Lab 4
CSC 2053 - Platform Based Computing
Grading: Non-graded
In class

Description:
The complexities of today require many people to engage in multitasking - performing more than one task at a time. In this lab we will explore concurrent programs - several interacting code sequences executing simultaneously, possibly through an interleaving of their statements by a single processor. To support our investigation of the topics, we will start with the counter class.

Filename: Counter.java

```
public class Counter {
  private int count;
  public Counter() {
    count = 0;
  }
  public void increment() {
    count++;
  }
  public int getCount() {
    return count;
  }
}
```

Proceed by writing a driver class that simply increments the count variable three times sequentially.

Filename: Demo01.java

```
public class Demo01 {
  public static void main(String[] args) {
    Counter c = new Counter();
    c.increment();
    c.increment();
    c.increment();
    System.out.println("Count is " + c.getCount());
  }
}
```

In order to create multi-threaded program, we need to create a Thread and spawn a Runnable object. Create the Increase class and override the run method to support multi-threading.

Filename: Increase.java

```
public class Increase implements Runnable {
  private Counter c;
  private int amount;
```

```
  public Increase(Counter c, int amount) {
    this.c = c;
    this.amount = amount;
  }
  public void run() {
    for(int i=0;i<amount;i++) {
      c.increment();
    }
  }
}
```

Now, let's create a new driver class that can spawn threads.

Filename: Demo02.java

```
public class Demo02 {
  public static void main(String[] args) throws InterruptedException {
    Counter c = new Counter();
    Runnable r1 = new Increase(c, 10000);
    Thread t1 = new Thread(r1);
    t1.start();
    System.out.println("Expected count is 10000");
    System.out.println("Count in reality is "+c.getCount());
  }
}
```

Is the expected count the same as the count in reality? Think about why that is. Use the thread join() method to ensure that you are able to get the expected count equal the count in reality. Now let us create multiple threads that interfere with each other.

Filename: Demo03.java

```
public class Demo03 {
  public static void main(String[] args) throws InterruptedException {
    Counter c = new Counter();
    Runnable r1 = new Increase(c, 5000);
    Runnable r2 = new Increase(c, 5000);
    Thread t1 = new Thread(r1);
    Thread t2 = new Thread(r2);
    t1.start();
    t2.start();
    t1.join();
    t2.join();
    System.out.println("Expected count is 10000");
    System.out.println("Count in reality is "+c.getCount());
  }
}
```

The expected count again doesn't match the count in reality. We can add the "synchronized" keyword to the increment method to prevent that method from being executed by multiple threads. All synchronized blocks synchronized on the same object can only have one thread executing inside

them at the same time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

Deliverables: No deliverable.