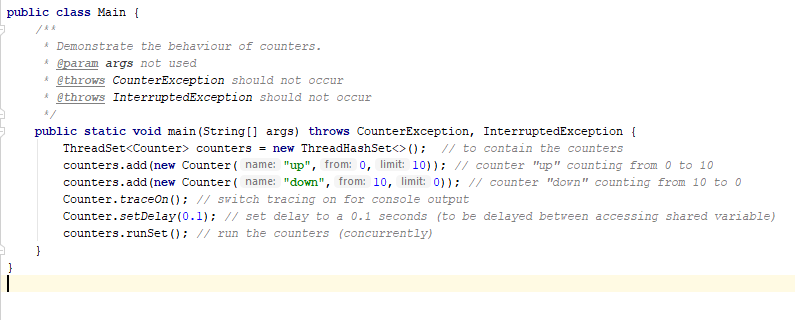
**Main.java**

**Implemented main method to run the threads concurrently with delay of 0.1 second.**



## Question 1. Will the test always terminate?

**I.e. is it certain that no matter how often you were to run the test it would always end in a finite length of time?**

This is the minimum output sequence from which the infinite loop starts to become possible.

down has started: 10

up has started: 0

up has stepped: 1

up has stepped: 2

When considering the above code, after those 4 lines, if both processes would take turns (Up, Down, Up, Down, Up, Down….) they would infinitely loop and change the value by either decreasing or increasing it by 1, which means they would be stuck for ever making shared variable value to be in range of 2-3. Now this is not the only case. While the shared variable value is between 1-9, there is a chance the test may not-terminate, as none of the numbers from the specified range are finish points for processes and **as both processes desperately wants to complete different tasks using the same shared variable, there is a chance the processes will end-up in infinity loop which could not be defined by finite length of time.**

Below is the test output. It did terminate; however, it shows, that compared to shortest output (14 lines) it is way longer and provides evidence that it is possible that test MAY NOT terminate in finite length of time.

down has started: 10

down has stepped: 9

up has started: 0

up has stepped: 1

down has stepped: 0

up has stepped: 1

up has stepped: 2

down has stepped: 1

up has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

down has stepped: 1

up has stepped: 2

down has stepped: 1

up has stepped: 2

down has stepped: 1

up has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

down has stepped: 2

up has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

up has stepped: 5

down has stepped: 4

down has stepped: 3

up has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

up has stepped: 7

down has stepped: 6

up has stepped: 7

down has stepped: 6

down has stepped: 5

up has stepped: 6

up has stepped: 7

down has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

up has stepped: 6

down has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

up has stepped: 5

down has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

down has stepped: 2

up has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

down has stepped: 3

up has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

down has stepped: 4

up has stepped: 5

up has stepped: 6

up has stepped: 7

down has stepped: 6

up has stepped: 7

down has stepped: 6

up has stepped: 7

up has stepped: 8

down has stepped: 7

down has stepped: 6

up has stepped: 7

down has stepped: 6

up has stepped: 7

up has stepped: 8

down has stepped: 7

up has stepped: 8

up has stepped: 9

down has stepped: 8

up has stepped: 9

up has stepped: 10

down has stepped: 9

up has stepped: 10

down has stepped: 9

up has stepped: 10

up has finished: 10

down has stepped: 9

down has stepped: 8

down has stepped: 7

down has stepped: 6

down has stepped: 5

down has stepped: 4

down has stepped: 3

down has stepped: 2

down has stepped: 1

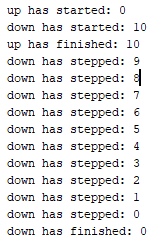
down has stepped: 0

down has finished: 0

**2. What is the shortest possible output for the test, in terms of the number**

**of lines output?**

Shortest output considering lines would be **14**. This happens, because one thread finish point is others thread start point, therefore, when both threads start in the parallel, the next to step in will finish, as previous thread made a shared counter variable to the others finish point.

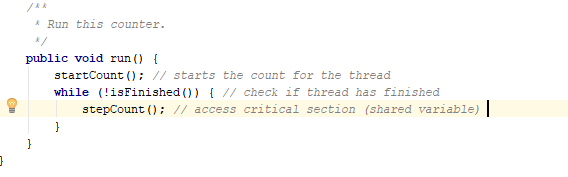


**3. What is the largest possible value that the count can reach when the test is run?**

**11**.



The process starts its counting and then is stuck in infinite loop of decreasing/increasing shared variable, until the shared variable value has reached its finish point. (See code below)



Now, because the shared variable is not secured from being accessed by one processor at any time, and many processors can reach or change it at the same time, this may cause problems, such as unexpected value of 11.

In this example:

**Process 1:** !isFinished() returns false, so the process enters method stepCount();

**Process 1:** Is delayed for a random amount of time (see code below)

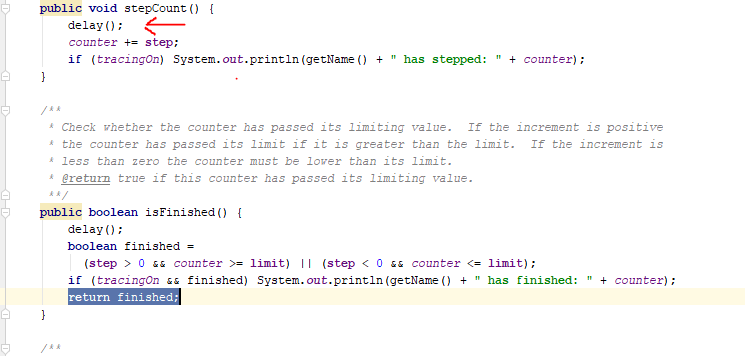
**Process 2:** !isFinished() returns false, as the value of shared variable has not been changed by Process 1 yet (Process 1 is still in delay)

**Process 2:** Is delayed, but wakes up faster then Process 1 and changes the resource value

**Process 1:** Is not informed about any changes of resource, so after waking up from delay, it changes the shared variable causing “unexpected value”

In brief:

While one of the process is in delay at stepCount(); the other process may change the value of shared variable, which means, that after finishing, the thread, which was stuck in delay actually thinks, the value have not been changed and behaves accordingly.



4. What is the lowest possible value that the count can reach when the test

is run?

For the same reasons, explained in the question 3, the minimum shared variable value may be -1.

Situation:

Down has started: 10

Up has started: 0

Down has stepped: -1

Up has stepped: 0

Self Evaluation; 5/5

Good analysis of termination question

Clear and detailed explanation of min & max values

Evidence provided for multiple runs & code implementation