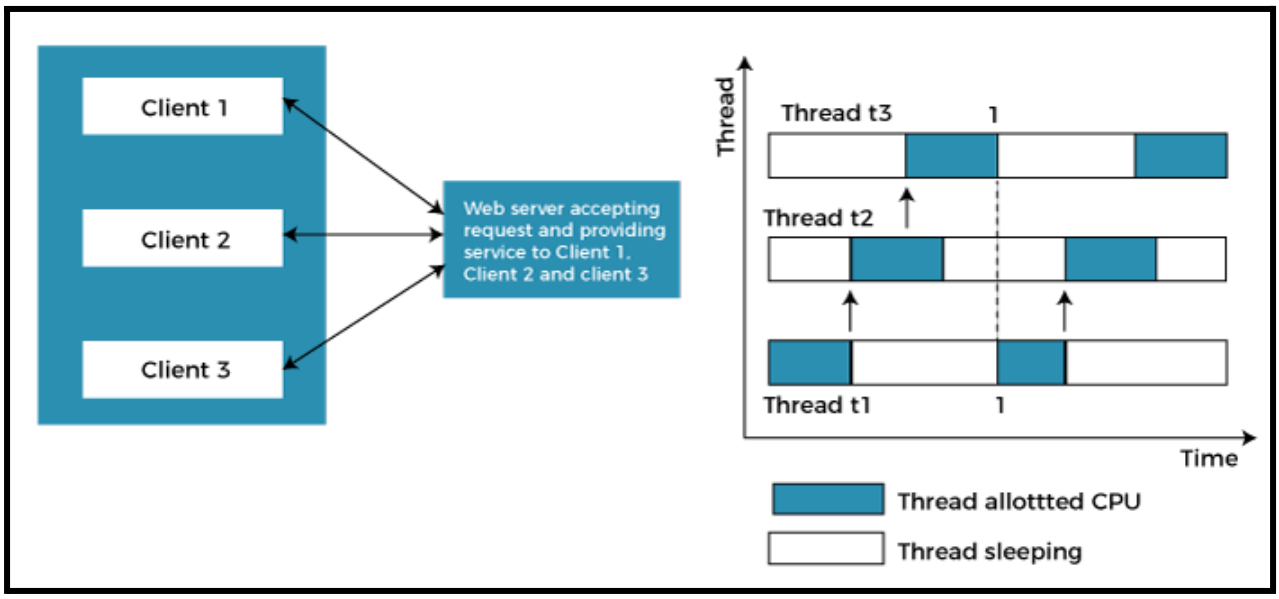
EXPERIMENT 2

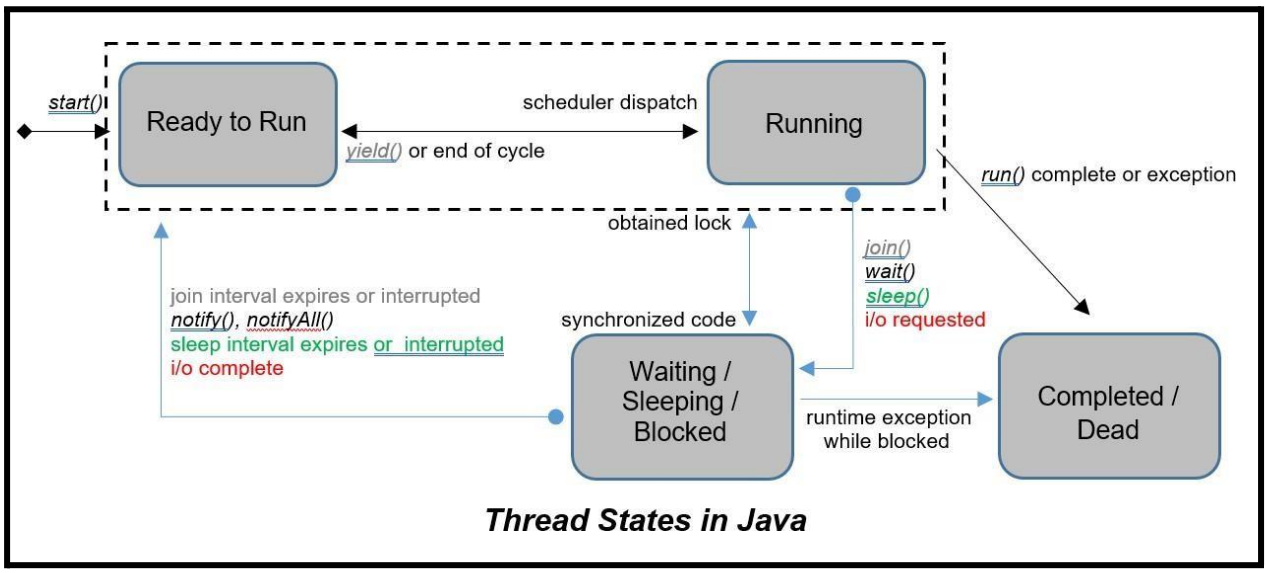
Aim: Implementation of multi thread application

Theory: Multi-threading:

Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer. Multithreading can also handle multiple requests from the same user.

Each user request for a program or system service is tracked as a thread with a separate identity. As programs work on behalf of the initial thread request and are interrupted by other requests, the work status of the initial request is tracked until the work is completed. In this context, a user can also be another program.





Types :

Single thread process: Only one thread in an entire process Multi-thread process: Multiple threads within an entire process.

Advantages of Multithreading:

- Improves execution speed (by combining CPU machine and I/O wait times).
- Multithreading can be used to achieve concurrency.
- Reduces the amount of time it takes for context switching.
- Improves responsiveness.
- Make synchronous processing possible (separates the execution of independent threads).
- Increases throughput.
- Performs foreground and background work in parallel.

Disadvantages of Multithreading:

- Because the threads share the same address space and may access resources such as open files, difficulties might arise in managing threads if they utilize incompatible data structures.
- If we don't optimize the number of threads, instead of gaining performance, we may lose performance.
- If a parent process requires several threads to function properly, the child processes should be multithreaded as well, as they may all be required.

Models :

1. Many-to-One: There will be a many-to-one relationship model between threads, as the name implies. Multiple user threads are linked or mapped to a single kernel thread in this case. Management of threads is done on a user-by-user basis, which makes it more efficient. It converts a large number of user-level threads into a single Kernel-level thread.

2. One-to-One: We can deduce from the name that one user thread is mapped to one separate kernel thread. The user-level thread and the kernel-level thread have a one-to-one relationship. The many-to-one model for thread provides less concurrency than this architecture. When a thread performs a blocking system call, it also allows another thread to run.

3. Many-to-Many: From the name itself, we may assume that there are numerous user threads mapped to a lesser or equal number of kernel threads. This model has a version called a two-level model, which incorporates both many-to-many and one-to-one relationships. When a thread makes a blocked system call, the kernel can schedule another thread for execution.

Code:

```
class MyThread extends Thread {
public void run() { try {
synchronized (System.out) {
            System.out.println("Thread " + Thread.currentThread().getId() + " is running");
        }
        Thread.sleep(1000); synchronized
        (System.out) {
           System.out.println("Thread " + Thread.currentThread().getId() + " has completed"); }
    } catch (InterruptedException e) {
        synchronized (System.out) {
           System.out.println("Thread interrupted");
        }}}}

class MyRunnable implements Runnable {
   public void run() { try {
        synchronized (System.out) {
           System.out.println("Runnable Thread " + Thread.currentThread().getId() + " is
running");
        }
```

```
        Thread.sleep(1000); synchronized
        (System.out) {
            System.out.println("Runnable Thread " + Thread.currentThread().getId() + " has
completed");
        }
      } catch (InterruptedException e) { synchronized
        (System.out) {
            System.out.println("Runnable Thread interrupted");
        }}}}

public class MultithreadingExample {
   public static void main(String[] args) {
   int n = 5; // Number of threads
      System.out.println("Using Thread class:"); for
      (int i = 0; i < n; i++) {
         MyThread thread = new MyThread();
         thread.start();
      }
      System.out.println("\nUsing Runnable interface:"); for
      (int i = 0; i < n; i++) {
         Thread thread = new Thread(new MyRunnable()); thread.start();
      }}}
```

Output:

```
computer@computer-ThinkCentre:~$  /usr/bin/env /usr/lib/jvm/j
Using Thread class:
Thread 12 is running
Thread 16 is running

Using Runnable interface:
Thread 15 is running
Thread 14 is running
Thread 13 is running
Runnable Thread 17 is running
Runnable Thread 20 is running
Runnable Thread 21 is running
Runnable Thread 19 is running
Runnable Thread 18 is running
Thread 16 has completed
Thread 13 has completed
Thread 15 has completed
Thread 14 has completed
Thread 12 has completed
Runnable Thread 17 has completed
Runnable Thread 21 has completed
Runnable Thread 20 has completed
Runnable Thread 18 has completed
Runnable Thread 19 has completed
```