CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: Java Programming
Semester: 3
Subject Code: CSE201
Academic year: 2024 - 25

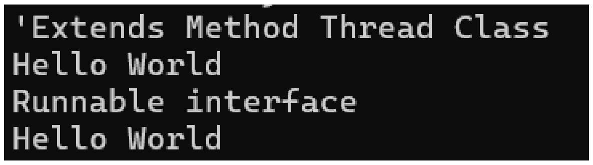
PART – 7 (Multithreading)

NO.	**Aim of the Practical**

32.	Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.
	
	PROGRAM CODE:
	
	// PRACTICAL 32
	import java.util.Scanner;
	
	class hellow extends Thread
	{
	public void run()
	{
	System.out.println("'Extends Method Thread Class");
	System.out.println("Hello World");
	}
	}
	class hworld implements Runnable
	{
	public void run()
	{
	System.out.println("Runnable interface");
	System.out.println("Hello World");
	}
	}
	public class pra32

```
 {
   public static void main(String[] args)
{
       // for  Thread class
       hellow thread1 = new hellow();
       thread1.start();

        hworld  runnable = new hworld();
       Thread thread2 = new Thread(runnable);
       thread2.start();
   }
}
```

OUTPUT:



```
'Extends Method Thread Class
Hello World
Runnable interface
Hello World
```

CONCLUSION:
In This Practical We Learnt About Extend And Runnable Interface Thread.

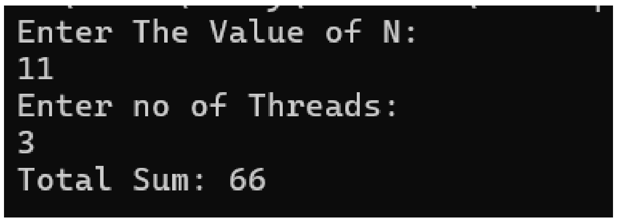
33. Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.

PROGRAM CODE :

```
import java.util.Scanner;

public class pra33
{
   public static void main(String[] args)
{
       Scanner sc = new Scanner(System.in);
       System.out.println("Enter The Value of N:");
       int N = sc.nextInt();
       System.out.println("Enter no of Threads:");
       int noThread = sc.nextInt();

       int[] numbers = new int[N];
       for (int i = 0; i < N; i++) {
           numbers[i] = i + 1;
       }
```

```
        SumTask[] tasks = new SumTask[noThread];
        Thread[] threads = new Thread[noThread];
        int total = N / noThread;
        int start = 0;

        for (int i = 0; i < noThread; i++)
{
            int end = (i == noThread - 1) ? N : start + total;
            tasks[i] = new SumTask(numbers, start, end);
            threads[i] = new Thread(tasks[i]);
            threads[i].start();
            start = end;
 }

        long totalSum = 0;
        for (int i = 0; i < noThread; i++)
 {
            Try
 {
                threads[i].join();
                totalSum = totalSum + tasks[i].getSum();
  }
catch (InterruptedException e)
{
                e.printStackTrace();
            }
        }

        System.out.println("Total Sum: " + totalSum);
    }
}

class SumTask implements Runnable
{
    int[] numbers;
    int start;
    int end;
    long sum;

    public SumTask(int[] numbers, int start, int end)
{
        this.numbers = numbers;
        this.start = start;
        this.end = end;
        this.sum = 0;
    }
    public long getSum()
{
        return sum;
    }
    public void run()
 {
        for (int i = start; i < end; i++) {
            sum += numbers[i];
        }
    }
}
```

OUTPUT:



```
Enter The Value of N:
11
Enter no of Threads:
3
Total Sum: 66
```

CONCLUSION:

In This Practical We Make N Number Of Thread To Complete Task.

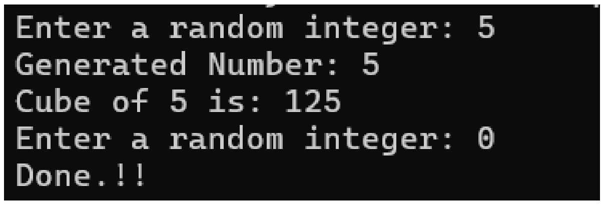
34. Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

PROGRAM CODE:

```
import java.util.Scanner;

class RandomNumberGenerator extends Thread
{
    private Scanner;

    public RandomNumberGenerator(Scanner scanner)
{
        this.scanner = scanner;
    }
    public void run()
{
        try
{
            while (true)
{
                System.out.print("Enter a random integer: ");
                int number = scanner.nextInt();
             if(number ==0)
            {
             System.out.println("Done.!!");
             break;
             }
                System.out.println("Generated Number: " + number);
                if (number % 2 == 0) {
                    new Thread(new Square(number)).start();
                } else {
                    new Thread(new Cube(number)).start();
                }
                Thread.sleep(1000);
            }
        }
catch (InterruptedException e)
{
            System.out.println("Generator interrupted");
        }
    }
}

class Square implements Runnable
 {
    private int number;

    public Square(int number)
 {
        this.number = number;
    }

    public void run()
{
        System.out.println("Square of " + number + " is: " + (number * number));
    }
}
```

```
class Cube implements Runnable {
    private int number;

    public Cube(int number)
{
        this.number = number;
    }

    public void run()
 {
        System.out.println("Cube of " + number + " is: " + (number * number * number));
    }
}

public class pra34
{
    public static void main(String[] args)
{
        Scanner = new Scanner(System.in);
        RandomNumberGenerator generator = new RandomNumberGenerator(scanner);
        generator.start();
    }
}
```

OUTPUT:



```
Enter a random integer: 5
Generated Number: 5
Cube of 5 is: 125
Enter a random integer: 0
Done.!!
```

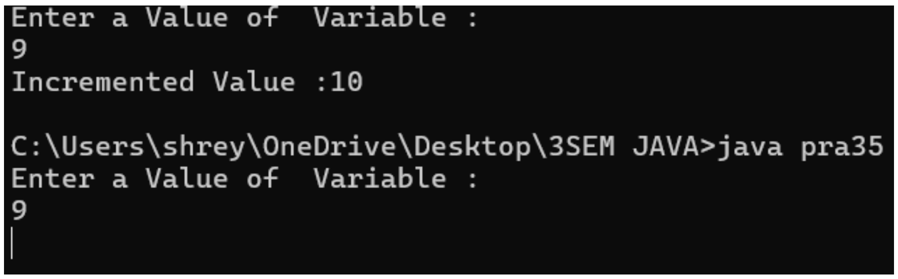
CONCLUSION:
In This Practical We Take Radom Integer And Print Square And Cube.

35. Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.

PROGRAM CODE

```
import java.util.Scanner;
public class pra35
{
public static void main(String[] args)
{
   Scanner sc = new Scanner(System.in);
   System.out.println("Enter a Value of  Variable :");
  int value = sc.nextInt();
Runnable increment = new Runnable()
{
public void run(){
try {
Thread.sleep(5000);
System.out.println("Incremented Value :" + (value+1));
}
catch (InterruptedException e)
{
System.out.println("The Interrupt was Occurred in Thread.");
}
}
};
Thread incrementThread = new Thread(increment);
incrementThread.start();
}
}
```

OUTPUT:



```
Enter a Value of  Variable :
9
Incremented Value :10

C:\Users\shrey\OneDrive\Desktop\3SEM JAVA>java pra35
Enter a Value of  Variable :
9
|
```

CONCLUSION:

In This Practical We use sleep method.

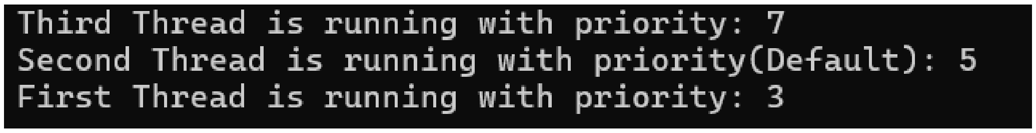
36. Write a program to create three threads 'FIRST', 'SECOND', 'THIRD'. Set the priority of the 'FIRST' thread to 3, the 'SECOND' thread to 5(default) and the 'THIRD' thread to 7.

PROGRAM CODE:

```
class FirstThread extends Thread
{
   public void run()
 {
       System.out.println("First Thread is running with priority: " + this.getPriority());
   }
}

class SecondThread extends Thread
{
   public void run()
{
       System.out.println("Second Thread is running with priority: " + this.getPriority());
   }
}

class ThirdThread extends Thread
{
   public void run()
{
       System.out.println("Third Thread is running with priority: " + this.getPriority());
   }
}

public class pra36
{
   public static void main(String[] args)
{
       FirstThread first = new FirstThread();
       SecondThread second = new SecondThread();
       ThirdThread third = new ThirdThread();

       first.setPriority(3);
       second.setPriority();
       third.setPriority(7);

       first.start();
       second.start();
```

```
        third.start();
    }
}
```

OUTPUT:



```
Third Thread is running with priority: 7
Second Thread is running with priority(Default): 5
First Thread is running with priority: 3
```

CONCLUSION:

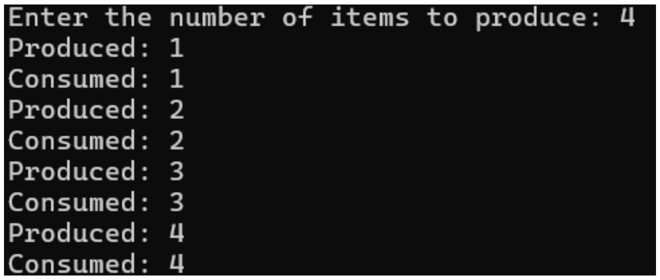
In This Practical We Put On Threads Priorities.

37. Write a program to solve producer-consumer problem using thread synchronization.

PROGRAM CODE:

```
import java.util.Scanner;

class ProducerConsumer {
    private int item = -1; // Shared item
    private boolean isProduced = false;

    synchronized void produce(int itemCount) {
        for (int i = 0; i < itemCount; i++) {
            while (isProduced) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
            }
            item = i + 1; // Produce item (starting from 1)
            isProduced = true;
            System.out.println("Produced: " + item);
            notifyAll();
        }
        synchronized (this) {
            while (isProduced) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
            }
            item = 0; // End signal for consumer
            isProduced = true;
            notifyAll();
        }
    }

    synchronized void consume() {
        while (true) {
            while (!isProduced) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
            }
            if (item == 0) {
                break; // End signal received
            }
            System.out.println("Consumed: " + item);
            isProduced = false;
            notifyAll();
        }
    }
}

class Producer extends Thread {
    ProducerConsumer pc;
    int itemCount;

    Producer(ProducerConsumer pc, int itemCount) {
        this.pc = pc;
        this.itemCount = itemCount;
    }

    public void run() {
        pc.produce(itemCount);
    }
}

class Consumer extends Thread {
    ProducerConsumer pc;

    Consumer(ProducerConsumer pc) {
        this.pc = pc;
    }

    public void run() {
        pc.consume();
    }
}

public class TestSynchronization {
    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of items to produce: ");
        int itemCount = scanner.nextInt();

        ProducerConsumer pc = new ProducerConsumer();
        Producer producer = new Producer(pc, itemCount);
        Consumer consumer = new Consumer(pc);

        producer.start();
        consumer.start();
    }
}
```

OUTPUT:



```
Enter the number of items to produce: 4
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
```

CONCLUSION:

In This Practical We Performed Synchronization with threads