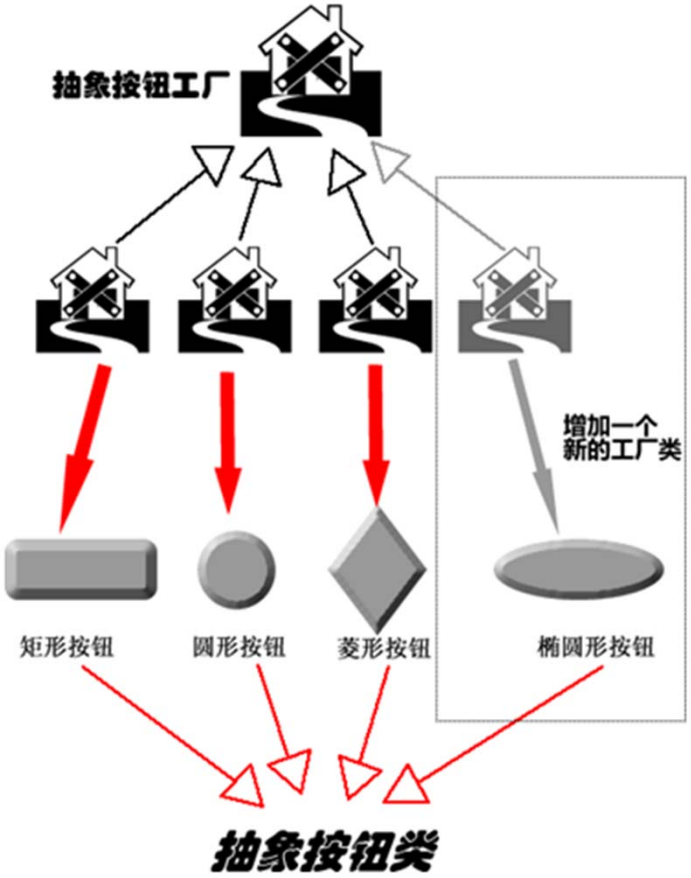
- **在以下情况下可以使用简单工厂模式：**
  - 工厂类负责创建的对象比较少：由于创建的对象较少，不会造成工厂方法中的业务逻辑太过复杂
  - 客户端只知道传入工厂类的参数，对于如何创建对象不关心：客户端既不需要关心创建细节，甚至连类名都不需要记住，只需要知道类型所对应的参数

工厂方法模式

- **工厂方法模式的模式动机**



简单工厂模式



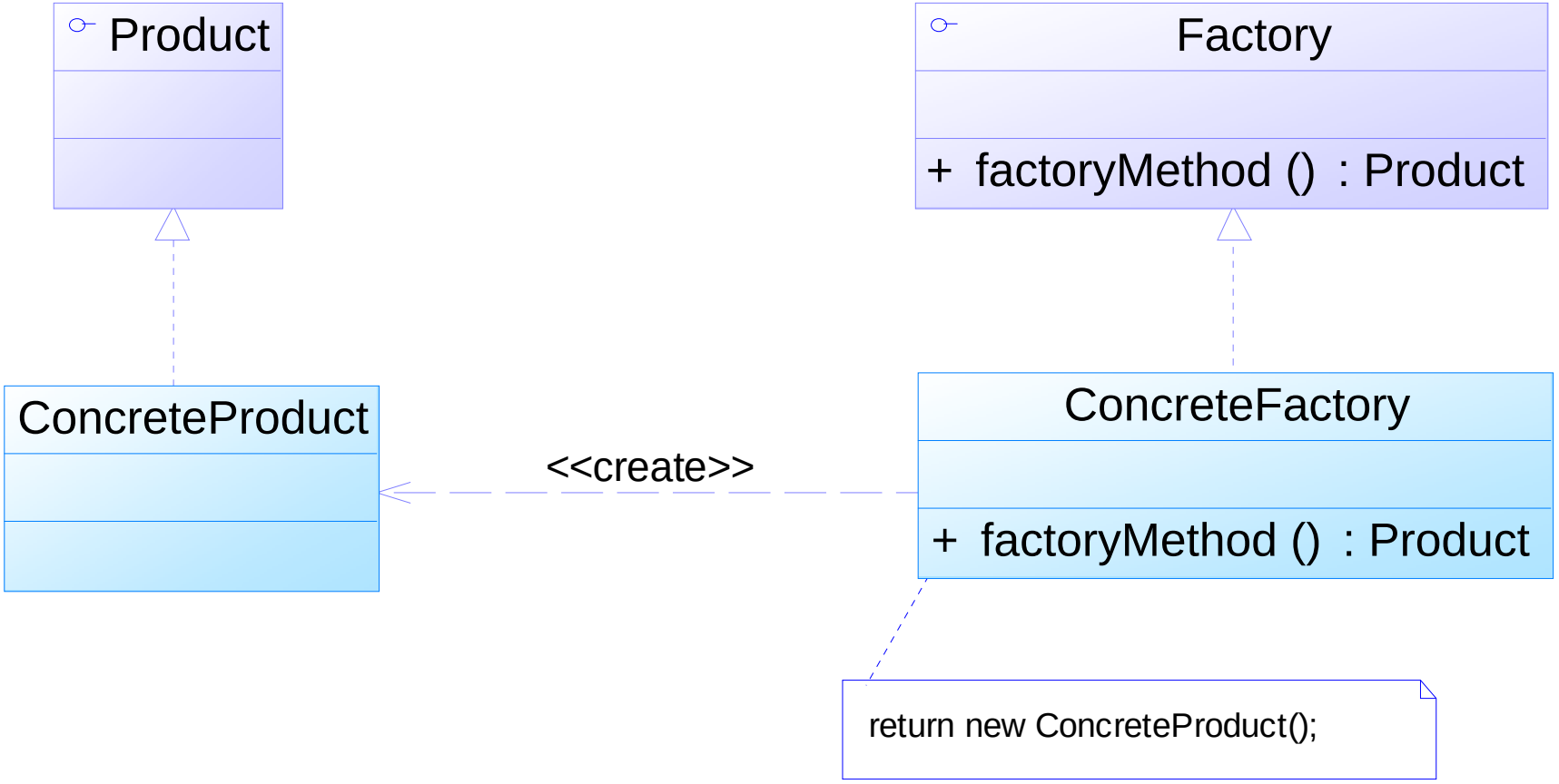
工厂方法模式

工厂方法模式的模式定义

- 工厂方法模式(Factory Method Pattern)简称工厂模式，也叫虚拟构造器(Virtual Constructor)模式或者多态工厂(Polymorphic Factory)模式，它属于类创建型模式。
- 在工厂方法模式中，工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生成具体的产品对象，这样做的目的是将产品类的实例化操作延迟到工厂子类中完成，即通过工厂子类来确定究竟应该实例化哪一个具体产品类。

工厂方法模式的模式结构与分析

- **工厂方法模式的模式结构（**模式包含如下角色）：
  - Product：抽象产品 ， ConcreteProduct：具体产品
  - Factory：抽象工厂， ConcreteFactory：具体工厂



工厂方法模式的模式分析

- 工厂方法模式是简单工厂模式的进一步抽象和推广
- 工厂方法模式保持了简单工厂模式的优点，并克服了它的缺点
- 核心的工厂类不再负责所有产品的创建，而是将具体创建工作交给其子类去完成
- 可以允许系统在不修改工厂角色的情况下引进新产品
- 增加具体产品-->增加具体工厂，符合“开闭原则”

工厂方法模式的模式分析

- Java反射(Java Reflection)机制的应用

```
//创建一个字符串类型的对象
Class c = Class.forName("String");
Object obj = c.newInstance();
return obj;
```

工厂方法模式的模式分析

- 配置文件的应用

```
<?xml version="1.0"?>
<config>
        <className>NameOfFactory</className>
</config>
```

工厂方法模式的模式分析

- 工具类XMLUtil代码片段

```
//创建DOM文档对象
DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = dFactory.newDocumentBuilder();
Document doc;
doc = builder.parse(new File("config.xml"));

//获取包含类名的文本节点
NodeList nl = doc.getElementsByTagName("className");
Node classNode=nl.item(0).getFirstChild();
String cName=classNode.getNodeValue();

//通过类名生成实例对象并将其返回
Class c=Class.forName(cName);
Object obj=c.newInstance();
return obj;
```

工厂方法模式的模式实例

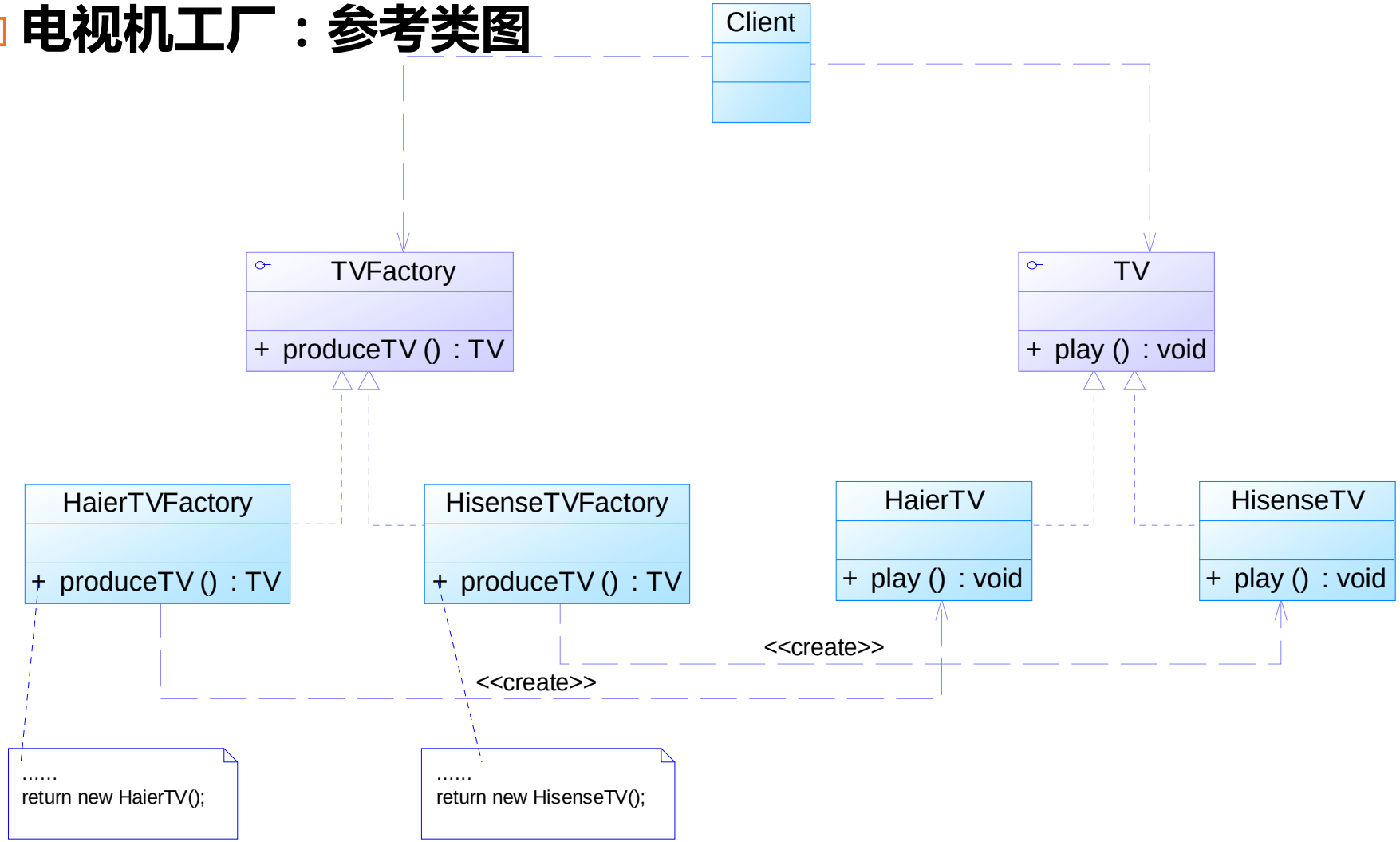
电视机工厂：实例说明

- 将原有的电视机工厂进行分割，为每种品牌的电视机提供一个子工厂，海尔工厂专门负责生产海尔电视机，海信工厂专门负责生产海信电视机，如果需要生产TCL电视机或创维电视机，只需要对应增加一个新的TCL工厂或创维工厂即可，原有的工厂无须做任何修改，使得整个系统具有更加的灵活性和可扩展性。

工厂方法模式的模式实例与解析

工厂方法模式的模式实例

电视机工厂：参考类图



工厂方法模式的模式实例与解析

- **工厂方法模式的模式实例**
  - **电视机工厂：参考代码**
- DesignPatterns之factorymethod包

TV.java

```
1  package factorymethod;
2
3  public interface TV
4  {
5      public void play();
6  }
```

TVFactory.java

```
1  package factorymethod;
2
3  public interface TVFactory
4  {
5      public TV produceTV();
6  }
```

HaierTV.java

```java
package factorymethod;

public class HaierTV implements TV
{
    public void play()
    {
        System.out.println("海尔电视机播放中.....");
    }
}
```

HisenseTV.java

```java
package factorymethod;

public class HisenseTV implements TV
{
    public void play()
    {
        System.out.println("海信电视机播放中.....");
    }
}
```

HaierTVFactory.java

```java
package factorymethod;

public class HaierTVFactory implements TVFactory
{
    public TV produceTV()
    {
        System.out.println("海尔电视机工厂生产海尔电视机。");
        return new HaierTV();
    }
}
```

HisenseTVFactory.java

```java
package factorymethod;

public class HisenseTVFactory implements TVFactory
{
    public TV produceTV()
    {
        System.out.println("海信电视机工厂生产海信电视机。");
        return new HisenseTV();
    }
}
```

XMLUtil.java

```java
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import java.io.*;
public class XMLUtil
{
//该方法用于从XML配置文件中提取具体类类名，并返回一个实例对象
    public static Object getBean()
    {
        try
        {//创建文档对象
            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = dFactory.newDocumentBuilder();
            Document doc;
            doc = builder.parse(new File("FactoryMethodconfig.xml"));
            //获取包含类名的文本节点
            NodeList nl = doc.getElementsByTagName("className");
            Node classNode=nl.item(0).getFirstChild();
            String cName=classNode.getNodeValue();
            //通过类名生成实例对象并将其返回
            Class c=Class.forName(cName);
            Object obj=c.newInstance();
            return obj;
           }
```

XMLUtil.java

```java
            catch(Exception e)
            {
                e.printStackTrace();
                return null;
            }
        }
}
```

FactoryMethodconfig.xml

```xml
<?xml version="1.0"?>
<config>
    <className>factorymethod.HisenseTVFactory</className>
</config>
```

Client.java

```java
public class Client
{
    public static void main(String args[])
    {
        try
        {
            TV tv;
            TVFactory factory;
            factory=(TVFactory)XMLUtil.getBean();
            tv=factory.produceTV();
            tv.play();
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

工厂方法模式的模式效果与应用

- **工厂方法模式优点：**
  - 工厂方法用来创建客户所需要的产品，同时还向客户隐藏了哪种具体产品类将被实例化这一细节
  - 能够让工厂自主确定创建何种产品对象，而如何创建这个对象的细节则完全封装在具体工厂内部
  - 在系统中加入新产品时，完全符合开闭原则

工厂方法模式的模式效果与应用

- **工厂方法模式缺点：**
  - 系统中类的个数将成对增加，在一定程度上增加了系统的复杂度，会给系统带来一些额外的开销
  - 增加了系统的抽象性和理解难度

工厂方法模式的模式效果与应用

- **在以下情况下可以使用工厂方法模式：**
  - 客户端不知道它所需要的对象的类（客户端不需要知道具体产品类的类名，只需要知道所对应的工厂即可，具体产品对象由具体工厂类创建）
  - 抽象工厂类通过其子类来指定创建哪个对象

谢谢