# 模板方法模式简介

模板方法模式是结构最简单的行为型设计模式，它是一种类行为模式，在某结构中只存在父类与子类之间的继承关系。通过使用模板方法模式可以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。它提供了具体的模板方法来定义算法结构，而具体步骤的实现可以在其子类中完成。

## 概述

1. 在现实生活中很多事情都包含几个实现步骤，例如请客吃饭，无论吃什么，一般都包含点单、吃东西、买单等几个步骤，通常情况下这几个步骤的次序是点单-->吃东西-->买单。在这3个步骤中，点单和买单大同小异，最大的区别在于第二步----吃什么？吃面条和吃满汉全席可大不相同。在软件开发中有时也会遇到类似的情况，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“吃什么”）。为了提高代码的复用性和系统的灵活性，可以使用一种称为模板方法模式的设计模式来对这类情况进行设计，在模板方法中将实现功能的每一个步骤所对应的方法称为基本方法（例如“点单”，“吃东西”，“买单”），而将调用这些基本方法同时定义基本方法执行次序的方法称为模板方法（例如“请客”）。在模板方法模式中可以将相同的代码放在父类中，例如将模板方法“请客”以及基本方法“点单”和“买单”的实现放在父类中，而对于基本方法“吃东西”在父类中只做一个声明，将其具体实现放在不同的子类中，在一个子类中提供“吃面条”的实现，而另一个子类提供“吃满汉全席”的实现。通过使用模板方法模式，一方面提高了代码的复用性，另一方面还可以利用面向对象的多态性，在运行时一种具体子类，实现完整的“请客”方法，提高系统的灵活性和可扩展性。
2. 模板方法模式（Template Method Pattern）的定义如下：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可以不改变一个算法的结构即可重定义该方法的某些特定步骤。
3. 模板方法模式是一种基于继承的代码复用技术，它是一种类行为型模式。
4. 模板方法模式是结构最简单的行为型设计模式，在其结构中只存在父类与子类之间的继承关系。通过使用模板方法模式可以将一些复杂流程的实现步骤封装在系列基本方法中，在抽象父类中提供一个称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。模板方法模式提供一个模板方法来定义算法框架，而某些具体步骤的熟悉可以在其子类中完成。

## 结构

模板方法模式的结构比较简单，其核心是抽象类和其中的模板方法的设计，其结构如下

图所示：



由图可知，模板方法模式包含以下角色：

1. AbstractClass（抽象类）：在抽象类中定义了一系列基本操作（Primitive Operations），这些基本操作可以是具体的，也可以是抽象的，每一个基本操作对应算法的一个步骤，在其子类中可以重定义或实现这些步骤。同时在抽象类中定义一个模板方法（Template Method），用于定义一个算法的框架，模板方法不仅可以调用在抽象类中实现的基本方法，也可以调用在抽象类的子类中实现的基本方法，还可以调用其他对象中的方法。
2. ConcreteClass（具体子类）：它是抽象类的子类，用于实现在父类中声明的抽象基本操作以完成子类特定算法的步骤，也可以覆盖在父类中已经实现的具体基本操作。

# 实现

## 实现原理

在实现模板方法模式时，开发抽象类的软件设计师和开发具体子类的软件设计师之间可以进行协作。一个设计师负责给出一个算法的轮廓和框架，另一些设计师则负责给出这个方法的各个逻辑步骤。实现这些具体逻辑步骤的方法即为基本方法，而将这些基本方法汇总起来的方法即为模板方法，模板方法模式的名字也因此而来。

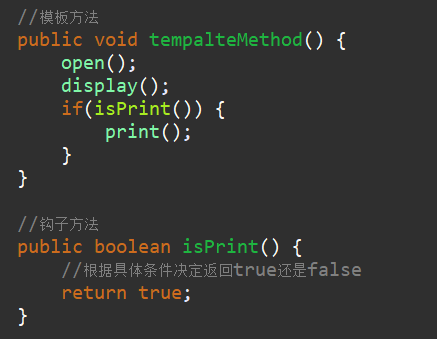
1. **模板方法**

一个模板方法是定义在抽象类中的把基本操作组合在一起形成一个总算法或一个总行为的方法。这个模板方法定义在抽象类中，并由子类不加修改地完全继承下来（在Java语言中可以将模板方法定义为final类型）。模板方法是一个具体方法，它给出一个顶层逻辑框架，而逻辑的组成步骤在抽象类中可以是具体方法，也可以是抽象方法。由于模板方法是具体方法，因此模板方法模式中抽象层只能是抽象类，而不是接口。

1. **基本方法**

基本方法是实现算法各个步骤的方法，是模板方法的组成部分。基本方法又可以分为3种，即抽象方法（Abstract Method）、具体方法（Concrete Method）和钩子方法（Hook Method）。

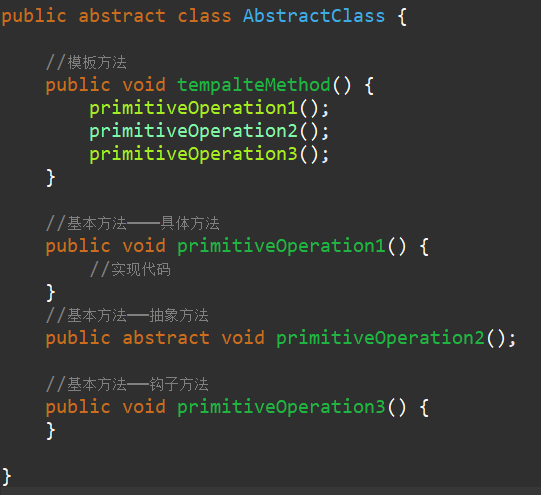
1. 抽象方法：一个抽象方法由抽象类声明，由其具体子类实现。在Java语言中抽象方法使用Abstract关键字声明。
2. 具体方法：一个具体方法由一个抽象类或者具体类声明并实现，其子类可以进行覆盖也可以直接继承。
3. 钩子方法：一个钩子方法由一个抽象类或具体类声明并实现，而其子类可能会加以扩展。通常在父类中给出的实现是一个空实现，并以空实现作为方法的默认实现。当然，钩子方法也可以提供一个非空的默认实现。在模板方法模式中的钩子方法有两类，第一类钩子方法可以与一些具体步骤“挂钩”，以实现在不同条件下执行模板方法中的不同步骤，这类钩子方法的返回类型通常是boolean类型，方法名一般为isXXX( )，用于对某个条件进行判断，如果条件满足则主席某一步骤，否则将不执行，如下代码片段所示：



在这段代码中，isPrint( )方法即为钩子方法，它可以决定print( )方法是否执行。一般情况下，钩子方法的返回值为true，如果不希望某方法执行，可以在其子类中覆盖钩子方法，将其返回值改为false即可，这种类型的钩子方法可以控制方法的执行，对一个算法进行约束。

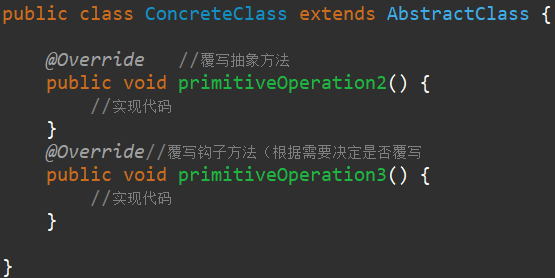
还有一类的钩子方法就是实现体为空的具体方法，子类可以根据需要覆盖或继承这些钩子方法。与抽象方法相比，这类钩子方法的好处在于子类如果没有覆盖父类中定义的钩子方法，编译可以正常通过，但是如果没有覆盖父类中声明的抽象方法，编译将会报错。

1. **原理**
2. 在模板方法模式中，抽象类的典型代码如下：



在抽象类中，模板方法templateMethod( )定义了算法的框架，在模板方法中调用基本方法以实现完整的算法，每一个基本算法（如primitiveOperation1( )、primitiveOperation2( )等）均实现了算法的一部分，对于所有子类都相同的基本方法可以在父类中提供具体实现，例如primitiveOperation1( )，否则在父类中将其声明为抽象方法或钩子方法，由不同的子类提供不用的实现，例如primitiveOperation2( )、primitiveOperation3( )。

1. 用户在抽象类的子类中提供抽象步骤的实现，也可覆盖父类中已经实现的具体方法。具体子类的典型代码如下：



在模板方法模式中，由于面向对象的多态性，子类对象在运行时将覆盖父类对象，子类中定义的方法也将覆盖父类中定义的方法，因此程序在运行时具体子类的基本方法将覆盖父类中定义的基本方法，子类的钩方法也将覆盖父类中的钩子方法，从而可以通过在子类中实现的钩子方法对父类方法进行约束，实现子类对父类行为的反向控制。

## 实例

实例说明：某软件公司要为某银行的业务吃撑系统开发一个利息计算模块，利息计算流程如下：

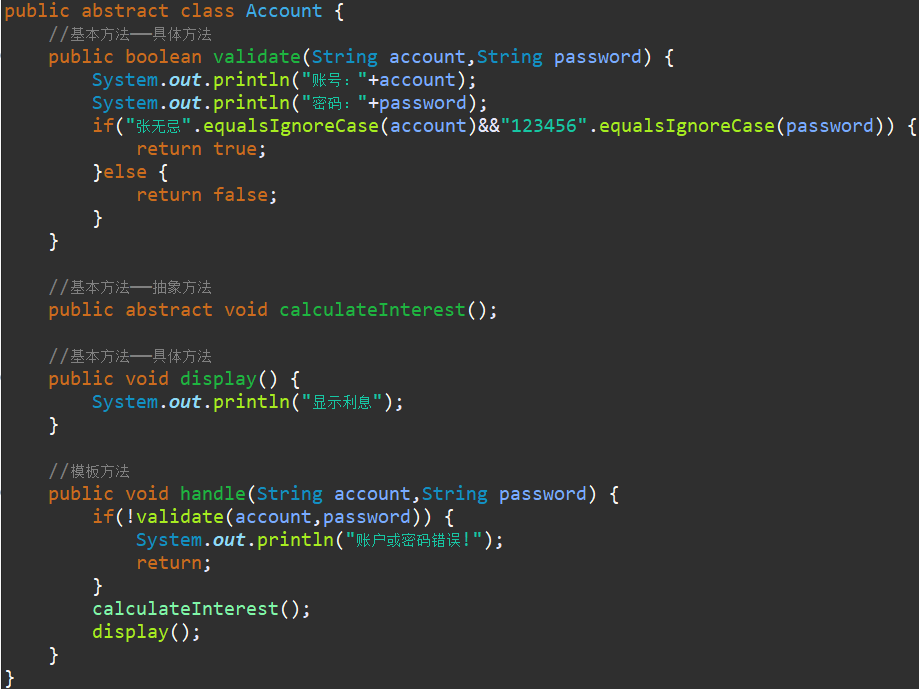
1. 系统根据账号和密码验证用户信息，如果用户信息错误，系统显示出错提示。
2. 如果用户星星正确，则根据用户类型的不同使用不同的利息计算公司计算利息（如活期账户和定期账户具有不同的利息计算公式）。
3. 系统显示利息。

现使用模板方法模式设计该利息计算模块，通过分析，本实例的类结构图如下所示：

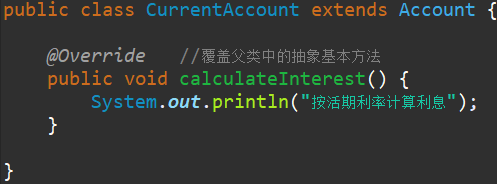


在图中，Account充当抽象类角色，CurrentAccount和SavingAccount充当具体子类角色。

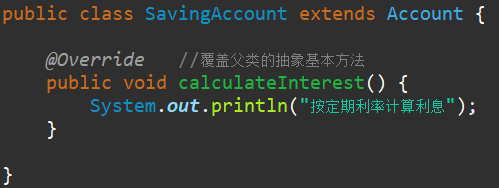
1. Account：账户类，充当抽象类



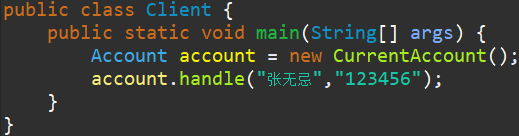
1. CurrentAccount：活期账户类，充当具体子类



1. SavingAccount：定期账户类，充当具体子类



1. Client：客户端测试类



运行结果如下：



如果需要增加新的具体子类（新的账户类型），原有代码无须修改，完全符合开闭原则。

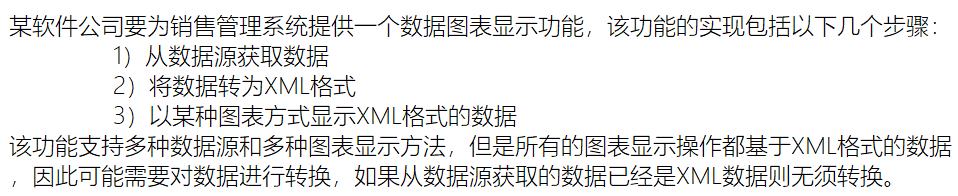
## 钩子方法的使用

在模板方法模式中，父类提供一个定义算法框架的模板方法，还提供了一系列抽象方法、具体方法和钩子方法，其中钩子方法的引入使得子类可以控制父类的行为。最简单的钩子方法就是空方法，代码如下：



当然，用户也可以在钩子方法中定义一个默认的实现，如果子类不覆写钩子方法，则执行父类的默认实现代码。另一种钩子可以对其他方法进行约束，这种钩子方法通常返回

一个boolean类型的值，即返回true或false，用来判断是否执行某一个基本方法。下面通过一个实例来说明这种钩子方法的使用。



由于该数据图表显示功能的3个步骤次序都是固定的，且存在公共代码（例如数据格式转换代码），满足模板方法模式的适用条件，可以使用模板方法模式对其进行设计。因为数据格式的不同，XML数据可以直接显示，而其他格式的数据需要记性转换，因此第（2）步“将数据转换为XML格式”的执行存在不确定性，为了解决这个问题，可以定义一个钩子方法isNotXMLData( )对数据转换方法进行控制。通过分析，该图标显示功能的基本结构如下所示：



1. 用户可以将公共方法和框架代码放在抽象父类中，其代码如下；



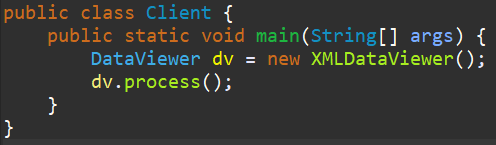
在上面代码中引入了一个钩子方法isNotXMLData( )，其返回值为boolean类型，在模板方法中通过它对数据转换方法convertData( )进行约束。该钩子方法的默认返回值为true，在子类中可以根据实际情况覆盖该方法，

1. 其中用于显示XML格式数据的具体子类XMLDataViewer的代码如下：

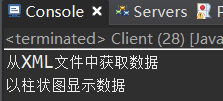


在具体子类XMLDataViewer中覆盖了钩子方法isNotXMLData( )，返回false，表示该数据已为XML格式，无须执行数据转换方法convertData( )。

1. 客户端测试类代码如下：



运行结果如下：



# 优缺点及适用环境

模板方法模式是基于继承的代码复用技术，它体现了面向对象的诸多重要思想，是一种较为频繁的设计模式。模板方法模式广泛应用于框架设计（例如Spring，Junit等）中，以确保通过父类来控制处理流程的逻辑顺序（例如框架的初始化、测试流程的设置等。

## 优点

模板方法模式的优点主要如下：

1. 在父类中形式地定义一个算法，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。
2. 模板方法模式是一种代码复用技术，在类库设计中尤为重要，它提取了类库中的公共行为，将公共行为放在父类中，而通过其子类实现不同的行为，它鼓励用户恰当的使用继承来实现代码复用。
3. 模板方法模式可以实现一种反向控制，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。
4. 在模板方法模式中可以通过子类来覆盖父类的基本方法，不同的子类可以提供基本方法的不同实现，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。

## 缺点

模板方法模式的缺点主要如下：在模板方法中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，系统更加庞大，设计也更加抽象，此时可结合桥接模式进行设计。

## 适用环境

在以下情况下可以考虑适用模板方法模式：

1. 对一些复杂的算法进行分割，将其算法中固定不变的部分设计为模板方法和父类的具体方法，而一些可以改变的细节由其子类来实现。即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。
2. 各子类中的公共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。
3. 需要通过子类来决定父类算法中的某个步骤是否执行，实现子类对象父类的反向控制。