尚硅谷Java基础实战—Bank项目

【简　介】

《Bank项目》是尚硅谷版权的 "Java 基础阶段" 代码实战之一。该项目在讲授 JavaSE 时以讲师引导、学员实战的方式完成。同时，此项目也可作为Java从业人员、Java自学者自测的经典项目！

【项目特点】

1. **更多技术涵盖：**由 8 组由浅入深的模块构成，应用如下技术：面向对象的封装性、构造器、引用类型的成员变量、异构数组、继承、多态、方法的重载、方法的重写、包装类、单子模式、异常、集合。

2. **涉及较复杂业务：**以银行业务为背景，包含：添加客户，创建异构账户、存钱、取钱、透支保护等业务。

3. **类之间多重引用、依赖关系：**该项目包含 Bank、Customer、Account、SavingsAccount、CheckingAccount、OverdraftException、CustomerReport、TestBanking 等 8 个类构成，Bank 和 Customer、Customer 和 Account(SavingsAccount、CheckingAccount)、CheckingAccount 和 OverdraftException、CustomerReport 和 Bank 及 TestBanking 之间以方法参数、成员变量的方式建立引用、依赖关系。

【项目需求】

Bank项目需求下载地址：

http://www.atguigu.com/download.shtml

【项目源代码】

Bank项目源代码下载地址：

http://www.atguigu.com/download.shtml

【项目视频】

尚硅谷佟刚老师全程讲解Bank项目 8 个子模块。详见下载地址：

http://www.atguigu.com/download.shtml

尚硅谷，鼓励每一位Java爱好者，自学成才。网站提供更多、更全面、更细致、更新的学习资料。同时，提供免费在线答疑！

尚硅谷讲师，时刻伴你同行！www.atguigu.com

实验题目 1:

创建一个简单的银行程序包

实验目的:

Java 语言中面向对象的封装性及构造器的创建和使用。

实验说明:

在这个练习里, 创建一个简单版本的 Account 类。将这个源文件放入 banking 程序包中。在创建单个帐户的默认程序包中, 已编写了一个测试程序 TestBanking。这个测试程序初始化帐户余额, 并可执行几种简单的事物处理。最后, 该测试程 序显示该帐户的最终余额。

提示:

1. 创建 banking 包

2. 在 banking 包下创建 Account 类。该类必须实现上述 UML 框图中的模型。
   a. 声明一个私有对象属性:balance, 这个属性保留了银行帐户的当前(或 即时)余额。
   b. 声明一个带有一个参数(init_balance)的公有构造器, 这个参数为 balance 属性赋值。
   c. 声明一个公有方法 geBalance, 该方法用于获取经常余额。
   d. 声明一个公有方法 deposit,该方法向当前余额增加金额。
   e. 声明一个公有方法 withdraw 从当前余额中减去金额。

3. 打开TestBanking.java文件，按提示完成编写，并编译 TestBanking.java 文件。

4. 运行 TestBanking 类。可以看到下列输出结果:

```
Creating an account with a 500.00 balance
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
The account has a balance of 324.88
```

实验题目 2:

扩展银行项目，添加一个 Customer 类。Customer 类将包含一个 Account对象。

实验目的:

使用引用类型的成员变量。

提 示:

1. 在banking包下的创建Customer类。该类必须实现上面的UML图表中的模型。

a. 声明三个私有对象属性:firstName、lastName 和 account。
b. 声明一个公有构造器，这个构造器带有两个代表对象属性的参数(f 和 l)
c. 声明两个公有存取器来访问该对象属性，方法 getFirstName 和 getLastName 返
回相应的属性。
d. 声明 setAccount 方法来对 account 属性赋值。
e. 声明 getAccount 方法以获取 account 属性。

2. 在 exercise2 主目录里，编译运行这个 TestBanking 程序。应该看到如下输出结果:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
Customer [Smith, Jane] has a balance of 324.88
```

实验题目 3:

修改 withdraw 方法以返回一个布尔值，指示交易是否成功。

实验目的:

使用有返回值的方法。

提 示:

1. 修改 Account 类
   a. 修改 deposit 方法返回 true(意味所有存款是成功的)。
   b. 修改 withdraw 方法来检查提款数目是否大于余额。如果amt小于balance, 则从余额中扣除提款数目并返回 true, 否则余额不变返回false。

2. 在 exercise3 主目录编译并运行 TestBanking 程序, 将看到下列输出;

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Smith, Jane] has a balance of 324.88
```

实验题目 4:

将用数组实现银行与客户间的多重关系。

实验目的:

在类中使用数组作为模拟集合操作。

提示:

对银行来说, 可添加 Bank 类。 Bank 对象跟踪自身与其客户间的关系。用 Customer 对象的数组实现这个集合化的关系。还要保持一个整数属性来跟踪银 行当前有多少客户。

a. 创建 Bank 类

b. 为 Bank 类增加两个属性：customers(Customer对象的数组) 和 numberOfCustomers(整数, 跟踪下一个 customers 数组索引)

c. 添加公有构造器, 以合适的最大尺寸(至少大于 5)初始化 customers 数组。

d. 添加 addCustomer 方法。该方法必须依照参数(姓, 名)构造一个新的 Customer
对象然后把它放到 customer 数组中。还必须把 numberofCustomers 属性的值加 1。

e. 添加 getNumOfCustomers 访问方法, 它返回 numberofCustomers 属性值。

f. 添加 getCustomer方法。它返回与给出的index参数相关的客户。

g. 编译并运行 TestBanking 程序。可以看到下列输出结果:
Customer [1] is Simms,Jane
Customer [2] is Bryant,Owen
Customer [3] is Soley,Tim
Customer [4] is Soley,Maria

实验题目 5:

在银行项目中创建 Account 的两个子类:SavingAccount 和 CheckingAccount

实验目的:

继承、多态、方法的重写。

提 示:

创建 Account 类的两个子类:SavingAccount 和 CheckingAccount 子类

a. 修改 Account 类;将 balance 属性的访问方式改为 protected

b. 创建 SavingAccount 类, 该类继承 Account 类

c. 该类必须包含一个类型为 double 的 interestRate 属性

d. 该类必须包括带有两个参数(balance 和 interest_rate)的公有构造器。该构 造器必须通过调用 super(balance)将 balance 参数传递给父类构造器。

实现 CheckingAccount 类

1. CheckingAccount 类必须扩展 Account 类

2. 该类必须包含一个类型为 double 的 overdraftProtection 属性。

3. 该类必须包含一个带有参数(balance)的共有构造器。该构造器必须通过调用 super(balance)将 balance 参数传递给父类构造器。

4. 给类必须包括另一个带有两个参数(balance 和 protect)的公有构造器。该构造器必须通过调用 super(balance)并设置 overdragtProtection 属性, 将 balance 参数传递给父类构造器。

5. CheckingAccount 类必须覆盖 withdraw 方法。此方法必须执行下列检查。如 果当前余额足够弥补取款 amount,则正常进行。如果不够弥补但是存在透支 保护, 则尝试用 overdraftProtection 得值来弥补该差值(balance-amount). 如果弥补该透支所需要的金额大于当前的保护级别。则整个交易失败, 但余 额未受影响。

6. 在主 exercise1 目录中, 编译并执行 TestBanking 程序。输出应为:
Creating the customer Jane Smith.
Creating her Savings Account with a 500.00 balance and 3% interest.

Creating the customer Owen Bryant.
Creating his Checking Account with a 500.00 balance and no overdraft protection.
Creating the customer Tim Soley.
Creating his Checking Account with a 500.00 balance and 500.00 in overdraft protection.
Creating the customer Maria Soley.
Maria shares her Checking Account with her husband Tim.

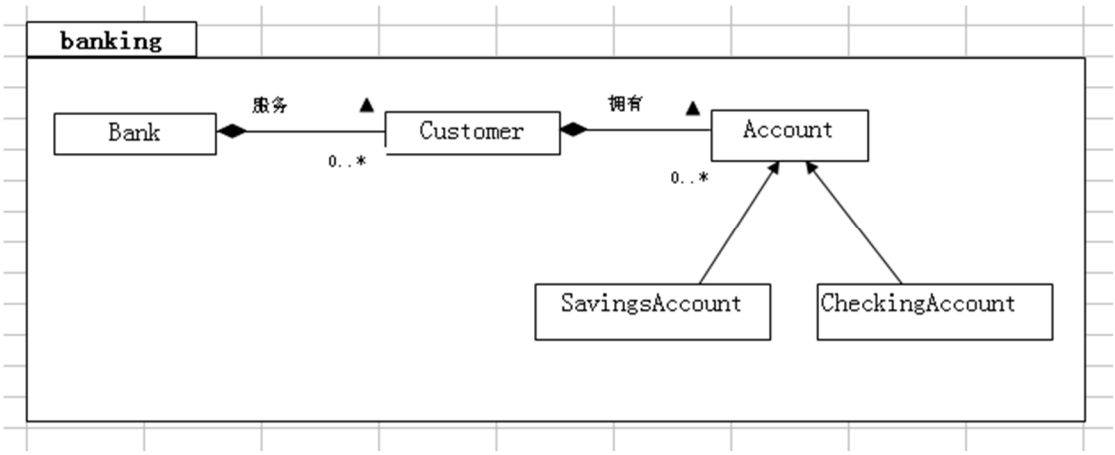
Retrieving the customer Jane Smith with her savings account.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Simms, Jane] has a balance of 324.88

Retrieving the customer Owen Bryant with his checking account with no overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Bryant, Owen] has a balance of 324.88

Retrieving the customer Tim Soley with his checking account that has overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: true
Customer [Soley, Tim] has a balance of 0.0

Retrieving the customer Maria Soley with her joint checking account with husband Tim.
Deposit 150.00: true
Withdraw 750.00: false
Customer [Soley, Maria] has a balance of 150.0

创建客户账户



修改 Customer 类

1. 修改 Customer 类来处理具有多种类型的联合账户。(例如用数组表示多重性一节所作的，该类必须包括以下的公有方法：addAccount(Account),getAccount(int)和 getNumOfAccount)
2. 完成 TestBanking 程序：该程序创建一个客户和账户的集合，并生成这些客户及其账户余额的报告。在 TestBanking。Java 文件中，你会发现注释块以/***…***/来开头和结尾。这些注释只是必须提供的代码的位置。
3. 使用 instanceof 操作符测试拥有的账户类型,并且将 account_type 设置为适当的值，例如:“Savings Account”或“Checking Account”。
4. 编译并运行该程序，将看到下列结果

CUSTOMERS REPORT

==========

Customer: Simms, Jane

Savings Account: current balance is ￥500.00

Checking Account: current balance is ￥200.00

Customer: Bryant, Owen

Checking Account: current balance is ￥200.00

Customer: Soley, Tim

Savings Account: current balance is ￥1,500.00

Checking Account: current balance is ￥200.00

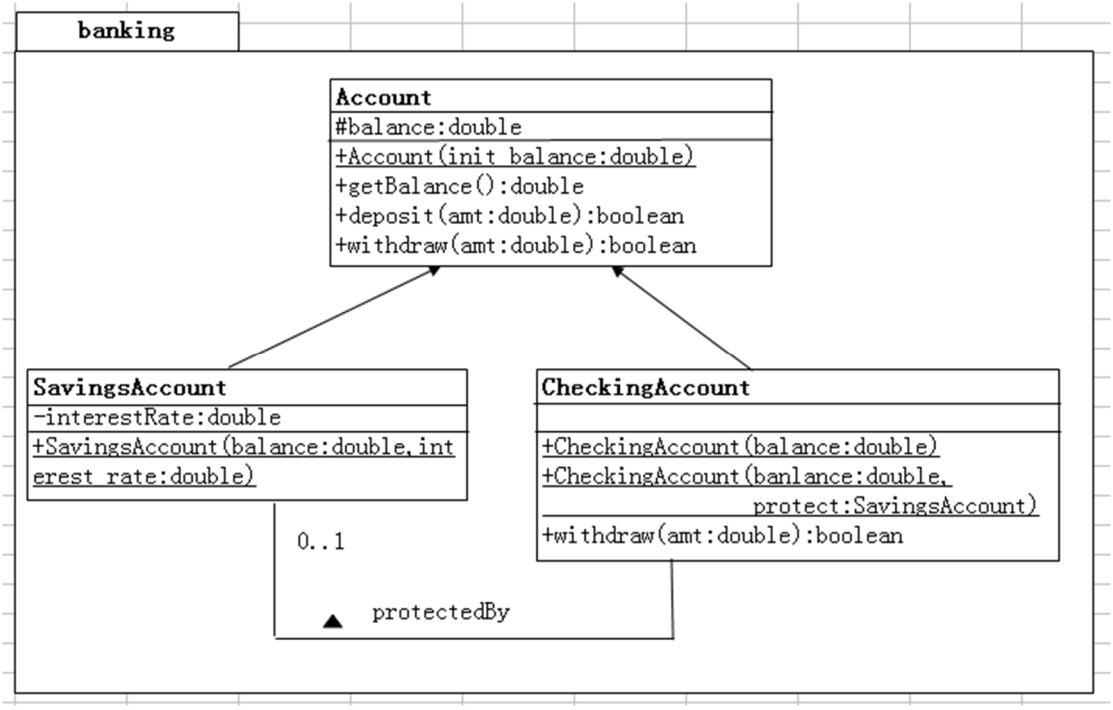
Customer: Soley, Maria

Checking Account: current balance is ￥200.00

Savings Account: current balance is ￥150.00

实现更为复杂的透支保护机制

注释-这是练习 1 的选择练习。它包括了更为复杂的透支保护机制模型。它使用客户储蓄。它使用客户储蓄账户完成透支保护。本练习必须在完成上述连个练习后进行



1．仿照练习 1“Account 类的两个子类”一节实现 SavingsAccount 类。

2．SavingsAccount 类必须扩展 Account 类。

3．该类必须包含一个类型为 double 的 interestRate 属性

4．该类必须包括一个带有两个参数(balance 和 interest_rate)的公有构造器。该构造器必须通过调用 super（balance）来将 balance 参数传递给父类构造器

5．仿照练习 1“Account 类的两个子类”一节实现 CheckingAccount 类。

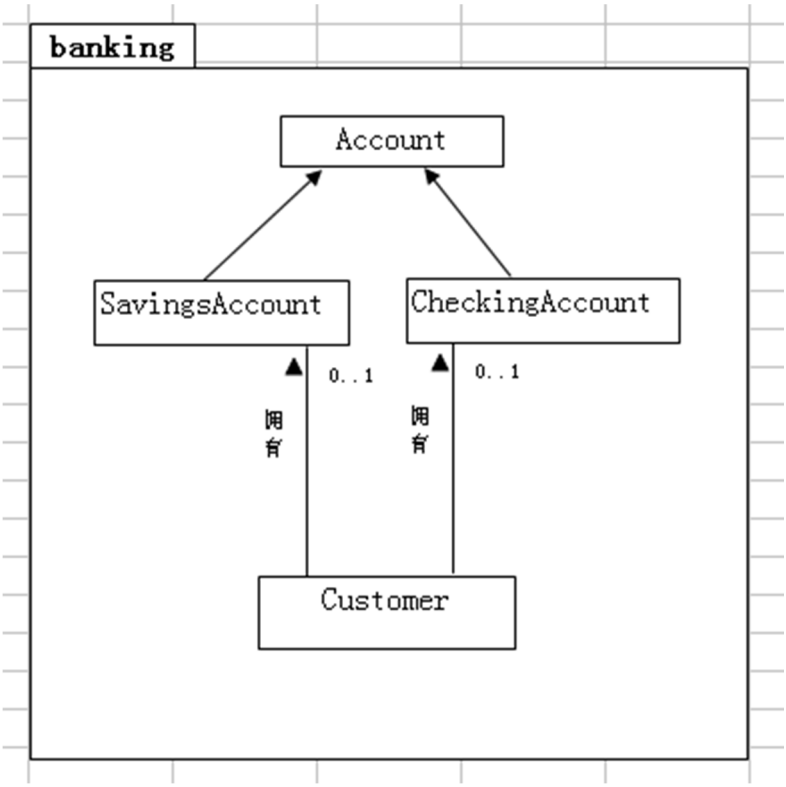
6．CheckingAccount 类必须扩展 Account 类

7．该类必须包括一个关联属性，称作 protectedBy，该属性的类型为 SavingsAccount，缺省值必须是 null。除此之外该类没有其他的数据属性

8．该类必须包括一个带有参数（balance）的公有构造器，该构造器必须通过调用 super（balance）将 balance 参数传递到父类构造器

9．该类必须包括另外一个带有两个参数的（balance 和 protect）的公有构造器。该构造器必须通过调用 super（balance）将 balance 参数传递给父类构造器。

10． CheckingAccount 类必须覆盖 withdraw 方法。该类必须执行下面的检查：如果当前余额足够弥补取款 amount，则正常进行。如果不够弥补但是存在透支保护则尝试用储蓄账户来弥补这个差值（balance-amount）。如果后一个交易失败，则整个交易一定失败，但余额未受影响。



修改客户使其拥有两个账户

修改 Customer 类使其拥有两个银行账户：一个储蓄账户和一个支票账户：两个都是可选的。

11．　原来的 Customer 类包含一个称作 account 的关联属性,可用该属性控制一个 Account 对象。重写这个类，使其包含两个关联属性：savingsAccount 和 checkingAccount，这两个属性的缺省值是 null

12．　包含两个访问方法：getSavings 和 getChecking，这两个方法分别返回储蓄和支票总数。

13．　包含两个相反的方法：SetSavings 和 setChecking，这两个方法分别为 savings 和 checking account 这两个关联属性赋值。

14．　编译执行 TestBanking 程序。输出应为：

Customer [Simms, Jane] has a checking balance of 200.0 and a savings balance of

500.0

Checking Acct [Jane Simms] : withdraw 150.00 succeeds? true

Checking Acct [Jane Simms] : deposit 22.50 succeeds? true

Checking Acct [Jane Simms] : withdraw 147.62 succeeds? true

Customer [Simms, Jane] has a checking balance of 0.0 and a savings balance of 42

4.88

Customer [Bryant, Owen] has a checking balance of 200.0

Checking Acct [Owen Bryant] : withdraw 100.00 succeeds? true

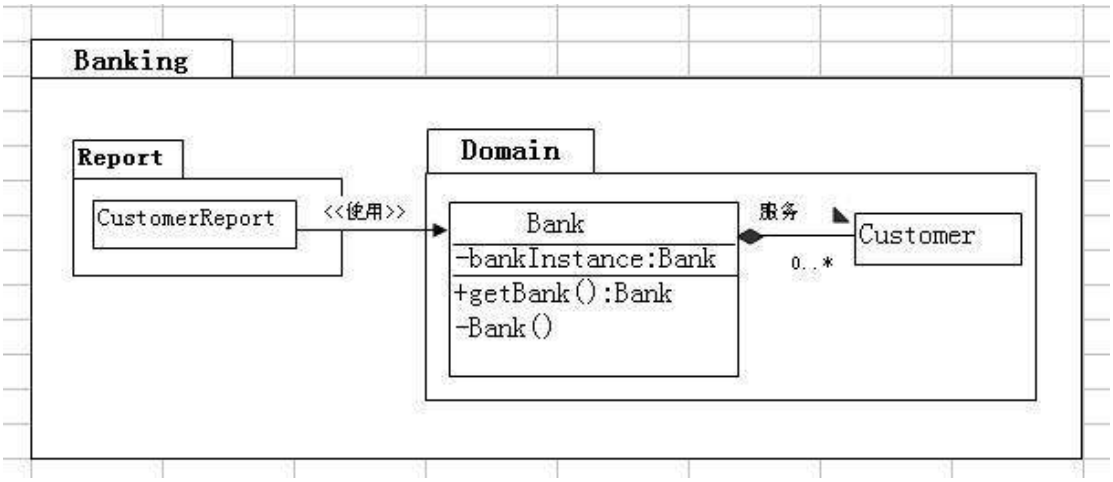
Checking Acct [Owen Bryant] : deposit 25.00 succeeds? true

Checking Acct [Owen Bryant] : withdraw 175.00 succeeds? false

Customer [Bryant, Owen] has a checking balance of 125.0

实验题目 6:(在5_续1的基础上修改)

修改 Bank 类来实现单子设计模式:



实验目的:

单子模式。

提示:

1. 修改 Bank 类, 创建名为 getBanking 的公有静态方法, 它返回一个 Bank 类的实例。
2. 单个的实例应是静态属性, 且为私有。同样, Bank 构造器也应该是私有的

创建 CustomerReport 类

1.在前面的银行项目练习中, "客户报告"嵌入在 TestBanking 应用程序的 main 方法中。在这个练习中, 改代码被放在, banking.reports 包的 CustomerReport 类中。您的任务是修改这个类, 使其使用单一银行对象。

2. 查找标注为注释块/*** ***/的代码行.修改该行以检索单子银行对象。

编译并运行 TestBanking 应用程序
看到下列输入结果:

```
                    CUSTOMER REPORT
                ========================
Customer:simms,jane
        Savings Account:current balance is
        $500.00 Checking Account:current
        balance is $200.00

Customer:Bryant,owen
```

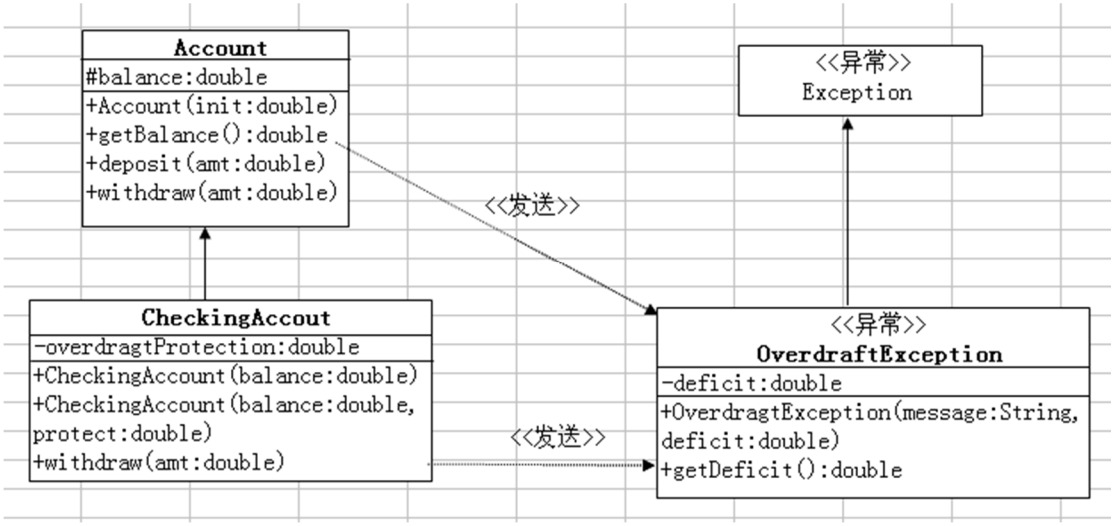
Checking Account:current balance is $200.00

Customer: Soley,Tim
Savings Account:current balance is $1,500.00
Checking Account:current balance is $200.00

Customer:Soley ,Maria
Checking Account:current balance is $200.00
Savings Account:current balance is $150.00

实验题目 7：(在 6 基础上修改)

将建立一个 OverdraftException 异常，它由 Account 类的 withdraw()方法抛出。



创建 OverdraftException 类

1．在 banking.domain 包中建立一个共有类 OverdraftException. 这个类扩展 RuntimeException 类。

2．添加一个 double 类型的私有属性 deficit.增加一个共有访问方法 getDeficit

3．添加一个有两个参数的共有构造器。deficit 参数初始化 deficit 属性

修改 Account 类

4．重写 withdraw 方法使它不返回值（即 void）.

5．修改代码抛出新异常，指明“资金不足”以及不足数额（当前余额扣除请求的数额）

修改 CheckingAccount 类

6．重写 withdraw 方法使它不返回值（即 void）

7．修改代码使其在需要时抛出异常。两种情况要处理：第一是存在没有透支保护的赤字，对这个异常使用“no overdraft protection”信息。第二是 overdraftProtection 数额不足以弥补赤字：对这个异常可使用 ”Insufficient funds for overdraft protection” 信息

编译并运行 TestBanking 程序

Customer [simms,Jane]has a checking balance of 200.0 with a 500.0 overdragt protection

Checking Acct[Jane Simms]: withdraw 150.00

Checking Acct[Jane Simms]: deposit 22.50

Checking Acct[Jane Simms]: withdraw 147.62

Checking Acct[Jane Simms]: withdraw 470.00

Exception: Insufficient funds for overdraft protection

Deifcit:470.0

Customer [Simms,Jane]has a checking balance of 0.0

Customer [Bryant,Owen]has a checking balance of 200.0

Checking Acct[Bryant,Owen]: withdraw 100.00

Checking Acct[Bryant,Owen]: deposit25.00

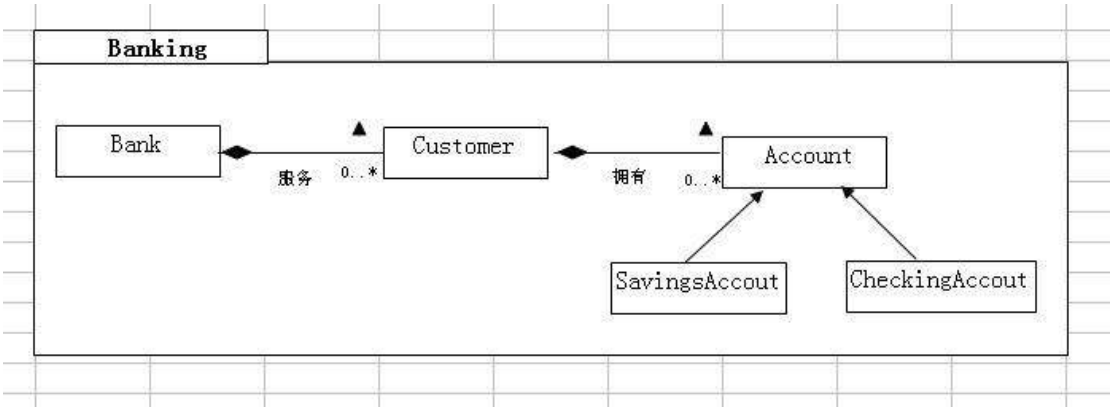
Checking Acct[Bryant,Owen]: withdraw 175.00

Exception: no overdraft protection Deficit:50.0

Customer [Bryant,Owen]has a checking balance of 125.0

实验题目 8:

将替换这样的数组代码:这些数组代码用于实现银行和客户间, 以及客户与他们的帐户间的关系的多样性。



实验目的:

使用集合

实验说明:

修改 Bank 类

修改 Bank 类, 利用 ArrayList 实现多重的客户关系, 不要忘记倒入必须的 java.uti类

1. 将 Customer 属性的声明修改为List 类型, 不再使用 numberOfCustomers 属性。
2. 修改 Bank 构造器, 将 customers 属性的声明修改为List 类型, 不再使用 numberOfcustomers 属性
3. 修改 addCustomer 方法, 使用 add 方法
4. 修改 getCustomer 方法, 使用 get 方法
5. 修改 getNumofCustomer 方法, 使用 size 方法

修改 Customer 类

6. 修改 Customer 类, 使用 ArrayList 实现多重的账户关系。修改方法同上。

编译运行 TestBanking 程序

这里, 不必修改 CustomerReport 代码, 因为并没有改变 Bank 和 Customer 类的接口。编译运行TestBanking

应看到下列输出结果:

```
                    CUSTOMERS REPORT
                    ====================
Customer:Simms,Jane
    Savings Account:current balance is
```

$500.00 Checking Account:current
balance is $200.00

Customer:Bryant,Owen
Checking Accout:current balance is $200

Customer:Soley,Tim
Savings Account:current balance is $1,500.00
Checking Account:current balance is $200.00

Customer:Soley,Tim
Checking Account:current balance is $200.00
Savings Account :current balance is $150.00

可选:修改 CustomerReport 类
修改 CustomerReport 类, 使用 Iterator 实现对客户的迭代

1. 在 Bank 类中, 添加一个名为 getCustomers 的方法, 该方法返回一个客户列 表上的 iterator
2. 在 Customer 类中, 添加一个名为个体 Accounts 的方法, 该方法返回一个帐 户列表上的 iterator
3. 修改 CustomerReport 类, 使用一对嵌套的 while 循环(而不是使用嵌套的 for 循环), 在客户的 iterator 与帐户的 iterator 上进行迭代
4. 重新编译运行 TestBanking 程序, 应看到与上面一样的输出结果