Exercises Multithreading - 2

1. Download the file RaceCondition.java from Moodle and add it to a new/existing project. The project implements a race condition between two threads. Modify the program, so that the race condition is resolved. (Hint: use a static class and the keyword synchronized for resolving the race condition).

```java
public class RaceCondition {
    private static class Counter
    {
        private long value = 0;
        void inceaseValue()
        {
            value++;
        }
        public long getValue()
        {
            return value;
        }
    }
    public static void main(String[] args) {
            Counter total = new Counter();
        Thread t1 = new Thread(){
            @Override
            public void run() {
                for (int i=0;i<10000000;i++){
                    synchronized(total)
                    {
                        total.inceaseValue();}}}
        };
        Thread t2 = new Thread(){
            @Override
            public void run() {
                for (int i=0;i<10000000;i++){
                    synchronized(total)
                    {
                        total.inceaseValue();}}}
        };
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {}
        System.out.println(total.getValue());
    }
}
```

2. Change the RaceCondition.java program (save it as ThreadPool.java), so that the Thread1 and Thread2 are substituted by two threads from a thread pool. Resolve the race condition by using locks this time.

```java
import java.util.concurrent.ExecutorService;

public class ThreadPool {
    public static long total = 0;



    public static void main(String[] args) {

        ExecutorService pool = Executors.newCachedThreadPool();
        ReentrantLock lock = new ReentrantLock();

        Runnable myTask =  () ->
        {
            lock.lock();
            try
            {
                for(int i =0; i< 100000;i++)
                {
                    System.out.println(total++);

                }

            }
            finally
            {
                lock.unlock();
            }

        };


        for(int j = 0; j<2;j++)
        {
            pool.submit(myTask);
        }

        pool.shutdown();

    }
}
```

3. Download the file SemaphoreDemo.java from Moodle and add it to a new/existing project. What is the output of this program and why? Describe at least three ways to fix the program, so that all threads acquire the semaphore.

It first creates 5 Semaphores and then outputs that it has made them. It then sleeps for 1 second. And tries for 5 additional tries to complete the full 10 it is told to run.

a. Make the number of Semaphores into 10

(ExecutorService executor = Executors.newFixedThreadPool(10); -> ExecutorService executor = Executors.newFixedThreadPool(5);)

b. Make the number of thread pools into 5

(Semaphore semaphore = new Semaphore(5); -> Semaphore semaphore = new Semaphore(10);)

c. Make the sleep thread the same amount as tryAcquire so 10000 milliseconds is 10 seconds.

(permit = semaphore.tryAcquire(1, TimeUnit.SECONDS); if(permit) { System.out.println("Semaphore acquired"); Thread.sleep(5000); } -> permit = semaphore.tryAcquire(5, TimeUnit.SECONDS); if(permit) { System.out.println("Semaphore acquired"); Thread.sleep(5000); })

4. Write a program where the Thread class is extended to define a custom Thread class. In the run() method of this class, the message “Thread-0 is in control” is printed five times by use of a for loop. Thread0 is the thread’s name and it can be different for your own program, but you have to get the thread’s name before printing. In the main method, a new custom Thread object is defined and started. Then, the main thread yields control to the custom thread and then prints the message “main in control” five times (main is also the thread’s name, so again you have to get it before printing. Run the program several times and observe its output. Can you explain it? Try to pause one or both threads for a while (preferably not for the same amount of time) by invoking the sleep method and see how this affects the output.

```
class MyThread extends Thread implements Runnable
{
    public void run()
    {
        for (int i=0; i<5 ; i++)

            System.out.println(Thread.currentThread().getName()
                               + " in control");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

// Driver Class
public class killme
{
    public static void main(String[]args)
    {
        MyThread t = new MyThread();
        t.start();

        for (int i=0; i<5; i++)
        {
            // Control passes to child thread
            try {
                Thread.sleep(4000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            // After execution of child Thread
            // main thread takes over
            System.out.println(Thread.currentThread().getName()
                               + " in control");
        }
    }
}
```