|  |
| --- |
| Experiment No. 10 |
| Implement program on Multithreading |
| Date of Performance: |
| Date of Submission: |

**Aim:** Implement program on Multithreading

**Objective**:

## Theory:

**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Java provides **Thread class** to achieve thread programming. Thread class provides [constructors](https://www.javatpoint.com/java-constructor) and methods to create and perform operations on a thread. Thread class extends [Object class](https://www.javatpoint.com/object-class) and implements Runnable interface.

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### **Thread class:**

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

### **1) Java Thread Example by extending Thread class**

**FileName:** Multi.java

**class** Multi **extends** Thread{

**public** **void** run(){

System.out.println("thread is running...");

}

**public** **static** **void** main(String args[]){

Multi t1=**new** Multi();

t1.start();

 }  }

**Output**

thread is running...

### **2) Java Thread Example by implementing Runnable interface**

**FileName:** Multi3.java

**class** Multi3 **implements** Runnable{

**public** **void** run(){

System.out.println("thread is running...");

}

**public** **static** **void** main(String args[]){

Multi3 m1=**new** Multi3();

Thread t1 =**new** Thread(m1);   // Using the constructor Thread(Runnable r)

t1.start();

 }

}

**Output:**

thread is running...

**Code:**

class MultithreadingDemo extends Thread {

public void run()

{

try {

// Displaying the thread that is running

System.out.println(

"Thread " + Thread.currentThread().getId()

+ " is running");

}

catch (Exception e) {

// Throwing an exception

System.out.println("Exception is caught");

}

}}

// Main Class

public class Multithread {

public static void main(String[] args)

{

int n = 8; // Number of threads

for (int i = 0; i < n; i++) {

MultithreadingDemo object

= new MultithreadingDemo();

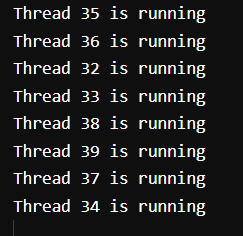
object.start();

}

}

}

**Output:**

****

## Conclusion:

Multithreading in Java enables programs to perform multiple tasks simultaneously, making better use of available resources. By creating and managing threads using the `Thread` class or the `Runnable` interface, you can achieve parallel execution. This is particularly useful for tasks involving I/O operations or concurrent processing. To ensure program stability, it's crucial to address synchronization and thread safety. Overall, multithreading enhances program performance and responsiveness, making it an essential concept in Java for efficient and responsive applications.