设计模式课程实验报告（项目三）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程名称** | 设计模式 | **班级** | 20软件2 | **实验日期** | 第10-14周周三第一二节 |
| **姓名** | 徐小龙 | **学号** | B20200306207 | **指导老师** | 张肖霞 |
| **实验评分** |  | | | | |
| **教师评语** |  | | | | |
| **实验名称** | 行为型设计模式 | | | | |
| 实验目的及要求 | 掌握职责链模式、命令模式，迭代器模式，备忘录模式、观察者模式、状态模  式和策略模式的应用。 | | | | |
|
|
|
|
|
| 实验环境 | 1、硬件平台：计算机CPU 奔腾Ⅲ；内存64兆以上（最好128兆以上）。  2、软件：Windows 2000 server；Eclipse。 | | | | |
| 实验步骤 | 1. 掌握各个模式的定义； 2. 理解各个模式的结构图；   3、使用相关的工具绘制给定背景的UML类图；  4、依据类图使用合适的面向对象语言实现；  5、分析和总结各模式的优缺点。 | | | | |
| 实验内容及实验结果（截图） | 按要求完成以下各题（至少完成5道题）  第一题可以从一（1）和一（2）中任选一道完成。  一(1)、某OA系统需要提供一个假条审批的模块，如果员工请假天数小于3天，主任可以审批该请假条；如果员工请假天数大于等于3天，小于10天，经理可以审批；如果员工请假天数大于等于10天，小于30天，总经理可以审批；如果超过30天，总经理也不能审批，提示相应的拒绝信息。  要求绘制相应的类图并使用Java语言编程模拟实现。    代码：  package test3\_1\_1;  import java.util.Scanner;  public class Client {  public static void main(String []args) {  DirectorHandler dir=new DirectorHandler();  ManagerHandler man=new ManagerHandler();  DirectorGeneralManagerHandler dirma=new DirectorGeneralManagerHandler();  dir.setNextHandler(man);  man.setNextHandler(dirma);  while(true) {  Scanner sc=new Scanner(System.in);  int re=sc.nextInt();  dir.handler(re);  }  }  }  DirectorGeneralManagerHandler.java:  package test3\_1\_1;  public class DirectorGeneralManagerHandler implements Handler{  private Handler nextHandler;    public void setNextHandler(Handler handler) {  this.nextHandler=handler;  }  public void handler(int request) {  if(request<30) {  System.out.println("总经理审批"+request+"天假期，请假成功！");  }  else {  System.out.println("请假时间太长，总经理无权审批！");  }  }  }  DirectorHandler.java:  package test3\_1\_1;  public class DirectorHandler implements Handler{  private Handler nextHandler;    public void setNextHandler(Handler handler) {  this.nextHandler=handler;  }  public void handler(int request) {  if(request<3) {  System.out.println("主任审批"+request+"天假期，请假成功！");  }  else {  nextHandler.handler(request);  }  }  }  Handler.java:  package test3\_1\_1;  public interface Handler {  public void setNextHandler(Handler handler);  public void handler(int request);  }  ManagerHandler.java:  package test3\_1\_1;  public class ManagerHandler implements Handler{  private Handler nextHandler;    public void setNextHandler(Handler handler) {  this.nextHandler=handler;  }  public void handler(int request) {  if(request<10) {  System.out.println("经理审批"+request+"天假期，请假成功！");  }  else {  nextHandler.handler(request);  }  }    }  类图：    测试结果：     1. （1）现有一个文本编辑器，在该文本编辑器中为客户提供了不同的对文本编辑的展示方式，有按钮型、菜单型，快捷键等。文本编辑主要包括新建文件、打开文件、保存文件等功能，根据客户在展示层的不同选择，该文本编辑器可实现不同的功能。请使用命令模式来模拟该问题的实现，要求绘制相应的类图并使用JAVA语言编程实现。   路径：    代码：  package test3\_2\_1;  public class Client {  public static void main(String []args) {  // 宏命令  FileInvoker fileInvoker = new FileInvoker();  FileReceiver fileReceiver = new FileReceiver();    fileInvoker.addCommands(new CreateFileCommand(fileReceiver, ".\\test\\Akazuki"));  fileInvoker.addCommands(new OpenFileCommand(fileReceiver, ".\\test\\Akazuki"));  fileInvoker.addCommands(new SaveFileCommand(fileReceiver, ".\\test\\Akazuki"));    fileInvoker.executeCommands();  }  }  Command.java:  package test3\_2\_1;  public interface Command {  public void execute();  }  CreateFileCommand.java:  package test3\_2\_1;  public class CreateFileCommand implements Command{  private FileReceiver fileReceiver;  private String src;    public CreateFileCommand(FileReceiver fileReceiver, String src) {  super();  this.fileReceiver = fileReceiver;  this.src = src;  }  public void execute() {  System.out.println("创建文件"+src+"中");  fileReceiver.createFile();  }  }  FileInvoker.java:  package test3\_2\_1;  import java.util.ArrayList;  import java.util.List;  public class FileInvoker {  private List<Command> list = new ArrayList<>();  public void addCommands(Command command) {  list.add(command);  }  public void executeCommands() {  for(Command command:list) {  command.execute();  }  }  }  FileReceiver.java:  package test3\_2\_1;  public class FileReceiver {  public void createFile() {  System.out.println("创建文件！");  }  public void openFile() {  System.out.println("打开文件！");  }  public void saveFile() {  System.out.println("保存文件！");  }  }  OpenFileCommand.java:  package test3\_2\_1;  public class OpenFileCommand implements Command{  private FileReceiver fileReceiver;  private String src;    public OpenFileCommand(FileReceiver fileReceiver, String src) {  super();  this.fileReceiver = fileReceiver;  this.src = src;  }  public void execute() {  System.out.println("打开文件"+src+"中");  fileReceiver.openFile();  }  }  SaveFileCommand.java:  package test3\_2\_1;  public class SaveFileCommand implements Command{  private FileReceiver fileReceiver;  private String src;    public SaveFileCommand(FileReceiver fileReceiver, String src) {  super();  this.fileReceiver = fileReceiver;  this.src = src;  }  public void execute() {  System.out.println("保存文件"+src+"中");  fileReceiver.saveFile();  }  }  类图：    测试：     1. 电视机遥控器就是一个迭代器的实例，通过它可以实现对电视频道集合的遍历操作，电视机可以看成一个存储频道的聚合对象，试模拟电视机遥控器的实现，要求绘制相应的类图并使用Java语言编程模拟实现。   路径：    代码：  AbsIterator.java:  package test3\_3;  public interface AbsIterator {  public void begin();  public void end();  public boolean hasNext();  public boolean hasPrev();  public Object prev();  public Object next();  }  Client.java:  package test3\_3;  import java.util.ArrayList;  import java.util.List;  import java.util.Scanner;  public class Client {  public static void main(String []args) {  List<Object>list=new ArrayList<>();  for(int i=1;i<=12;i++) {  list.add("cctv"+i);  }  ConcreteTv tv = new ConcreteTv(list);  System.out.println("正向输出所有频道！");  AbsIterator it = tv.createIterator();  while(it.hasNext()) {  System.out.println(it.next());  }    System.out.println("-------------");  System.out.println("反向输出所有频道");  it = tv.createPrevIterator();  while(it.hasPrev()) {  System.out.println(it.prev());  }  while(true) {  System.out.println("请输入数字对应的频道，可以选择的频道为1-"+list.size());  Scanner sc = new Scanner(System.in);  int num=sc.nextInt();  if(num<1||num>list.size()) {  System.out.println("输出数字超过范围，请重新输入！");  continue;  }  int idx=1;  AbsIterator itt=tv.createIterator();  while(idx<num&&itt.hasNext()) {  idx++;  itt.next();  }  System.out.println(itt.next());  }  }  }  ConcreteTv.java:  package test3\_3;  import java.util.ArrayList;  import java.util.List;  public class ConcreteTv extends Tv{  List<Object>objlist=new ArrayList<>();  public AbsIterator createNextIterator() {  return new TvIterator(this).getNextIterator();  }    public ConcreteTv(List<Object> objs) {  this.objlist = objs;  }    public AbsIterator createIterator() {  return new TvIterator(this).getNextIterator();  }    public AbsIterator createPrevIterator() {  return new TvIterator(this).getPrevIterator();  }  public Object getCurrentItem(int idx) {    return objlist.get(idx);  }  public int size() {  return objlist.size();  }  }  Tv.java:  package test3\_3;  public abstract class Tv {  public abstract AbsIterator createNextIterator();  public abstract AbsIterator createPrevIterator();  }  TvIterator.java:  package test3\_3;  public class TvIterator implements AbsIterator{  private ConcreteTv tv;  private int index;  private int size;  public TvIterator() {  }  public TvIterator(ConcreteTv tv) {  this.tv = tv;  this.index = 0;  this.size = tv.size();  }  public void begin() {  index=0;  }  @Override  public void end() {  index=size;  }  @Override  public boolean hasNext() {  return index!=size;  }  @Override  public boolean hasPrev() {  return index!=-1;  }  @Override  public Object prev() {  return tv.getCurrentItem(index--);  }  @Override  public Object next() {  return tv.getCurrentItem(index++);  }  public AbsIterator getNextIterator() {  this.index = 0;  this.size = tv.size();  return this;  }  public AbsIterator getPrevIterator() {  this.size = tv.size();  this.index = this.size - 1;  return this;  }    }  类图：  测试：      四、某系统提供了用户信息操作模块，用户可以修改自己的各项信息。为了使操作过程更加人性化，现使用备忘录模式对系统进行改进，使得用户在进行了错误操作之后可以恢复到操作之前的状态。用户信息中包含账号、密码、电话号码等信息。要求绘制相应的类图并使用Java语言编程模拟实现。  结构：  代码：  Client.java:  package test3\_4;  public class Client {  public static void main(String []args) {  MementoCaretaker me =new MementoCaretaker();  UserInfo us = new UserInfo("111","222","333");  me.setMemento(us.save());  System.out.println("开始: "+us.getUsername()+" "+us.getPassword()+" "+us.getTel());    us.setUsername("1");  us.setPassword("2");  us.setTel("3");  System.out.println("保存: "+us.getUsername()+" "+us.getPassword()+" "+us.getTel());    us.restore(me.getMemento());  System.out.println("恢复: "+us.getUsername()+" "+us.getPassword()+" "+us.getTel());  }  }  MementoCaretaker.java:  package test3\_4;  import java.util.ArrayList;  import java.util.List;  public class MementoCaretaker {  private List<UserInfoMemento> list = new ArrayList<>();  public UserInfoMemento getMemento() {  return list.remove(list.size()-1);  }  public void setMemento(UserInfoMemento userInfoMemento) {  list.add(userInfoMemento);  }  }  UserInfo.java:  package test3\_4;  public class UserInfo {  private String username;  private String password;  private String tel;  public UserInfo() {    }    public UserInfo(String username, String password, String tel) {  super();  this.username = username;  this.password = password;  this.tel = tel;  }    public String getUsername() {  return username;  }  public String getPassword() {  return password;  }  public String getTel() {  return tel;  }      public void setUsername(String username) {  this.username = username;  }  public void setPassword(String password) {  this.password = password;  }  public void setTel(String tel) {  this.tel = tel;  }  public UserInfoMemento save() {  return new UserInfoMemento(username,password,tel);  }  public void restore(UserInfoMemento userInfoMemento) {  this.password=userInfoMemento.getPassword();  this.tel=userInfoMemento.getTel();  this.username=userInfoMemento.getUsername();  }  }  UserInfoMemento.java:  package test3\_4;  public class UserInfoMemento {  private String username;  private String password;  private String tel;  public UserInfoMemento(String username,String password,String tel) {  this.password=password;  this.tel=tel;  this.username=username;  }  public String getUsername() {  return username;  }  public String getPassword() {  return password;  }  public String getTel() {  return tel;  }  }  类图：  测试：    五、某实时在线股票软件需要提供以下功能：当股票购买者所购买的某只股票的价格变化幅度超过5%时，系统将自动发送通知（包括新价格）给购买该股票的所有股民。试使用观察者模式设计并实现该系统，要求绘制相应的类图并使用Java语言模拟实现。  结构：  代码：  Client.java:  package test3\_5;  import java.util.Vector;  public class Client {  public static void main(String []args) {  Subject s1=new StockSubject("1111", 100, 100, new Vector<>());  Subject s2=new StockSubject("2222", 100, 100, new Vector<>());    Observer o1=new StockObserver("o1");  Observer o2=new StockObserver("o2");    s1.attach(o1);  s1.attach(o2);  s2.attach(o1);    s1.updatePrice(106D);  s1.updatePrice(107D);  s2.updatePrice(101D);  s2.updatePrice(1000D);  }  }  Observer.java:  package test3\_5;  public interface Observer {  public void update(Subject subject) ;  }  StockObserver.java:  package test3\_5;  public class StockObserver implements Observer{  private String name;  public StockObserver(String name) {  super();  this.name = name;  }  public void update(Subject subject) {  System.out.println(name+"您好，检测到"+subject.name+"波动大于5%,新价格为"+subject.curPrice);  }  }  StockSubject.java:  package test3\_5;  import java.util.Vector;  public class StockSubject extends Subject{  public StockSubject(String name, double curPrice, double oldPrice, Vector<Observer> obV) {  super(name, curPrice, oldPrice, obV);  }  public void notifyObserver() {  for(Observer ob:obV) {  ob.update(this);  }  }    public void updatePrice(Double newPrice) {  this.oldPrice=this.curPrice;  this.curPrice=newPrice;  double dif=Math.abs(oldPrice-curPrice)/oldPrice;  if(dif>0.05) {  notifyObserver();  }  }  }  Subject.java:  package test3\_5;  import java.util.Vector;  public class Subject {  public String name;  public double curPrice;  public double oldPrice;  Vector<Observer> obV;    public Subject(String name, double curPrice, double oldPrice, Vector<Observer> obV) {  super();  this.name = name;  this.curPrice = curPrice;  this.oldPrice = oldPrice;  this.obV = obV;  }  public void attach(Observer observer) {  obV.add(observer);  }  public void detach(Observer observer) {  obV.remove(observer);  }  public void notifyObserver() {    }  public void updatePrice(Double newPrice) {    }  }  类图：  测试：    六、在酒店订房系统中，酒店的房间对象可以供用户预定bookRoom，入住checkInRoom，取消预定cancelRoom，退房checkOutRoom。该系统房间分为三种状态，分别是空闲状态，已预定状态，已入住状态。这三种状态下可以执行的操作如下：  （1）如果酒店是空闲状态，用户可以预定和入住。  （2）如果酒店是已预定状态，用户可以入住和取消预定。  （3）如果酒店是已入住状态，用户可以退房。  试使用状态模式解决上述问题，要求绘制相应的类图并使用Java语言编程模拟实现。  结构：    代码：  BookedStatus.java:  package test3\_6;  public class BookedStatus implements Status{  HotelContext hot;  public BookedStatus(HotelContext context){  this.hot = context;  }  @Override  public void bookRoom() {  System.out.println("预定失败， 房间已预定");  }  @Override  public void checkInRoom() {  System.out.println("入住成功。。。");  this.hot.setCurrentStatus(hot.getBusyStatus());  }  @Override  public void cancelRoom() {  System.out.println("取消预定成功。。。");  this.hot.setCurrentStatus(hot.getFreeStatus());  }  @Override  public void checkOutRoom() {  System.out.println("退房失败， 房间已预定");  }  }  BusyStatus.java:  package test3\_6;  public class BusyStatus implements Status{  HotelContext hot;  public BusyStatus(HotelContext context){  this.hot = context;  }    @Override  public void bookRoom() {  System.out.println("预定失败， 房间已入住");  }  @Override  public void checkInRoom() {  System.out.println("入住失败， 房间已入住");  }  @Override  public void cancelRoom() {  System.out.println("取消预定失败， 房间已入住");  }  @Override  public void checkOutRoom() {  System.out.println("退房成功。。。");  this.hot.setCurrentStatus(hot.getFreeStatus());  }  }  Client.java:  package test3\_6;  public class Client {  public static void main(String []args) {  System.out.println("\n正常流程:");  HotelContext h1=new HotelContext();  h1.bookRoom();  h1.checkInRoom();  h1.checkOutRoom();  h1.cancelRoom();  System.out.println("\n在取消预定后入住:");  HotelContext h3=new HotelContext();  h3.bookRoom();  h3.cancelRoom();  h3.checkInRoom();    System.out.println("\n在住房时预定:");  HotelContext h2=new HotelContext();  h2.checkInRoom();  h2.bookRoom();    System.out.println("\n在空房时退房:");  HotelContext h4=new HotelContext();  h4.checkOutRoom();    }  }  FreeStatus.java:  package test3\_6;  public class FreeStatus implements Status{  HotelContext hot;  public FreeStatus(HotelContext context) {  this.hot = context;  }  @Override  public void bookRoom() {  System.out.println("房间预定成功。。。");  this.hot.setCurrentStatus(hot.getBookedStatus());  }  @Override  public void checkInRoom() {  System.out.println("房间入住成功。。。");  this.hot.setCurrentStatus(hot.getBusyStatus());  }  @Override  public void cancelRoom() {  System.out.println("取消失败， 房间未预定");  }  @Override  public void checkOutRoom() {  System.out.println("退房失败， 房间未入住");  }  }  HotelContext.java:  package test3\_6;  public class HotelContext {  private Status currentStatus;  private Status freeStatus;  private Status busyStatus;  private Status bookedStatus;    public Status getCurrentStatus() {  return currentStatus;  }  public void setCurrentStatus(Status currentStatus) {  this.currentStatus = currentStatus;  }  public Status getFreeStatus() {  return freeStatus;  }  public void setFreeStatus(Status freeStatus) {  this.freeStatus = freeStatus;  }  public Status getBusyStatus() {  return busyStatus;  }  public void setBusyStatus(Status busyStatus) {  this.busyStatus = busyStatus;  }  public Status getBookedStatus() {  return bookedStatus;  }  public void setBookedStatus(Status bookedStatus) {  this.bookedStatus = bookedStatus;  }  public HotelContext() {  currentStatus=new FreeStatus(this);  freeStatus=new FreeStatus(this);  busyStatus=new BusyStatus(this);  bookedStatus=new BookedStatus(this);  }  public void bookRoom() {  bookedStatus.bookRoom();  }  public void checkInRoom() {  currentStatus.checkInRoom();  }  public void checkOutRoom() {  currentStatus.checkOutRoom();  }  public void cancelRoom() {  currentStatus.cancelRoom();  }  }  Status.java:  package test3\_6;  public interface Status {  public void bookRoom();  public void checkInRoom();  public void cancelRoom();  public void checkOutRoom();  }  类图：    测试：    七、设计一个网上书店，该系统中所有的计算机类图书（ComputerBook）每本都有10%的折扣，所有的语言类图书（LanguageBook）每本都有2元的折扣，小说类图书（NovelBook）每100元有10元的折扣。  现使用策略模式来设计该系统，绘制类图并编程实现。  结构：  代码： Book.java:  package test3\_7;  public class Book {  public Discount discount;  public double price;  public void setDiscount(Discount discount) {  this.discount = discount;  }  public void setPrice(double price) {  this.price = price;  }    public Discount getDiscount() {  return discount;  }  public double getPrice() {  return price;  }  public double actionDiscount() {  return 0;  }  }  Client.java:  package test3\_7;  public class Client {  public static void main(String []args) {  Book b1 = new LanguageBook();  b1.setDiscount(new TwoYuanDiscount());  b1.setPrice(100D);    Book b2 = new ComputerBook();  b2.setPrice(101D);  b2.setDiscount(new TenDiscount());    Book b3 = new NovelBook();  b3.setPrice(104D);  b3.setDiscount(new TenFullOneHundredDiscount());    System.out.println("语言类原价："+b1.getPrice()+",折扣后为："+b1.actionDiscount());  System.out.println("计算机类原价："+b2.getPrice()+",折扣后为："+b2.actionDiscount());  System.out.println("小说类原价："+b3.getPrice()+",折扣后为："+b3.actionDiscount());    }  }  ComputerBook.java:  package test3\_7;  public class ComputerBook extends Book{  public double actionDiscount() {  return discount.action(this.price);  }  }  Discount.java:  package test3\_7;  public interface Discount {  public double action(double price);  }  LanguageBook.java:  package test3\_7;  public class LanguageBook extends Book{  public double actionDiscount() {  return discount.action(this.price);  }  }  NovelBook.java:  package test3\_7;  public class NovelBook extends Book{  public double actionDiscount() {  return discount.action(this.price);  }  }  TenDiscount.java:  package test3\_7;  public class TenDiscount implements Discount{  @Override  public double action(double price) {  return price\*0.9;  }    }  TenFullOneHundredDiscount.java:  package test3\_7;  public class TenFullOneHundredDiscount implements Discount{  @Override  public double action(double price) {  return price-(int)price/100\*10;  }    }  TwoYuanDiscount.java:  package test3\_7;  public class TwoYuanDiscount implements Discount{  @Override  public double action(double price) {  return Math.max(price-2, 0);  }  }  类图：  测试： | | | | |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
| 小结 | 在这次实验中我学会了，职责链模式、命令模式，迭代器模式，备忘录模式、观察者模式、状态模式和策略模式的应用。 | | | | |
|
|
|
| 教师评语 |  | | | | |