SCC 211: Programming Coursework on Concurrency

Aim

The aim of this exercise to make students familiar with basic concepts in concurrency, such as threads, critical sections, and race conditions.

As a practical matter, the exercise will be done in Java and will make students familiar with the use of Java threads and synchronization via *joining* threads and the *synchronized* keyword in Java.

The use of any other synchronization mechanism is forbidden in this exercise.

Submission

1. Each submission must be a zip file that contains all the source code (.java) files. There must be no subfolders in the zip file.
2. Each submission must contain a Readme.txt with the student's full name and student ID number.
3. The Java main method by which the application is launched must be in a file named *InventoryMain.java*.
4. Test that the following works:
    a. Extracting the contents of the zip file into some folder and running *javac InventoryMain* from the command line in that folder compiles the application. This should generate the .class files in the same folder.
    b. Running *java InventoryMain* with the appropriate arguments from the command line runs the application as desired. Your program must take two arguments as described below in the program specification.

You must submit the coursework on Moodle by the submission deadline – it is not optional! If, for some reason, the in-lab evaluation cannot take place, the marks will be determined based solely on the submission.

Program Specification

Imagine a warehouse and some number of items in its inventory. We care only about the size of the inventory, not the nature of the items. In particular, we care about maintaining the size of the inventory correctly despite concurrent operations on it. The operations specifically are (1) **adding a single item** to the inventory, and (2) **removing a single item** from the inventory.

Your task is to implement a multithreaded Java program that supports concurrent addition and removal of items. Assume that the initial inventory size is 0.

Your program must take two command-line arguments.

1. The first argument is the number of **add** operations. **For example, giving 50 as the first argument means 50 add operations must be performed, each in a separate thread**.

2. The second argument is the number of **remove** operations. **For example, giving 20 as the second argument means 20 remove operations must be performed, each in a separate thread.**

Desired Output:

Let's say you ran *java InventoryMain 5 10*. This means your program will do 5 add and 10 remove operations, with each operation in a separate thread. Your program must produce the correct result (-5) after the operations.

Your program output must be printed to the console. It consists of a sequence of statements one for each operation. Each statement prints the operation performed and the inventory size resulting from it. In addition, you must print the final inventory size after all the add and remove threads have finished.

Both Figures 1 and 2 shows possible correct program outputs for *java InventoryMain 5 10*.

Figure 1.

```
Added. Inventory size = 1
Removed. Inventory size = 0
Removed. Inventory size = -1
Removed. Inventory size = -2
Removed. Inventory size = -3
Removed. Inventory size = -4
Removed. Inventory size = -5
Removed. Inventory size = -6
Removed. Inventory size = -7
Removed. Inventory size = -8
Removed. Inventory size = -9
Added. Inventory size = -8
Added. Inventory size = -7
Added. Inventory size = -6
Added. Inventory size = -5
Final inventory size = -5
```

Figure 2.

```
Added. Inventory size = 1
Removed. Inventory size = 0
Removed. Inventory size = -1
Removed. Inventory size = -2
Removed. Inventory size = -3
Removed. Inventory size = -4
Added. Inventory size = -3
Added. Inventory size = -2
Added. Inventory size = -1
Removed. Inventory size = -2
Removed. Inventory size = -3
Removed. Inventory size = -4
Removed. Inventory size = -5
Removed. Inventory size = -6
Added. Inventory size = -5
Final inventory size = -5
```

Examples of other valid invocations

java InventoryMain 10 0 (10 add threads; 0 remove threads

java InventoryMain 0 1 (0 add threads; 1 remove thread)

Evaluation

In the lab sessions in Weeks 4 and 5. Be ready to run your code with the arguments you are given. Be prepared to show and explain your code. You get full marks if we are fully satisfied with your outputs, code, and explanations.

The coursework is worth 20 points. This is the *rough* distribution of marks.

Up to 2 points for taking input via command line arguments as specified earlier. **Please no interactive inputs, e.g., via the Scanner class!**

Up to 9 marks: For being able to launch the threads corresponding to the number of add and remove operations as specified in the command line arguments.

Up to 9 marks: For getting the desired output via the correct synchronization.