CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science Engineering

Subject Name: Java programming
Semester: III
Subject Code: CSE201
Academic year: 2024-25

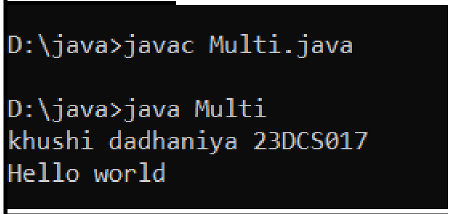
Part - 7

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:

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

        }
        public static void main(String args[])
        {
                Multi t1=new Multi();
                t1.start();
                 System.out.println("khushi dadhaniya 23DCS017");
        }
}
```

OUTPUT:



```
D:\java>javac Multi.java

D:\java>java Multi
khushi dadhaniya 23DCS017
Hello world
```

CONCLUSION:

In this Java program, we create a **Multi** class that extends the **Thread** class.

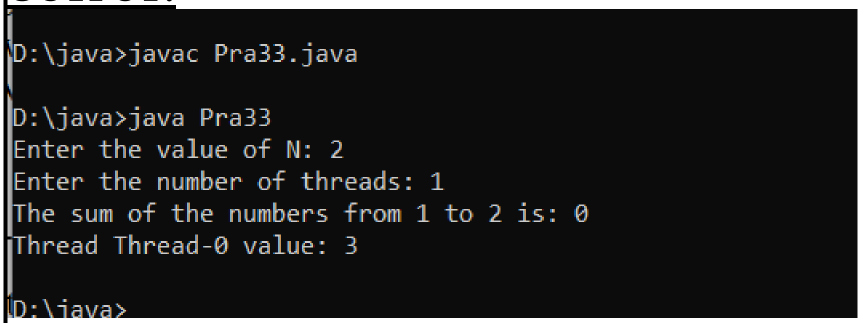
The **run()** method is overridden to print the "Hello world" message. In the **main()** method, we create an instance of the **Multi** class, **t1**, and start the thread using the **start()** method.

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;

class SumThread extends Thread {
     int start;
     int end;
     int result;


    public SumThread(int start, int end) {
        this.start = start;
        this.end = end;
    }

    public void run() {
        result = 0;
        for (int i = start; i <= end; i++) {
            result += i;
        }
        System.out.println("Thread " + Thread.currentThread().getName() + " value: "
+ result);
    }
}

public class Pra33 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the value of N: ");
        int n = s.nextInt();
        System.out.print("Enter the number of threads: ");
        int Threads = s.nextInt();

        int Size = n / Threads;
        SumThread[] threads = new SumThread[Threads];
        for (int i = 0; i < Threads; i++) {
            int start = i * Size + 1;
            int end = (i == Threads - 1) ? n : (i + 1) * Size;
            threads[i] = new SumThread(start, end);
            threads[i].start();
        }



        int sum = 0;
        for (SumThread thread : threads) {
            sum += thread.result;
        }

        System.out.println("The sum of the numbers from 1 to " + n + " is: " + sum);
    }
}
```

OUTPUT:



```
D:\java>javac Pra33.java

D:\java>java Pra33
Enter the value of N: 2
Enter the number of threads: 1
The sum of the numbers from 1 to 2 is: 0
Thread Thread-0 value: 3

D:\java>
```

CONCLUSION:

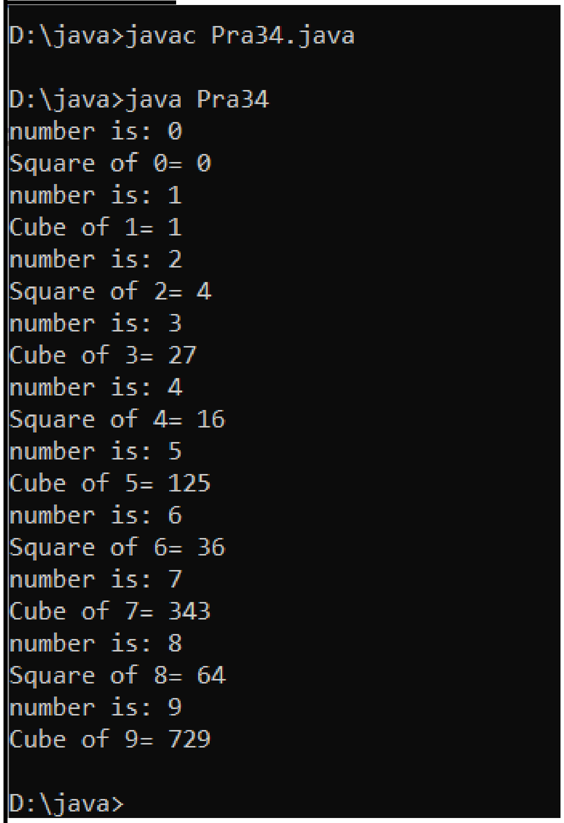
The provided Java program is designed to calculate the sum of numbers from 1 to **n** using multiple threads. The program prompts the user to input the value of **n** and the number of threads to use.

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 Even extends Thread {
    private int n;
    public Even(int n) {
        this.n = n;
    }

    public void run() {

            int square = n * n;
            System.out.println("Square of " + n + "= " + square);

    }
}

class Odd extends Thread {
    private int n;

    public Odd(int n) {
        this.n = n;
    }

    public void run() {

            int cube = n * n * n;
            System.out.println("Cube of " + n + "= " + cube);

    }
}

public class Pra34 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        for (int i = 0; i < 10; i++) {

            System.out.println("number is: " + i);
            if (i % 2 == 0) {
                Even e2 = new Even(i);
                e2.start();
            } else {
                Odd o3 = new Odd(i);
                o3.start();
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

OUTPUT:



```
D:\java>javac Pra34.java

D:\java>java Pra34
number is: 0
Square of 0= 0
number is: 1
Cube of 1= 1
number is: 2
Square of 2= 4
number is: 3
Cube of 3= 27
number is: 4
Square of 4= 16
number is: 5
Cube of 5= 125
number is: 6
Square of 6= 36
number is: 7
Cube of 7= 343
number is: 8
Square of 8= 64
number is: 9
Cube of 9= 729

D:\java>
```

CONCLUSION:

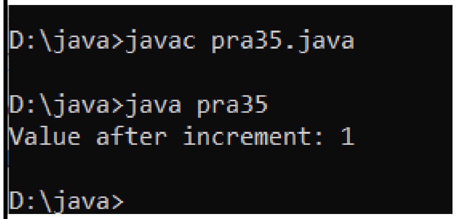
The provided Java program is designed to calculate the sum of numbers from 1 to **n** using multiple threads. The program prompts the user to input the value of **n** and the number of threads to use.

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:

```
class IncrementThread extends Thread {
    int value;

    public IncrementThread(int value) {
        this.value = value;
    }

    public void run() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted");
        }
        value++;
        System.out.println("Value after increment: " + value);
    }
}

public class pra35 {
    public static void main(String[] args) {
        IncrementThread thread = new IncrementThread(0);
        thread.start();
    }
}
```

OUTPUT:



```
D:\java>javac pra35.java

D:\java>java pra35
Value after increment: 1

D:\java>
```

CONCLUSION:

The provided Java program demonstrates the use of threads to increment a variable by one and display the result after a 1-second delay. The program utilizes the **sleep()** method to pause the thread's execution for 1 second, allowing for a delay between the start of the thread and the increment operation.

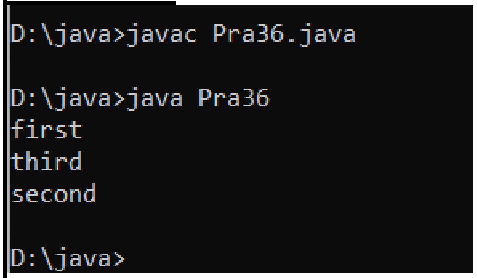
36. Write a program to create three threads ‘FIRST’, ‘SECOND’, ‘THIRD’. Set the priority of the ‘FIRST’ thread to 3, the ‘SECOND’ thread to 5(default) and the ‘THIRD’ thread to 7.

PROGRAM CODE:

```
class First extends Thread {


    public void run() {


            System.out.println("first" );
    }
}

class Second extends Thread {
    public void run() {


            System.out.println("second" );

    }
}
class Third extends Thread {

    public void run() {


            System.out.println("third");

    }
}

public class Pra36 {
    public static void main(String[] args) {
          First f1 = new First();
          Second s1 = new Second();
          Third t1 = new Third();


f1.getPriority();
s1.getPriority();
t1.getPriority();




f1.setPriority(5);
s1.setPriority(4);
t1.setPriority(8);

          f1.start();
    s1.start();
    t1.start();
```

```
        }
}
```

OUTPUT:



```
D:\java>javac Pra36.java

D:\java>java Pra36
first
third
second

D:\java>
```

CONCLUSION:

The program creates three threads, 'FIRST', 'SECOND', and 'THIRD', with varying priorities. The 'FIRST' thread has a priority of 3, the 'SECOND' thread has a default priority of 5, and the 'THIRD' thread has a priority of 7. This demonstrates the use of thread priorities in Java, allowing developers to control the order of thread execution based on their priority levels.

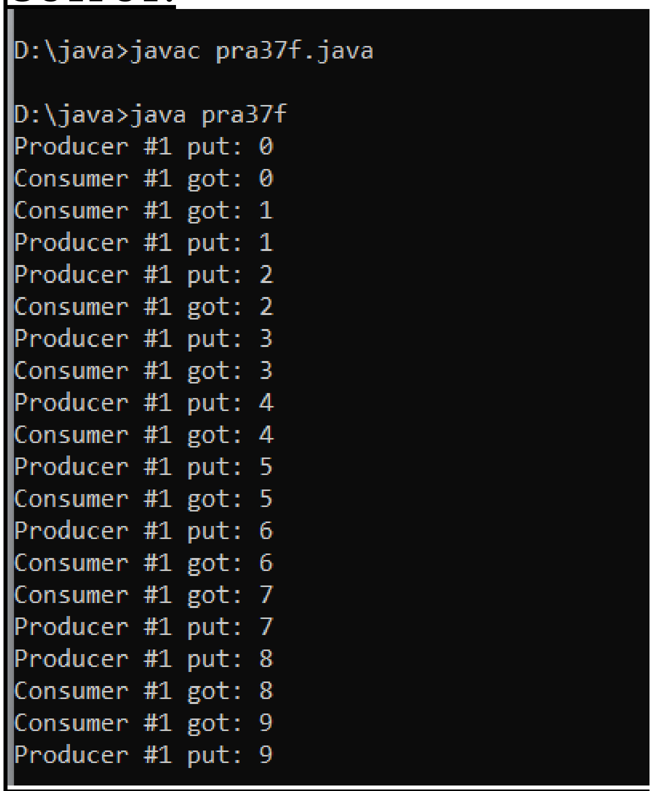
37. Write a program to solve producer-consumer problem using thread synchronization.

PROGRAM CODE:

```
public class pra37f {
  public static void main(String[] args) {
    check c = new check();
    Producer p1 = new Producer(c, 1);
    Consumer c1 = new Consumer(c, 1);
    p1.start();
    c1.start();
  }
}
class check {
  private int contents;
  private boolean available = false;

  public synchronized int get() {
    while (available == false) {
      try {
        wait();
```

```
            } catch (InterruptedException e) {}
        }
        available = false;
        notifyAll();
        return contents;
    }
    public synchronized void put(int value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        notifyAll();
    }
}
class Consumer extends Thread {
    private check ch;
    private int number;

    public Consumer(check c, int number) {
        ch = c;
        this.number = number;
    }
    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = ch.get();
            System.out.println("Consumer #" + this.number + " got: " + value);
        }
    }
}
class Producer extends Thread {
    private check ch;
    private int number;
    public Producer(check c, int number) {
        ch = c;
        this.number = number;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            ch.put(i);
            System.out.println("Producer #" + this.number + " put: " + i);
            try {
              sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
          }
        }
      }
```

OUTPUT:



```
D:\java>javac pra37f.java

D:\java>java pra37f
Producer #1 put: 0
Consumer #1 got: 0
Consumer #1 got: 1
Producer #1 put: 1
Producer #1 put: 2
Consumer #1 got: 2
Producer #1 put: 3
Consumer #1 got: 3
Producer #1 put: 4
Consumer #1 got: 4
Producer #1 put: 5
Consumer #1 got: 5
Producer #1 put: 6
Consumer #1 got: 6
Consumer #1 got: 7
Producer #1 put: 7
Producer #1 put: 8
Consumer #1 got: 8
Consumer #1 got: 9
Producer #1 put: 9
```

CONCLUSION:

The code provided is a classic example of the producer-consumer problem, where one thread (the producer) produces items and another thread (the consumer) consumes them. The check class acts as a shared buffer between the producer and consumer, and it uses synchronization to ensure that only one thread can access the buffer at a time.