**DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING**
**CONCORDIA UNIVERSITY**
**COMP 5461: Operating Systems**
**Winter 2019**
**Programming Assignment 1**
**Due date: Thursday February 7th before midnight**

**Total marks: 50**

## Objectives

This is an introductory assignment on Java Thread synchronization. You will use Java Threads while learning more about concurrency and achieving atomicity using Java's inbuilt mechanisms.

## Tools

You will use Java native threads for this assignment. You can work on the assignment on any machine you wish (e.g. personal laptop) as long as it uses native threads, and the assignment works in the lab as expected. You can also use any source code editing and compiling tools, such as TextPad, Emacs, Eclipse, JBuilder, etc. You need to make sure that your programs run as expected at the Concordia lab before you submit them.

## Source Code

There are four files provided with the assignment. You will need these files to work on the assignment. The files are available in the zip file PA1Code.zip in your Moodle page.

## File Checklist

Account.java
AccountManager.java
Depositor.java
Withdrawer.java

## Background

In this assignment, you will deal with concurrent executions of multiple threads operating on a shared data structure (Accounts). You will be learning more about Java threads and synchronization mechanisms as you work on this assignment.

An array of accounts holds 10 accounts information (imagine an OS shared resource of the same type). The account array has a valid state once it is initialized. You will employ 10 threads for deposit and 10 threads for withdrawal; each thread is bound to a specific account. There will be one *depositor* thread and one *withdrawer* thread that are bound to one specific account. The depositor is responsible for depositing X amount to the account and the withdrawer is responsible for withdrawing the same X amount from the same account. As a result of running 20 threads the final accounts' balance/state should be the same as initial accounts' balance/state. The main goal is to maintain the accounts' state valid regardless of the number of the concurrent threads accessing them, and regardless of their order.

A snapshot of a **consistent** run:

```
Print initial account balances
Account: 1234 Name: Mike Balance:  1000.0
Account: 2345 Name: Adam Balance:  2000.0
Account: 3456 Name: Linda Balance: 3000.0
Account: 4567 Name: John Balance:  4000.0
```

```
Account: 5678 Name: Rami Balance:  5000.0
Account: 6789 Name: Lee Balance:   6000.0
Account: 7890 Name: Tom Balance:   7000.0
Account: 8901 Name: Lisa Balance:  8000.0
Account: 9012 Name: Sam Balance:   9000.0
Account: 4321 Name: Ted Balance:   10000.0

Depositor and Withdrawal threads have been created
Print final account balances after all the child thread terminated...

Account: 1234 Name: Mike Balance:  1000.0
Account: 2345 Name: Adam Balance:  2000.0
Account: 3456 Name: Linda Balance: 3000.0
Account: 4567 Name: John Balance:  4000.0
Account: 5678 Name: Rami Balance:  5000.0
Account: 6789 Name: Lee Balance:   6000.0
Account: 7890 Name: Tom Balance:   7000.0
Account: 8901 Name: Lisa Balance:  8000.0
Account: 9012 Name: Sam Balance:   9000.0
Account: 4321 Name: Ted Balance:   10000.0
Elapsed time in milliseconds 13900
Elapsed time in seconds is 13.9
```

A snapshot of an **inconsistent** run:

```
Print initial account balances
Account: 1234 Name: Mike Balance:  1000.0
Account: 2345 Name: Adam Balance:  2000.0
Account: 3456 Name: Linda Balance: 3000.0
Account: 4567 Name: John Balance:  4000.0
Account: 5678 Name: Rami Balance:  5000.0
Account: 6789 Name: Lee Balance:   6000.0
Account: 7890 Name: Tom Balance:   7000.0
Account: 8901 Name: Lisa Balance:  8000.0
Account: 9012 Name: Sam Balance:   9000.0
Account: 4321 Name: Ted Balance:   10000.0

Depositor and Withdrawal threads have been created
Print final account balances after all the child thread terminated...

Account: 1234 Name: Mike Balance: -9180.0
Account: 2345 Name: Adam Balance:  2000.0
Account: 3456 Name: Linda Balance: -9.953971E7
Account: 4567 Name: John Balance:  1.00004E8
Account: 5678 Name: Rami Balance:  -3.682011E7
Account: 6789 Name: Lee Balance:  6000.0
Account: 7890 Name: Tom Balance:  -6.951315E7
Account: 8901 Name: Lisa Balance:  -9.087615E7
Account: 9012 Name: Sam Balance:  1.00009E8
Account: 4321 Name: Ted Balance:  -9.631527E7

Elapsed time in milliseconds 139
Elapsed time in seconds is 0.139
```

There are 20 threads accessing the accounts concurrently. One depositor threads deposits into account # 1234 the amount 10 CAD in 10000000 iterations. On the other side, one withdrawer thread withdraws from account # 1234 the same amount 10 CAD in 10000000 iterations. The rest 9 depositor threads and 9 withdrawer threads perform the same operations over the other 9 accounts.

As part of this assignment, you are required to perform the following tasks:

**Tasks**

**Task 1: Atomicity violation**

Compile and run the Java app given to you as it is. Explain why the main requirement above (i.e. consistent state of the account array) is not met. Find the bug(s) that cause(s) it and in a few sentences explain how it can be fixed.

**Task 2: Starting Order**

Explain in a few sentences what determines the starting order of the threads. Can the consistency of the accounts preserved by changing the starting order? Explain.

**Task 3: Critical section**

Identify the critical sections of the given code. You can "cut and paste" the relevant pieces of code to your answer.

**Task 4: Method level synchronization**

Create a package named Task4 and copy the provided java files into the new package. Use Java's *method level synchronization* mechanisms to provide a solution to the inconsistency problem at hand.

**Task 5: Block level synchronization (also known as: Synchronized statements)**

Create a package named Task5 and then copy the java files from Task 4 into the new package. This time, use Java's *block level synchronization* (synchronized statements) mechanisms to provide a solution to the inconsistency problem at hand.

**Task 6: Synchronized block versus synchronized method**

Considering the results of Task 4 and Task 5, what is(are) the advantage(s) of synchronized block over synchronized method, or vice versa? Explain.

**Deliverables**

Submit your assignment electronically via https://fis.encs.concordia.ca/eas/. A working copy of the code and a sample output should be submitted for the tasks that require them. A text or pdf file with answers to tasks 1 to 3 and 6 should be provided. Put it all in a file layout as explained below, and archive it with an archiving and compressing utility (e.g. zip). **You must keep a record of your submission confirmation from EAS.** This is your proof of submission, which you may need should a submission problem arises.

Possible file layout:
PA1.txt -- Written components for Tasks 1, 2, 3, and 6 (Note: can be .pdf as well)
Task4:
*.java -- Fixed Java code
before.out -- Buggy output
after.out -- Output after the code has been fixed
Task5:
*.java -- Fixed Java code
after.out -- Output after the code has been fixed according to Task 5

Zip all files into a single file called PA1.zip and submit electronically to EAS.

**Reference**

http://docs.oracle.com/javase/tutorial/essential/concurrency/