设计模式思想在《面向对象程序设计》课程教学应用与探索

李　涛

黔南民族师范学院计算机与信息学院　贵州都匀　558000

摘　要：在《C++ 面向对象程序设计》课程教学中，通过引入策略模式和模板方法模式等设计模式的案例，帮助学生深入理解继承和多态性等面向对象特性在实际应用中的具体体现和重要性。在教学中逐步引导学生完成相关的类和功能设计，并对其设计代码进行分析、比较和优化。实践表明，引入设计模式思想进行教学不仅有助于提升学生的技术水平，还能培养学生的设计思维和解决问题的能力，对塑造学生的编程思维、编程习惯和编程水平具有重要作用。

关键字：设计模式；面向对象；策略模式；模板方法模式

引言

C++ 面向对象程序设计中，继承和多态性是核心概念，它们为程序提供了丰富的扩展性和灵活性。继承是指特殊类（派生类或子类）的对象拥有其一般类（基类或父类）的全部属性和行为。多态是指在一般类（基类或父类）中定义的属性或行为，被特殊类（派生类或子类）继承之后，可以具有不同的数据类型或表现出不同的行为。从《C++ 面向对象程序设计》课程教学效果来看，由于面向对象编程思想复杂化，对于初学者来说很难理解这些抽象的概念，导致学生无法对面向对象的相关特性深入理解。

设计模式（Design Patterns）是软件开发人员在面向对象编程（OOP）过程中，针对常见问题的最佳实践解决方案。这些模式是在大量实践中总结出来的，它们不仅解决了特定的问题，还提高了代码的可重用性、可维护性和可扩展性。尤为重要一点，设计模式是建立在面向对象基本特性之上的，如封装、继承、多态等。通过引入相关设计模式案例，可以更加深入地理解这些特性在实际应用中的具体体现和重要性。文献 [1] 提出启发式的教学，通过模板方法模式的案例应用于多态性与虚函数的教学过程中，促进学生深入理解面向对象的编程思想，提升学生面向对象的分析和设计能力；文献 [2] 提出基于观察者模式进行 CoAP 课程内容讲解可帮助学生触类旁通，提高学习效率和再创造能力。因此，设计模式对面向对象编程学习具有重要的帮助作用，它不仅有助于提升开发者的技术水平，还有助于培养设计思维和解决问题的能力。

1.　教学内容

在《C++ 面向对象程序设计》课程教学过程中，通过实例演示相关继承和多态性的基本概念和用法，帮助学生理解这些概念在面向对象编程中的重要作用。强调继承和多态性在代码复用、扩展性和可维护性方面的优势，并引导学生思考如何在实际项目中应用这些概念。

针对教学内容选择设计模式中部分思想进行辅助教学，将原先注重讲解语法的教学模式，转化为引入精心设计教学案例，通过案例逐步引导学生完成相关的类和功能设计。在完成设计的基础上，引导学生对其设计代码进行分析、比较和优化。在此过程中，通过教师分析与讲解，学生回答问题及讨论，让学生加深对继承、多态、虚函数等概念的理解。

在《C++ 面向对象程序设计》课程教学中，　　根据继承和多态这些章节内容，设计一些设计模式相联系的实践案例。如策略模式、工厂方法方式、模板方法模式，可以与“继承与多态”相关章节中的接口、抽象类等内容相联系。这些模式展示了如何通过继承和接口实现多态性，以及如何在不同情况下灵活切换算法或行为。

2.　策略模式在教学应用

策略模式（Strategy Pattern），它定义了一系列算法，并将每一种算法封装起来，使它们可以互相替换。策略模式让算法的变化独立于使用算法的客户 [3]。这种类型的设计模式属于行为型模式。

2.1 引入案例

假设有一个场景，大学开学了需要交付相应学费、住

宿费和书本费，学生根据不同的支付方式（如信用卡支付、支付宝支付、微信支付）来处理支付逻辑。学生提出支付请求后，并根据当前设置的支付方式来执行支付操作。同时根据需要动态地改变支付方式。

2.2 模式引入

设计一个利用策略模式来处理不同支付方式（如信用卡支付、支付宝支付、微信支付）的支付学费程序时，首先需要定义支付策略接口，然后为每个支付方式实现具体的策略类，并在一个上下文类中管理这些策略。设计类图如图 1 所示

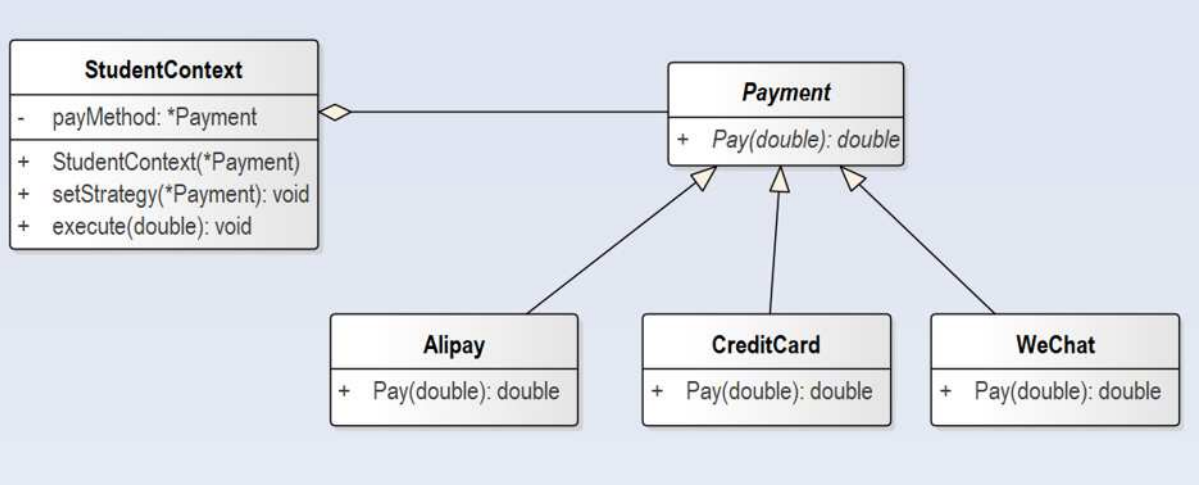

图 1 策略模式设计类图

2.3 实现

代码如下：

```
#include <iostream>
using namespace std;
class Payment
{    public:
                virtual void pay(double tuition) = 0;
};
class  CreditCard : public Payment
{    public:
                void pay(double tuition)        {
                        cout << “信用卡支付学费：” << tuition << endl;
                }
};
class Alipay : public Payment{
      public:
                void pay(double tuition)        {
                        cout << “支付宝支付学费：” << tuition << endl;
                }
```

```
};
 class WeChat : public Payment{
      public:
                void pay(double tuition)        {
                          cout << “微信支付学费：” << tuition << endl;
                }
};
 class StudentContext{
      private:
        Payment  *payMethod;
      public:
                StudentContext(Payment *payMethod)                {
                          this->payMethod = payMethod;
                }

                void setStrategy(Payment *payMethod){
                          this->payMethod = payMethod;
                }

                void execute(double tuition){
                          payMethod->pay(tuition);
                }
};
int main()
{
      StudentContext   stu1(new Alipay());  // 使用支付宝方式支付
      stu1.execute(5400);

      stu1.setStrategy(new  CreditCard()) ;  // 更改信用卡方式支付
      stu1.execute(4100);

stu1.setStrategy(new  WeChat()) ;   // 更改微信方式支付
      stu1.execute(3700);
      return 0;
}
```

在这个案例中，展示了如何使用策略模式来根据不同的支付方式（信用卡、支付宝、微信支付）来处理支付学费的逻辑。通过 Payment 抽象类和具体的支付类，可以轻松地扩展新的支付方式，而不需要修改现有的代码结构。同时，上下文类（StudentContex）负责在运行时管理支付方式，使得代码能够灵活地选择不同的支付方式进行支付。

3. 模板方法模式在教学应用

模板方法模式是一种行为设计模式，它定义了一个操作中的算法的框架，而将一些步骤延迟到子类中。这种方式使得子类可以重新定义算法的某些步骤而不改变算法的结构。

在模板方法模式中，通常会有一个抽象类定义了模板方法和一系列抽象方法，而具体的子类则实现这些抽象方法以完成算法。模板方法模式的结构通常包括一个抽象类和一个或多个具体子类。抽象类定义了算法的骨架和需要子类实现的方法，而具体子类则通过实现这些方法来完成算法的具体步骤。

模板方法模式一般应用在具有以下条件的应用中：

✓ 具有统一的操作步骤或操作过程

✓ 具有不同的操作细节

✓ 存在多个具有同样操作步骤的应用场景，但某些具体的操作细节却各不相同

3.1 引入案例

准备星巴兹饮料时，请精确地遵循下面的冲泡。

第一种：星巴兹咖啡冲泡法

- 把水煮沸
- 用沸水冲泡咖啡
- 把咖啡倒进杯子
- 加糖和牛奶

第二种：星巴兹茶冲泡法

- 把水煮沸
- 用沸水浸泡茶叶
- 把茶倒进杯子
- 加柠檬

在这里，泡茶和冲咖啡过程是如此相似，只是在某些细节存在不同，应该将共同的部分抽取出来，放进一个基类中。

3.2 模式引入

两种冲泡法是基本相同的，只是一些步骤需要不同的实现。以泛化了冲泡方法，设计一个基类（CaffeineBeverage），该基类包含了以下步骤：①把水煮沸②冲泡③把饮料倒进杯子④加调料，而一些细节步骤依赖具体子类进行[4]。设计类图如图 2 所示。

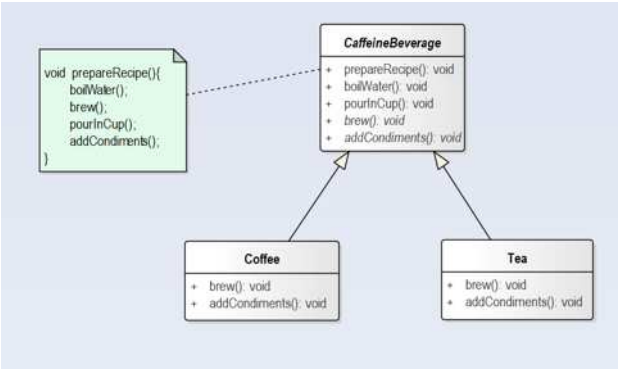

图 2 模板方法模式设计类图

3.3 实现

代码如下：

```
#include <iostream>
using namespace std;
class  CaffeineBeverage{
    public:
    void  prepareRecipe(){
            boilWater();
            brew();
            pourInCup();
            addCondiments();
    }
    void boilWater(){
```

```
        cout << “Boiling water” << endl;
    }
    void pourInCup(){
        cout << “Pouring into cup” <<endl;
    }
    virtual  void brew() = 0;
    virtual  void addCondiments() = 0;
};
class Coffee: public CaffeineBeverage{
    void  brew(){
        cout << “Dripping Coffee through filter” << endl;
    }
    void addCondiments(){
        cout << “Adding Sugar and Milk” << endl;
    }
};
class Tea : public CaffeineBeverage{
    void  brew(){
        cout << “Steeping the tea” << endl;
    }
    void addCondiments(){
        cout << “Adding Lemon” << endl;
    }
};
int main(){
    CaffeineBeverage  *qingTea = new Tea();
    qingTea->prepareRecipe();

    CaffeineBeverage  *lanshanCaff = new Coffee();
    lanshanCaff->prepareRecipe();

    return 0;
}
```

prepareRecipe() 称之为模板方法，它控制了算法，没有人能够改变它。这个方法也会依赖子类来提供某些或所有步骤的实现。模板方法主要提供了一个框架，模板方法定义了一连串的步骤，每个步骤由一个方法代表。新的咖啡因饮料只需要实现自己的方法就可以了。

4. 结语

通过引入设计模式作为辅助教学的内容，可以有效地帮助学生理解并应用继承和多态性等核心概念。设计模式不仅提供了解决常见问题的最佳实践，还展示了面向对象编程思想的精髓。通过具体的设计模式案例，学生可以直

观地看到如何在代码中实现这些概念，并学会如何在实际项目中应用它们，使学生更好地理解和掌握面向对象编程的思想和方法。

参考文献：

[1] 李涛 , 任廷艳 , 罗刚 , 等 . 模板方法模式在 C++ 多态性教学中的运用 [J]. 现代计算机 ,2019,(33):69-72.

[2] 张笑非 , 段先华 , 王长宝 , 等 . 基于观察者模式的 CoAP 课程教学与实验设计 [J]. 软件导刊 ,2019,18(12):199-203.

[3] 秦小波 . 设计模式之禅第 2 版 [M]. 北京：机械工业出版社，2014.

[4] 埃里克 . 弗里曼，伊丽莎白 . 罗布森 .Head First 设计模式（第二版）[M]. 北京 : 中国电力出版社 ,2022.

基金项目：

黔南民族师范学院 2020 年度校改项目（2020xjg0105）