DAY 23:

ASSIGNMENT 5:

Task 5: Thread Pools and Concurrency Utilities

Create a fixed-size thread pool and submit multiple tasks that perform complex calculations or I/O operations and observe the execution.

ANSWER:

Step-by-Step Example

1. *Create a Fixed-Size Thread Pool*
2. *Submit Tasks to the Thread Pool*
3. *Perform Complex Calculations or I/O Operations*
4. *Observe Execution and Shutdown the Pool*

Example Code

java

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

public class ThreadPoolExample {

    public static void main(String[] args) {
        // Step 1: Create a fixed-size thread pool with 4 threads
        ExecutorService executorService = Executors.newFixedThreadPool(4);

        // Step 2: Submit multiple tasks to the thread pool
        for (int i = 1; i <= 10; i++) {
            int taskId = i;
            executorService.submit(() -> {
                // Step 3: Perform complex calculation or I/O operation
```

```
            performTask(taskId);
        });
    }

    // Step 4: Shut down the executor service
    executorService.shutdown();
    try {
        // Wait for all tasks to complete before terminating
        if (!executorService.awaitTermination(60, TimeUnit.SECONDS)) {
            executorService.shutdownNow();
        }
    } catch (InterruptedException e) {
        executorService.shutdownNow();
        Thread.currentThread().interrupt();
    }
  }

  // Method to perform a task (e.g., complex calculation or I/O operation)
  private static void performTask(int taskId) {
    System.out.println("Task " + taskId + " started by " + Thread.currentThread().getName());
    try {
        // Simulate a complex calculation or I/O operation with sleep
        Thread.sleep((long) (Math.random() * 1000));
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
    System.out.println("Task " + taskId + " completed by " + Thread.currentThread().getName());
  }
}
```

Explanation

1. Fixed-Size Thread Pool Creation:

java

```
ExecutorService executorService = Executors.newFixedThreadPool(4);
```

This line creates a thread pool with 4 threads. The pool can execute up to 4 tasks concurrently.

2. Submitting Tasks

java

```
executorService.submit(() -> { performTask(taskId); });
```

This submits tasks to the thread pool. Each task is represented by a lambda function that calls performTask with the task ID.

3. Task Execution:

java

```
private static void performTask(int taskId) { ... }
```

The performTask method simulates a complex calculation or I/O operation. It prints when the task starts and completes and includes a Thread.sleep to simulate time-consuming operations.

4. Shutdown and Await Termination:

java

```
executorService.shutdown();
```

The shutdown method initiates an orderly shutdown in which previously submitted tasks are executed but no new tasks will be accepted. awaitTermination waits for all tasks to complete or the timeout to occur.

Observing Execution

When you run the program, you should see output indicating which task is being executed by which thread. The thread names will typically be like pool-1-thread-1, pool-1-thread-2, etc. You can observe how tasks are picked up by different threads in the pool.