**模式的秘密---模板方法模式**

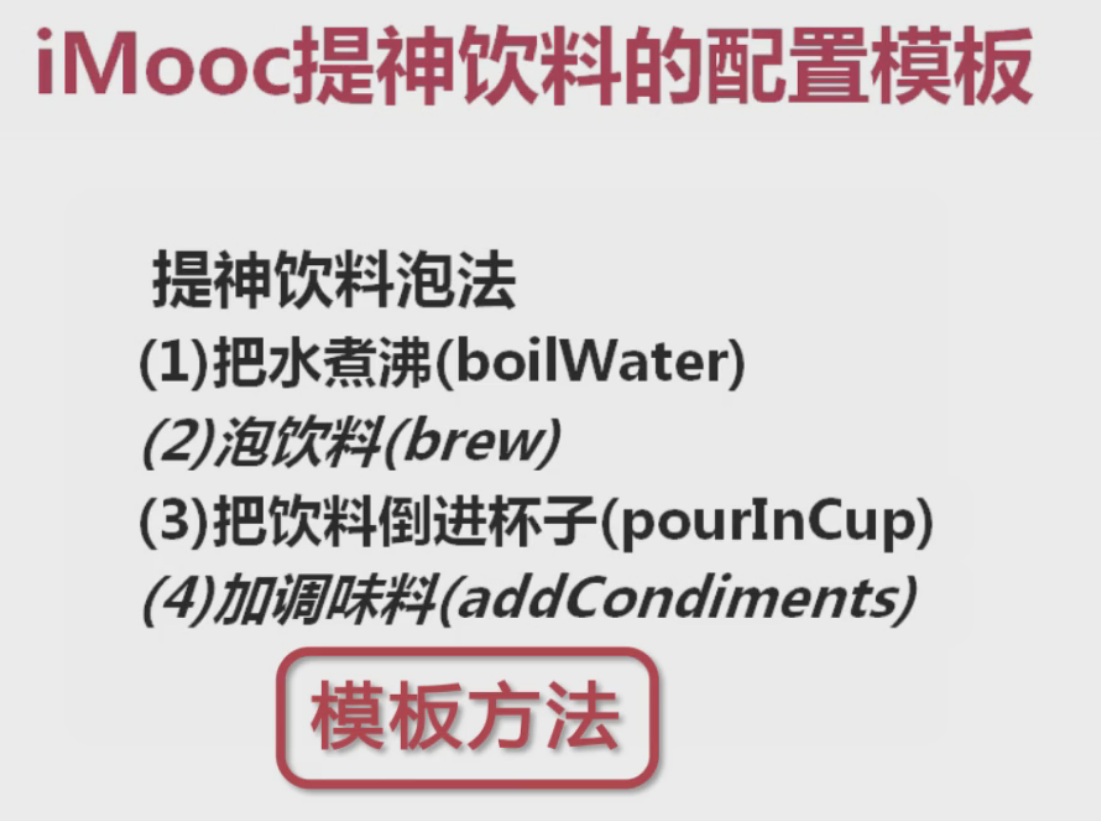




模板方法：在抽象基类定义final修饰的方法封装算法骨架，并实现其中的共性的步骤；某些没有共性的步骤（可定义为protected或默认的abstract方法）延迟到子类实现。这样做可以在不改变算法骨架的同时，又能在后期实现算法骨架中的非共性步骤。







模板模式

定义了一个操作中的算法骨架,而将一些步骤延迟到子类中实现,

使得子类在不改变一个算法结构的同时,就重新定义该算法的特定步骤.

模板方法的基本实现

思想

1、 算法框架中分离出变与不变的部分

2、将变化的算法，延迟实现（交由具体的子类实现）

模板方法的基本实现

思想

1、一份算法框架，大家共同遵守

2、 算法框架中分离出变与不变的部分

3、将变化的算法，延迟实现（交由具体的子类实现）

基本实现

1、用一个抽象基类，一个public final方法定义好算法框架

2、不变的部分，用private方法加以实现。（基本方法）

3、变化的部分，用protected abstract加以定义（抽象方法）

使用

1、面向接口编程

2、传入实际的实现子类给接口变量

3、接口变量调用框架方法

模板方法（使用抽象父类定义框架）

抽象父类，为所有子类提供一个算法框架

步骤：

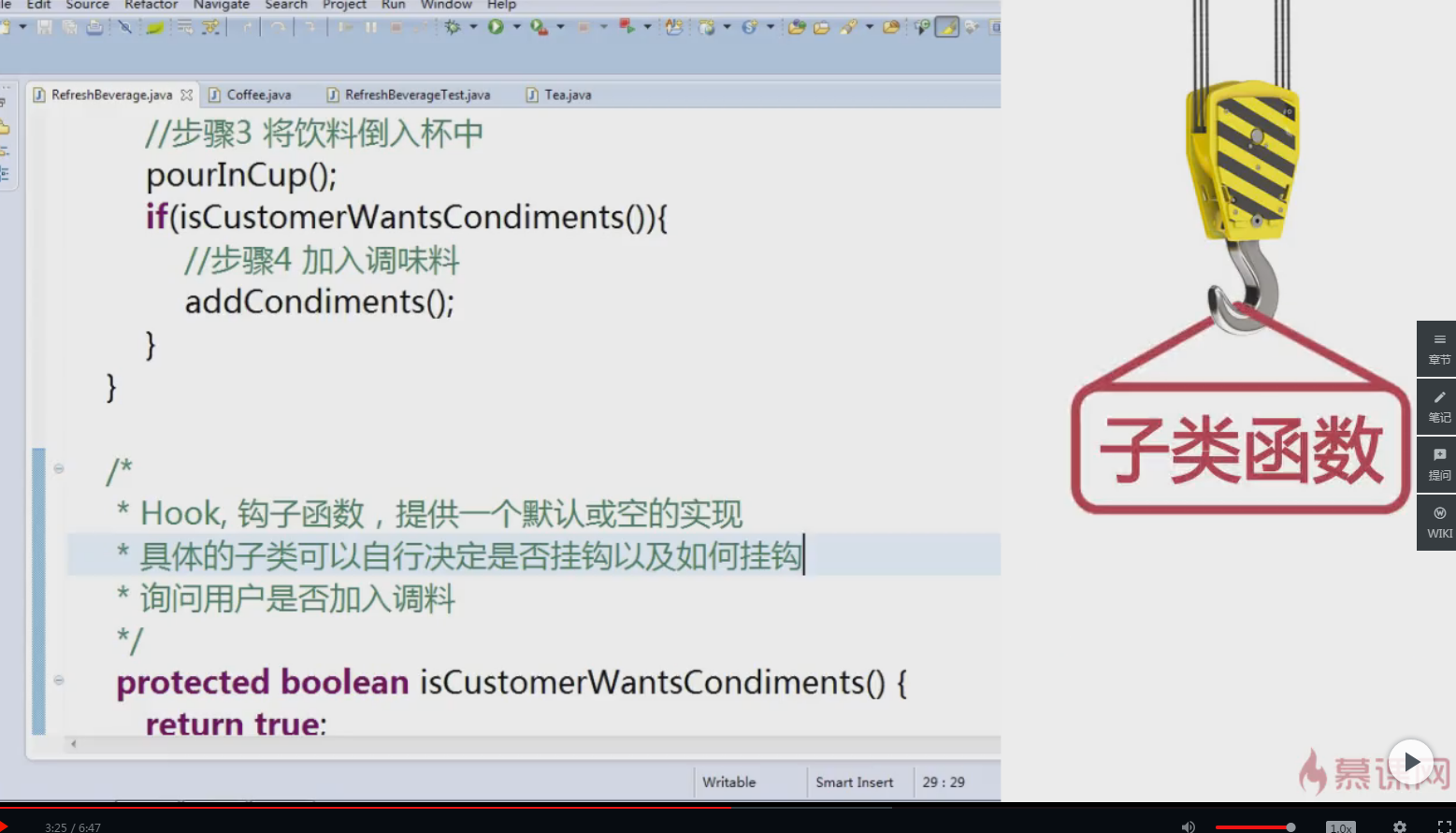
1、定义一个公共的抽象父类

2、定义一个公共 final 的方法（封装所有子类都要遵循的算法框架）

3、算法框架中定义的方法子类实现相同的使用 private 修饰该方法并且实现，子类实现方法不一样的使用 protected abstact 修饰该方法并且不实现

4、让子类重写父类未实现的方法，实现各自的方法

5、在调用的实例对象采用 父类 实例名 = new 子类，在直接调用遵循的框架方法



钩子函数：提供一个默认的或者空的实现。

具体的子类可以自行决定是否挂钩以及如何挂钩。

也就是提供一个方法，子类想更改就更改，不更改就走默认的。

采用钩子函数来让子类自行判断要不要挂钩，在父类的实现框架中添加if语句做判断，然后再添加一个钩子函数来返回是否需要挂钩，在子类中也要重写这个方法来决定这个子类是否挂钩

钩子函数的使用，使固定的父类模版显得更加灵活，因为并不是所有子类的过程都遵循着父类的规则。而存在的这些些许的差异，因为钩子函数的引入，使得子类的过程可以和父类的过程存在变通性

用钩子(Hook)函数实现子类对算法框架个性化的扩展

1、思想

框架通过提供一个个的钩子，使框架具备了更大的灵活性。不想执行算法框架中的某些个步骤，我们可以脱钩，如果想执行的话，我们可以挂钩。

2、实现

在抽象类中，提供protected钩子方法。这是个实现的或空的方法。这样子类就可以选择覆写-持钩，也可以选择不覆写-脱勾。

3、使用

提供一个isXXX类型的钩子方法。用该方法控制算法框架中

4、某个步骤是否执行

子类不覆写这个方法，就是脱钩，仍按框架逻辑执行，一旦覆写，就是挂钩，将改变框架算法方向，按子类逻辑执行。

模式的秘密---模板方法模式——总结

一、模板方法模式的实现要素：

准备一个抽象类，将部分逻辑以具体方法的形式实现，然后声明一些抽象方法交由子类实现剩余逻辑，用钩子方法给予子类更大的灵活性。最后将方法汇总构成一个不可改变的模板方法。

二、从类的角度看：

（1）抽象基类

1、基本方法。

2、抽象方法【只知道具体原则，而不知道实现细节，需要将其延迟到子类中实现的一些步骤】。

3、可选钩子（Hook，钩子函数，提供一个默认或空的实现。具体的子类可以自行决定是否挂钩以及如何挂钩）。

4、Template方法（final 使其不能被子类所覆写 模板方法模式要遵循的原则：子类可以替换掉父类中的可变逻辑，但不能改变整体逻辑结构））。

（2）具体子类

1、实现基类中的抽象方法。

2、覆盖钩子方法。

三、模板方法的优点：

（1）封装性好。

（2）复用性好。

（3）屏蔽细节。

（4）便于维护。

四、模板方法的缺点：

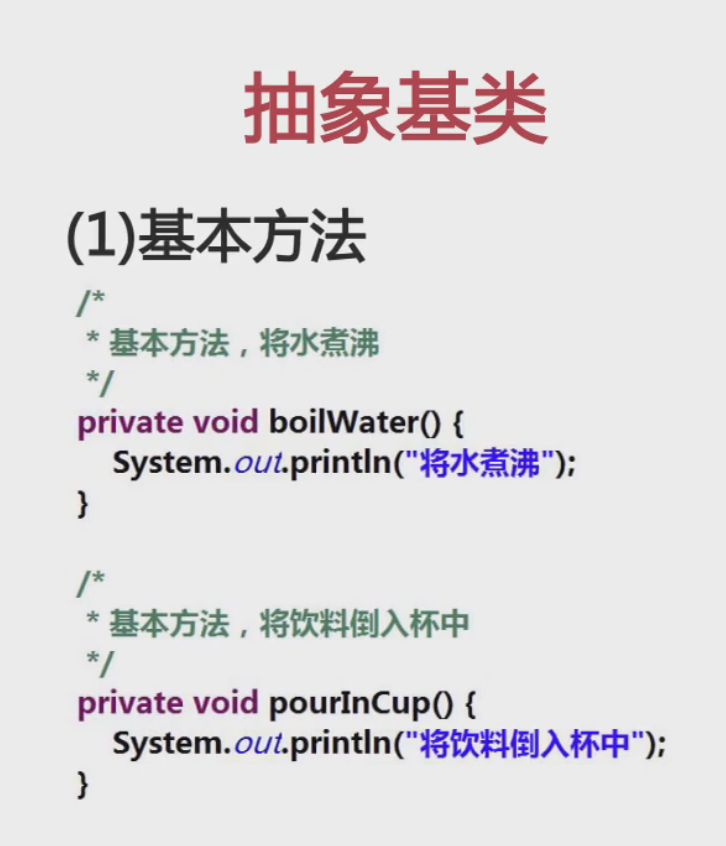
（1）继承限制（Java语言是单继承语言），单继承会使得更多情况不可用，新类引入困难。

五、模板方法模式的适用场景：

（1）算法或操作遵循相似的逻辑。

（2）重构时（把相同的代码抽取到父类中）。

（3）重要、复杂的算法，核心算法设计为模板方法。



模版方法模式的实现要素:准备一个抽象类,将部分逻辑以具体方法的形式实现,然后声明一些抽象方法交由子类实现剩余逻辑,用钩子方法给予子类更大的灵活性,最后将方法汇总构成一个不可改变的模版方法.

使用模版方法模式一般有两种类型的类, 分别是抽象基类与具体子类.

抽象基类中包括:

1.基本方法,即不同子类的共性,可以直接在抽象基类中定义.

2.抽象方法,代表一些只知道具体原则,不知道具体实现细节.需要延迟到子类实现的步骤.

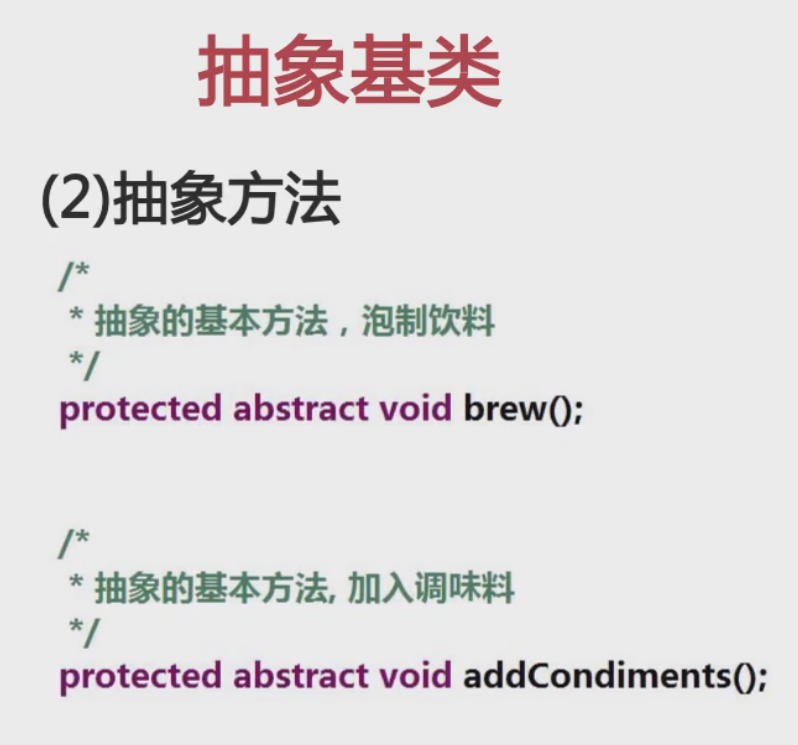
3.可选的构子函数,即在抽象基类中提供任意个默认或空的实现方法,以供给子类决定是否挂钩以及如何挂钩.

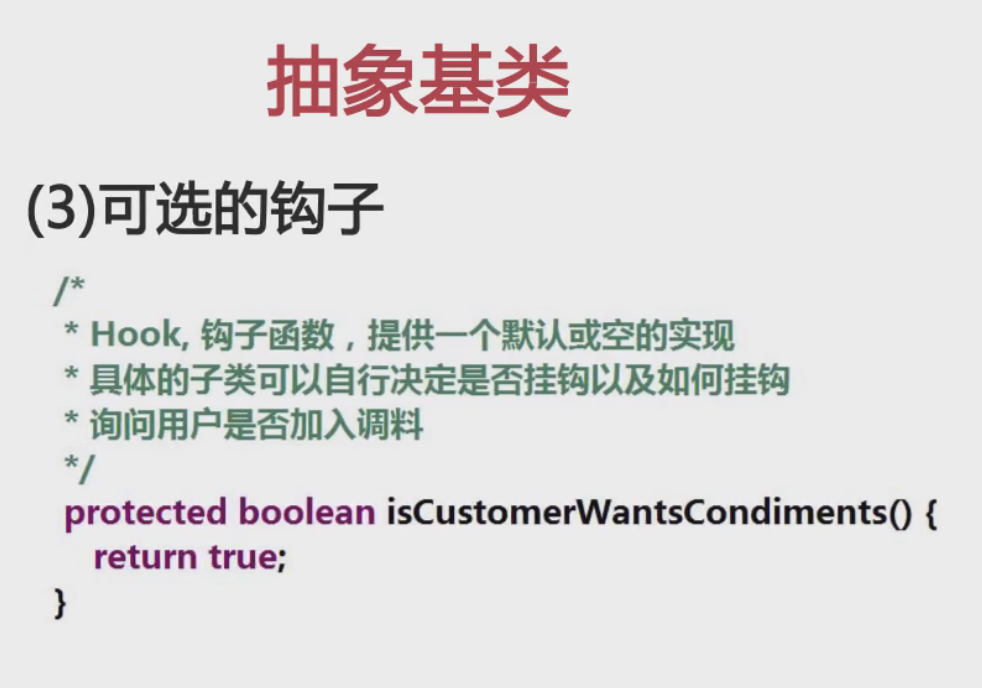
4.模版方法,用final关键字修饰.

具体子类中包括:

1.实现基类中的抽象方法.

2.可选的覆盖钩子方法.







原则：子类可以替换掉父类中的可变逻辑，但不能改变整体逻辑结构（好莱坞原则）





