LAB EXERCISE ON JAVA THREADS
NAME : SHERAL SIMON WASKAR
REG. NO : 20BCE1182
COURSE CODE : - CSE 1007 LAB
COURSE : JAVA PROGRAMMING LAB
SLOT : L13-L14
FACULTY : J V THOMAS ABRAHAM
DATE : 23/03/2022

1. Write a short program that prints "Hello world" from a thread.
Now modify the program to print "Hello world" five times, once from each of five different threads.
Now modify the printed string to include the thread number; ensure that all threads have a unique thread number.

Code :

```
class Print extends Thread
{

public void run()

{

System.out.print("Hello World ");

}

}

public class Test
{

public static void main(String[] args)

{

Print p = new Print();

p.start();


}
}
```

Output :

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

Code :

```
class Print extends Thread
{

public void run()

{

System.out.println("Hello World ");

}

}

public class Test
{

public static void main(String[] args)

{

Print p1 = new Print();
p1.start();

Print p2 = new Print();
p2.start();

Print p3 = new Print();
p3.start();

Print p4 = new Print();
p4.start();

Print p5 = new Print();
p5.start();



}
}
```

Output :

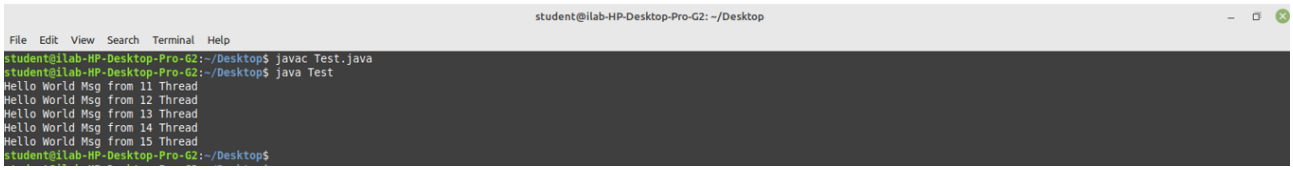
```
student@ilab-HP-Desktop-Pro-G2:~/Desktop$ javac Test.java
student@ilab-HP-Desktop-Pro-G2:~/Desktop$ java Test
Hello World
Hello World
Hello World
Hello World
Hello World
student@ilab-HP-Desktop-Pro-G2:~/Desktop$
```

Code :

```
import java.util.Scanner;

class Print extends Thread
{

public void run()

{

System.out.println("Hello World  Msg from " +Thread.currentThread().getId() + " Thread ");


}

}

public class Test
{

public static void main(String[] args) throws InterruptedException

{

Print p1 = new Print();
p1.start();
p1.join();

Print p2 = new Print();
p2.start();
p2.join();


Print p3 = new Print();
p3.start();
p3.join();


Print p4 = new Print();
p4.start();
p4.join();
```

```
Print p5 = new Print();
p5.start();




}
}
```

Output :



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

Without Synchronization

Code:

```
public class WithoutSyn
{

public static void main(String[] args)throws InterruptedException
{

Counter c = new Counter();

Thread t1  = new Thread(new Runnable()
{
public void run()
{

for(int i=0;i<1000000;i++)
c.incr();
}

}
);

Thread t2 = new Thread(new Runnable()
{
public void run()
{

for(int i=0;i<1000000;i++)
c.incr();
}
```

```
}
);
t1.start();
t2.start();
t1.join();
t2.join();

System.out.println("Count = " + c.count);
}

}
class Counter
{

int count ;

void incr()
{
count++;
}
}
```

Output :

```
D:\SEM 4\CSE1007>javac WithoutSyn.java

D:\SEM 4\CSE1007>java WithoutSyn
Count = 1912655
```

With Synchronization

Code:

```
public class SharedInteger
{

public static void main(String[] args)throws InterruptedException
{

Counter c = new Counter();

Thread t1  = new Thread(new Runnable()
{
public void run()
{

for(int i=0;i<1000000;i++)
c.incr();
}

}
```

```
);

Thread t2 = new Thread(new Runnable()
{
public void run()
{

for(int i=0;i<1000000;i++)
c.incr();
}

}
);
t1.start();
t2.start();
t1.join();
t2.join();

System.out.println("Count = " + c.count);
}

}
class Counter
{

int count ;

synchronized void incr()
{
count++;
}
}
```

Output :

```
D:\SEM 4\CSE1007>javac SharedInteger.java

D:\SEM 4\CSE1007>java SharedInteger.java
Count = 2000000
```

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 :

```
import java.util.LinkedList;

public class ThreadExample {
```

```
public static void main(String[] args)
    throws InterruptedException
{

    final PC pc = new PC();

    // Create producer thread
    Thread t1 = new Thread(new Runnable() {

        public void run()
        {
            try {
                pc.produce();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    });

    // Create consumer thread
    Thread t2 = new Thread(new Runnable() {

        public void run()
        {
            try {
                pc.consume();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    });

    // Start both threads
    t1.start();
    t2.start();

    // t1 finishes before t2
    t1.join();
    t2.join();
}


public static class PC {


    LinkedList<Integer> list = new LinkedList<>();
    int capacity = 2;

    // Function called by producer thread
    public void produce() throws InterruptedException
```

```
        {
            int value = 0;
            while (true) {
                synchronized (this)
                {

                    while (list.size() == capacity)
                        wait();

                    System.out.println("Producer produced-"
                                       + value);


                    list.add(value++);


                    notify();


                    Thread.sleep(1000);
                }
            }
        }

        // Function called by consumer thread
        public void consume() throws InterruptedException
        {
            while (true) {
                synchronized (this)
                {

                    while (list.size() == 0)
                        wait();


                    int val = list.removeFirst();

                    System.out.println("Consumer consumed-"
                                       + val);

                    // Wake up producer thread
                    notify();

                    // and sleep
                    Thread.sleep(1000);
                }
            }
        }
    }
}
```

Output :

```
D:\SEM 4\CSE1007>javac ThreadExample.java

D:\SEM 4\CSE1007>java ThreadExample
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
Producer produced-6
Producer produced-7
Consumer consumed-6
Consumer consumed-7
Producer produced-8
Producer produced-9
Consumer consumed-8
Consumer consumed-9
Producer produced-10
Producer produced-11
Consumer consumed-10
Consumer consumed-11
Producer produced-12
Producer produced-13
Consumer consumed-12
Consumer consumed-13
Producer produced-14
Producer produced-15
Consumer consumed-14
Consumer consumed-15
Producer produced-16
Producer produced-17
Consumer consumed-16
Consumer consumed-17
Producer produced-18
Producer produced-19
Consumer consumed-18
Producer produced-20
Consumer consumed-19
Consumer consumed-20
Producer produced-21
Producer produced-22
Consumer consumed-21
```

Infinite loop is created

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

Code:

```
public class DataRace extends Thread
{

private static volatile int count = 0 ;

public void run()
{

int x = count;
count = x + 1;
System.out.println(count);
}

public static void main(String[] args)
{

Thread t1  = new DataRace();

Thread t2 = new DataRace();

t1.start();

t2.start();
```

```
}

}
```

Output :

```
D:\SEM 4\CSE1007>javac DataRace.java

D:\SEM 4\CSE1007>java DataRace
2
2
```

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 MaxNumOfDivisors {

  public static void main(String[] args) {

    int N;
    int maxDivisors;
    int numWithMax;

    maxDivisors = 1;
    numWithMax = 1;



    for ( N = 2;  N <= 10000;  N++ ) {

      int D;
      int divisorCount;

      divisorCount = 0;

      for ( D = 1;  D <= N;  D++ ) {
        if ( N % D == 0 )
          divisorCount++;
      }

      if (divisorCount > maxDivisors) {
        maxDivisors = divisorCount;
        numWithMax = N;
      }

    }

    System.out.println("Among integers between 1 and 10000,");
    System.out.println("The maximum number of divisors is " + maxDivisors);
    System.out.println("A number with " + maxDivisors + " divisors is " + numWithMax);
```

```
  }

}
```

Output :

```
D:\SEM 4\CSE1007>javac MaxNumOfDivisors.java

D:\SEM 4\CSE1007>java MaxNumOfDivisors
Among integers between 1 and 10000,
The maximum number of divisors is 64
A number with 64 divisors is 7560
```

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 :

```
public class ThreadDiv {

  public static void main(String[] args) {
  long startTime = System.currentTimeMillis();


    Thread t1 = new Thread(new Runnable()
{
public void run()
{

  int N;
     int maxDivisors;
     int numWithMax;

     maxDivisors = 1;
     numWithMax = 1;
     for ( N = 1;  N <= 20000;  N++ ) {

        int D;
        int divisorCount;

        divisorCount = 0;

        for ( D = 1;  D <= N;  D++ ) {
          if ( N % D == 0 )
             divisorCount++;
        }

        if (divisorCount > maxDivisors) {
          maxDivisors = divisorCount;
```

```
            numWithMax = N;
          }

        }



    }

}
);

     Thread t2 = new Thread(new Runnable()
{
public void run()
{
   int N;
       int maxDivisors;
       int numWithMax;

       maxDivisors = 1;
       numWithMax = 1;

       for ( N = 20001;  N <= 40000;  N++ ) {

          int D;
          int divisorCount;

          divisorCount = 0;

          for ( D = 1;  D <= N;  D++ ) {
            if ( N % D == 0 )
               divisorCount++;
          }

          if (divisorCount > maxDivisors) {
            maxDivisors = divisorCount;
            numWithMax = N;
          }

        }



    }

}
);
 Thread t3 = new Thread(new Runnable()
{
public void run()
{
```

```
      int N;
      int maxDivisors;
      int numWithMax;

      maxDivisors = 1;
      numWithMax = 1;
      for ( N = 40001;  N <= 60000;  N++ ) {

         int D;
         int divisorCount;

         divisorCount = 0;

         for ( D = 1;  D <= N;  D++ ) {
           if ( N % D == 0 )
              divisorCount++;
         }

         if (divisorCount > maxDivisors) {
           maxDivisors = divisorCount;
           numWithMax = N;
         }

      }



   }

}
);
 Thread t4 = new Thread(new Runnable()
{
public void run()
{
   int N;
      int maxDivisors;
      int numWithMax;

      maxDivisors = 1;
      numWithMax = 1;

      for ( N = 60001;  N <= 80000;  N++ ) {

         int D;
         int divisorCount;

         divisorCount = 0;

         for ( D = 1;  D <= N;  D++ ) {
           if ( N % D == 0 )
              divisorCount++;
```

```
        }

        if (divisorCount > maxDivisors) {
          maxDivisors = divisorCount;
          numWithMax = N;
        }

      }


    }

}
);
 Thread t5 = new Thread(new Runnable()
{
public void run()
{
        int N;
      int maxDivisors;
      int numWithMax;

      maxDivisors = 1;
      numWithMax = 1;
      for ( N = 80001;  N <= 100000;  N++ ) {

        int D;
        int divisorCount;

        divisorCount = 0;

        for ( D = 1;  D <= N;  D++ ) {
          if ( N % D == 0 )
            divisorCount++;
        }

        if (divisorCount > maxDivisors) {
          maxDivisors = divisorCount;
          numWithMax = N;
        }

      }

      System.out.println("Among integers between 1  and 100000,");
      System.out.println("The maximum number of divisors is " + maxDivisors);
      System.out.println("A number with " + maxDivisors + " divisors is " + numWithMax);

    }

}
);
```

```
t1.start();
t2.start();
t3.start();
t4.start();
t5.start();

try
{
t1.join();
t2.join();
t3.join();
t4.join();
t5.join();
}
catch(InterruptedException e)
{
System.out.println(e);
}

long elapsedTime = System.currentTimeMillis() - startTime;
System.out.println("Total elapsed time:  " +
        (elapsedTime/1000.0) + " seconds.\n");
}
}
```

Output :

```
D:\SEM 4\CSE1007_LAB>javac ThreadDiv.java

D:\SEM 4\CSE1007_LAB>java ThreadDiv.java
Among integers between 1  and 100000,
The maximum number of divisors is 128
A number with 128 divisors is 83160
Total elapsed time:  5.556 seconds.
```