Programming Lab Exercise 3

Before you start:
Create a folder called **lab3** inside your personal **java** folder you created at the start. Save all your work for lab 3 in this folder. As usual name your files according to the question e.g. **lab3aq1.java** unless otherwise requested in the question.

Using Eclipse:

Your understanding of the classes and methods in Java is tested. In particular:

- Write class and method definitions
- Write driver programs to test these definitions by instantiating objects
- Differentiate between static and non-static methods and variables

Complete each question (successfully!) before you move on to the next one.

Exercises:

Q1.
The definition for a class Time is contained in Time.java on Blackboard. There is also a driver program called TimeTest.java. Copy both these files to your folder. Compile them and run the driver program. Can you follow what is happening? If not, ask for help.
Provide another driver program Clock.java that will create a Time object - you should pass to the Time constructor method the current time in hours and minutes. Hint: use `java.util.Calendar` to create the time object as follows:

```
Calendar cal = Calendar.getInstance();
Time t = new Time (cal.get(Calendar.HOUR_OF_DAY), cal.get(Calendar.MINUTE));
```

Next, write a loop that calls method `tick()` every second and then prints the stored time. The loop (and program) should terminate when the stored time advances to the next minute.
Hint: to find out when a second has passed you will need to use `System.currentTimeMillis()` which returns the number of milliseconds elapsed since January 1, 1970. There are 1000 milliseconds in 1 second.

Q2.
Create a class `SavingsAccount`. Each `SavingsAccount` should have a unique number that is automatically assigned by the constructor method, i.e. the number is not to be passed as a parameter to the constructor. The account numbers should start at 1 and count upwards in increments of 1.
Use a static class variable to store the `annualInterestRate` for each of the savers. Each object of the class contains a private instance variable `savingsBalance` indicating the amount the saver currently has on deposit. Provide method `calculateMonthlyInterest`() to calculate the monthly interest by multiplying the balance by `annualInterestRate` divided by 12; this

interest should be added to `savingsBalance`. Provide a static method `modifyInterestRate`() that sets the `annualInterestRate` to a new value.

Driver program: Write a driver program to test class `SavingsAccount`. Instantiate two different savingsAccount objects, `saver1` and `saver2`, with balances of €2000.00 and €3000.00, respectively. Set `annualInterestRate` to 4%, then calculate the monthly interest and print the new balances for each of the savers. Then set the `annualInterestRate` to 5% and calculate the next month's interest and print the new balances for each of the savers.

Q3.

Create a class `BankCustomer`. Each BankCustomer has a name, address and can have up to three `SavingsAccounts`. The BankCustomer constructor method should only accept the name and address of the customer. Provide a method called `addAccount` that accepts one SavingsAccount parameter – the BankCustomer object should keep track of how many valid SavingsAccounts have been added so far. Provide a method called `balance` that computes and returns the BankCustomers total savings. Provide a method summary that prints each account number and corresponding balance.

Create a driver program that fully tests all of the above methods.