**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** | SD11100701 |
| **实验名称** | 类的关联 | | |
| **实验类型** | 验证与编程 | **实验学时** | 4学时 |
| **实验地点** | 研究院1号楼二楼中厅计算机公共机房2017.10.17 周二晚18:40开始 | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **学号** | 161110407 | **姓名** | 马宇斌 | **开始时间** | 2017.10.17 |
| **成绩** |  | | | **结束时间** | 2017.10.24 |
| **批语** |  | | | | |

目录

[1 实验目的、内容和要求 1](#_Toc494470285)

[1.1 实验目的 1](#_Toc494470286)

[1.2 实验内容 1](#_Toc494470287)

[1.3 实验要求 1](#_Toc494470288)

[2实验前准备 2](#_Toc494470289)

[3 实验内容 4](#_Toc494470290)

[3.1 冒险游戏设计与实现 4](#_Toc494470291)

[3.2 超市发票系统的设计与实现 7](#_Toc494470292)

[4 实验总结 13](#_Toc494470293)

# 1 实验目的、内容和要求

## 1.1 实验目的

程序设计编码理论上只要跑得起来就没有问题，但实际上程序是为需求服务的，需求无时无刻都有可能变更，程序也需要做出相应改变。这时如果程序设计混乱，那么大多数时候都是没有办法轻微改动程序就达到需求目标的，而此时给程序大开刀也不允许，最后进入了两难境地，所以最好一开始就依托一套优秀原则来设计程序，这就是软件设计的七原则，包括：开闭原则、里氏代换原则、依赖倒转原则、合成/聚合复用原则、迪米特法则、接口隔离原则，单一职责原则。

七原则都和接口与抽象类相关。在语法层面，它们充分的利用了接口和抽象类的灵活性；在应用层面，考虑了未来需求变更。因此遵循七原则的软件设计，在可扩展性、鲁棒性、高内聚、低耦合等方面有优秀的表现。

基于以上原因，本次实验，着重训练学生面向对象的应用，主要是类的关联关系的应用。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、同一类系中的垂直关联等类的垂直关联关系

## 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。

## 1.3 实验要求

1. 本次实验采用Eclipse快速开发工具；
2. 代码按照《Java面向对象程序设计》附录1中的Java编写规范书写；
3. 除了增删表格内的行，不允许改变本文档的页边距及其他格式；
4. 程序代码要求直接从Eclipse中拷贝代码，并粘贴到本报告相应的表格内，并调整为consolas字体，五号；不得对程序代码进行截图；
5. 采用StarUML设计时，不得截图后再粘贴到本报告中。个别同学采用mac系统，必须采用截图形式，请在报告中需要粘贴类图的位置进行说明
6. 实验结果可以截图；
7. 中文宋体五号；
8. 由于每个同学写的程序长短不一，必然导致页码的变换，因此在正式提交本文档前，需要在目录部分，右键更新目录的页码，否则会出现“未定义标签”错误。

# 2实验前准备

本次实验包含两个部分：一是SimuDuck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。

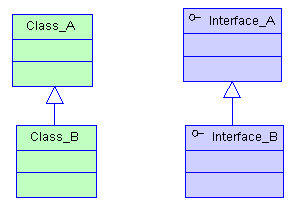


图1 继承的UML表达

* 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。

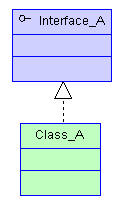


图2 实现的UML表达

* 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。

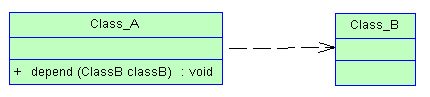


图3 依赖的UML表达

* 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：

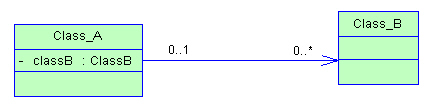


图4 关联关系的UML表达

* 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。

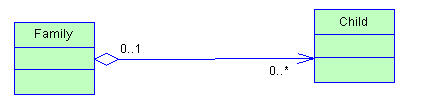


图5 聚合关系的UML表达

* 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。

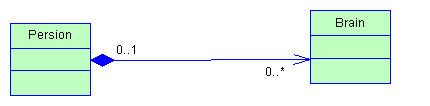


图6 聚合关系的UML表达

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。

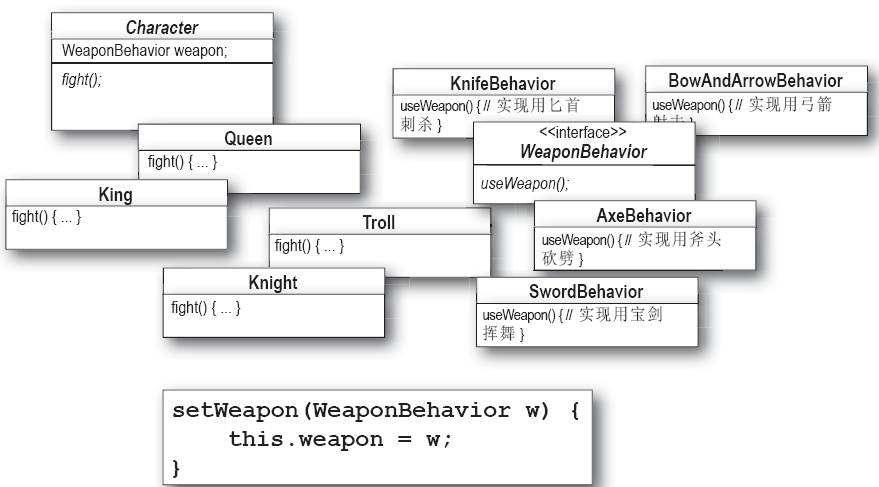


图7 某冒险游戏部分类

* **实验内容1：参考讲义中《SimuDuck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。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

**Charact类的代码：**

|  |
| --- |
| **public** **class** **Character** {  **public** WeaponBehavior weapon;  **public** **void** **fight**() {  **System**.***out***.println("这是超类的fight");  }    **public** **void** **setWeapon**(WeaponBehavior w) {  **this**.weapon=w;  }  } |

**King类的代码**

|  |
| --- |
| **public** **class** **King** **extends** **Character** {  **public** **void** **fight**() {  **System**.***out***.println("King Fight");  }  } |

**Queen类的代码**

|  |
| --- |
| **public** **class** **Queen** **extends** **Character** {  **public** **void** **fight**() {  **System**.***out***.println("Queen Fight");  }  } |

**Knight类的代码**

|  |
| --- |
| **public** **class** **Knight** **extends** **Character** {  **public** **void** **fight**() {  **System**.***out***.println("Knight Fight");  }  } |

**Troll类的代码**

|  |
| --- |
| **public** **class** **Troll** **extends** **Character** {  **public** **void** **fight**() {  **System**.***out***.println("Troll Fight");  }  } |

**WeaponBehavior接口的代码**

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

**KnifeBehavior类的代码**

|  |
| --- |
| **public** **class** **KnifeBehavior** **implements** WeaponBehavior {  **public** **void** **useWeapon**() {  **System**.***out***.println("用匕首刺杀");  }  } |

**BowAndArrowBehavior类的代码**

|  |
| --- |
| **public** **class** **BowAndArrowBehavior** **implements** WeaponBehavior {  **public** **void** **useWeapon**() {  **System**.***out***.println("用弓箭射击");  }  } |

**AxeBehavior类的代码**

|  |
| --- |
| **public** **class** **AxeBehavior** **implements** WeaponBehavior {  **public** **void** **useWeapon**(){  **System**.***out***.println("用斧头砍劈");  }  } |

**SwordBehavior类的代码**

|  |
| --- |
| **public** **class** **SwordBehavior** **implements** WeaponBehavior {  **public** **void** **useWeapon**() {  **System**.***out***.println("用宝剑挥舞");  }  } |

**测试类的代码：**

|  |
| --- |
| **public** **class** **Test** {  **public** **static** **void** **main**(**String**[] args) {  WeaponBehavior **weapon1**=**new** KnifeBehavior();  WeaponBehavior **weapon2**=**new** SwordBehavior();  WeaponBehavior **weapon3**=**new** BowAndArrowBehavior();  WeaponBehavior **weapon4**=**new** AxeBehavior();    **Character** **person1**=**new** King();  person1.setWeapon(weapon1);  person1.fight();  person1.weapon.useWeapon();  person1.setWeapon(weapon2);  person1.fight();  person1.weapon.useWeapon();  person1.setWeapon(weapon3);  person1.fight();  person1.weapon.useWeapon();  person1.setWeapon(weapon4);  person1.fight();  person1.weapon.useWeapon();  }  } |

**运行结果截图：**

|  |
| --- |
|  |

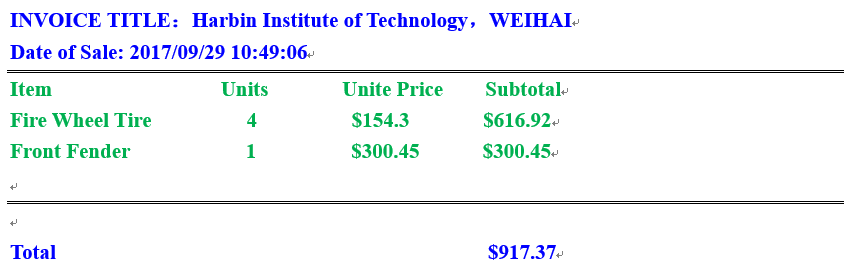
## 3.2 超市发票系统的设计与实现

**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：**

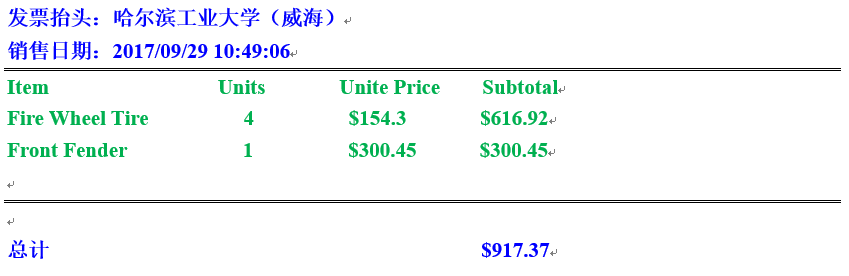
* 发票头部（Header）：上面有顾客的名字（打印时输入的），销售的日期（系统自动获取的）。
* 发票主部：销售的货物清单，包括商品的名字（Item）、购买的数量（Units）、单价（Unit Price）、小计（Sub Total）。
* 发票的尾部（Footer）：商品的总金额（Total）；

**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大致的样子（颜色是为了区分头尾和主部的，不必程序实现颜色）：

* 英文发票：



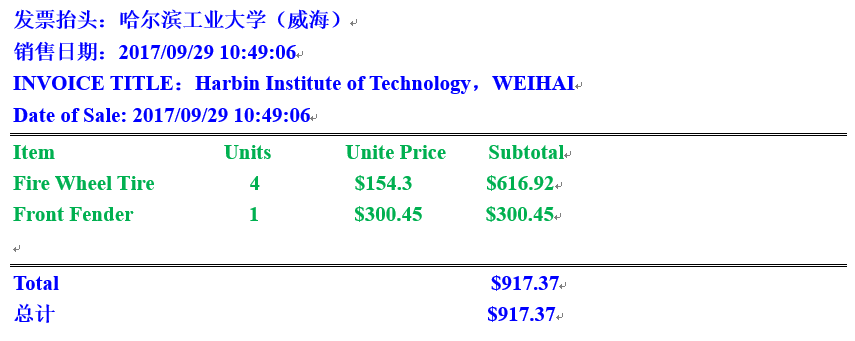
* 中文发票的样式



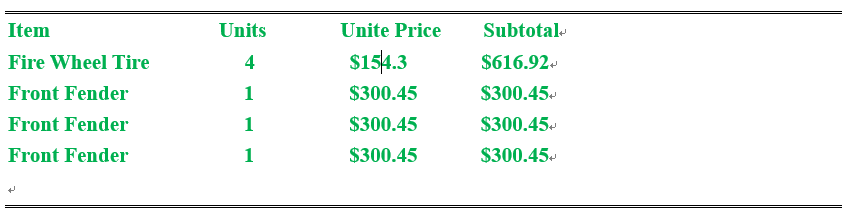
* 中英文结合的样式（1）



* 中英文结合的样式（2）



* **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。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；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，因此发票发票头尾部分，必须设计成可组合的，例如中英，英中，中法，法中，中英法等等，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

**\*\*\*类的代码：**

|  |
| --- |
| import java.text.SimpleDateFormat;  import java.util.Date;  public abstract class Invoice {  private String topic="unknow ";  private String date;  public String getTopic() {  return topic;  }  public void setTopic(String topic) {  this.topic = topic;  }  public String getDate() {  return date;  }  public void setDate(String date) {  this.date = date;  }  public void setSystemTime(Date date2){  SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//设置日期格式  this.setDate(df.format(date2));  }  public abstract void addInvoiceLine(Goods goods);  public abstract float carculatePrice();  public abstract void print();  } |

**\*\*\*类的代码：**

|  |
| --- |
| **import** java.util.ArrayList;  **public** **class** **Mainpart** **extends** **Invoice**{    **private** **ArrayList**<Goods> goods=**new** ArrayList<Goods>();  **public** **float** **carculatePrice**(){  **float** **totalprice**=0;  **for**(**Goods** **g** : goods)  {  totalprice+=g.getSubtotal();  }  **return** totalprice;  }    **public** **void** **addInvoiceLine**( **Goods** goods){  **this**.goods.add(goods);  }    **public** **void** **print**(){  **System**.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  **System**.***out***.println("\* Item Units Unite Price Subtotal");  **for**(**Goods** **g**: goods)  {  **System**.***out***.printf("\* %-30s %-20d $%-23.2f%.2f\n",g.getItem(),g.getUnits(),g.getUnitePrice(),g.getSubtotal());  }  **System**.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  }  } |

**\*\*\*类的代码：**

|  |
| --- |
| **public** **class** **Goods** {  **private** **String** item;  **private** **float** unitePrice;  **private** **int** units;  **private** **float** subtotal;  **Goods**(**String** item,**float** unitePrice, **int** units)  {  **this**.item=item;  **this**.unitePrice=unitePrice;  **this**.units=units;  setSubtotal();  }  **public** **String** **getItem**() {  **return** item;  }  **public** **float** **getUnitePrice**() {  **return** unitePrice;  }  **public** **int** **getUnits**() {  **return** units;  }  **public** **void** **setSubtotal**(){  **this**.subtotal=(unitePrice\*units);  }  **public** **float** **getSubtotal**() {  **return** **this**.subtotal;  }  } |

**\*\*\*类的代码：**

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

**\*\*\*类的代码：**

|  |
| --- |
| **public** **class** **EnglishStyle** **extends** **HeaderAndFooter**{  **EnglishStyle**(**Invoice** invoice){  **this**.setInvoice(invoice);  }  **public** **String** **getDate**(){  **return** **this**.getInvoice().getDate();  }  **public** **void** **print**(){  **System**.***out***.println("INVOICE TITLE:"+**this**.getTopic());  **System**.***out***.println("Date of Sale:"+**this**.getInvoice().getDate());  **this**.getInvoice().print();  **System**.***out***.printf("Total:"+"%92.2f\n",**this**.getInvoice().carculatePrice());  }  ***@Override***  **public** **void** **addInvoiceLine**(**Goods** goods) {  // **TODO** Auto-generated method stub  }  ***@Override***  **public** **float** **carculatePrice**() {  **return** **this**.getInvoice().carculatePrice();  }  } |

**\*\*\*类的代码：**

|  |
| --- |
| **public** **class** **ChineseStyle** **extends** **HeaderAndFooter**{    **public** **ChineseStyle**(**Invoice** invoice){  **this**.setInvoice(invoice);  }  **public** **String** **getDate**(){  **return** **this**.getInvoice().getDate();  }  **public** **void** **print**(){  **System**.***out***.println("发票抬头："+**this**.getTopic());  **System**.***out***.println("销售日期："+**this**.getInvoice().getDate());  **this**.getInvoice().print();  **System**.***out***.printf("总计："+"%91.2f\n",**this**.getInvoice().carculatePrice());  }  ***@Override***  **public** **void** **addInvoiceLine**(**Goods** goods) {  // **TODO** Auto-generated method stub  }  ***@Override***  **public** **float** **carculatePrice**() {  **return** **this**.getInvoice().carculatePrice();  }  } |

根据设计情况，自行拷贝放置代码的表格

**测试类的代码：**

|  |
| --- |
| **import** java.util.Date;  **public** **class** **Test** {  **public** **static** **void** **main**(**String**[] args) {  **Invoice** **invoice1**=**new** Mainpart();  invoice1.setTopic("哈尔滨工业大学（威海）");  invoice1.setSystemTime(**new** Date());    **Goods** **good1**=**new** Goods("Fire Wheel Tire", 154.3f, 4 );  invoice1.addInvoiceLine(good1);  **Goods** **good2**=**new** Goods("Front Fender",300.45f,1);  invoice1.addInvoiceLine(good2);    **Invoice** **invoice2**=**new** ChineseStyle(invoice1);  invoice2.setTopic("哈尔滨工业大学（威海）");  **Invoice** **invoice3**=**new** EnglishStyle(invoice2);  invoice3.setTopic("Harbin Institute of Technology");  **Invoice** **invoice4**=**new** ChineseStyle(invoice3);  invoice4.setTopic("哈尔滨工业大学（威海）");  invoice4.print();  }  } |

* **运行结果（1）：中文的发票头尾的发票截图**

|  |
| --- |
|  |

* **运行结果（2）：英文头发票尾的发票截图**

|  |
| --- |
|  |

* **运行结果（3）：中英文结合的样式（1）**

|  |
| --- |
|  |

**运行结果（4）：中英文结合的样式（2）**

|  |
| --- |
|  |

# 4 实验总结

在下面的空表中撰写本次实验的总结

|  |
| --- |
| 一．思考过程  1.刚开始做这个实验的时候，看到题目不知道从何处下手，只知道肯定有一个Goods商品类，所以先是尝试着写了一个Goods商品类  2.然后发现这种从细枝末节出发写程序的方式是不对的，然后我又再一次把书本上的咖啡计费系统装饰者模式案例看了一遍，了解了个大概之后，我在本子上画出了一个大概的类图，就是上面所截图的，当然，里面的方法还没有这么全。  3.因为发票是中文头对应中文尾，英文头对应英文尾，所以我把Header与Footer两个原本应该分开写的类写在了一起。  二．实验过程出现的困难  1.当我把整个类图的大概在纸上画了出来，但是我不知道应该添加一些什么方法将各个类图链接起来  2.在ChineseStyle与EnglishStyle中的print方法中不知道如何才能打印出上一次装饰的发票头与发票尾。  三．解决方法  1.经过思考与查阅资料，了解了如何用关联与继承把各个类之间都串联起来  2.经过询问同学，知道了采用递归调用的方式可以实现  四．总结  1.通过这次实验，我学到了当面对一个程序设计时候，要先从大局出发将整个类图画出来，然后再去添加一些类中的方法等等  2.这次设计让我更加深刻的了解了装饰者模式的算法思想，了解了如何构造类去链接装饰者与被装饰者。 |