Java程序设计·实验指导

董玉涛

天津理工大学

JAVA程序设计

目录

[1、试卷分析软件设计（第1次实验，2+4学时） 2](#_Toc500034794)

[2、工资管理软件设计（第2次实验，3+6学时） 3](#_Toc500034795)

[3、银行业务管理软件（第3次实验，3+6学时） 5](#_Toc500034796)

[4、打印机管理器模拟（第4次实验，4+8学时） 7](#_Toc500034797)

### 1、试卷分析软件设计（第1次实验，2+4学时）

使用if语句、while语句等基本语言要素设计一个Java程序，帮助教师进行试卷分析。该程序必须能：

* 从键盘接受一系列成绩
* 计算平均分
* 确定最高和最低分
* 显示平均、最高、最低分
* 确定每个分数档（每10分为一档，不及格为一档）的人数和百分比
* 显示每个分数档的人数和百分比

### 2、工资管理软件设计（第2次实验，3+6学时）

某公司由6人组成，他们分别是：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号**  **No.** | **姓名**  **name** | **地 址**  **address** | **电 话**  **phone** | **职 务**  **title** | **社会安全号**  **socialSN** | **基本工资**  **payRate** |
| 1 | Sam | 123 Main Line | 555-0469 | manager | 123-45-6789 | $2423.07 |
| 2 | Peter | 456 Off Line | 555-0101 | employee | 987-65-4321 | $1246.15 |
| 3 | Mary | 789 Off Rocker | 555-0690 | employee | 010-20-3040 | $1169.23 |
| 4 | Cliff | 678 Fifth Ave. | 555-0000 | hourly | 958-47-3625 | $10.55 |
| 5 | Al | 987 Suds Ave. | 555-8374 | volunteer | 无 | $0.00 |
| 6 | Gus | 321 Off Line | 555-7282 | volunteer | 无 | $0.00 |

其中，Sam是经理，每月除得到基本工资外，还获得$500.00的红利；Peter和Mary是合同工，每月拿基本工资；Cliff是小时工，他的每月的工资额=基本工资×工作小时数，他的工作小时数为每月40小时；而Al和Gus是义工，不拿工资。图1反应了上述情况。

*StaffMember*是抽象类，有一个抽象方法*pay*。

Manager类中的bonus代表红利；awardBonus方法用来增加红利。

Hourly类中的hoursWorked代表一个月工作的小时数；addHours方法用来增加小时数。

Staff类中的main方法用来驱动整个程序的运行；payDetail方法用来显示所有人的详细信息；构造方法用来初始化上述6个人，并将他们存储在staffMember类型的数组中。

每个类都有toString方法，用来返回该类的详细信息。要求你按照此图的继承关系定义每一个类，并最终显示所有人的详细信息。显示结果如下：

姓名：Sam

地址：123 Main Line

电话：555-0469

社会安全号：123-45-6789

工资：$2923.07

-----------------------------------------

姓名：Peter

地址：456 Off Line

电话：555-0101

社会安全号：987-65-4321

工资：$1246.15

-----------------------------------------

姓名：Mary

地址：789 Off Rocker

电话：555-0690

社会安全号：010-20-3040

工资：$1169.23

-----------------------------------------

姓名：Cliff

地址：678 Fifth Ave.

电话：555-0000

社会安全号：958-47-3625

工作小时数：40

工资：$422.00

-----------------------------------------

姓名：Al

地址：987 Suds Ave.

电话：555-8374

多谢！

-----------------------------------------

姓名：Gus

地址：321 Off Line

电话：555-7282

多谢！

-----------------------------------------



图1

### 3、银行业务管理软件（第3次实验，3+6学时）

某银行有许多储户，每个储户可以拥有一个或多个帐号。现要求你开发一个软件，管理银行储户与帐号业务。见图2

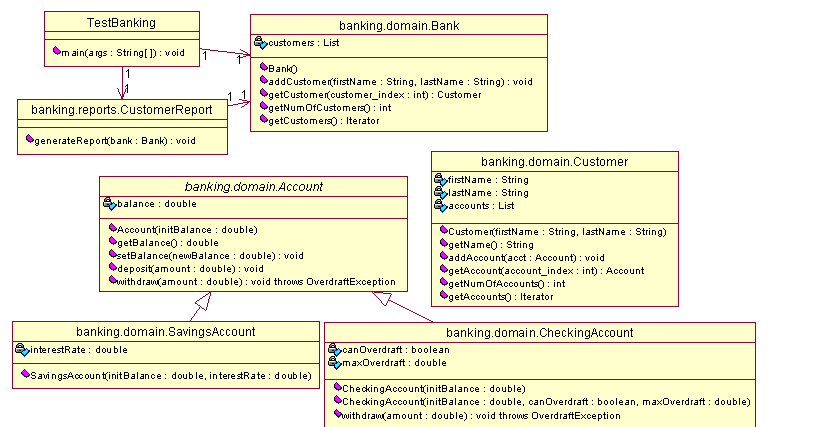


图2

Bank类中有一个customers集合，用来存储所有的Customer（储户）对象，addCustomer方法用于向该集合中加储户，getCustomer方法根据下标值取某个储户，getNumOfCustomers方法取储户总数，getCustomers方法返回储户的Iterator，以便获得每个储户对象。

Customer类有一个accounts集合，用来存储某个储户的所有Account（帐号）对象，addAccount方法用于向该集合中加帐号，getAccount方法根据下标值取该储户的某个帐号，getNumOfAccounts方法取该储户的帐号总数，getAccounts方法返回该储户的帐号的Iterator，以便获得每个帐号对象。

*Account*类是抽象类，有一个balance属性，代表余额。deposit方法表示存款，amount参数是存款额。withdraw方法表示取款，取款额amount如果超出了余额就会抛出透支异常，我们需要自己定义一个OverdraftException类来表示这个异常，当抛出透支异常时，不进行取款交易，并报告用户此项错误。

SavingsAccount类表示存款帐号，继承*Account*，新增一个属性interestRate, 代表利率。

CheckingAccount类表示大额存款帐号，也继承*Account*，它有一个叫canOverdraft的属性，是一个boolean值，代表该帐号能否透支（true－能，false－否）；它还有一个叫maxOverdraft的属性，表示该帐号允许的最大透支额。这个类的withdraw（取款）方法需要考虑的因素比较多：在发生透支时，如果帐号不允许透支（canOverdraft=false），则抛出OverdraftException异常并退出交易；如果允许透支（canOverdraft=true），但透支额（amount-balance）超过最大透支额的话，也抛出OverdraftException异常并退出交易；只有在不发生透支或透支额小于最大透支额时交易才能正常进行。另外，在每次进行透支交易时，最大透支额(maxOverdraft)应做调整，以便使该帐号的最大透支额随透支次数的增加而不断减少，从而可以避免透支的滥用，阻止信用膨胀。CheckingAccount类有两个构造方法，只带一个参数的构造方法用来初始化balance，同时设定canOverdraft=false，maxOverdraft=0.00。

CustomerReport类用来显示每个储户的姓名以及他所持有的帐号的类别和余额，以报表的形式输出。

根据以上描述，创建一个TestBanking类，并在其main方法中添加若干储户和帐号，然后模拟存款、取款业务，并最终输出一张完整的报表。

采用下表中的数据进行模拟：

|  |  |  |
| --- | --- | --- |
| **储户姓名** | **帐 号** | |
| **Savings Account** | **Checking Account** |
| Jane Simms | SavingsAccount (500.00, 0.05) | CheckingAccount (200.00, true, 400.00) |
| Owen Bryant | 无 | CheckingAccount (200.00) |
| Tim Soley | SavingsAccount (1500.00, 0.05) | CheckingAccount (300.00) |
| Maria Soley | SavingsAccount (150.00, 0.05) | 与Tim Soley共享一个CheckingAccount |

输出报表格式如下：

CUSTOMERS REPORT

==================

储户姓名：Jane Simms

Savings Account：当前余额是￥XXXX

Checking Account：当前余额是￥XXXX

储户姓名：Owen Bryant

Checking Account：当前余额是￥XXXX

储户姓名：Tim Soley

Savings Account：当前余额是￥XXXX

Checking Account：当前余额是￥XXXX

储户姓名：Maria Soley

Savings Account：当前余额是￥XXXX

Checking Account：当前余额是￥XXXX

### 4、打印机管理器模拟（第4次实验，4+8学时）

#### 目标

本实验要求你使用wait/notify和其他线程控制方法创建一个打印机管理器（printer manager）。要求你创建几个并发线程产生打印作业，一个打印管理器线程处理这些作业。

#### 指导

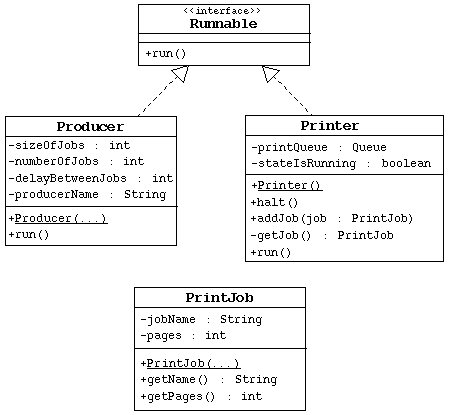


图3

##### 定义PrintJob类

1. 基于上述UML类图定义PrintJob类，代表一个打印作业。

##### 定义Producer类

1. 基于上述UML类图定义Producer类。它必须实现Runnable接口。
2. 写一个构造方法初始化4个实例变量。
3. run方法必须创建数量为numberOfJobs的打印作业（PrintJob对象）。根据producerName属性和作业号给每个作业起一个作业名（即jobName=producerName#N）。每个打印作业的页数pages由sizeOfJobs 属性确定。
4. 用Printer对象的addJob方法把打印作业加到打印机管理器的打印队列中去。该方法可能抛出FullQueueException异常（队列满异常），要求编程时处理这种异常情况，直至队列非满为止，然后把作业加到队列中去。
5. 在相邻两次打印期间，producer睡眠delayBetweenJobs长时间。

##### 定义Printer类

1. 基于上述UML类图定义Printer类。
2. 写构造方法初始化属性。打印队列大小为5，即最多能容纳5个作业。
3. printQueue属性是一个FIFO队列，它包含所有的被提交的打印作业。Queue是一个接口，代表FIFO队列。CircularQueue实现该接口。FullQueueException和EmptyQueueException异常分别可以由addBack和removeFront方法抛出。
4. 写public、synchronized方法addJob。该方法可能抛出FullQueueException异常。在该方法中，加一个作业以后要通知Printer对象可以继续打印。
5. 写private、synchronized方法getJob。该方法可能抛出EmptyQueueException异常。
6. 写public、synchronized方法halt（停止）。该方法设置stateIsRunning属性为false。StateIsRunning属性代表打印机是否正在运行。
7. 写run方法。它是打印管理器的核心。它始终处于连续循环之中，直到stateIsRunning=false为止。它从队列中取打印作业，输出Starting job ‘作业名’，然后处理作业（睡眠一段时间，500毫秒/页×页数），然后打印Completed job ‘作业名’。如果队列中没有作业，输出Waiting on a job to print并等待。当打印机被halt后，应先处理剩余作业，然后输出Print manager is halted并退出run方法。
8. 本题最困难的部分是wait和notify的使用。

##### 创建TestPrinter程序

1. 这个类只包含main方法。
2. 让main方法创建3个Producer对象，它们有不同的打印特征：作业大小（5页－25页），作业间的延迟（2秒－10秒）。下面是模拟用的三个Producer对象：

|  |  |  |  |
| --- | --- | --- | --- |
| **producerName** | **numberOfJobs** | **sizeOfJobs** | **delayBetweenJobs** |
| Fred | 5 | 5 | 8秒 |
| Elizabeth | 3 | 25 | 20秒 |
| Simon | 3 | 5 | 10秒 |

1. 为打印机管理器创建一个线程。设置它的优先级为最大，然后启动该线程。
2. 为每个producer创建线程，然后启动该线程。
3. 用join方法等待每个producer线程结束。
4. 最后，停止打印机管理器。

下面是一个可能的输出结果：

C: Print manager is starting up.

C: Waiting on a job to print.

P: Adding job 'Fred#1' to the queue

C: Starting job 'Fred#1'

P: Adding job 'Elizabeth#1' to the queue

P: Adding job 'Simon#1' to the queue

C: Completed job 'Fred#1'

C: Starting job 'Elizabeth#1'

P: Adding job 'Fred#2' to the queue

P: Adding job 'Simon#2' to the queue

C: Completed job 'Elizabeth#1'

C: Starting job 'Simon#1'

P: Adding job 'Fred#3' to the queue

C: Completed job 'Simon#1'

C: Starting job 'Fred#2'

P: Adding job 'Elizabeth#2' to the queue

P: Adding job 'Simon#3' to the queue

C: Completed job 'Fred#2'

C: Starting job 'Simon#2'

C: Completed job 'Simon#2'

C: Starting job 'Fred#3'

P: Adding job 'Fred#4' to the queue

C: Completed job 'Fred#3'

C: Starting job 'Elizabeth#2'

P: Adding job 'Fred#5' to the queue

C: Completed job 'Elizabeth#2'

C: Starting job 'Simon#3'

P: Adding job 'Elizabeth#3' to the queue

C: Completed job 'Simon#3'

C: Starting job 'Fred#4'

C: Completed job 'Fred#4'

C: Starting job 'Fred#5'

C: Completed job 'Fred#5'

C: Starting job 'Elizabeth#3'

C: Completed job 'Elizabeth#3'

C: Waiting on a job to print.

C: Print manager is halted.