Ex 3	Multiple threads - Thread class, Runnable interface, Thread lifecycle, Priorities, and Synchronization

Aim

To implement multiple threads in Java using the Thread class and Runnable interface and to demonstrate thread lifecycle, priorities, and synchronization.

Definitions

Threads

In Java, a thread represents a single, independent path of execution within a program. It is the smallest unit of execution that can be scheduled by the operating system.

Multiple Threads

Java is a multi-threaded language, meaning it provides built-in support for creating and managing multiple threads, allowing programs to perform multiple tasks concurrently or in parallel. This capability is known as multithreading.

Thread Class

The Thread class in Java, found within the java.lang package, is the fundamental building block for implementing multithreading. It represents a single thread of execution within a program, allowing multiple tasks to run concurrently.

Runnable Interface

The Runnable interface in Java is a core component of its multithreading model, allowing classes to define tasks that can be executed concurrently by separate threads.

Tread Lifecycle

The life cycle of a thread in Java describes the various states a thread transitions through from its creation to its termination. These states are defined by the Thread.State enum and include:

1. **New:** A thread is in the New state immediately after it is created using the new Thread() constructor but before the start() method is invoked. At this point, the thread object exists, but it is not yet eligible to be scheduled for execution by the JVM.
2. **Runnable:** After calling the start() method on a thread, it enters the Runnable state. In this state, the thread is considered "ready to run" and is waiting for the CPU to allocate time for its execution. The thread scheduler determines which Runnable thread gets CPU time. A thread in the Runnable state may or may not be actively executing, depending on the scheduler.
3. **Running:** When the thread scheduler selects a thread from the Runnable pool and allocates CPU time to it, the thread enters the Running state. In this state, the thread is actively executing the code within its run() method.

4. **Blocked:** A thread enters the Blocked state when it is temporarily inactive and waiting to acquire a monitor lock to access a synchronized block or method that is currently held by another thread. It remains blocked until the lock becomes available.
5. **Waiting:** A thread enters the Waiting state when it is waiting for another thread to perform a specific action before it can resume execution. This often involves methods like Object.wait() (without a timeout) or Thread.join() (without a timeout). The thread will remain in this state until it is explicitly notified by another thread or the joined thread terminates.
6. **Timed Waiting:** Similar to Waiting, a thread in the Timed Waiting state is waiting for another thread to perform an action, but with a specified timeout. This occurs when calling methods like Thread.sleep(), Object.wait(long timeout), or Thread.join(long timeout). The thread will transition out of this state either when the action occurs or the timeout expires.
7. **Terminated:** A thread enters the Terminated state (also known as the Dead state) when it has finished its execution, either by completing the run() method or due to an unhandled exception. Once a thread is terminated, it cannot be restarted.

Thread Priorities

Thread priority in Java is a numeric value assigned to a thread, influencing the thread scheduler's decision on which thread to execute next when multiple threads are in a runnable state.

Thread Synchronization

Thread synchronization in Java is a mechanism used to control access to shared resources by multiple threads in a concurrent environment. Its primary purpose is to prevent data inconsistency and race conditions that can arise when multiple threads attempt to modify the same shared data simultaneously.

Procedure

Open NetBeans IDE.

To create a Project go to File Menu → choose New Project → choose Java from Categories →choose Java Application from Projects →click next →specify the project name as Thread → uncheck Create Main Class Check-box → click Finish.

Right click on Source Packages Folder → choose New → select Java Class → specify the class name as ThreadDemo →click Finish.

Type the following codes in ThreadDemo.java:

ThreadDemo.java

```
import java.util.concurrent.TimeUnit;


// Shared resource for synchronization demonstration
class SharedResource {
    private int counter = 0;


    // Synchronized method to ensure atomic updates
    public synchronized void increment() {
        counter++;
        System.out.println(Thread.currentThread().getName() + " incremented counter to: " + counter);
    }


    public int getCounter() {
        return counter;
    }
}
// Thread creation by extending Thread class
class MyThread extends Thread {
    private String threadName;
    private SharedResource resource;
    public MyThread(String name, SharedResource resource) {
        super(name); // Set thread name
```

```
        this.threadName = name;
        this.resource = resource;
        System.out.println("Creating " + threadName + " - State: " + this.getState()); // NEW state
    }
    @Override
    public void run() {
        System.out.println("Running " + threadName + " - State: " + this.getState());
// RUNNABLE/RUNNING state
        try {
            for (int i = 0; i < 3; i++) {
                System.out.println(threadName + ": " + i);
                TimeUnit.MILLISECONDS.sleep(100); // Simulate work and demonstrate BLOCKED/TIMED_WAITING
                resource.increment(); // Accessing shared resource
            }
        } catch (InterruptedException e) {
            System.out.println(threadName + " interrupted.");
        }
        System.out.println(threadName + " exiting. - State: " + this.getState());
// TERMINATED state
    }
}
// Thread creation by implementing Runnable interface
class MyRunnable implements Runnable {
    private String threadName;
    private SharedResource resource;

    public MyRunnable(String name, SharedResource resource) {
        this.threadName = name;
        this.resource = resource;
    }
```

```
    @Override
    public void run() {
        System.out.println("Running " + threadName + " (Runnable) - State: " + Thread.currentThread().getState());
        try {
            for (int i = 0; i < 3; i++) {
                System.out.println(threadName + " (Runnable): " + i);
                TimeUnit.MILLISECONDS.sleep(150);
                resource.increment();
            }
        } catch (InterruptedException e) {
            System.out.println(threadName + " (Runnable) interrupted.");
        }
        System.out.println(threadName + " (Runnable) exiting. - State: " + Thread.currentThread().getState());
    }
}
public class ThreadDemo {
    public static void main(String[] args) throws InterruptedException {
        SharedResource sharedResource = new SharedResource();

        // Demonstrate Thread class
        MyThread thread1 = new MyThread("Thread-A", sharedResource);
        thread1.setPriority(Thread.MAX_PRIORITY); // Setting priority
        System.out.println("Thread-A Priority: " + thread1.getPriority());

        // Demonstrate Runnable interface
        Thread thread2 = new Thread(new MyRunnable("Thread-B", sharedResource));
        thread2.setPriority(Thread.MIN_PRIORITY); // Setting priority
        System.out.println("Thread-B Priority: " + thread2.getPriority());
```

```
        // Start threads
        thread1.start(); // Moves from NEW to RUNNABLE
        thread2.start(); // Moves from NEW to RUNNABLE

        // Main thread waiting for other threads to finish
        thread1.join(); // Main thread WAITING for thread1
        thread2.join(); // Main thread WAITING for thread2

        System.out.println("Main thread exiting. Final counter value: " + sharedResource.getCounter());
    }
}
```

Right click on ThreadDemo.java file → choose Run File. You can see the following result in the output window.

Output

```
run:
Creating Thread-A - State: NEW
Thread-A Priority: 10
Thread-B Priority: 1
Running Thread-A - State: RUNNABLE
Thread-A: 0
Running Thread-B (Runnable) - State: RUNNABLE
Thread-B (Runnable): 0
Thread-A incremented counter to: 1
Thread-A: 1
Thread-A incremented counter to: 2
Thread-A: 2
Thread-0 incremented counter to: 3
Thread-B (Runnable): 1
Thread-A incremented counter to: 4
Thread-A exiting. - State: RUNNABLE
```

Thread-0 incremented counter to: 5

Thread-B (Runnable): 2

Thread-0 incremented counter to: 6

Thread-B (Runnable) exiting. - State: RUNNABLE

Main thread exiting. Final counter value: 6

BUILD SUCCESSFUL (total time: 1 second)

Result

Thus, multiple threads in Java using the Thread class and Runnable interface has been implemented and also thread lifecycle, priorities, and synchronization have been demonstrated.