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.

Step-by-Step Example

1. *Create a Fixed-Size Thread Pool*
2. *Submit Tasks to the Thread Pool*
3. *Perform I/O Operations (Reading from Files)*
4. *Observe Execution and Shutdown the Pool*

Example Code

java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ThreadPoolIOExample {
```

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

        // Step 2: Define file paths to read from (replace with your actual file paths)
        String[] filePaths = {
            "file1.txt",
            "file2.txt",
            "file3.txt",
            "file4.txt",
            "file5.txt",
            "file6.txt"
        };

        // Step 3: Submit tasks to read files
        for (String filePath : filePaths) {
            executorService.submit(() -> {
                // Step 4: Perform I/O operation (reading file)
                readFile(filePath);
            });
        }

        // Step 5: 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 read a file and print its contents
    private static void readFile(String filePath) {
        System.out.println("Reading file " + filePath + " started by " +
Thread.currentThread().getName());
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(Thread.currentThread().getName() + " - " + line);
            }
        } catch (IOException e) {
            System.err.println("Error reading file " + filePath + ": " + e.getMessage());
        }
        System.out.println("Reading file " + filePath + " 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, allowing up to 4 concurrent file-reading tasks.

2. *File Paths*:

   java

   String[] filePaths = { "file1.txt", "file2.txt", "file3.txt", "file4.txt", "file5.txt", "file6.txt" };

   Replace these placeholders with actual file paths that you want to read.

3. *Submitting Tasks*:

   java

   executorService.submit(() -> { readFile(filePath); });

   This submits file-reading tasks to the thread pool. Each task is a lambda function that calls readFile with the file path.

4. *Reading Files*:

   java

   private static void readFile(String filePath) { ... }

The readFile method reads the content of the file line by line and prints it. It logs when the reading starts and completes. The try-with-resources statement ensures the BufferedReader is closed properly.

5. *Shutdown and Await Termination*:

java

executorService.shutdown();

The shutdown method initiates an orderly shutdown. awaitTermination waits for all tasks to finish or for the timeout to occur.

Observing Execution

When you run the program, you should see output indicating which thread is reading which file. This allows you to observe how tasks are being processed concurrently by different threads.