Programming - Part II

Ricard Solé Casas

June 8, 2017

Declaration

I confirm that the submitted coursework is my own work and that all material attributed to others (whether published or unpublished) has been clearly identified and fully acknowledged and referred to original sources. I agree that the College has the right to submit my work to the plagiarism detection service. TurnitinUK for originality checks.

Acknowledgements

I'd like to thank my partner Shannon for her continued support and challenges that help me grow, both professionally and personally. I would also like to thank all of you who also helped me get here.

Contents

Chapter 1

Submission

1.1 Synopsis

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

1.2 Exercises

1.2.1 Foreword

The complete code implementations can be found on Appendix A.

1.2.2 Exercise I

Console output

```
> 450.0
> 100.0
> 525.0
> 20230715  Barack Obama    $525.00
> 31558040  Bill Gates  $100.00
> 44003050  Tom Cruise  $600.00
```

1.2.3 Exercise II

Console output

```
> Total in bank is: 1225.0
```

Code snippet

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

  private static double getTotal(List<Account> accounts) {
    return accounts
      .stream()
      .map(Account::getBalance)
      .reduce(0.0, (total, balance) -> total + balance);
  }
```

1.2.4 Exercise III

Code snippet

```
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;
}
```

1.2.5 Exercise II

Code snippet

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

1.2.6 Exercise I

Code snippet

```
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;
}
```

1.2.7 Exercise VI

Code snippet

Constructor:

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

Usage:

```
Account inlandRevenue = new Account("Inland Revenue", 1);
```

1.2.8 Exercise VII

Code snippet

```
inlandRevenue.deposit(tax(accounts));
```

1.2.9 Exercise VIII

Console output

```
Added 1.5% interest to account #20230715
New balance is: 669.375
Added 1.5% interest to account #31558040
New balance is: 127.49999999999999
Added 1.5% interest to account #44003050
New balance is: 765.0
Added 1.5% interest to account #1
New balance is: 275.625
```

Code snippet

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

1.2.10 Exercise II

Code snippet

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

1.2.11 Exercise I

Code snippet

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

public double getMaxOverdraft() {
  return maxOverdraft;
}

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

Chapter 2

Appendix A

2.1 Account.java

```
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);
    }
  }
}
```

2.2 ManageAccount.java

```
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(400);
      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;
  }
}
```