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

Aim: Implement program on Multithreading **Objective**:

Theory:

Multithreading in Java 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 and methods to create and perform operations on a thread. Thread class extends 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 Multithreading {   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();
 }
 }
}
class MultithreadingDemo extends Thread {
public void run()
 {
tr
y
{
 // 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");
 }
 }}
```

Conclusion:

Multi-threading is supported through the use of the Thread class and the Runnable interface. To create a new thread, we can either extend the Thread class or implement the Runnable interface and override the run() method. Once the thread is created, we can start it by calling the start() method. This allows the thread to run concurrently with other threads in the program. Java also provides synchronization mechanisms such as synchronized blocks and locks to handle thread safety and prevent race conditions. Multi-threading in Java allows for parallel execution of tasks, which can improve performance and responsiveness in applications that require concurrent processing or handling of multiple tasks simultaneously.