Problem 1: matrix multiplication

Develop an execution driven simulator for a multiprocessor that solves a 3x3 matrix multiply (i.e., A x B=C, where A, B, C are 3 x 3 matrixes) following these steps:

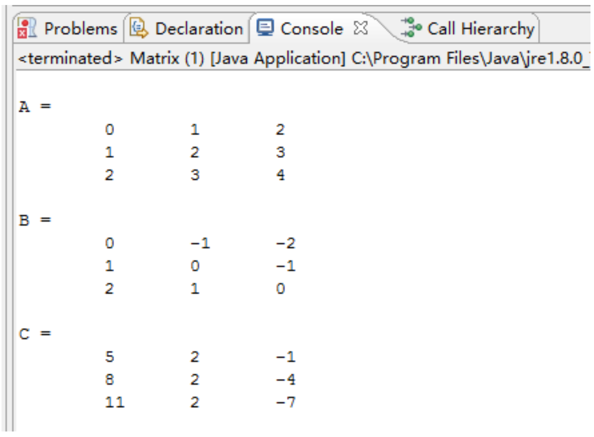
1. Define A, B, C as instance variables using a suitable data-structure
2. Set the values in the constructor
3. Create a thread for every processor so that each processor will compute one element of the resulting matrix.
4. In each thread calculate one element (i.e., multiply the elements of one raw from A and one column from B, and add the results to get the value of C[i][j]).
5. The main thread should wait until all the other threads are done and then print the matrix C.

We provide two source files named *Matrix.java* and *MultThread.java,* respectively. You need to write some codes in each file.

Task:

1. In *MultThread.java*, you need to write how to calculate C[i][j] in *run()* function.
2. In *MultThread.java*, you need to write the code to start the work of each thread in *dowork()* function.

Output is showed as follow:

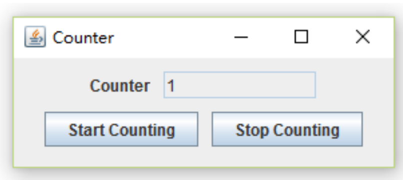


```
<terminated> Matrix (1) [Java Application] C:\Program Files\Java\jre1.8.0_

A =
        0       1       2
        1       2       3
        2       3       4

B =
        0       -1      -2
        1       0       -1
        2       1       0

C =
        5       2       -1
        8       2       -4
        11      2       -7
```

Problem 2: Counter

Description

The GUI program has two buttons. Pushing the "Start Counting" button starts the counting. Pushing the "Stop Counting" button is supposed to stop (pause) the counting. The two button-handlers communicate via a boolean flag called stop. The stop-button handler sets the stop flag; while the start-button handler checks if stop flag has been set before continuing the next count. You should modify the code we provide to make it run correctly (we have provided the demo of the correct version on the website.)



Hint:
What you need to do is to use a thread in *addActionListener* function.

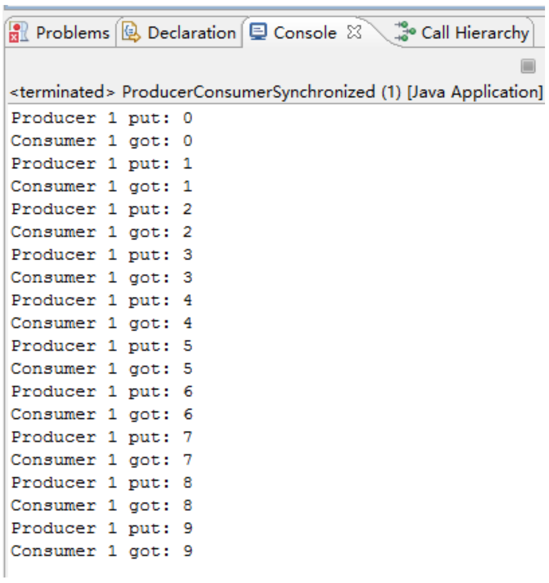
Problem 3: Inter-thread communication

Description

This is a producer-consumer problem.
The results of the code we provide to you are unpredictable;
The correct result should be that:
a number may be read before a number has been produced or multiple numbers may be produced with only one or two being read adding synchronization ensures that a number is first produced, then read in the correct order.
So you need to modify the code to achieve the correct result.

Hint:
In this problem, you need to perform inter-thread communication by the usage of *wait()*, *notify()*, and *notifyAll()* methods.

Output is showed as follow:



```
<terminated> ProducerConsumerSynchronized (1) [Java Application]
Producer 1 put: 0
Consumer 1 got: 0
Producer 1 put: 1
Consumer 1 got: 1
Producer 1 put: 2
Consumer 1 got: 2
Producer 1 put: 3
Consumer 1 got: 3
Producer 1 put: 4
Consumer 1 got: 4
Producer 1 put: 5
Consumer 1 got: 5
Producer 1 put: 6
Consumer 1 got: 6
Producer 1 put: 7
Consumer 1 got: 7
Producer 1 put: 8
Consumer 1 got: 8
Producer 1 put: 9
Consumer 1 got: 9
```