软件工程 II 课程设计课程指导书

杨蒙召 编著

上海海洋大学海洋智能信息实验教学示范中心

实验一 面向抽象编程基础

一、实验目的

（1）掌握类和对象的设计
（2）掌握面向对象程序设计的基本方法
（3）掌握面向抽象编程的原则

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

有一个 Circle 类（圆类），该类创建的对象 Circle 调用 getArea（）方法可以计算圆的面积。Circle 类的代码如下：

```
public class Circle {
    double r;
    Circle(double r){
        this.r=r;
    }
    public double getArea() {
        return(3.14*r*r);
    }
}
```

现在要设计一个 Pillar 类（柱类），该类的对象调用 getVolume（）方法可以计算柱体的体积。Pillar 类的代码如下：

```
public class Pillar {
    Circle bottom;    //bottom 是用具体类 Circle 声明的对象
    double height;
    Pillar (Circle bottom,double height){
        this.bottom=bottom;
```

```
        this.height=height;
    }
    public double getVolume() {
        return bottom.getArea()*height;
    }
}
```

请补充 main 函数，完成圆柱体的体积计算。

2、编程题

上述 Pillar 类中，bottom 是用具体类 Circle 声明的对象，如果不涉及用户需求的变化，上面 Pillar 类的设计没有什么不妥，但是在某个时候，用户希望 Pillar 类能创建出底是三角形的柱体。显然上述 Pillar 类无法创建出这样的柱体，即上述设计的 Pillar 类不能应对用户的这种需求（软件设计面临的最大问题是用户需求的变化）。

利用面向抽象编程原则实现柱体体积的计算。程序参考如下：

```
public abstract class Geometry {

    public abstract double getArea();//只是单纯的构造了一个 abstract 方法，不用加任何东西。

}

class Circle extends Geometry{

    double r;

    Circle(double r){//构造函数

        this.r = r;

    }

    public double getArea() {   //方法的重写

        return r * r * 3.14;

    }

}
```

```
class Rectangle extends Geometry{
    double a,b;
   Rectangle(double a,double b){
        this.a = a;
        this.b = b;
    }
    public double getArea() {
        return a*b;
    }
}

public class Pillar {
   Geometry bottom;//声明 Geometry 对象（抽象对象）
    double hight;
   Pillar(Geometry bottom,double h){//构造函数，获取值
        this.bottom = bottom;
        this.hight = h;
    }

    public double getVolume(){//返回体积值
        if(bottom == null){
            System.out.println("没有底，无法计算面积");
```

```
            return 0;
        }
        return bottom.getArea()*hight;//这里的调用的 getArea()函数根据对 bottom 的上转型对象来确定。
    }
}
public class Application {
    public static void main(String[] args) {
        Pillar pillar;
        Geometry bottom;

        bottom = null;
        pillar = new Pillar(bottom,5);
        System.out.println("体积：" + pillar.getVolume());

        bottom = new Circle(2);//上转型对象
        pillar = new Pillar(bottom,1);
        System.out.println("体积：" + pillar.getVolume());//调用的是 Circle 里面的 getVolume()。

        bottom = new Rectangle(5,2);//上转型对象
        pillar = new Pillar(bottom,1);
        System.out.println("体积：" + pillar.getVolume());//调用的是 Rectangle 里面的 getVolume()。
```

```
    }
}
```

若有余力的同学，请增加底面积是三角形的柱体体积的计算。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求：

实验报告参考模板如下：

实验报告

题目：_____

学 院：信息学院

专 业：

班 级：

学 号：

姓 名：

年　　月　　日

一、实验目的

正文（正文 宋体小四，1.5 倍行距）

二、实验环境

三、实验内容

四、实验步骤（图文方式叙述）

五、实验结果及分析（遇到的问题与解决）

六、实验体会

实验二 面向抽象与接口编程

一、实验目的

（1）掌握抽象类和接口的设计
（2）掌握面向对象编程的基本方法
（3）掌握面向抽象编程的原则

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

要求有一个 abstract 类，类名为 Employee。Employee 的子类有 YearWorker，MonthWorker 和 WeekWorker。YearWorker 对象按年领取薪水，MonthWorker 按月领取薪水，WeekWorker 按周领取薪水。主程序 HardWork 能输出一年需要支付的薪水总额。

Employee 类有一个 abstract 方法：public abstract double earnings(); 子类必须重写父类的 earnings()方法，给出各自领取报酬的具体方法。

```
Employee[] e=new Employee[3];
e[0]=new YearWorker();
e[1]=new MonthWorker();
e[2]=new WeekWorker ();
```

程序设计思路：

一个 abstract 类名为 Employee，Employee 的子类有 YearWorker，MonthWorker 和 WeekWorker，子类必须重写父类的 earnings()方法。

```
package java_experiment_five;
abstract class Employee{
    abstract double earnings();
}
class YearWorker extends Employee{
    int year;
    double yearSalary;
    YearWorker(int y,double s){
        this.year=y;
        this.yearSalary=s;
    }
    double earnings() {
        return year*yearSalary;
    }

}
class MonthWorker extends Employee{
    int month;
```

```
    double monthSalary;
    MonthWorker(int m,double s){
        this.month=m;
        this.monthSalary=s;
    }
    double earnings() {
        return month*monthSalary;
    }

}
class WeekWorker extends Employee{
    int week;
    double weekSalary;
    WeekWorker(int w,double s){
        this.week=w;
        this.weekSalary=s;
    }
    double earnings() {
        return week*weekSalary;
    }

}
public class HardWork {
    public static void main(String[] args) {
        Employee[] e=new Employee[3];
        e[0]=new YearWorker(1, 100000);
        e[1]=new MonthWorker(12, 6000);
        e[2]=new WeekWorker (48, 1400);
        double sum;
        sum=e[0].earnings()+e[1].earnings()+e[2].earnings();
        System.out.println("总工资为："+sum);
    }
}
```

2、编程题

要求有一个 ComputeTotalSales 接口，该接口中有一个方法：public double totalSalesByYear();有三个实现该接口的类:Television,Computer 和 Mobile。这三个类通过实现 ComputeTotalSales 接口，给出自己的年销售额。定义一个 Shop 类,该类用 ComputeTotalSales 数组作为成员,该数组可以存放 Television，Computer 或 Mobile 对象的引用。利用接口回调技术计算 Shop 对象的年销售额。

程序设计思路：

ComputeTotalSales 接口，三个实现该接口的类：Television，Computer 和 Mobile，定义一个 Shop 类里面写一个方法计算年销售额，主类 Sale 利用接口回调技术计算 Shop 对象的年销售额。

程序参考如下：

```
package java_experiment_five;
interface ComputeTotalSales{
    public double totalSalesByYear( );
}
class Television implements ComputeTotalSales{
    public double totalSalesByYear() {
        return 10000;
    }
}
class Computer implements ComputeTotalSales{
    public double totalSalesByYear() {
        return 20000;
    }
}
class Mobile implements ComputeTotalSales{
    public double totalSalesByYear() {
        return 30000;
    }
}
class Shop{
    double totalSales;
    ComputeTotalSales[] goods;
    Shop(ComputeTotalSales[] goods){
        this.goods=goods;
```

```
    }
    public double giveTotalSales(){
        totalSales=0;
        for(int i=0;i<goods.length;i++){
            totalSales=totalSales+goods[i].totalSalesByYear();
        }
        return totalSales;
    }
}
public class Sale {
    public static void main(String[] args) {
        ComputeTotalSales[] goods=new ComputeTotalSales[40];
        for(int i=0;i<goods.length;i++){
            if(i%3==0) {
                goods[i]=new Television();
            }
            else if(i%3==1) {
                goods[i]=new Computer();
            }
            else if(i%3==2) {
                goods[i]=new Mobile();
            }
        }
        Shop shop=new Shop(goods);
        System.out.println("所有商品的年销售额为:"+shop.giveTotalSales());
    }
}
```

四、实验步骤

1. 打开 Eclipse 软件
2. 参考实验内容，新建项目和程序文件
3. 在程序文件中编写程序代码
4. 程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验三 命令模式

一、实验目的

（1）掌握 UML 的概念
（2）掌握命令模式的思想
（3）掌握命令模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

在许多设计中，经常涉及到一个对象请求另一个对象调用其方法到达某种目的。如果请求者不希望或无法直接和被请求者打交道，即不希望或无法含有被请求者的引用，那么就可以使用命令模式。比如军队中军官让连长“偷袭敌人”的作战命令。参考代码如下：

```
public class CompanyArmy{
   public void sneakAttack(){
      System.out.println("我们知道如何偷袭敌人,保证完成任务");
   }
}

public interface Command {
   public abstract void execute();
}

public class ConcreteCommand implements Command{
   CompanyArmy army;            //含有接收者的引用
   ConcreteCommand(CompanyArmy army){
```

```
        this.army=army;
    }
    public  void execute(){    //封装着指挥官的请求
        army.sneakAttack();     //偷袭敌人
    }
}

public class ArmySuperior{
  Command command;                        //用来存放具体命令的引用
  public void setCommand(Command command){
      this.command=command;
  }
  public void startExecuteCommand(){   //让具体命令执行execute()方法
      command.execute();
  }
}

public class Application{
    public static void main(String args[]){
        CompanyArmy 三连=new CompanyArmy();          //创建接收者
        Command command=new ConcreteCommand(三连);  //创建具体命令并指定接收者
        ArmySuperior 指挥官=new ArmySuperior();     //创建请求者
        指挥官.setCommand(command);
        指挥官.startExecuteCommand();
    }
}
```

2、编程题

结合设计模式实训教程 P137 和 P151，理解并验证命令模式之公告板系统，参考代码如下：

```
import java.util.*;
//抽象命令
```

```
interface Command
{
	public void execute();
}

//菜单项类：发送者（调用者）
class MenuItem
{
	private String name;
	private Command command;
	public MenuItem(String name)
	{
		this.name = name;
	}
	public String getName()
	{
		return this.name;
	}
	public void setName(String name)
	{
		this.name = name;
	}
	public Command getCommand()
	{
		return this.command;
	}
	public void setCommand(Command command)
	{
		this.command = command;
	}
	public void click()
	{
		command.execute();
	}
```

```
}

//菜单类
class Menu
{
	public ArrayList itemList = new ArrayList();
	public void addMenuItem(MenuItem item)
	{
		itemList.add(item);
	}
}

//打开命令：具体命令
class OpenCommand implements Command
{
	private BoardScreen screen;
	public OpenCommand(BoardScreen screen)
	{
		this.screen = screen;
	}
	public void execute()
	{
		screen.open();
	}
}

//新建命令：具体命令
class CreateCommand implements Command
{
	private BoardScreen screen;
	public CreateCommand(BoardScreen screen)
	{
		this.screen = screen;
	}
```

```
    public void execute()
    {
        screen.create();
    }
}

//编辑命令：具体命令
class EditCommand implements Command
{
    private BoardScreen screen;
    public EditCommand(BoardScreen screen)
    {
        this.screen = screen;
    }
    public void execute()
    {
        screen.edit();
    }
}

//公告板系统界面：接收者
class BoardScreen
{
    private Menu menu;
    private MenuItem openItem,createItem,editItem;
    public BoardScreen()
    {
        menu = new Menu();
        openItem = new MenuItem("打开");
        createItem = new MenuItem("新建");
        editItem = new MenuItem("编辑");
        menu.addMenuItem(openItem);
        menu.addMenuItem(createItem);
        menu.addMenuItem(editItem);
```

```
    }
    public void display()
    {
        System.out.println("主菜单选项： ");
        for(Object obj:menu.itemList)
        {
            System.out.println(((MenuItem)obj).getName());
        }
    }
    public void open()
    {
        System.out.println("显示打开窗口！ ");
    }
    public void create()
    {
        System.out.println("显示新建窗口！ ");
    }
    public void edit()
    {
        System.out.println("显示编辑窗口！ ");
    }
    public Menu getMenu()
    {
        return menu;
    }
}

//客户端测试类
class Client
{
    public static void main(String args[])
    {
        BoardScreen screen = new BoardScreen();
        Menu menu = screen.getMenu();
```

```
        Command openCommand,createCommand,editCommand;
        openCommand = new OpenCommand(screen);
        createCommand = new CreateCommand(screen);
        editCommand = new EditCommand(screen);
        MenuItem openItem,createItem,editItem;
        openItem = (MenuItem)menu.itemList.get(0);
        createItem = (MenuItem)menu.itemList.get(1);
        editItem = (MenuItem)menu.itemList.get(2);
        openItem.setCommand(openCommand);
        createItem.setCommand(createCommand);
        editItem.setCommand(editCommand);
        screen.display();
        openItem.click();
        createItem.click();
        editItem.click();
    }
}
```

四、实验步骤

****

五、实验报告要求

见实验一中的实验报告要求

实验四 观察者模式

一、实验目的

（1）理解多对一的依赖关系
（2）掌握观察者模式的思想
（3）掌握观察者模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

在许多设计中，经常涉及到多个对象都对一个特殊对象中的数据变化感兴趣，而且这多个对象都希望跟踪那个特殊对象中的数据变化。请大家参考教材，验证求职中心和二位求职者的观察者模式程序。

2、编程题

请验证设计模式实训教程 P174 页的股票变化的案例（**例子 5.2.7**），许多股民关注股市的变化，体现了多对一的**观察者模式。**

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验五 装饰模式

一、实验目的

（1）理解装饰模式的思想
（2）掌握多次装饰的思想
（3）掌握装饰模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

装饰模式是动态地扩展一个对象的功能，而不需要改变原始类代码的一种成熟模式。在装饰模式中，“具体组件”类(被装饰的)和“具体装饰”类（装饰者）是该模式中的最重要的两个角色。请大家验证鸟类飞翔（安装电子翅膀）的装饰模式程序。

2、编程题

请验证《设计模式实训教程》P87 描述和 P102 页的界面显示构建库的案例（实训实例 4.2.4），装饰的思想用在这个案例中，体现了装饰模式的应用。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验六 策略模式

一、实验目的

（1）理解策略模式的思想
（2）掌握策略模式的应用场景
（3）掌握策略模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

策略模式是处理算法的不同变体的一种成熟模式，策略模式通过接口或抽象类封装算法的标识，即在接口中定义一个抽象方法，实现该接口的类将实现接口中的抽象方法。在策略模式中，封装算法标识的接口称作策略，实现该接口的类称作具体策略。请验证如下程序，场景描述：比赛中若干裁判，给选手不同得分。请给出几种计算选手得分的评分方案（方法/策略）。

2、编程题

请参考《设计模式实训教程》P144 中对策略模式应用案例的描述，验证 P182 页的电影票打折策略的案例（参见实训实例 5.2.9）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验七 适配器模式和责任链模式

一、实验目的

（1）理解适配器模式的思想
（2）理解责任链模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

适配器模式的关键是建立一个适配器，这个适配器实现了目标接口并包含有被适配者的引用。请参考教材第八章例子 1，验证如下程序，电视机采用如何使用三相插座接通电流。

2、编程题

责任链模式的关键是将用户的请求分派给许多对象，这些对象被组织成一个责任链，即每个对象含有后继对象的引用，并要求责任链上的每个对象，如果能处理用户的请求，就做出处理，不再将用户的请求传递给责任链上的下一个对象；如果不能处理用户的请求，就必须将用户的请求传递给责任链上的下一个对象。请参考教材第九章例子 1，验证如下程序，用户提交身份证号验证他是否在北京、上海、天津居住。

3、编程题

请参考《设计模式实训教程》P136 中对责任链模式应用案例的描述，验证 P148 页的在线文档帮助系统的案例（参见实训教程案例 5.2.1）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验八 外观模式和迭代器模式

一、实验目的

（1）理解外观模式的思想
（2）理解迭代器模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

外观模式是简化用户和子系统进行交互的成熟模式，外观模式的关键是为子系统提供一个称作外观的类，该外观类的实例负责和子系统中类的实例打交道。当用户想要和子系统中的若干个类的实例打交道时，可以代替地和子系统的外观类的实例打交道。请参考教材第十章例子 1，验证如下程序，报社广告子系统如何代理客户实现广告的字符统计、费用计算和排版设计。

2、编程题

迭代器模式是遍历集合的成熟模式，迭代器模式的关键是将遍历集合的任务交给一个称作迭代器的对象。请参考教材第十一章例子 1，验证如下程序，点钞机实现对假币的剔除，对真币总额的计算。

3、编程题

请参考《设计模式实训教程》P139 中对迭代器模式应用案例的描述，验证 P161 页的商品名称遍历的案例（参见实训教程案例 5.2.4）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验九 中介者模式和工厂模式

一、实验目的

（1）理解中介者模式的思想
（2）理解工厂方法和抽象工厂模式的对比
（3）掌握几种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

中介者模式是封装一系列的对象交互的成熟模式，其关键是将对象之间的交互封装在称作中介者的对象中，中介者使各对象不需要显示地相互引用，这些对象只包含中介者的引用。当系统中某个对象需要和系统中另外一个对象交互时，只需将自己的请求通知中介者即可。请参考教材第十二章例子 1，验证如下程序，A、B 和 C 三方通过中介者（调停方）表达不同的停战诉求，从而达成停战协议。

2、编程题

抽象工厂模式的关键是在一个抽象类或接口中定义若干个抽象方法，这些抽象方法分别返回某个类的实例，该抽象类或接口让其子类或实现该接口的类重写这些抽象方法,为用户提供一系列相关的对象。请参考教材第十四章例子 1，验证如下程序，用户（商店 shop）无法直接生产（创建）西服和牛仔等衣服，但是他可以借用一个两个工厂类生产（创建）北京牌西服和上海牌牛仔。

3、编程题

请参考《设计模式实训教程》P41 中对工厂方法模式应用案例的描述，验证 P48 页的日志记录器的案例（参见实训教程案例 3.2.2）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十 生成器模式和原型模式

一、实验目的

（1）理解生成器模式的思想
（2）理解原型模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

生成器模式的关键是将一个包含有多个组件对象的创建分成若干个步骤，并将这些步骤封装在一个称作生成器的接口中。请参考教材第十五章例子 1，验证如下程序，指挥者指挥不同的具体生成器创建产品，即面板上三个组件：按钮、标签和文本框出现的顺序和名称不一致。

2、编程题

原型模式是从一个对象出发得到一个和自己有相同状态的新对象的成熟模式，该模式的关键是将一个对象定义为原型，并为其提供复制自己的方法。请参考教材第十六章例子 2（P162-163）浅克隆和深度克隆，注意对于 Geometry 类中成员变量是 rectangle 要再克隆一次。

3、编程题

请参考《设计模式实训教程》P43 中对生成器模式（建造者模式）应用案例的描述，验证 P53 页的游戏人物角色的案例（参见实训教程案例 3.2.4）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十一 单件模式和组合模式

一、实验目的

（1）理解单件模式的思想
（2）理解组合模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

单件模式是关于怎样设计一个类，并使得该类只有一个实例的成熟模式，该模式的关键是将类的构造方法设置为 private 权限，并提供一个返回它的唯一实例的类方法。。请参考教材第十七章例子 1，验证如下程序，创建唯一的月亮对象，不同的人看到同一个月亮。

2、编程题

组合模式是关于怎样将对象形成树形结构来表现整体和部分的层次结构的成熟模式。使用组合模式，可以让用户以一致的方式处理个体对象和组合对象，组合模式的关键在于无论是个体对象，还是组合对象都实现了相同的接口或都是同一个抽象类的子类。请参考教材第十八章例子 1，建立士兵的树形结构，计算一个连队、一个排或一个班的工资。

3、编程题

请参考《设计模式实训教程》P44 中对单件模式（单例模式）应用案例的描述，验证 P61 页的文档的案例（参见实训教程案例 3.2.6）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十二 桥接模式和状态模式

一、实验目的

（1）理解桥接模式的思想
（2）理解状态模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

桥接模式是关于怎样将抽象部分与它的实现部分分离，使得它们都可以独立地变化的成熟模式。请参考教材第十九章例子 1，验证如下程序，计算建筑楼房的成本（住宅楼和商业楼）。

2、编程题

状态模式的关键是将对象的状态封装成为独立的类，对象调用方法时，可以委托当前对象所具有的状态调用相应的方法，使得当前对象看起来好像修改了它的类。请参考教材第二十章例子 1，设计带文字提示信息的温度计。

3、编程题

请参考《设计模式实训教程》P143 中对状态模式应用案例的描述，验证 P177 页的银行账户的案例（参见实训教程案例 5.2.8）。

四、实验步骤

1.打开 Eclipse 软件

2.参考实验内容，新建项目和程序文件

3.在程序文件中编写程序代码

4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十三 模板模式和代理模式

一、实验目的

（1）理解模板模式的思想
（2）理解代理模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

模板方法是关于怎样将若干个方法集成到一个方法中，以便形成一个解决问题的算法骨架。模板方法模式的关键是在一个抽象类中定义一个算法的骨架，即将若干个方法集成到一个方法中，并称该方法为一个模板方法，或简称为模板。请参考教材第二十一章例子 1，验证如下程序，按照不同排序方式，显示某个目录下全部文件的名字。

2、编程题

代理模式是为对象提供一个代理，代理可以控制对它所代理的对象的访问。模式的结构中包括三种角色：抽象主题、实际主题、代理。请参考教材第二十二章例子 1，代理验证三边是否可以构成三角形，进而转交实际主题计算三角形的面积。

3、编程题

请参考《设计模式实训教程》P90 中对代理模式应用案例的描述，验证 P111 页的日志记录代理的案例（参见实训教程案例 4.2.7）。

四、实验步骤

1.打开 Eclipse 软件

2.参考实验内容，新建项目和程序文件

3.在程序文件中编写程序代码

4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十四 享元模式和访问者模式

一、实验目的

（1）理解享元模式的思想
（2）理解访问者模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

一个类中的成员变量表明该类所创建的对象所具有的属性，在某些程序设计中我们可能用一个类创建若干个对象，但是我们发现这些对象的一个共同特点：它们有一部分属性的取值必须是完全相同的，那么可以将这些相同的属性单独构建为一个类，以共享这些元素，称为享元模式。请参考教材第二十三章例子 1，验证如下程序，相同型号的车具有相同的长宽高，不同的颜色和功率，这些相同的元素可以构成共享的类。

2、编程题

当一个集合中有若干个对象时，习惯上将这些对象称作集合中的元素，访问者模式可以使得我们在不改变集合中各个元素的类的前提下,定义作用于这些元素上的新操作。请参考教材第二十四章例子 1，验证如下程序，大学生和研究生参加公司面试时，要接受公司的考核，但考核录用的标准是由公司来定的。

3、编程题

请参考《设计模式实训教程》P89 中对代理模式应用案例的描述，验证 P107 页的围棋棋子的案例（参见实训教程案例 4.2.6）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十五 备忘录模式和解释器模式

一、实验目的

（1）理解备忘录模式的思想
（2）理解解释器模式的思想
（3）掌握两种模式的编程实现

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

备忘录模式是关于怎样保存对象状态的成熟模式，其关键是提供一个备忘录对象，该备忘录负责存储一个对象的状态，程序可以在磁盘或内存中保存这个备忘录。请参考教材第二十五章例子 1，验证如下程序，从 phrase 文件中挑选自己喜欢的成语写入 favorphrase 中，但要记录下每次读写的地址（位置），以便能以恢复。

2、编程题

解释模式是关于怎样实现一个简单语言的成熟模式，其关键是将每一个语法规则表示成一个类。请参考教材第二十六章例子 1，验证如下程序，语言的中英文翻译和解释。

3、编程题

请参考《设计模式实训教程》P141 中对备忘录模式应用案例的描述，验证 P170 页的游戏恢复点设置的案例（参见实训教程案例 5.2.6）。

四、实验步骤

1.打开 Eclipse 软件
2.参考实验内容，新建项目和程序文件
3.在程序文件中编写程序代码
4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求

实验十六 综合案例设计

一、实验目的

（1）理解软件设计模式的思想
（2）掌握核心设计模式的框架
（3）学会利用设计模式实现综合案例的设计

二、实验环境

Windows 7
Eclipse+JDK

三、实验内容

1、编程题

1．应用场景与案例描述
选取或设计一个应用场景或案例，以适合你抽取的设计模式给以解决，首先用文字或图表，给出类似于《设计模式实训教程》中的描述。（鼓励自己结合实际生活和学习，提出原创的应用案例。或者也可以改编一些案例、或者网上其他材料的修改等。）。
2．案例分析与解决问题
对你描述的应用场景或案例，先进行分析（如同《设计模式实训教程》的分析），分析的目的，是为了提出应用场景或案例存在的潜在需求和需要解决的问题（为了你的模式有用武之地），针对该需求和问题，说明采用你的模式可以很好解决这些问题。
3．各个角色描述与UML图示
首先给出模式的几种角色描述(参考课本中固定的几种角色，用你设计每个类，描述对应角色)；然后仿照书本，画出你的UML图（用你的角色和成员替换书中UML内容，但UML的框架保持不变），角色之间关系和每种角色的主要成员，以及之间的调用关系不变。

四、实验步骤

1.打开 Eclipse 软件

2.参考实验内容，新建项目和程序文件

3.在程序文件中编写程序代码

4.程序调试，如果结果无误，则运行程序出结果

五、实验报告要求

见实验一中的实验报告要求