



深圳大学  
Shenzhen University

# 第17-1讲

## 模板方法模式

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

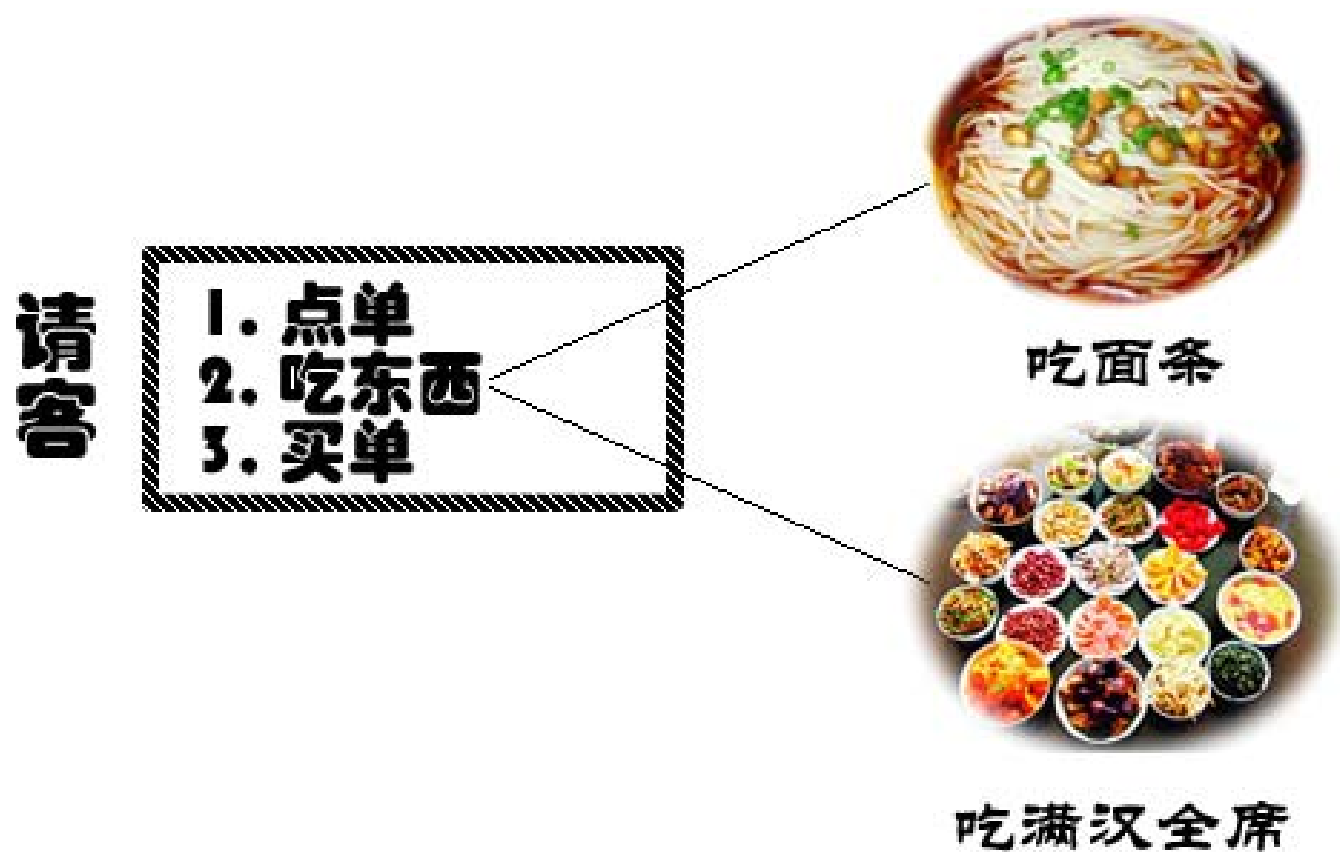


# 主要内容

- ◆ 模板方法模式动机与定义
- ◆ 模板方法模式结构与分析
- ◆ 模板方法模式实例与解析
- ◆ 模板方法模式效果与应用

# 模板方法模式动机

## ■ 请客吃饭示意图





## 模板方法模式动机

- 请客吃饭：(1) 点单 → (2) 吃东西 → (3) 买单
- 软件开发：某个方法的实现需要多个步骤（类似“请客”），其中有些步骤是固定的（类似“点单”和“买单”），而有些步骤并不固定，存在可变性（类似“吃东西”）
- 模板方法模式：
  - 基本方法（“点单”、“吃东西”和“买单”）
  - 模板方法（“请客”）

## 模板方法模式定义

### ■ 类行为型模式

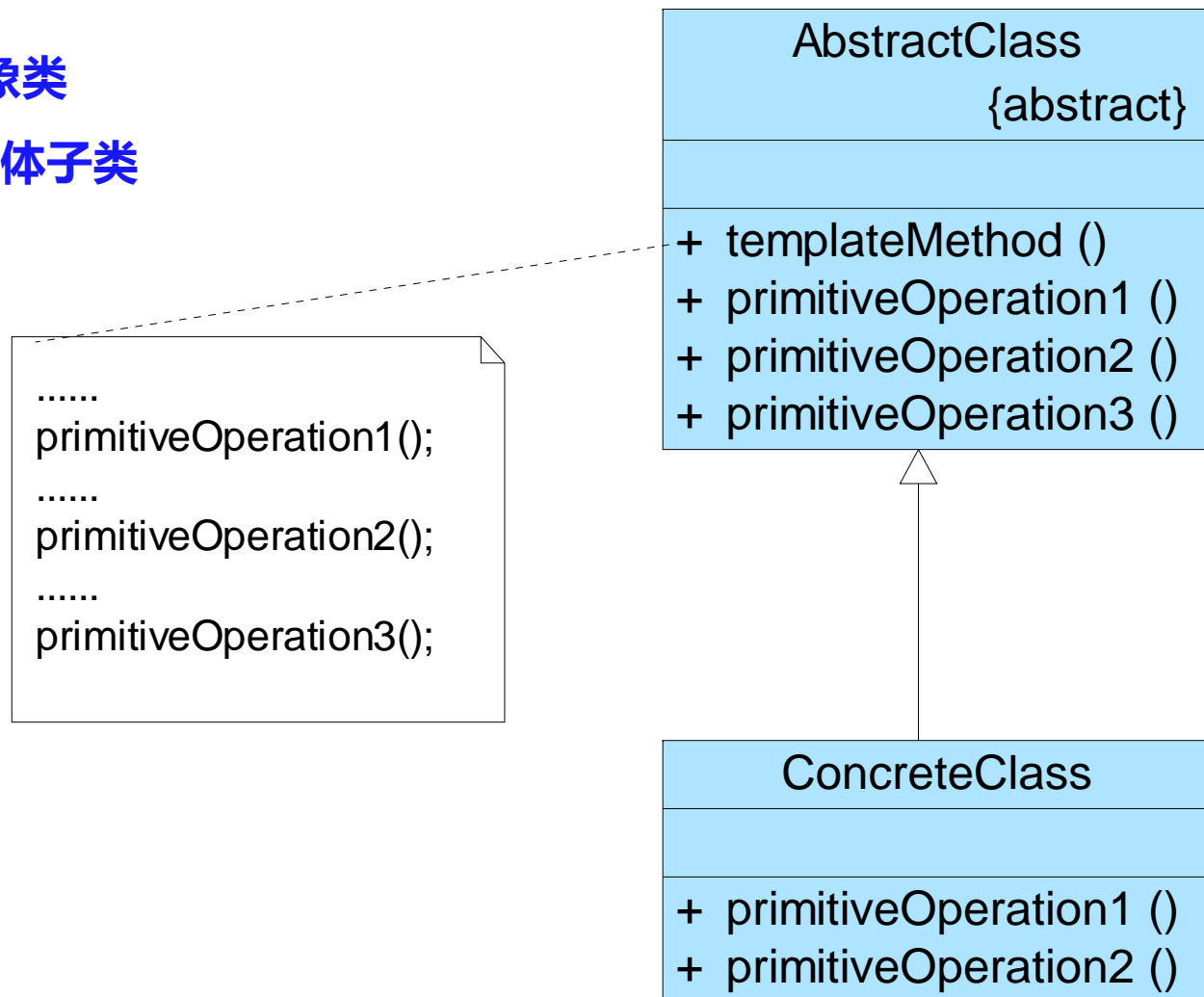
**模板方法模式：**定义一个操作中算法的框架，而将一些步骤延迟到子类中。模板方法模式使得子类不改变一个算法的结构即可重定义该算法的某些特定步骤。

**Template Method Pattern:** Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

- 是一种基于继承的代码复用技术
- 将一些复杂流程的实现步骤封装在一系列基本方法中
- 在抽象父类中提供一个称之为模板方法的方法来定义这些基本方法的执行次序，而通过其子类来覆盖某些步骤，从而使得相同的算法框架可以有不同的执行结果

## 模板方法模式结构

- 模板方法模式
- 包含如下角色：
  - **AbstractClass**: 抽象类
  - **ConcreteClass**: 具体子类





## 模板方法模式分析

- 模板方法 (Template Method)
- 基本方法 (Primitive Method)
  - 抽象方法(Abstract Method)
  - 具体方法(Concrete Method)
  - 钩子方法(Hook Method)

# 模板方法模式分析

## ■ 钩子方法

□ (1) “挂钩” 方法：isXXX()或hasXXX()，返回类型为boolean类型

□ (2) 空方法

```
.....  
//模板方法  
public void template() {  
    open();  
    display();  
    //通过钩子方法来确定某一步骤是否执行  
    if(isPrint()) {  
        print();  
    }  
}  
//钩子方法  
public boolean isPrint() {  
    return true;  
}  
.....
```



# 模板方法模式分析

## ■ 典型抽象类示例代码：

```
public abstract class AbstractClass {  
    //模板方法  
    public void templateMethod() {  
        primitiveOperation1();  
        primitiveOperation2();  
        primitiveOperation3();  
    }  
  
    //基本方法—具体方法  
    public void primitiveOperation1() {  
        //实现代码  
    }  
  
    //基本方法—抽象方法  
    public abstract void primitiveOperation2();  
  
    //基本方法—钩子方法  
    public void primitiveOperation3()  
    { }  
}
```



## 模板方法模式分析

- 典型子类示例代码：

```
public class ConcreteClass extends AbstractClass {  
    public void primitiveOperation2() {  
        //实现代码  
    }  
  
    public void primitiveOperation3() {  
        //实现代码  
    }  
}
```



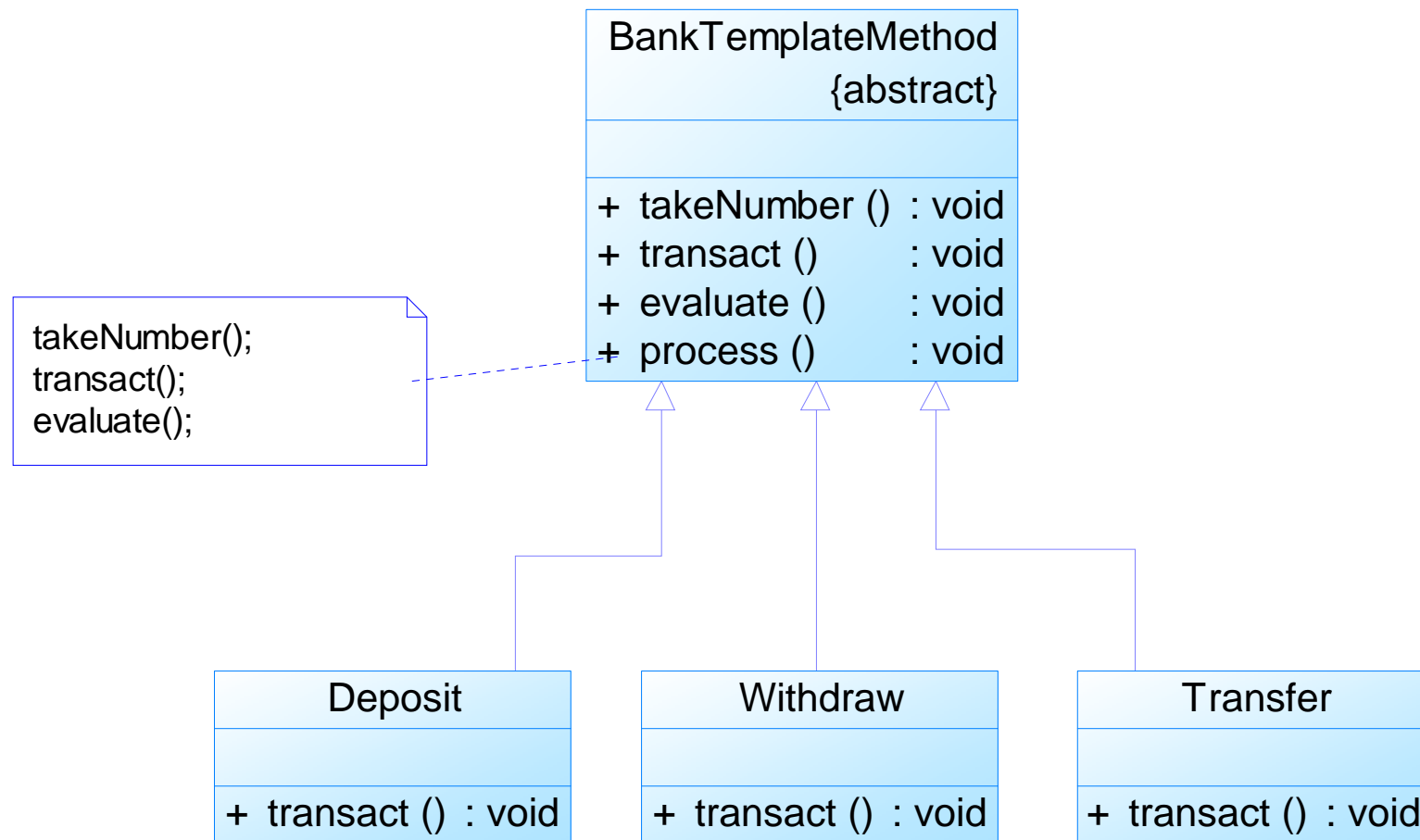
## 模板方法模式实例

### ■ 银行业务办理流程：实例说明

- 在银行办理业务时，一般都包含几个基本步骤，首先需要取号排队，然后办理具体业务，最后需要对银行工作人员进行评分。无论具体业务是取款、存款还是转账，其基本流程都一样。现使用模板方法模式模拟银行业务办理流程。

## 模板方法模式实例与解析

### ■ 银行业务办理流程：参考类图



## 模板方法模式实例

- 银行业务办理流程：参考代码
- DesignPatterns之templatemethod包

```
BankTemplateMethod.java ✕  
3 public abstract class BankTemplateMethod  
4 {  
5     public void takeNumber()  
6     {  
7         System.out.println("取号排队。");  
8     }  
9  
10    public abstract void transact();  
11  
12    public void evaluate()  
13    {  
14        System.out.println("反馈评分。");  
15    }  
16  
17    public void process()  
18    {  
19        this.takeNumber();  
20        this.transact();  
21        this.evaluate();  
22    }  
23 }
```

Deposit.java ✕

```
3 public class Deposit extends BankTemplateMethod
4 {
5     public void transact()
6     {
7         System.out.println("存款");
8     }
9 }
```

Transfer.java ✕

```
3 public class Transfer extends BankTemplateMethod
4 {
5     public void transact()
6     {
7         System.out.println("转账");
8     }
9 }
```

Withdraw.java ✕

```
3 public class Withdraw extends BankTemplateMethod
4 {
5     public void transact()
6     {
7         System.out.println("取款");
8     }
9 }
```

```
7 public class XMLUtil
8 {
9     //该方法用于从XML配置文件中提取具体类名，并返回一个实例对象
10    public static Object getBean()
11    {
12        try
13        {
14            //创建文档对象
15            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
16            DocumentBuilder builder = dFactory.newDocumentBuilder();
17            Document doc;
18            doc = builder.parse(new File("Templatemethodconfig.xml"));
19            //获取包含类名的文本节点
20            NodeList nl = doc.getElementsByTagName("className");
21            Node classNode=nl.item(0).getFirstChild();
22            String cName=classNode.getNodeValue();
23            Class c=Class.forName(cName); //通过类名生成实例对象并将其返回
24            Object obj=c.newInstance();
25            return obj;
26        }
27        catch(Exception e)
28        {
29            e.printStackTrace();
30            return null;
31        }
32    }
33 }
```

Templatemethodconfig.xml

```
1 <?xml version="1.0"?>
2 <config>
3     <className>templatemethod.Transfer</className>
4 </config>
```

Client.java

```
1 package templatemethod;
2
3 public class Client
4 {
5     public static void main(String a[])
6     {
7         BankTemplateMethod bank;
8         bank=(BankTemplateMethod)XMLUtil.getBean();
9         bank.process();
10        System.out.println("-----");
11    }
12 }
```





## 模板方法模式效果与应用

### ■ 模板方法模式优点：

- 在父类中形式化地定义一个算法，而由它的子类来实现细节的处理，在子类实现详细的处理算法时并不会改变算法中步骤的执行次序
- 提取了类库中的公共行为，将公共行为放在父类中，而通过其子类来实现不同的行为
- 可实现一种反向控制结构，通过子类覆盖父类的钩子方法来决定某一特定步骤是否需要执行
- 更换和增加新的子类很方便，符合单一职责原则和开闭原则



## 模板方法模式效果与应用

### ■ 模板方法模式缺点：

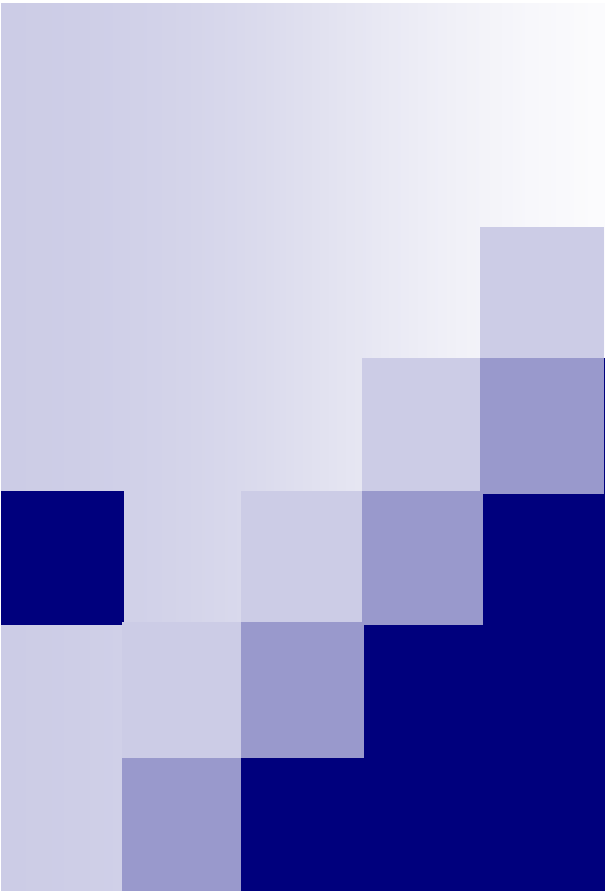
- 需要为每一个基本方法的不同实现提供一个子类，如果父类中可变的基本方法太多，将会导致类的个数增加，系统会更加庞大，设计也更加抽象



## 模板方法模式效果与应用

### ■ 在以下情况下可以使用模板方法模式：

- 一次性实现一个算法的不变部分，并将可变的行为留给子类来实现
- 各子类中公共的行为应被提取出来，并集中到一个公共父类中，以避免代码重复
- 需要通过子类来决定父类算法中某个步骤是否执行，实现子类对父类的反向控制



谢谢