COS 226 Practical Assignment 4

- Date Issued: Monday, 29 August 2016, 13:30h
- Date Due: Friday, 2 September 2016, 17:00h
- Submission Procedure: Upload the specified required files for a given task.
- This assignment consists of **2 tasks** with marks stated alongside each question.

1. Introduction

From here on forward, the practical assignments assume that you know the basics of how threads work. You must complete this assignment individually.

You may ask the Teaching Assistants for help but they will not be able to give you the solutions. They will be able to help you with coding, debugging and understanding the concepts explained both in the textbook and during lectures.

2. Mark Allocation

This assignment is divided into **two (2) tasks**. Each task will be marked by a Teaching Assistants manually. Please upload your code to the provided upload slots. Your program must adhere to the following:

1. Your program must produce the expected output
2. Your program must not throw any exceptions
3. Your program must implement the correct mechanisms or algorithms, as specified by the assignment.

3. Source code

Before you begin, you should download the source code from the CS website. You are allowed to make changes to some of the files in the source code.

4. Task 1 – Atomic Account Balance Update (4 marks)

In this task, you will have to study the code that is in the *Task1* folder. The files provided are **Bank.java, Account.java** and **Company.java**.

You will have to study the Java language *java.util.concurrent.atomic* class. Furthermore, useful details for understanding this can be found in Chapter 5 of the prescribed textbook.

You will have to implement the methods of the *Account* class from **Account.java**. They include a constructor, setters and getters. Use relevant atomic functions to ensure proper access and modification of variables. You are allowed to create helper functions. You will use your own test suite to test the execution of the functions.

Compress all your files (especially **Account.java** file) and upload tar-ball (or *.zip* file) to the CS website to the **Practical4 Task1** submission box. Make sure that the uploaded files do not belong to any package.

5. Task 2 –Quick Sort (6 marks)

Implement the quick sort algorithm using multiple threads. The algorithm must sort integers in ***ascending order***. You are provided with the **QuickSort.java** file. The *QuickSort* class contains a function with the following prototype.

public static Runnable parallelQuickSort(int[] list, int numberOfThreads)

The arguments include an array of integers that will be sorted and the number of threads that will be used. The return value may be a null runnable object.

The above method may be called as a function without creating a *QuickSort* object. Also, note that you need not have to provide a copy of the array from your *QuickSort* class, the function must modify the array passed from the caller method. An example for calling the function from *Main* can be done by the following line of code:

QuickSort.parallelQuickSort (list,7);

The *parallelQuickSort* algorithm can be run concurrently, by taking advantage of the algorithm's recursion and, the divide and conquer approach. At each recursive method call, a thread is allocated to sort a sub-array until all threads have been allocated. The threads sort the sub arrays and then combine to create the final sorted array. You are allowed to create helper functions. You are expected to test if your functions work thoroughly.

Compress the file (**QuickSort.java**) and upload the tar-ball or zip to the CS website to the **Practical4 Task2** submission box. Make sure that the uploaded file does not belong to any package.