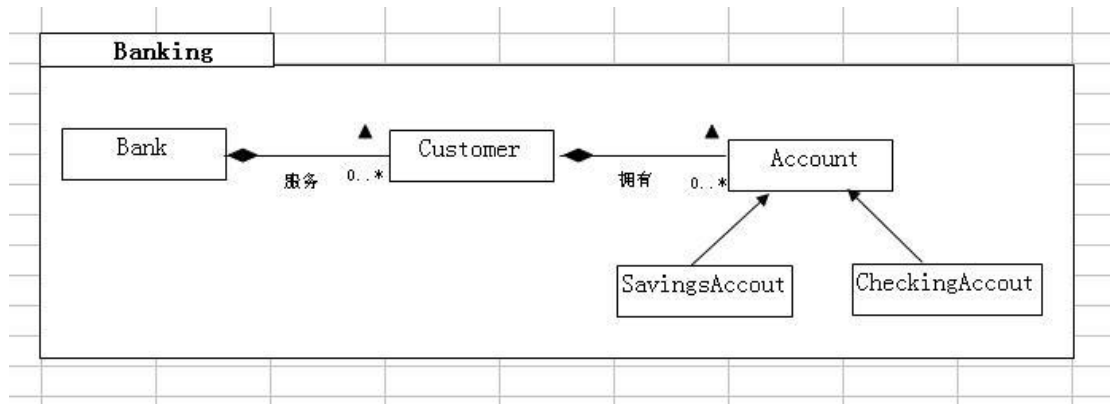


实验题目 8:

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.util 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 Account: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 `getAccounts` 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