Lab Class 2

1. Locate your Java program from previous lab class. It illustrates the interference between two concurrent threads, both incrementing a single shared variable. This lab will provide solutions to this problem.

2. Transform your program in the following way (avoid using ***static*** keyword):

2.1 Define a class **SharedClass** with 2 private attributes (you may use different identifiers):

```
private long SharedValue;
private Lock LocalLock;
```

2.2 Define 2 constructors:

First constructor initialises the shared value `SharedValue`

Second constructor initialises the shared value `SharedValue` and the explicit lock `LocalLock`.

2.3 Define the **Increment Member Method** using the following steps:

```
long TempValue;
TempValue = this.SharedValue;
TempValue++;
this.SharedValue = TempValue;
```


(Using `TempValue` increases the chance of **interference**)

2.4 Define member method ***public long getSharedValue()***, which will return the private attribute **SharedValue**.

3. Once your class is ready, create a second class, which will extend the class ***Thread***. (`class MyThread extends Thread`). Alternatively, you can implement the Runnable Interface.

3.1 The constructor of the Thread class should capture an instance of the previously defined ***class SharedClass***

3.2 The run() method of the Thread class will call the **Increment Method** multiple times in the following fashion:

```
for(i=0; i<1000000; i++)
{
	SharedLocal.ExplicitLockIncrement();
}
```

4. In your **main method**:

4.1 Create an Instance of the Shared class (`SharedClass MySharedClass)`

4.2 Create 2 instances (MyThreadA and MyThreadB) of the **thread class:**

```
MyThread MyThreadA = new MyThread(MySharedClass);
MyThread MyThreadB = new MyThread(MySharedClass);
```

4.3 Run both threads. After the threads complete (ensure this by using ***join***), print the shared value of `MySharedClass` by invoking `getSharedValue()` method. Observe the result.

5. Implement synchronization by adding additional methods to the SharedClass.

(The only modification to the Thread Class should be calling the corresponding **Increment Method**.)

5.1 By adding **synchronized** method:

```
public synchronized void SynchronizedMethodIncrement()
```

5.2 By adding **synchronized** block

```
synchronized(this)
{
// increment code
}
```


5.3 By using Explicit Lock

This will require an instance of the Lock as follows:

```
Lock MyLock = new ReentrantLock();
```

`MyLock` will be passed during instantiation, invoking the second class constructor:


```
SharedClass MySharedClass = new SharedClass(0,MyLock);
```

Explicit lock can be implemented by calling:

1. Lock and unlock methods, or
2. tryLock and unlock methods

Use both and observe the result.

Finally, you will have 5 versions of the Increment method:

1. No synchronization

Synchronization achieved by:

2. Synchronized Method
3. Synchronized Block
4. Explicit Lock using Lock/unlock
5. Explicit Lock using tryLock/unlock