Advanced Java: Lab 5  
  
Multithreading

**Members:** Alexandre CASARA | Benoit Chaurand |Vincent JACOB

**Group:** Maths & Java

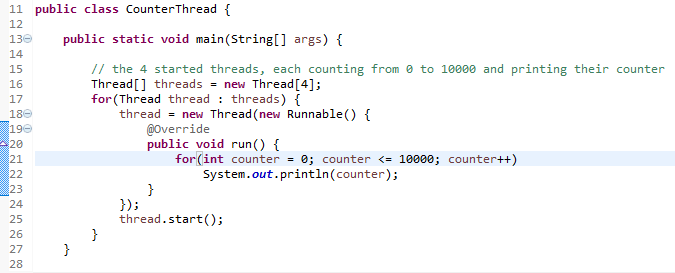
**Date:** 11 February 2018

This document aims to bring additional comments concerning some of the lab’s questions. However, the majority of the questions were answered as usual by commenting the source code.

Exercise 1

**Question 2**

By executing the following code:



We could see in the console:

*Beginning:*



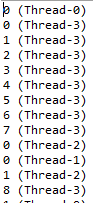
*A little after…:*



This behavior seems normal: all 4 threads are executing simultaneously, some might execute more than others (and thus count more) during some time lapses, that is why we can observe offsets between the several counters.

**Question 3**

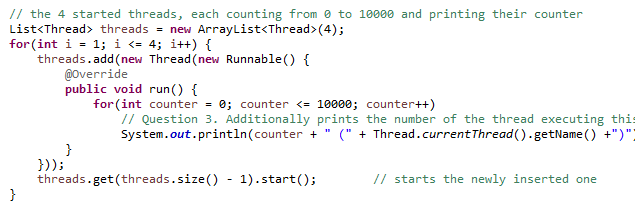
We could verify this by printing the thread numbers:



Here Thread-3 is executing more often than other threads, at least at the beginning.

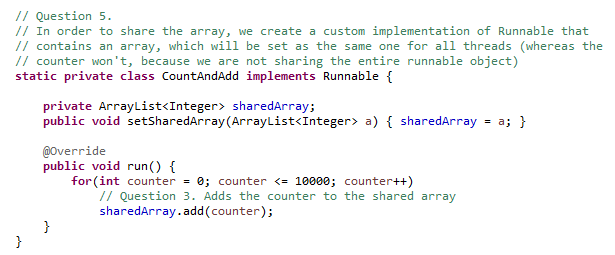
**Question 4**

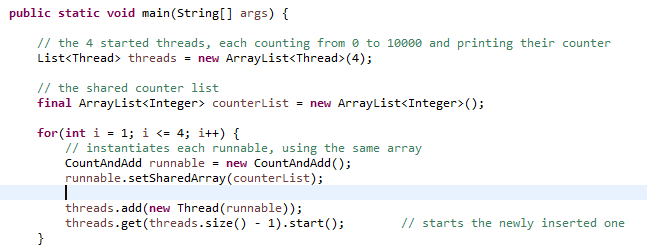
The call to the *join()* method in the main thread actually caused null pointer exceptions. Therefore we had to switch the data structure from a traditional array to an *ArrayList* for this question:

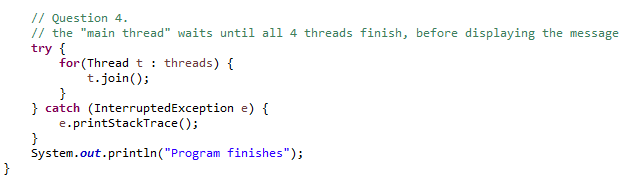


**Question 5**

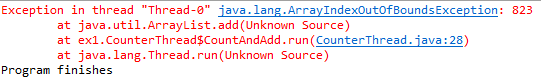
When running this code:



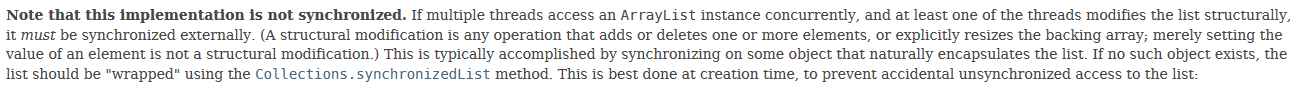




We actually get a different behavior each time. But most of the time we get the following Exception:



This could be explained by looking at the *ArrayList* class in the API Documentation:

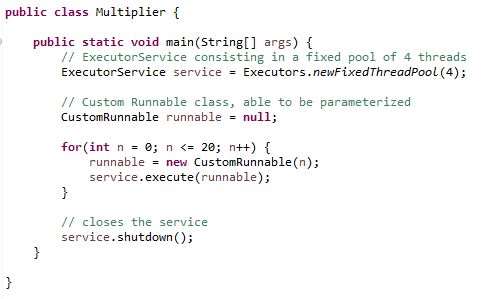


Which states that *ArrayList* operations are **not *thread-safe***, therefore using them with multiple threads at the same time without synchronization causes undefined behavior.

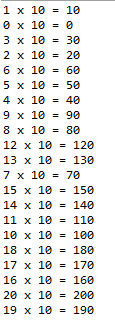
Exercise 5

**Question 2**

Executing the following code:



Gives the following result:



Which shows that the operations have effectively been distributed among the service’s threads (because not always executed in the order that they have been submitted).