**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 | 类间关系 | | |
| 实验类型 | 验证与编程 | 实验学时 | 2学时 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 171110516 | 姓名 | 刘宇奇 | 开始时间 | 2018.10.05 |
| 成绩 |  | | | 结束时间 | 2018.10.12 |
| 批语 |  | | | | |

目录

[1 实验目的、内容和要求 1](#_Toc526878907)

[1.1 实验目的 1](#_Toc526878908)

[1.2 实验内容 1](#_Toc526878909)

[1.3 实验要求 1](#_Toc526878910)

[2实验前准备 2](#_Toc526878911)

[3 实验内容 4](#_Toc526878912)

[3.1 冒险游戏设计与实现 4](#_Toc526878913)

[3.2 超市发票系统的设计与实现 8](#_Toc526878914)

# 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。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、垂直关联等类的垂直方向上的关系

## 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。

## 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。

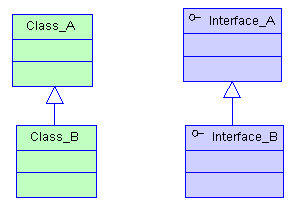


图1 继承的UML表达

* 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。

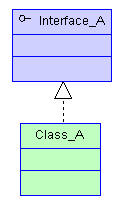


图2 实现的UML表达

* 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。

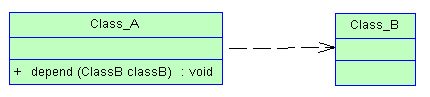


图3 依赖的UML表达

* 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：

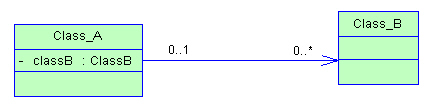


图4 关联关系的UML表达

* 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。

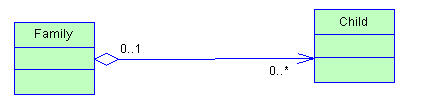


图5 聚合关系的UML表达

* 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。

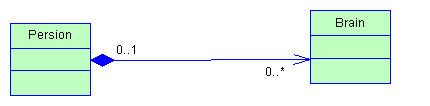


图6 聚合关系的UML表达

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。

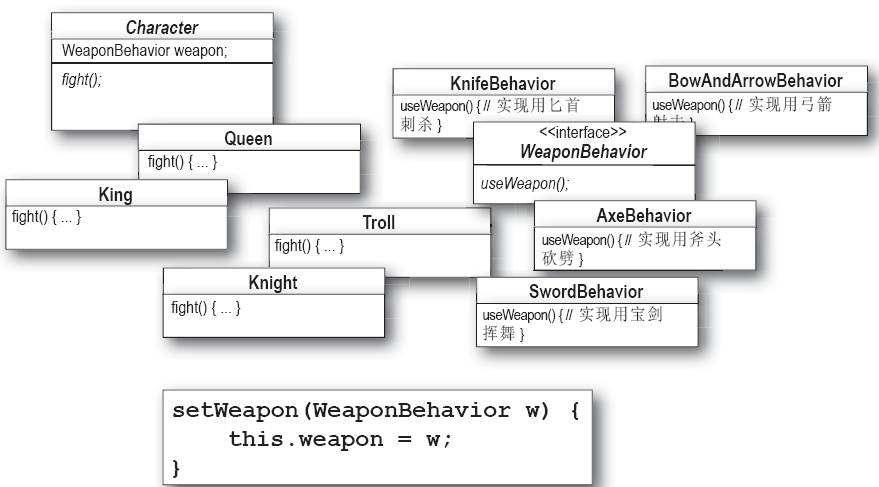


图7 某冒险游戏部分类

* **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：**

|  |
| --- |
| **package** Adventure\_Game;  **abstract** **public** **class** Character {  WeaponBehavior weapon;  **abstract** **void** fight();  **void** setWeapon(WeaponBehavior w) {  **this**.weapon = w;  }  } |

**King类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** King **extends** Character {  **public** King() {  weapon = **new** SwordBehavior();  }  **public** **void** fight() {  weapon.useWeapon();  System.***out***.println("国王在打斗！");  }  } |

**Queen类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** Queen **extends** Character {  **public** Queen() {  weapon = **new** BowAndArrowBehavior();  }  **public** **void** fight() {  weapon.useWeapon();  System.***out***.println("皇后在打斗！");  }  } |

**Knight类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** Knight **extends** Character {  **public** Knight() {  weapon = **new** KnifeBehavior();  }  **public** **void** fight() {  weapon.useWeapon();  System.***out***.println("骑士在打斗！");  }  } |

**Troll类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** Troll **extends** Character {  **public** Troll() {  weapon = **new** AxeBehavior();  }  **public** **void** fight() {  weapon.useWeapon();  System.***out***.println("巨人在打斗！");  }  } |

**WeaponBehavior接口的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **interface** WeaponBehavior {  **public** **void** useWeapon();  } |

**KnifeBehavior类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** KnifeBehavior **implements** WeaponBehavior {  **public** **void** useWeapon() {  System.***out***.println("该角色正在用匕首刺杀");  }  } |

**BowAndArrowBehavior类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** BowAndArrowBehavior **implements** WeaponBehavior {  **public** **void** useWeapon() {  System.***out***.println("该角色正在用弓箭射击");  }  } |

**AxeBehavior类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** AxeBehavior **implements** WeaponBehavior {  **public** **void** useWeapon() {  System.***out***.println("该角色正在用斧头劈砍");  }  } |

**SwordBehavior类的代码**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** SwordBehavior **implements** WeaponBehavior {  **public** **void** useWeapon() {  System.***out***.println("该角色正在用宝剑挥舞");  }  } |

**测试类的代码：**

|  |
| --- |
| **package** Adventure\_Game;  **public** **class** test\_01 {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  Queen t1 = **new** Queen();  t1.fight();  t1.setWeapon(**new** AxeBehavior());  t1.fight();  }  } |

**运行结果截图：**

|  |
| --- |
|  |

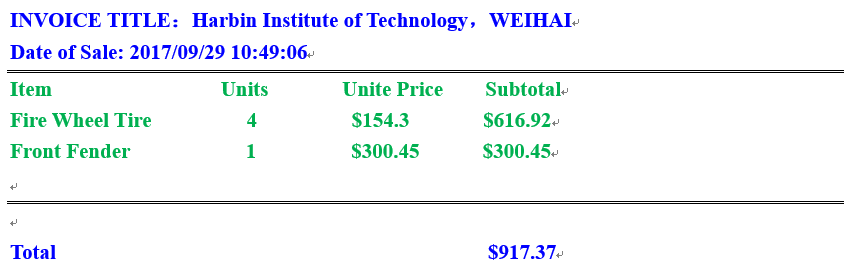
## 3.2 超市发票系统的设计与实现

**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：**

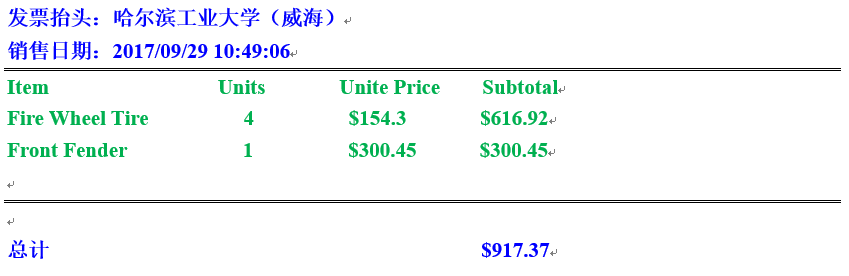
* 发票头部（Header）：上面有顾客的名字（打印时输入的），销售的日期（系统自动获取的）。
* 发票主部：销售的货物清单，包括商品的名字（Item）、购买的数量（Units）、单价（Unit Price）、小计（Sub Total）。
* 发票的尾部（Footer）：商品的总金额（Total）；

**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大致的样子（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：



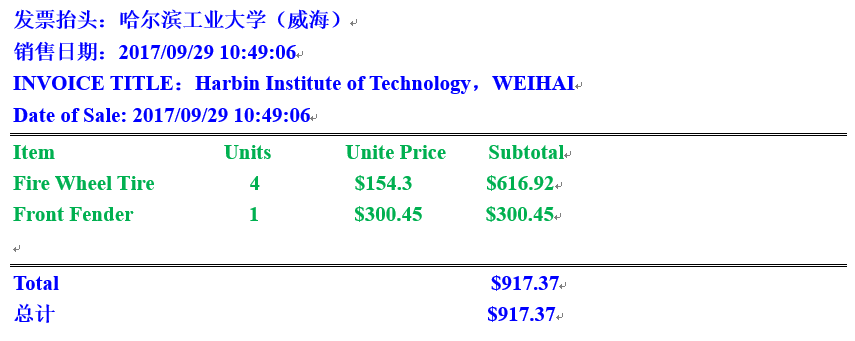
* 中文发票的样式



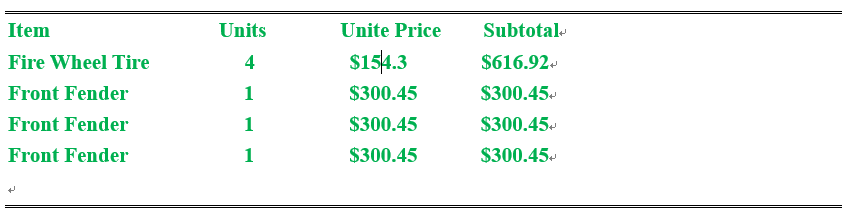
* 中英文结合的样式（1）



* 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

**Invoice类的代码：**

|  |
| --- |
| **package** invoice;  **import** java.util.Date;  **import** java.text.SimpleDateFormat;  **public** **abstract** **class** Invoice {  **private** String Title;  **private** String Date;  **private** String description = "Unknown";  **public** **void** setTitle(String s) {  Title = s;  }  **public** String getTitle() {  **return** Title;  }  **public** String getDate() {  SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//设置日期格式    **return** df.format(**new** Date());  }  **public** **void** setDescription(String d) {  **this**.description = d;  }  **public** String getDescription() {  **return** description;  }  **public** **abstract** **double** cost();  } |

**MainPart类的代码：**

|  |
| --- |
| **package** invoice;  **public** **class** MainPart **extends** Invoice {  **public** MainPart() {  **this**.setDescription("----------------------------------------------------------------------------\nItem\t\t\tUnits\t\tUnite Price\t\tSubtotal\n");  }  **public** **double** cost() {  **return** 0.0;  }  } |

**Goods类的代码：**

|  |
| --- |
| **package** invoice;  **public** **class** Goods **extends** Invoice{  **private** String item;  **private** **int** units;  **private** **double** price;  **private** **double** subtotal;  **private** Invoice invoice;  **public** Goods(Invoice invoice,String n,**int** u,**double** p){  **this**.setInvoice(invoice);  **this**.setItem(n);  **this**.setUnits(u);  **this**.setPrice(p);  }  **public** **void** setInvoice(Invoice i) {  **this**.invoice = i;  }  **public** **void** setItem(String s) {  **this**.item = s;  }  **public** **void** setUnits(**int** u) {  **this**.units = u;  }  **public** **void** setPrice(**double** p) {  **this**.price = p;  }  **public** **double** getsubtotal() {  **return** units\*price;  }  **public** **double** cost() {  **return** units\*price+**this**.invoice.cost();  }  **public** String getDescription() {  **return** **this**.invoice.getDescription()+**this**.item+"\t\t"+**this**.units+"\t\t$"+**this**.price+"\t\t\t$"+**this**.getsubtotal()+"\n";  }  } |

**HeaderAndFooter类的代码：**

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

**otherPart类的代码：**

|  |
| --- |
| **package** invoice;  **public** **class** otherPart **extends** Invoice {  **public** **double** cost() {  **return** 0.0;  }  } |

**ChineseStyle类的代码：**

|  |
| --- |
| **package** invoice;  **import** java.text.DecimalFormat;  **public** **class** ChineseStyle **extends** HeaderAndFooter {  ChineseStyle(Invoice invoice,String s){  **this**.setInvoice(invoice);  **this**.getInvoice().setTitle(s);  }  **public** **void** print() {  DecimalFormat decimalFormat=**new** DecimalFormat(".00");//构造方法的字符格式这里如果小数不足2位,会以0补足.  String p= decimalFormat.format(**this**.getInvoice().cost());//format 返回的是字符串  System.***out***.println("发票抬头："+**this**.getInvoice().getTitle());  System.***out***.println("销售日期："+**this**.getInvoice().getDate());  System.***out***.println(**this**.getInvoice().getDescription());  System.***out***.println("----------------------------------------------------------------------------");  System.***out***.println("总计\t\t\t\t\t$"+p);  }  } |

**EnglishStyle类的代码：**

|  |
| --- |
| **package** invoice;  **import** java.text.DecimalFormat;  **public** **class** EnglishStyle **extends** HeaderAndFooter {  EnglishStyle(Invoice invoice,String s){  **this**.setInvoice(invoice);  **this**.getInvoice().setTitle(s);  }  **public** **void** print() {  DecimalFormat decimalFormat=**new** DecimalFormat(".00");//构造方法的字符格式这里如果小数不足2位,会以0补足.  String p= decimalFormat.format(**this**.getInvoice().cost());//format 返回的是字符串  System.***out***.println("INVOICE TITLE："+**this**.getInvoice().getTitle());  System.***out***.println("Date of Sale："+**this**.getInvoice().getDate());  System.***out***.println(**this**.getInvoice().getDescription());  System.***out***.println("----------------------------------------------------------------------------");  System.***out***.println("Total\t\t\t\t\t$"+p);  }  } |

**EnglishAndChinese类的代码：**

|  |
| --- |
| **package** invoice;  **import** java.text.DecimalFormat;  **public** **class** EnglishAndChinese **extends** HeaderAndFooter {  **private** String title1;  **private** String title2;  EnglishAndChinese(Invoice invoice,String s1,String s2){  **this**.setInvoice(invoice);  title1 = s1;  title2 = s2;  }  **public** **void** print() {  DecimalFormat decimalFormat=**new** DecimalFormat(".00");//构造方法的字符格式这里如果小数不足2位,会以0补足.  String p= decimalFormat.format(**this**.getInvoice().cost());//format 返回的是字符串  System.***out***.println("发票抬头："+**this**.title2);  System.***out***.println("销售日期："+**this**.getInvoice().getDate());  System.***out***.println("INVOICE TITLE："+**this**.title1);  System.***out***.println("Date of Sale："+**this**.getInvoice().getDate());  System.***out***.println(**this**.getInvoice().getDescription());  System.***out***.println("----------------------------------------------------------------------------");  System.***out***.println("总计\t\t\t\t\t$"+p);  System.***out***.println("Total\t\t\t\t\t$"+p);  }  } |

**测试类的代码：**

|  |
| --- |
| **package** invoice;  **public** **class** test {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  Invoice invoice = **new** MainPart();  invoice = **new** Goods(invoice,"Fire Wheel Tire",4,154.3);  invoice = **new** Goods(invoice,"Front Fender",1,300.45);  ChineseStyle test1 = **new** ChineseStyle(invoice,"哈尔滨工业大学");  Test1.print();  EnglishStyle test2 = **new** EnglishStyle(invoice,"Harbin Institute of Technology");  test2.print();  EnglishAndChinese test3 = **new** EnglishAndChinese(invoice,"HIT","哈工大");  test3.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）**

|  |
| --- |
|  |