**哈尔滨工业大学（威海）**

**Java语言实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | Java语言 | 课程编号 | SE33601 |
| 实验名称 | 类的关联及其应用 | | |
| 实验类型 | 验证与编程 | 实验学时 | 2学时 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 班号 | 1811103 | 学号 | 181110315 | 姓名 | 王少博 |
| 开始 | 2019.10.09 | 截止 | 2019.10.16 | 成绩 |  |
| 批语 |  | | | | |

目录

[1 实验目的、内容和要求 1](#_Toc20385685)

[1.1 实验目的 1](#_Toc20385686)

[1.2 实验内容 1](#_Toc20385687)

[2实验前准备 2](#_Toc20385688)

[3 实验内容 4](#_Toc20385689)

[3.1 冒险游戏设计与实现 4](#_Toc20385690)

[3.2 超市发票系统的设计与实现 6](#_Toc20385691)

# 1 实验目的、内容和要求

## 1.1 实验目的

本次实验着重训练学生面向对象的灵活应用，体现在两个方面：类的水平关联关系和垂直关联关系。同时，继续训练利用UML设计后再进行编码的软件开发过程。因此本次实验的主要目的有三：

1. 训练UML中类的关系的表达
2. 掌握类中水平关联关系
3. 掌握类的继承、垂直关联等类垂直方向上的关系

## 1.2 实验内容

1. 根据某游戏的业务需求，设计类以及类之间的水平关联并实现和测试代码
2. 根据某信息管理系统的业务需求，设计类及类之间的关系，并实现和测试。

# 2实验前准备

本次实验包含两个部分：一是Duck Simulator案例为原型的游戏设计；二是咖啡计费系统为原型的超市收费系统的设计。因此，需要认真吃透两个案例的设计思想和相关面向对象的概念。

此外，本案例中类的个数和难度与第1次，第2次实验相比，有了很大的提高，在类图表达上也越来越复杂，在此，将类与类之间的关系介绍一下，为同学们绘制类图提供参考。

在java中类和类之间的关系基本上有继承、实现、依赖、关联、聚合、组合六种关系。它们在UML中的表达方式如下：

* 继承关系：继承指的是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力。在Java中继承关系通过关键字extends明确标识，在设计时一般没有争议性。在UML类图设计中，继承用一条带空心三角箭头的实线表示，从子类指向父类，或者子接口指向父接口，如图1所示。

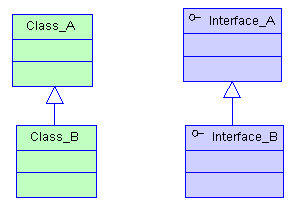


图1 继承的UML表达

* 实现关系：实现指的是一个class类实现interface接口（可以是多个）的功能，实现是类与接口之间最常见的关系。在Java中此类关系通过关键字implements明确标识，在设计时一般没有争议性。在UML类图设计中，实现用一条带空心三角箭头的虚线表示，从类指向实现的接口，如图2所示。

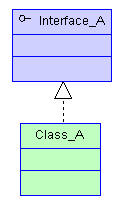


图2 实现的UML表达

* 依赖关系：简单的理解，依赖就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、临时性的、非常弱的，但是类B的变化会影响到类A。比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖。表现在代码层面，为类B作为参数被类A在某个method方法中使用。在UML类图设计中，依赖关系用由类A指向类B的带箭头虚线表示，如图3所示。

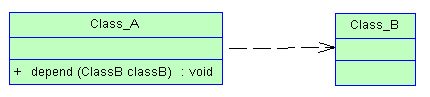


图3 依赖的UML表达

* 关联关系：关联体现的是两个类之间语义级别的一种强依赖关系，比如我和我的朋友，这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，而且双方的关系一般是平等的。关联可以是单向、双向的。表现在代码层面，为被关联类B以类的属性形式出现在关联类A中，也可能是关联类A引用了一个类型为被关联类B的全局变量。在UML类图设计中，关联关系用由关联类A指向被关联类B的带箭头实线表示，在关联的两端可以标注关联双方的角色和多重性标记，如图4所示：

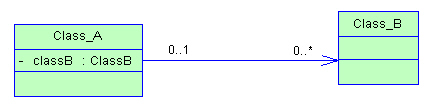


图4 关联关系的UML表达

* 聚合关系： 聚合是关联关系的一种特例，它体现的是整体与部分的关系，即has-a的关系。此时整体与部分之间是可分离的，它们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享。比如计算机与CPU、公司与员工的关系等，比如一个航母编队包括海空母舰、驱护舰艇、舰载飞机及核动力攻击潜艇等。表现在代码层面，和关联关系是一致的，只能从语义级别来区分。在UML类图设计中，聚合关系以空心菱形加实线箭头表示，如图5所示。

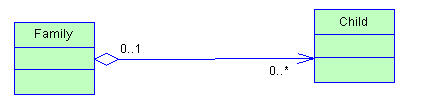


图5 聚合关系的UML表达

* 组合关系：组合也是关联关系的一种特例，它体现的是一种contains-a的关系，这种关系比聚合更强，也称为强聚合。它同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束，比如人和人的大脑。表现在代码层面，和关联关系是一致的，只能从语义级别来区分。在UML类图设计中，组合关系以实心菱形加实线箭头表示，如图6所示。

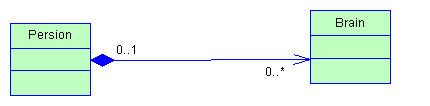


图6 聚合关系的UML表达

对于继承、实现这两种关系没多少疑问，它们体现的是一种类和类、或者类与接口间的纵向关系。其他的四种关系体现的是类和类、或者类与接口间的引用、横向关系，是比较难区分的，有很多事物间的关系要想准确定位是很难的。前面也提到，这四种关系都是语义级别的，所以从代码层面并不能完全区分各种关系，但总的来说，后几种关系所表现的强弱程度依次为：组合>聚合>关联>依赖。

# 3 实验内容

## 3.1 冒险游戏设计与实现

在下面图7中，你将看到一堆杂乱的类与接口，这是取自一个交互式的冒险游戏。你将看到代表游戏角色的类，以及武器行为的类。每个角色一次只能使用一个武器，但是可以在游戏的过程中换武器。

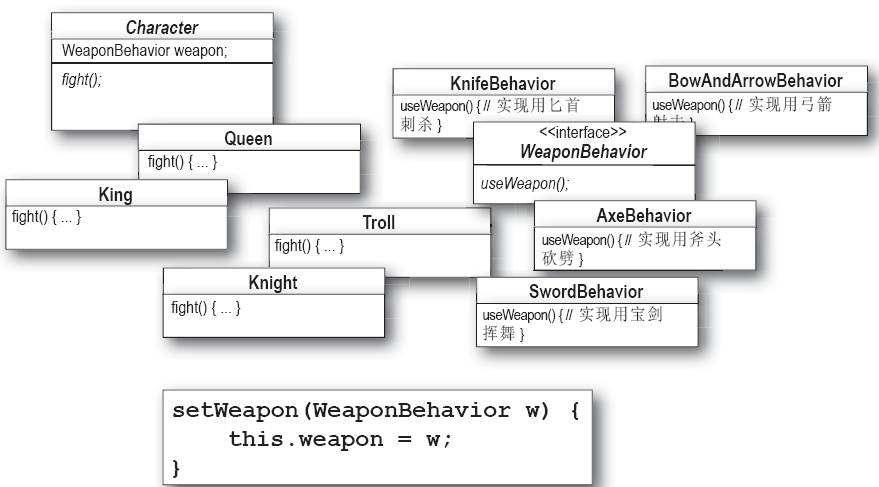


图7 某冒险游戏部分类

* 实验内容1：参考讲义相关案例，对上述的杂乱的类与接口进行整理，形成完整的设计。要求利用StarUML工具绘制出整理完毕的类图。类图绘制过程中，需要把上图中的setWeapon()方法放到上图中某类中，从而实现武器运行时更换功能，不用画客户端。绘制时特别注意abstract class和interface的绘制。类图绘制如下：

|  |
| --- |
|  |

实验内容2根据绘制的类图，实现各个类，请编写代码。调通后，粘贴在下方相应位置

Character类的代码：

|  |
| --- |
| public abstract class Character {  private WeaponBehavior weapon;  public Character(WeaponBehavior w){  setWeapon(w);  }  public abstract void fight();  public void setWeapon(WeaponBehavior w) {  this.weapon = w;  }  public WeaponBehavior getWeapon(){  return weapon;  }  } |

King类的代码

|  |
| --- |
| public class King extends Character{  public King(WeaponBehavior w){  super(w);  }  public void fight(){  System.out.print("King fight!");  getWeapon().useWeapon();  }  } |

Queen类的代码

|  |
| --- |
| public class Queen extends Character{  public Queen(WeaponBehavior w){  super(w);  }  public void fight(){  System.out.print("Queen fight!");  getWeapon().useWeapon();  }  } |

Knight类的代码

|  |
| --- |
| public class Knight extends Character{  public Knight(WeaponBehavior w){  super(w);  }  public void fight(){  System.out.print("Knight fight!");  getWeapon().useWeapon();  }  } |

Troll类的代码

|  |
| --- |
| public class Troll extends Character{  public Troll(WeaponBehavior w){  super(w);  }  public void fight(){  System.out.print("Troll fight!");  getWeapon().useWeapon();  }  } |

WeaponBehavior接口的代码

|  |
| --- |
| public interface WeaponBehavior{  public abstract void useWeapon();  } |

KnifeBehavior类的代码

|  |
| --- |
| public class KnifeBehavior implements WeaponBehavior{  public void useWeapon(){  System.out.println("Use Knife!");  }  } |

BowAndArrowBehavior类的代码

|  |
| --- |
| public class BowAndArrowBehavior implements WeaponBehavior{  public void useWeapon(){  System.out.println("Use BowAndArrow!");  }  } |

AxeBehavior类的代码

|  |
| --- |
| public class AxeBehavior implements WeaponBehavior{  public void useWeapon(){  System.out.println("Use Axe!");  }  } |

SwordBehavior类的代码

|  |
| --- |
| public class SwordBehavior implements WeaponBehavior{  public void useWeapon(){  System.out.println("Use Sword!");  }  } |

测试类的代码：

|  |
| --- |
| public class Test {  public static void main(String[] args) {  Character pig = new King(new BowAndArrowBehavior());  pig.fight();  pig.setWeapon(new KnifeBehavior());  pig.fight();  pig.setWeapon(new SwordBehavior());  pig.fight();  pig.setWeapon(new AxeBehavior());  pig.fight();    }  } |

运行结果截图：

|  |
| --- |
|  |

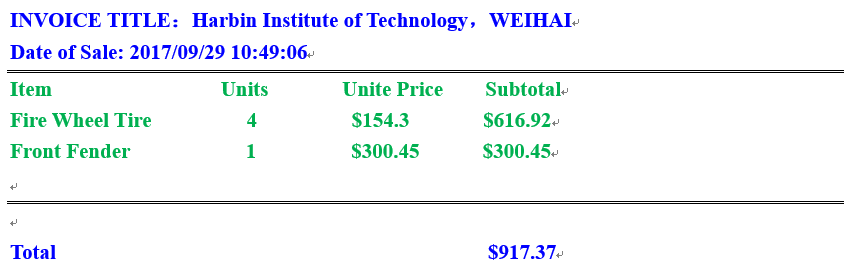
## 3.2 超市发票系统的设计与实现

**有一个电子销售系统需要打印顾客所购买的商品的发票（**Invoice）**，一张发票可以分成三个部分：**

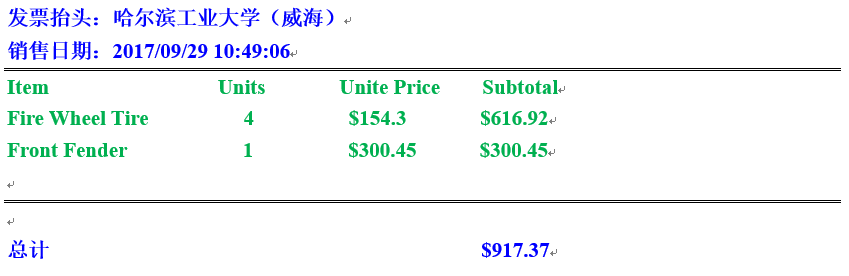
* 发票头部（Header）：上面有顾客的名字（打印时输入的），销售的日期（系统自动获取的）。
* 发票主部：销售的货物清单，包括商品的名字（Item）、购买的数量（Units）、单价（Unit Price）、小计（Sub Total）。
* 发票的尾部（Footer）：商品的总金额（Total）；

**客户可能要求中文发票或者英文发票。**下面是打印出的发票的样子（颜色是为了区分头尾和主部的，程序不必实现颜色）：

* 英文发票：



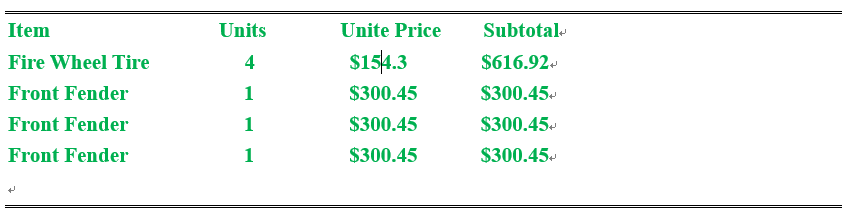
* 中文发票的样式



* 中英文结合的样式



* **需要提醒的是：**
* 为简化设计，发票的主体部分，都采用英文；
* 根据购买物品的不同，发票主体会产生不同的行，这部分必须设计成能够不断添加的，而不是如上图所显示的仅仅两项（可以在测试类中用程序语句实现各行的添加，不必采用键盘输入），除外，同一物品不必合并。例如下图的Front Fender：



* 发票的头尾可能同时是中文，也可能同时是英文，但发票的头尾同时只能使用一种语言。例如：只能头和尾同时是中文，或者同时是英文，不可能出现中文头、英文尾的混合头尾形式
* 根据业务变换，后续会添加其他多种语言的头尾，因此头尾部分必须可扩展；
* 为了进行验证，本实验要求打印中文的发票和英文的发票，以及同时打印中文头尾英文头尾的发票，因此发票头尾部分，必须设计成可组合的，例如中英，中法等等，这一部分的设计需要认真考虑。
* **实验内容1：参考讲义中《咖啡计费系统案例》，绘制出完整的类图** ，**要求必须符合OCP**

|  |
| --- |
|  |

* **实验内容2**根据绘制的类图，实现各个类，并测试。调通后，粘贴在下方相应位置

**Invoice类的代码：**

|  |
| --- |
| import java.util.ArrayList;  public abstract class Invoice {  private String name;  private String time;  private ArrayList<InvoiceLine> invoiceLineList;  public abstract void print();  public void addInvoiceLine(InvoiceLine invoiceLine) {  invoiceLineList.add(invoiceLine);  }  public void removeInvoiceLine(InvoiceLine invoiceLine) {  invoiceLineList.remove(invoiceLine);  }  public ArrayList<InvoiceLine> getInvoiceLineList() {  return invoiceLineList;  }  public double getTotal() {  double total = 0;  for (int i = 0; i < getInvoiceLineList().size(); i++) {  total += getInvoiceLineList().get(i).getSubtotal();  }  return total;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getTime() {  return time;  }  public void setTime(String time) {  this.time = time;  }  public Invoice() {  invoiceLineList = new ArrayList<InvoiceLine>();  }  } |

**InvoiceLine类的代码：**

|  |
| --- |
| public class InvoiceLine {  private String item;  private double unitPrice;  private int units;  public InvoiceLine(String item, double unitPrice, int units) {  setItem(item);  setUnitPrice(unitPrice);  setUnits(units);  }  public void printLine() {  System.out.println(getItem() + "\t\t" + getUnits() + "\t\t$" + getUnitPrice() + "\t\t\t$" + String.format("%.3f", getSubtotal()));  }  public double getSubtotal() {  return getUnitPrice() \* getUnits();  }  public String getItem() {  return item;  }  public void setItem(String item) {  this.item = item;  }  public double getUnitPrice() {  return unitPrice;  }  public void setUnitPrice(double unitPrice) {  this.unitPrice = unitPrice;  }  public int getUnits() {  return units;  }  public void setUnits(int units) {  this.units = units;  }  } |

**BodyDecorator类的代码：**

|  |
| --- |
| public abstract class BodyDecorator extends Invoice {  private Invoice invoice;  public double getTotal() {  return getInvoice().getTotal();  }  public BodyDecorator(Invoice invoice) {  setInvoice(invoice);  }  public Invoice getInvoice() {  return invoice;  }  public void setInvoice(Invoice invoice) {  this.invoice = invoice;  }  } |

**Body类的代码：**

|  |
| --- |
| public class Body extends Invoice {  public void print() {  System.out.println("Item" + "\t\t" + "Units" + "\t\t" + "Unite Price" + "\t\t" + "Subtotal" + "\t\t");  for (int i = 0; i < getInvoiceLineList().size(); i++) {  getInvoiceLineList().get(i).printLine();  }  }  } |

**ChineseStyle类的代码：**

|  |
| --- |
| public class ChineseStyle extends BodyDecorator {  public ChineseStyle(Invoice invoice) {  super(invoice);  }  public void print() {  System.out.println("发票抬头: " + getName());  System.out.println("销售日期: " + getTime());  getInvoice().print();  System.out.println("总计:" + "\t\t\t\t\t\t\t$" + String.format("%.3f", getTotal()));  }  } |

**EnglishStyle类的代码：**

|  |
| --- |
| public class EnglishStyle extends BodyDecorator {  public EnglishStyle(Invoice invoice) {  super(invoice);  }  public void print() {  System.out.println("INVOICE TITLE: " + getName());  System.out.println("Date Of Sale: " + getTime());  getInvoice().print();  System.out.println("Total:" + "\t\t\t\t\t\t\t$" + String.format("%.3f", getTotal()));  }  } |

测试类的代码：

|  |
| --- |
| import java.text.SimpleDateFormat;  import java.util.Date;  public class Client {  public static void main(String[] args) {  Invoice invoice = new Body();    invoice.addInvoiceLine(new InvoiceLine("Computer",23.5,13));  invoice.addInvoiceLine(new InvoiceLine("Coke",5.3,50));  invoice.addInvoiceLine(new InvoiceLine("Coffee",20.0,50));  String date = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(new Date());    Invoice invoice1 = new ChineseStyle(invoice);  invoice1.setName("哈工大（威海）");  invoice1.setTime(date);  invoice1.print();    Invoice invoice2 = new EnglishStyle(invoice);  invoice2.setName("HIT(WH)");  invoice2.setTime(date);  invoice2.print();    Invoice invoice3 = new EnglishStyle(invoice1);  invoice3.setName("HIT(WH)");  invoice3.setTime(date);  invoice3.print();  }  } } |

* 运行结果（1）：中文的发票头尾的发票截图

|  |
| --- |
| 发票抬头: 哈工大（威海）  销售日期: 2019/10/16 23:48:55  Item Units Unite Price Subtotal  Pig 13 $23.5 $305.500  Coke 50 $5.3 $265.000  Coffee 50 $20.0 $1000.000  总计: $1570.500 |

* 运行结果（2）：英文头发票尾的发票截图

|  |
| --- |
| INVOICE TITLE: HIT(WH)  Date Of Sale: 2019/10/16 23:48:55  Item Units Unite Price Subtotal  Pig 13 $23.5 $305.500  Coke 50 $5.3 $265.000  Coffee 50 $20.0 $1000.000  Total: $1570.500 |

* 运行结果（3）：中英文结合的样式

|  |
| --- |
| INVOICE TITLE: HIT(WH)  Date Of Sale: 2019/10/16 23:48:55  发票抬头: 哈工大（威海）  销售日期: 2019/10/16 23:48:55  Item Units Unite Price Subtotal  Pig 13 $23.5 $305.500  Coke 50 $5.3 $265.000  Coffee 50 $20.0 $1000.000  总计: $1570.500  Total: $1570.500 |