

Lab CC 6: Race Conditions

Objectives

In this lab, you will create a race condition with multiple threads accessing a shared resource. In part two, you will use the synchronized keyword to manage the shared resource

Part One: Race Condition

Step 1: Create the Counter class

1. The counter class is passed a reference to a counter that it will increment then decrement.
2. This counter will be shared by 10 counter threads and will initially be set to zero
3. Since each thread adds 1 then subtracts 1 from the counter, after all of the threads have run, the counter should be back to 0.

```
2
3 class Counter implements Runnable{
4     private int myCount = 0;
5
6     public void increment() {
7         try {
8             Thread.sleep(10);
9         } catch (Exception e) {
10            System.err.println(e);
11        }
12        this.myCount++;
13    }
14
15    public void decrement() {
16        this.myCount--;
17    }
18
19    public int getValue() {
20        return this.myCount;
21    }
```

4. The run() method for the class is defined as follows

```
@Override
public void run() {
    this.increment();
    System.out.println("Value for Thread After increment "
        + Thread.currentThread().getName() + " " + this.getValue());
    this.decrement();
    System.out.println("Final value for Thread "
        + Thread.currentThread().getName() + " " + this.getValue());
}
```

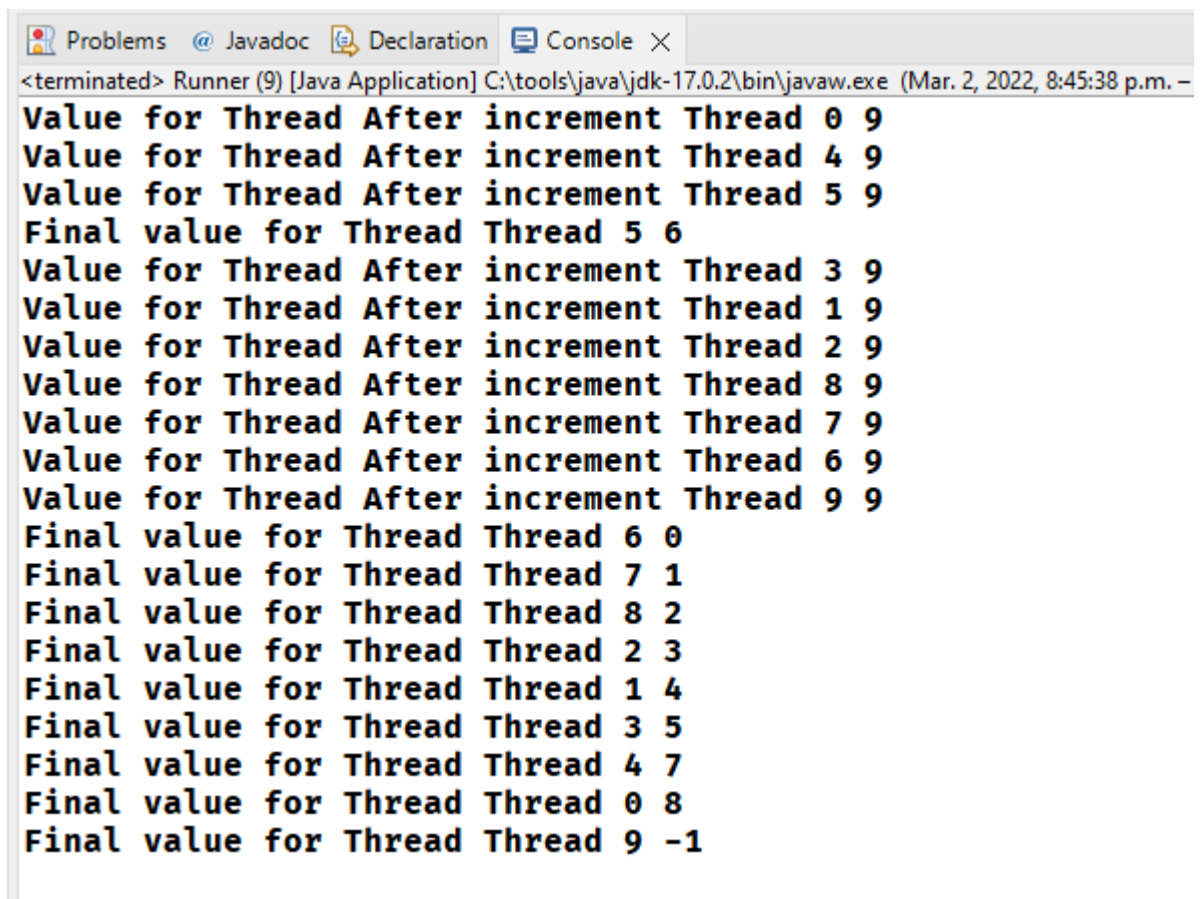
Step 2: Create the runner class

1. This class starts 10 counter threads running and passes to each a reference to a single Counter object.

```
public class Runner {
    public static void main(String[] args) {
        Counter counter = new Counter();
        for (int i = 0; i < 10; i++) {
            Thread t = new Thread(counter, "Thread " + i);
            t.start();
        }
    }
}
```

Step 3: Run the code

1. Eventually the counter may be zero. Notice that according to the code, it should never be more than one but the interleaving of operations means that several threads are incrementing before decrementing.
2. This is non-deterministic since we can never predict what the value of the counter is at any specific point in execution of the code. In the example below the final result is -1
3. Run the code several times and notice that the results will vary.



```
<terminated> Runner (9) [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Mar. 2, 2022, 8:45:38 p.m. -
Value for Thread After increment Thread 0 9
Value for Thread After increment Thread 4 9
Value for Thread After increment Thread 5 9
Final value for Thread Thread 5 6
Value for Thread After increment Thread 3 9
Value for Thread After increment Thread 1 9
Value for Thread After increment Thread 2 9
Value for Thread After increment Thread 8 9
Value for Thread After increment Thread 7 9
Value for Thread After increment Thread 6 9
Value for Thread After increment Thread 9 9
Final value for Thread Thread 6 0
Final value for Thread Thread 7 1
Final value for Thread Thread 8 2
Final value for Thread Thread 2 3
Final value for Thread Thread 1 4
Final value for Thread Thread 3 5
Final value for Thread Thread 4 7
Final value for Thread Thread 0 8
Final value for Thread Thread 9 -1
```

Part Two – Synchronize

This part adds the synchronized block to ensure that only one thread at a time can access the code that manipulates the counter block. This is in a class called SynchCounter

```
2
3 class SynchCounter implements Runnable{
4     private int myCount = 0;
5 }
```

The only change to wrap the code in run() method in a synchronization block

```
@Override
public void run() {
    synchronized (this) {
        this.increment();
        System.out.println(
            "Value for Thread After increment " + Thread.currentThread().getName()
            + " " + this.getValue());
        this.decrement();
        System.out
            .println("Final value for Thread " + Thread.currentThread().getName()
                + " " + this.getValue());
    }
}
```

And we change the runner class to use this new class.

```
2
3 public class Runner2 {
4     public static void main(String[] args) {
5         SynchCounter counter = new SynchCounter();
6         for (int i = 0; i < 10; i++) {
7             Thread t = new Thread(counter, "Thread " + i);
8             t.start();
9         }
10    }
11 }
12
13 }
```

Running the code shows that now the counter objects work exactly as we would expect.

```
Problems @ Javadoc Declaration Console X
<terminated> Runner2 (1) [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Mar. 2,
Value for Thread After increment Thread 0 1
Final value for Thread Thread 0 0
Value for Thread After increment Thread 9 1
Final value for Thread Thread 9 0
Value for Thread After increment Thread 8 1
Final value for Thread Thread 8 0
Value for Thread After increment Thread 7 1
Final value for Thread Thread 7 0
Value for Thread After increment Thread 6 1
Final value for Thread Thread 6 0
Value for Thread After increment Thread 5 1
Final value for Thread Thread 5 0
Value for Thread After increment Thread 4 1
Final value for Thread Thread 4 0
Value for Thread After increment Thread 3 1
Final value for Thread Thread 3 0
Value for Thread After increment Thread 2 1
Final value for Thread Thread 2 0
Value for Thread After increment Thread 1 1
Final value for Thread Thread 1 0
```