Java Programming (JOOSE2)

Laboratory Sheet 3

This Lab Sheet contains material based on Lectures 1 – 6 (up to *8 October 2014*), and contains the submission information for Laboratory 3 (week 4, *13 – 17 October 2014*).

Be sure to look over the material of Lectures 5 - 6 before Laboratory 3, and bring this sheet to your Laboratory.

The deadline for submission is 24 hours after the end of your scheduled laboratory session in week 4 *(13 – 17 October 2014)*.

Aims and objectives

- Reinforcement of the basic concepts of objects and classes in Java
- Design and implementation of a simple class

Set up

When you download `Laboratory3.zip` from moodle, please unzip this file. You will obtain a folder `Laboratory3`, containing a subfolder entitled `Submission3_1`. Remember that for this Laboratory you will have to switch your Eclipse workspace to the `Laboratory3` folder.

In subfolder `Submission3_1` will be a file `BankAccount.java` that contains a skeleton bank account class and a file `TestBankAccount.java` that contains a set of JUnit tests. Create a project entitled `Submission3_1`; the given files will become part of this project when you follow the usual project creation steps

Submission material

Preparatory work for these programming exercises, prior to your scheduled lab session, is expected and essential to enable you to submit satisfactory attempts.

Submission

You are to design and implement a class `BankAccount` representing a (simplified) bank account. Each account is to have the following `private` fields:

- `accountNumber` (an `int`)
- `accountHolder` (a `String`)
- `currentBalance` (a `double`)
- `overdraftLimit` (a `double`)

There should be two `public` constructor methods:

- a no-args constructor `BankAccount()`
- a constructor that takes an `accountHolder` name and an `overdraftLimit`, i.e. `BankAccount(String accountHolder, double overdraftLimit)`

For all created `BankAccount` objects, each object should have a unique `accountNumber`. All `BankAccount` objects are initialized with a balance of 0.0. Unless otherwise specified via the constructor, the `overdraftLimit` should also be initialized to 0.0.

There should be methods to withdraw and deposit a given amount:

- `public void deposit(double amount)`

- `public boolean withdraw(double amount)` – returns `true` if the withdrawal is successful, `false` if unsuccessful. A withdrawal should not succeed if it would cause the overdraft limit to be exceeded.
- If the `amount` parameter for `deposit()` or `withdraw()` is negative, then a `java.lang.Exception` should be thrown.

Please also define relevant getter methods for the BankAccount fields:

- `public int getAccountNumber()`
- `public String getAccountHolder()`
- `public double getCurrentBalance()`
- `public double getOverdraftLimit()`

and setter methods for two fields:

- `public void setAccountHolder(String accountHolder)`
- `public boolean setOverdraftLimit(double overdraftLimit)` – return value indicates success. An `overdraftLimit` is a non-negative value. If the `currentBalance` is negative, then the new `overdraftLimit` must be greater than the absolute value of the `currentBalance` for the `setOverdraftLimit` method to succeed.

Also override the inherited `toString` method, to display the values of the four object fields. Sums of money (e.g. the `currentBalance` reported in `toString()`) should be displayed in a conventional way e.g. `£100.00` or `-£3.99` (this can be done using the `String.format` or `System.out.printf` – see Java API docs[1]). The correct £ symbol is available via this method call: `java.util.Currency.getInstance("GBP").getSymbol(java.util.Locale.UK);`

Also override the inherited `equals` method. Two `BankAccount` instances are only equal if they have the same `accountNumber` value.

Use the JUnit test driver class, `TestBankAccount`, for this class to check that your code works. (Remember how to add the JUnit library to your project build path, as in previous labs.) The test driver creates a small number of accounts, carries out some operations on these accounts, and then displays the final details of each account before terminating.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 3 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code ...\Laboratory3\Submission3_1\ and drag *only* the single Java file BankAccount.java into the drag-n-drop area on the moodle submission page. **Your markers only want to read your java file, not your class file.** Then click the blue save changes button. Check the single .java file is uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your file and return feedback to you via moodle.

[1] E.g. http://docs.oracle.com/javase/7/docs/api/java/lang/String.html or http://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html

Guidelines for Mark Scheme

A1: All methods completed, with correct types and behaviour. All JUnit tests pass.

B1: Majority of methods completed, with correct types and behaviour. Some JUnit tests pass.

C1: Some methods attempted, possible problems with types and behaviour. Code may not compile.

D1: Minimal effort.