

Lab Exercise on Java Threads

1. Write a short program that prints "Hello world" from a thread.

CODE:

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Hello world");
    }
}

public class q1
{
    public static void main(String[] args)
    {
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

OUTPUT:

```
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ javac q1.java
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ java q1
Hello world
```

Now modify the program to print "Hello world" five times, once from each of five different threads.

CODE:

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Hello world");
    }
}

public class q1
{
    public static void main(String[] args)
    {
        MyThread t1 = new MyThread();
        t1.start();

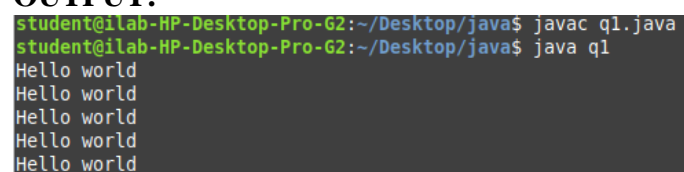
        MyThread t2 = new MyThread();
        t2.start();

        MyThread t3 = new MyThread();
        t3.start();

        MyThread t4 = new MyThread();
        t4.start();

        MyThread t5 = new MyThread();
        t5.start();
    }
}
```

OUTPUT:



```
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ javac q1.java
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ java q1
Hello world
Hello world
Hello world
Hello world
Hello world
```

Now modify the printed string to include the thread number; ensure that all threads have a unique thread number.

CODE:

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName()+ " Hello world" );
    }
}

public class q1
{
    public static void main(String[] args)
    {
        MyThread t1 = new MyThread();
        t1.start();
        MyThread t2 = new MyThread();
        t2.start();
        MyThread t3 = new MyThread();
        t3.start();
        MyThread t4 = new MyThread();
        t4.start();
        MyThread t5 = new MyThread();
        t5.start();
    }
}
```

OUTPUT:

```
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ javac q1.java
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ java q1
Thread-0 Hello world
Thread-1 Hello world
Thread-2 Hello world
Thread-4 Hello world
Thread-3 Hello world
```

2. Write a short program in which two threads both increment a shared integer repeatedly, without proper synchronisation, 1,000,000 times, printing the result at the end of the program.

CODE:

```
class counterClass
{
    static int sharedInt=0;
    void count()
    {
        for(int i=0; i<10000; i++)
            sharedInt++;
    }
}

class MyThread extends Thread
{
    counterClass c;
    MyThread(counterClass c)
    {
        this.c = c;
    }
    public void run()
    {
        c.count();
    }
}

public class q2
{
    public static void main(String[] args)
```

```
{  
    counterClass c = new counterClass();  
    MyThread t1 = new MyThread(c);  
    t1.start();  
    MyThread t2 = new MyThread(c);  
    t2.start();  
    System.out.println("Result: "+c.sharedInt);  
}  
}
```

OUTPUT:

```
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ javac q2.java  
student@ilab-HP-Desktop-Pro-G2:~/Desktop/java$ java q2  
Result: 1578
```

Now modify the program to use synchronized to ensure that increments on the shared variable are atomic.

CODE:

```
class counterClass  
{  
    public static int sharedInt=0;  
    synchronized void count()  
    {  
        for(int i=0; i<10000; i++)  
            sharedInt++;  
    }  
}  
  
class MyThread extends Thread  
{  
    counterClass c;  
    MyThread(counterClass c)
```

```
{
    this.c = c;
    start();
    try
    {
        join();
    }catch(Exception e){}
}
public void run()
{
    c.count();
}
}

public class q2
{
    public static void main(String[] args)
    {
        counterClass c = new counterClass();
        MyThread t1 = new MyThread(c);
        MyThread t2 = new MyThread(c);
        System.out.println("Result: "+c.sharedInt);
    }
}
```

OUTPUT:

```
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>javac q2.java
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>java q2
Result: 20000
```

- 3. We have seen several examples of producer-consumer implemented using a number of different synchronisation primitives in pseudo-code. Implement a Producer-Consumer class using synchronized, wait(), and notify() in Java.**

CODE:

```
class Cart
{
    private int materials;
    private boolean available = false;
    public synchronized int get()
    {
        while (available == false)
        {
            try
            {
                wait();
            }
            catch (InterruptedException ie)
            {
            }
        }
        available = false;
        notifyAll();
        return materials;
    }

    public synchronized void put(int value)
    {
        while (available == true)
        {
```



```
        try
        {
            wait();
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
    materials = value;
    available = true;
    notifyAll();
}
}

class Consumer extends Thread
{
    private Cart Cart;
    private int number;
    public Consumer(Cart c, int number)
    {
        Cart = c;
        this.number = number;
    }
    public void run()
    {
        int value = 0;
        for (int i = 0; i < 5; i++)
        {
            value = Cart.get();
            System.out.println("Consumed "+ value);
        }
    }
}
```

```
        }  
    }  
}  
class Producer extends Thread  
{  
    private Cart Cart;  
    private int number;  
  
    public Producer(Cart c, int number)  
    {  
        Cart = c;  
        this.number = number;  
    }  
    public void run()  
    {  
        for (int i = 0; i < 5; i++)  
        {  
            Cart.put(i);  
            System.out.println("Produced " + i);  
            try  
            {  
                sleep((int)(Math.random() * 100));  
            }  
            catch (InterruptedException ie)  
            {  
                ie.printStackTrace();  
            }  
        }  
    }  
}
```

```
public class q3
{
    public static void main(String[] args)
    {
        Cart c = new Cart();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}
```

OUTPUT:

```
C:\Goku1\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>javac q3.java
C:\Goku1\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>java q3
Consumed 0
Produced 0
Consumed 1
Produced 1
Produced 2
Consumed 2
Consumed 3
Produced 3
Consumed 4
Produced 4
```

- 4. Data races occur when there is insufficient synchronization around composite operations. Write a short program that illustrates a data race.**

CODE:

```
class Counter
{
    int count=0;

    public void increment()
    {
        count++;
    }

    public void decrement()
    {
        count--;
    }
}

class CounterThread extends Thread
{
    Counter c = new Counter();
    CounterThread(Counter c)
    {
        this.c=c;
        start();
    }

    public void run()
    {
        c.increment();
    }
}
```

```
        System.out.println(Thread.currentThread().getName() + ": Beg value: "+ c.count);  
        c.decrement();  
        System.out.println(Thread.currentThread().getName() + ": End value: " +c.count);  
    }  
}
```

```
public class q4  
{  
    public static void main(String args[])  
    {  
        Counter c = new Counter();  
        CounterThread t1 = new CounterThread(c);  
        CounterThread t2 = new CounterThread(c);  
        CounterThread t3 = new CounterThread(c);  
    }  
}
```

OUTPUT:

```
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>javac q4.java  
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>java q4  
Thread-0: Beg value: 1  
Thread-1: Beg value: 2  
Thread-1: End value: 1  
Thread-2: Beg value: 3  
Thread-0: End value: 2  
Thread-2: End value: 0
```

- 5. Which integer between 1 and 10000 has the largest number of divisors, and how many divisors does it have? Write a program to find the answers and print out the results. It is possible that several integers in this range have the same, maximum number of divisors. Your program only has to print out one of them.**

CODE:

```
public class q5
{
    public static void main(String[] args)
    {
        int max=0, maxDivisors=0, divisors;
        for(int j=1;j<=10000;j++)
        {
            divisors=0;
            for(int i=1;i<=j;i++)
            {
                if(j%i==0)
                    divisors++;
            }
            if(divisors>maxDivisors)
            {
                max=j;
                maxDivisors=divisors;
            }
        }

        System.out.println("Max Number: "+max);
        System.out.println("Max no:of Divisors: "+maxDivisors);
    }
}
```

OUTPUT:

```
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>javac q5.java
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>java q5
Max Number: 7560
Max no:of Divisors: 64
```

6. Now write a program that uses multiple threads to solve the same problem, but for the range 1 to 100000. By using threads, your program will take less time to do the computation when it is run on a multiprocessor computer. At the end of the program, output the elapsed time, the integer that has the largest number of divisors, and the number of divisors that it has.

CODE:

```
class Divisors
{
    int maxDivisors = 0, maxNumber=0;
    long startingTime, totalTime;

    class DivisorsThread extends Thread
    {
        int startIndex, endIndex;

        DivisorsThread(int startIndex, int endIndex)
        {
            this.startIndex = startIndex;
            this.endIndex = endIndex;
        }

        public void run()
        {
            int maxDivisors = 0;
```

```
int maxNumber = 0;
int divisors;
for (int i = startIndex; i < endIndex; i++)
{
    divisors = 0;

    for (int j = 1; j <= i ; j++)
    {
        if (i%j == 0 )
            divisors++;
    }

    if (divisors > maxDivisors)
    {
        maxDivisors = divisors;
        maxNumber = i;
    }
}
compare(maxDivisors,maxNumber);
}

public void compare(int ThreadMaxDivisors, int ThreadMaxNumber)
{
    if (ThreadMaxDivisors > maxDivisors)
    {
        maxDivisors = ThreadMaxDivisors;
        maxNumber = ThreadMaxNumber;
    }
}
```



```
public void countDivisors()
{
    int range = 1000000 ,noOfThreads=10;

    startingTime = System.currentTimeMillis();

    DivisorsThread[] threads = new DivisorsThread[noOfThreads];
    int n = range/noOfThreads;
    int start = 1;
    int end = start-1 + n;
    for (int i = 0; i < noOfThreads; i++)
    {
        threads[i] = new DivisorsThread(start, end);
        start = end+1;
        end = start + n - 1;

        threads[i].start();

        try
        {
            threads[i].join();
        }
        catch (Exception e) {}
    }
    totalTime = System.currentTimeMillis() - startingTime;
}

public class q6
```

```
{  
    public static void main(String[] args)  
    {  
        Divisors d = new Divisors();  
        d.countDivisors();  
        System.out.println("Max Number: " + d.maxNumber);  
        System.out.println("Max no:of Divisors " + d.maxDivisors);  
        System.out.println("Elapsed time: " + (d.totalTime/1000.0) + " sec");  
    }  
}
```

OUTPUT:

```
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>javac q6.java  
C:\Gokul\VIT\SEM-4\CSE1007 - Java\Lab\Lab10>java q6  
Max Number: 83160  
Max no:of Divisors 128  
Elapsed time: 2.729 sec
```