Java Coursework

Ricard Solé Casas

June 6, 2017

Contents

Chapter 1

Submission

Synopsis

Hand-in document for the part 2 of the Programming Coursework at Ada College.

Tasks

1. Create a new application named `ManageAccount.java` (similar to `Transactions.java`) that uses the `Account` class as follows:
2. Creates three new `Account`s initialized as specified below:
   - "Barack Obama" with £400 as initial balance and account number: 20230715
   - "Bill Gates" with £500 as initial balance and account number 31558040
   - "Tom Cruise" with £600 as initial balance and account number 44003050
3. Deposits £50 in the first account, and prints resulting balance
4. Withdraws £400 from the second account, and prints resulting balance
5. Deposits £75 in the first account, and prints resulting balance
6. Prints out Account info for all three accounts
7. Examine the `getBalance()` method in the `Account` class. Note that it returns the balance in the account. Add some more code in `ManageAccount.java` to use the `getBalance()` method to get the balances of the three accounts and add them together to obtain the total amount of money in the bank. Print the total and verify that you are getting the right amount
8. Modify the `withdraw()` method to print an appropriate message, when there is not sufficient fund in the account.
9. Add another version of the `withdraw()` method. This version does NOT charge a withdrawal fee, so it has only one parameter. (Java allows you to define alternative versions of methods using the same method name as long as the different versions also have different a different number of parameters)
10. Use this version of the method in `ManageAccount.java` to `withdraw` the taxes from the accounts. (Reminder: The name of this method should still be `withdraw()`. You need to write some additional code in `ManageAccount.java` to "tax" the accounts by withdrawing 15% from each of the first three accounts
11. Add another version of the constructor, which takes only 2 parameters: `name` and `account number` (i.e., no initial balance). This constructor creates an `Account` object with initial balance £0. Modify `ManageAccount.java` to use this version of the constructor to create the "Inland revenue" account.

12. Add additional code to the `ManageAccount.java` to deposit the total tax to the "Inland revenue" account.

13. Create a new method that adds interest to the account. The amount added should be computed according to the rate given by its parameter. For example, if the `acct1` balance is £100.00 and the method is invoked as follows: `acct1.addInterest(0.015);` the balance of `acct1` should increase by 1.5% (so £100 + £1.50 = £101.50). Test your method by invoking it four times to add interest to all the accounts (including Inland Revenue's!).

14. Add a method to record the date when the account is opened and test it using the `Account.java`.

15. Provide an Overdraft facility in the `Account.java` file which allows a withdraw as long as the current balance has not reached the overdraft limit. Test this using the `Account.java`.

Chapter 2

Appendix

Account.java

```
package me.rsole;

import java.text.NumberFormat;
import java.util.Date;

public class Account {
  private int accountNumber;
  private double balance;
  private String name;
  private Date openingDate;
  private double maxOverdraft;
  private String INSUFFICIENT_FUNDS_MSG = "Insufficient funds on account # %d";

  Account(String name, int accountNumber) {
    this(name, accountNumber, 0);
  }

  Account(String name, int accountNumber, double initialBalance) {
    this.name = name;
    this.accountNumber = accountNumber;
    this.balance = initialBalance;
    this.openingDate = new Date();
    this.maxOverdraft = 0;
  }

  void deposit(double amount) {
    balance += amount;
  }

  double withdraw(double amount) throws InsufficientFundsException {
    return withdraw(amount, 0);
  }

  double withdraw(double amount, double fee) throws InsufficientFundsException {
    double withdrawal = amount + fee;
    double newBalance = balance - withdrawal;

    if (newBalance < (0 - maxOverdraft)) {
      throw new InsufficientFundsException(String.format(
```

```
        INSUFFICIENT_FUNDS_MSG, accountNumber
      ));
    }

    balance = newBalance;
    return withdrawal;
  }

  double getBalance() {
    return balance;
  }

  int getAccountNumber() {
    return accountNumber;
  }

  void addInterest(double i) {
    balance = balance * i * 100;
  }

  Date getOpeningDate() {
    return openingDate;
  }

  public double getMaxOverdraft() {
    return maxOverdraft;
  }

  public void setMaxOverdraft(double maxOverdraft) {
    this.maxOverdraft = maxOverdraft;
  }

  public String toString() {
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return (accountNumber + "\t" + name + "\t" + fmt.format(balance));
  }

  class InsufficientFundsException extends Exception {
    InsufficientFundsException() {
    }

    InsufficientFundsException(String message) {
      super(message);
    }
  }
}
```

ManageAccount.java

```
package me.rsole;

import java.util.ArrayList;
import java.util.List;

public class ManageAccount {
  public static void main(String[] args) {
    List<Account> accounts = new ArrayList<>();
```

```
    accounts.add(new Account("Barack Obama", 20230715, 400));
    accounts.add(new Account("Bill Gates", 31558040, 500));
    accounts.add(new Account("Tom Cruise", 44003050, 600));

    accounts.get(0).deposit(50);
    System.out.println(accounts.get(0).getBalance());

    try {
      accounts.get(1).withdraw(10000);
      System.out.println(accounts.get(1).getBalance());
    } catch (Account.InsufficientFundsException e) {
      System.out.println(e.getMessage());
    }

    accounts.get(0).deposit(75);
    System.out.println(accounts.get(0).getBalance());

    for (Account a : accounts) {
      System.out.println(a);
    }

    System.out.println("Total in bank is: " + getTotal(accounts));

    Account inlandRevenue = new Account("Inland Revenue", 1);
    inlandRevenue.deposit(tax(accounts));

    for (Account a : accounts) {
      a.addInterest(.015);
      System.out.println(
        "Added 1.5% interest to account #" + a.getAccountNumber()
      );
      System.out.println("New balance is: " + a.getBalance());
    }

    inlandRevenue.addInterest(.015);
    System.out.println(
      "Added 1.5% interest to account #" + inlandRevenue.getAccountNumber()
    );
    System.out.println("New balance is: " + inlandRevenue.getBalance());

    for (Account a : accounts) {
      System.out.format(
        "Account # %d was created on %s.\n",
        a.getAccountNumber(),
        a.getOpeningDate()
      );
    }

    System.out.format(
      "Account # %d was created on %s.\n",
      inlandRevenue.getAccountNumber(),
      inlandRevenue.getOpeningDate()
    );
  }

  private static double getTotal(List<Account> accounts) {
    return accounts
      .stream()
```

```
      .map(Account::getBalance)
      .reduce(0.0, (total, balance) -> total + balance);
  }

  private static double tax(List<Account> accounts) {
    double t = 0;

    for (Account a : accounts) {
      try {
        t += a.withdraw(a.getBalance() * .15);
      } catch (Account.InsufficientFundsException e) {
        System.out.format(
          "Account # %d had insufficient funds to pay their taxes.\n",
          a.getAccountNumber()
        );
      }
    }

    return t;
  }
}
```