Assignment 2 – CE202 – 2015/16

Deadline: Wednesday 9th December 2015, 11:59:59

Weight. 20% of final mark

What to submit:

Your assignment-related files should be included in a single .zip file (WinZip is installed and ready to use in all CS lab machines) and only the .zip file should be submitted. The name of the file should include your name and 'CE202', for example: SamLowryCE202.zip. The Zip file should include the following:

Document. Answers to Questions 1 and 3 in the assignment should be combined in one file saved either in Microsoft Word (.doc) or Portable Document Format (.pdf) format. Text must be single-spaced 12 point Times font. Diagrams must be included in the document itself.

Programs. Answers to Question 2 should be submitted in the form of a collection/directory of .java files as indicated, including all other files as necessary to execute the test and run it exactly as prescribed. Place your full name in a comment at the top of each .java file. The Java code MUST COMPILE and run as you claim it does.

1. Code-coverage test

The following class implements a simple bank account system.

The bankAccount() method is passed a customer number, and a starting balance, and it returns the final balance.

For the purposes of this program the bank account balance has been simplified to only deal with integer values.

Sample program

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;

public class Bank {


public int bankAccount(int customerNumber, int startingBalance) {

    Scanner stdin = new Scanner(System.in);
    System.out.println("Welcome to the Bank.");

    //Get the customer
    Customer myCustomer = Customer.getWithCustomerNumber(customerNumber);

    // Get the customer name.
```

```
    String name = myCustomer.name;

    // Get the starting balance.
    int balance = startingBalance;

    // Get the date and time
    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    Date date = new Date();

    // Get the user's choice.
    // 1. Make a deposit
    // 2. Make a withdrawal
    // 3. Print your balance
    // 4. Quit
    int choice = 0;

    // Continue processing choices until the user quits.
    while (choice != 4) {
      // Reprompt the menu.
      System.out.println("Here are your menu options:");
      System.out.println("1. Make a deposit.");
      System.out.println("2. Make a withdrawal.");
      System.out.println("3. Print your balance.");
      System.out.println("4. Quit.");
      System.out.println("Please enter the number of your choice.");
      choice = stdin.nextInt();

      switch (choice) {
        case 1: {
          // Handle a deposit.
          System.out.println("Enter the amount of your deposit.");
          int deposit = stdin.nextInt();
          balance = balance + deposit;
          System.out.println(dateFormat.format(date) + " Customer " + name + " new balance is $" + balance + ".");
          break;
        }
        case 2: {
          // Handle a withdrawal.
          System.out.println("Enter the amount of your withdrawal.");
          int withdrawal = stdin.nextInt();
          if (balance - withdrawal >= 0) {
            balance = balance - withdrawal;
            System.out.println(dateFormat.format(date) + " Customer " + name + " new balance is $" + balance + ".");
          } else {
            System.out.println(dateFormat.format(date) + " Customer " + name + " Insufficient funds for withdrawal.
Balance is $" + balance + ".");
          }
          break;
        }
        case 3: {
          // Print out the current balance.
          System.out.println(dateFormat.format(date) + " Customer " + name + " balance is $" + balance + ".");
          break;
        }
        case 4: {
```

```
                //Do nothing, we quit
                break;
            }
            default: {
                System.out.println("Sorry that is not a valid choice.");
            }
          }
      }
      System.out.println("Thank you for using the bank!");
      return balance;
   }
}
```

(i) Using the bankAccount() method, draw its flow graph and list all the possible paths in the flow of the method.

You should list paths that include up to 2 menu choices followed by a quit eg. 'make a deposit', followed by 'print a balance' then 'quit' could be one valid path.

(ii) For each path, design one test case as follows:

Define which values each field (such as keys and values), method argument (such as key, value) and the relevant local variables must take so that the path is chosen. Write the expected return value from the method and the value of each field after the conclusion of the particular path for the test case you suggested.

The best way to do this is to create a table that defines each test case based on the following format:

Path	**Test Case**	**Expected Result**
Path through the flow graph for test case 1	*Input values for variables*	*Output values for variables*
For test case 2 …	*Etc*	*Etc*
Etc	*Etc*	*Etc*

2. Unit Test

Write a program that carries out the code-coverage test you designed in question 1 and notes any failures. Your test program should run all the test cases you designed in question 1, detect failures and calculate the Probability of Failure on Demand (POFOD) as defined in the lecture. See hints below.

To do so, define classes BankTestDriver and BankTestOracle. The methods in these two classes should implement the tests using the test cases you defined above.

The Bank class (as supplied above) makes use of a Customer class to retrieve information about the customer. Replace all occurrences of this with appropriate stubs to enable you to isolate and test the Bank class. Hint: the stub should make use of the test case values that you have defined above.

You should also replace the use of the Date class with an appropriate stub.

The answer to this section should be entirely in terms of three Java files:

- Bank.java (based on the code supplied above but revised with stubs)
- BankTestDriver.java
- BankTestOracle.java

3. Design

Assess the quality of the design of the program you implemented in Question 2:

(i) Have you applied either one of the software design principles of--

- Abstraction
- Modularity
- Loose Coupling or
- Cohesion?

For each one of these principles answer: If No, explain why not. If Yes, explain where exactly and give the details of one example.

(ii) Have you employed the object-oriented design mechanisms of--

- Inheritance
- Encapsulation
- Information hiding
- Polymorphism

For each one of these mechanisms answer: If No, explain why not. If Yes, explain where exactly and give the details of one example.

Hints for BankTestDriver.java

BankTestDriver.java should look something like this:

```
public class BankTestDriver {
    /**
     * Main function calculating the POFOD on the basis of the return statement
results of all test cases
     * POFOD is being printed out
*/
    public static void main(String[] args) {
        //insert your code to execute your tests here
        //you should implement each test as a separate method below which
returns a Boolean value indicating whether the test is successful or not
       //to calculate the POFOD you will need to keep a count of the total
number of tests and the number of successful tests


      <insert your code here>



       System.out.println("POFOD is : "); //output your POFOD

    }


/**
* test case implementations
* return boolean variables which indicate how successful the cases are
*/
    public static boolean test1(){
        //code to setup the variables and execute the test case 1
        //your code will create an instance of the Bank class and execute the
bankAccount method here
        //you should then pass the output from the Bank test to the Oracle which
will compare the output with the expected output and return a Boolean value
indicating success or failure
        <insert your code here>




      return BankTestOracle.bankAccount(<insert your parameters>);
    }

    public static boolean test2(){
      //etc

    }
```

Hints for BankTestOracle.java

BankTestOracle.java should look something like this:

```
public class BankTestOracle {

   public static boolean bankAccount(<parameters>){

      /**

       * implement tests (oracles) for each of the test cases to compare the result with the expected result and return a Boolean value indicating success of failure

       */

      <insert your code here>

 }
```