模板设计模式定义:

定义一个操作中算法的骨架，而将一些步骤延迟到子类中，模板方法使得子类可以不改变算法的结构即可重定义该算法的某些特定步骤。

抽象父类: 实现了模板方法，定义了算法的骨架。

具体类: 实现抽象类中的抽象方法，即不同的对象的具体实现细节。

抽象父类:

|  |
| --- |
| public abstract class Beverage {  //注意这里的final,表示子类不 可以重写这个方法  public final void prepareBeverageTemplate(){  //1.烧开水  boilWater();  //2.放到杯中  putIntoCup();  //3.加入开水  addHotWater();  //4.加入调味应有剂,这个方法并不是所有的类都需要的  if(isCustomered()) {  addCondiments();  }  }  //1.  protected void boilWater() {  System.out.println("烧开水");  }  //2.  protected abstract void putIntoCup();  //3.  private void addHotWater(){  System.out.println("加水");  }  //4.  protected abstract void addCondiments();  //钩子函数  protected boolean isCustomered(){  return true;  }  } |

实现类1-Coffee:

|  |
| --- |
| public class Coffee extends Beverage{  @Override  protected void putIntoCup() {  System.out.println("把咖啡放到水杯中");  }  @Override  protected void addCondiments() {  System.out.println("加入糖和牛奶");  }  } |

实现类2-Tea:

|  |
| --- |
| package adaptor;  public class Tea extends Beverage{  @Override  protected void putIntoCup() {  System.out.println("把茶放入到水杯中");  }  @Override  protected void addCondiments() {  }  @Override  protected boolean isCustomered() {  return false;  }  } |

测试:

|  |
| --- |
| public class Test {  public static void main(String args[]){  Beverage coffee = new Coffee();  coffee.prepareBeverageTemplate();  System.out.println("===========");  Beverage tea = new Tea();  tea.prepareBeverageTemplate();  }  } |

不足: 每个不同的实现都需要定义一个子类，会导致类的个数增加，系统更加庞大。