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

Aim: Implement program on Multithreading

Objective: To Implement program on Multithreading.

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 A extends Thread {public void run() {
      for (int i = 1; i<= 5; i++) { System.out.println("\n From Thread A:i=" + i);
      }

System.out.println("Exit from A");

   }

}

class B extends Thread {public void run()
```

```
for (int j = 1; j <= 5; j++) {
 System.out.println("\n From Thread B:j=" + j);
        }
 System.out.println("Exit from B");
    }
 }
 class C extends Thread {public void run() {
        for (int k = 1; k <= 5; k++) { System.out.println("\n From Thread C:k=" + k);
        }
 System.out.println("Exit from C");
    }
 }
 class Multithreading {
    public static void main(String args[]) {new A().start();
        new B().start();
        new C().start();
    }
 }
```

Output:

```
java -cp /tmp/fUXfxaeVjS Multithreading
From Thread A:i=1
From Thread A:i=2

 From Thread A:i=3

 From Thread A:i=4

 From Thread A:i=5
Exit from A
From Thread B:j=1From Thread B:j=2
From Thread B:j=3
From Thread C:k=1

 From Thread C:k=2

 From Thread C:k=3From Thread B:j=4
From Thread C:k=4

 From Thread B:j=5
Exit from B
```

```
 From Thread B:j=5
Exit from B

 From Thread C:k=5
Exit from C
```

Conclusion:

Java supports multithreading through the Thread class. The Thread class represents a thread of execution. A thread is a lightweight process that can execute independently of other threads. Threads share the same memory space as the main process, but they have their own stack.

Threads can communicate with each other through shared variables. However, it is important to synchronize access to shared variables to avoid race conditions. A race condition is a situation where two threads are trying to access and modify the same variable at the same time.