Bharati Vidyapeeth (Deemed to be University)
Department of Engineering and Technology
Kharghar, Navi Mumbai

Department of Artificial Intelligence Machine Learning

Experiment No. 04

Name: Kamran Khan	Roll No: 11
Subject: High Performance Parallel Computing	Class/Batch: CSE-AIML / B1
Date of Performance: __ / __ / 2024	Date of Submission: __ / __ / 2024

AIM

To implement and observe the behavior of thread priorities in Java multithreading.

Theory/Procedure/Algorithm

Multithreading in Java allows concurrent execution of two or more threads. Each thread in Java is assigned a priority, an integer value ranging from 1 to 10, where:

- **Minimum Priority**: 1
- **Normal Priority (Default)**: 5
- **Maximum Priority**: 10

The priority of a thread is used by the thread scheduler to decide the order in which threads should be executed. However, thread priorities do not guarantee the exact order of execution. The underlying operating system decides which thread runs first, based on its scheduling policy.

Methods used for thread priority manipulation:

- `getPriority()` : Returns the priority of the thread.
- `setPriority(int priority)` : Sets the priority of the thread within the valid range (1–10). Trying to set a priority outside this range throws an `IllegalArgumentException` .

The thread scheduler generally favors threads with higher priority, but it is not deterministic and can vary depending on the system.

In [ ]:
```
// Main class to demonstrate thread priorities with a real task
class PriorityThreadExample extends Thread {

    // Constructor to set thread name
    public PriorityThreadExample(String name) {
        super(name);
    }

    // run() method for the thread that is called when start() is invoked
    public void run() {
        // Print the current thread executing the run method
        System.out.println("Executing run method of thread: " + Thread.currentThread().getNam
```

```
        // Simulating a time-consuming task (e.g., loop over a large range)
        for (int i = 1; i <= 1000000; i++) {
            if (i % 100000 == 0) {
                System.out.println(getName() + " has completed " + (i / 10000) + "% of the ta
            }
        }

        // Indicating that the thread has completed its task
        System.out.println(getName() + " has finished execution.");
    }

    // Main driver method
    public static void main(String[] args) {
        // Creating threads with meaningful names
        PriorityThreadExample thread1 = new PriorityThreadExample("Thread-1");
        PriorityThreadExample thread2 = new PriorityThreadExample("Thread-2");
        PriorityThreadExample thread3 = new PriorityThreadExample("Thread-3");

        // Displaying the default priority of threads
        System.out.println("Thread 1 default priority: " + thread1.getPriority());
        System.out.println("Thread 2 default priority: " + thread2.getPriority());
        System.out.println("Thread 3 default priority: " + thread3.getPriority());

        // Setting custom priorities for threads
        thread1.setPriority(2); // Lower priority
        thread2.setPriority(5); // Normal priority
        thread3.setPriority(8); // Higher priority

        // Display the updated priority of threads
        System.out.println("Updated Thread 1 priority: " + thread1.getPriority());
        System.out.println("Updated Thread 2 priority: " + thread2.getPriority());
        System.out.println("Updated Thread 3 priority: " + thread3.getPriority());

        // Start the threads to observe their execution behavior
        thread1.start();
        thread2.start();
        thread3.start();

        // Display main thread's current state
        System.out.println("Currently executing thread: " + Thread.currentThread().getName())
        System.out.println("Main thread default priority: " + Thread.currentThread().getPrior

        // Set and display updated main thread priority
        Thread.currentThread().setPriority(10);
        System.out.println("Updated Main thread priority: " + Thread.currentThread().getPrior

        // Main thread doing some work too
        for (int i = 1; i <= 500000; i++) {
            if (i % 100000 == 0) {
                System.out.println("Main thread has completed " + (i / 5000) + "% of its task
            }
        }

        System.out.println("Main thread has finished execution.");
    }
}
```

```
Thread 1 default priority: 5
Thread 2 default priority: 5
Thread 3 default priority: 5
Updated Thread 1 priority: 2
Updated Thread 2 priority: 5
Updated Thread 3 priority: 8
Executing run method of thread: Thread-3
Executing run method of thread: Thread-2
Currently executing thread: main
Main thread default priority: 5
Executing run method of thread: Thread-1
Updated Main thread priority: 10
Main thread has completed 20% of its task
Main thread has completed 40% of its task
Main thread has completed 60% of its task
Main thread has completed 80% of its task
Main thread has completed 100% of its task
Main thread has finished execution.
Thread-3 has completed 10% of the task
Thread-2 has completed 10% of the task
Thread-3 has completed 20% of the task
Thread-3 has completed 30% of the task
Thread-2 has completed 20% of the task
Thread-3 has completed 40% of the task
Thread-1 has completed 10% of the task
Thread-2 has completed 30% of the task
Thread-3 has completed 50% of the task
Thread-1 has completed 20% of the task
Thread-3 has completed 60% of the task
Thread-2 has completed 40% of the task
Thread-1 has completed 30% of the task
Thread-2 has completed 50% of the task
Thread-3 has completed 70% of the task
Thread-2 has completed 60% of the task
Thread-1 has completed 40% of the task
Thread-3 has completed 80% of the task
Thread-2 has completed 70% of the task
Thread-3 has completed 90% of the task
Thread-1 has completed 50% of the task
Thread-2 has completed 80% of the task
Thread-3 has completed 100% of the task
Thread-1 has completed 60% of the task
Thread-2 has completed 90% of the task
Thread-1 has completed 70% of the task
Thread-2 has completed 100% of the task
Thread-1 has completed 80% of the task
Thread-1 has completed 90% of the task
Thread-3 has finished execution.
Thread-2 has finished execution.
Thread-1 has completed 100% of the task
Thread-1 has finished execution.
```

Why do we have the same priority for different threads initially?

All threads created using the default constructor inherit the default priority, which is **5**. This explains why, at first, the priority for all three threads is the same. We must explicitly set different priorities using the `setPriority()` method if we want them to differ.

If the system assigns the same priority to different threads, how is that implemented?

When multiple threads have the same priority, the thread scheduler uses a **round-robin scheduling** or **time-slicing** strategy. This ensures that all threads with the same priority are given CPU time in a cyclic fashion, so none of them is starved for resources. However, the actual behavior depends on the underlying operating system and JVM implementation, and thus, the execution order is not guaranteed.

Reference materials:

- https://www.geeksforgeeks.org/java-thread-priority-multithreading
- https://www.geeksforgeeks.org/java-multithreading-tutorial

CONCLUSION

In this experiment, we successfully implemented thread priority manipulation in Java using the `getPriority()` and `setPriority()` methods. We observed that the default priority for all threads is 5 and that modifying the priority allows us to influence (but not control) the thread scheduling behavior. Threads with higher priority are more likely to be scheduled sooner, but the exact execution order is dependent on the system's thread scheduler.

ASSESSMENT

Timely Submission (7)	Presentation (06)	Understanding (12)	Total (25)	Sign