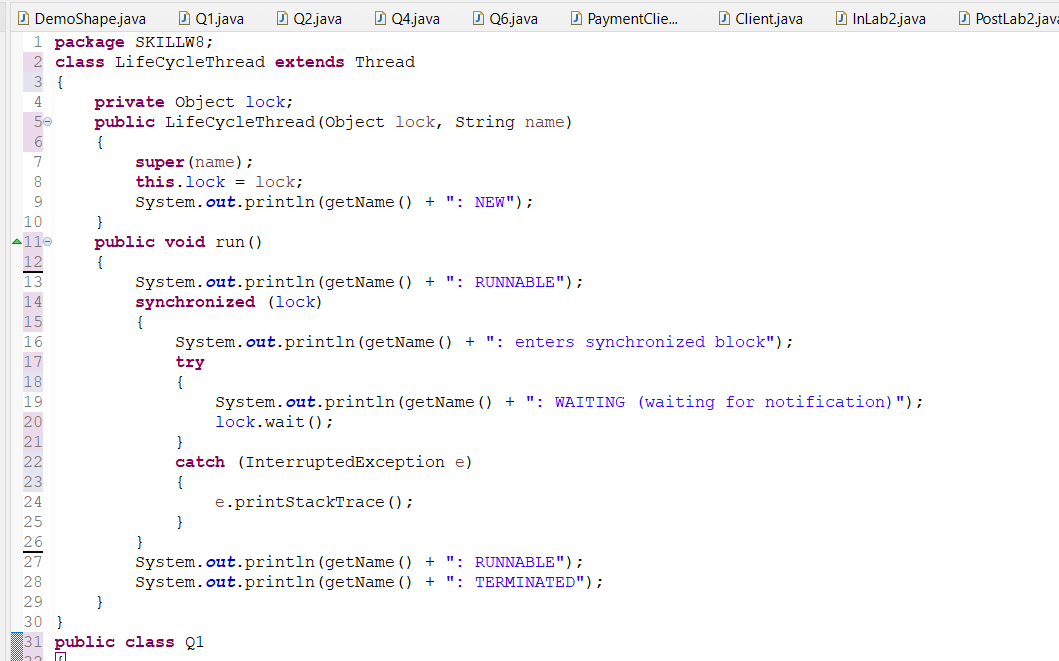
1. Thread Life Cycle Monitoring Implement a Java program where you create and start multiple threads, each of which simulates different stages of a thread's life cycle (e.g., NEW, RUNNABLE, BLOCKED, WAITING, TERMINATED). The program should print a log of each thread's state transitions. Use appropriate thread synchronization techniques to simulate the BLOCKED and WAITING states. Expected Output: The output should provide a detailed log of each thread’s state transitions, indicating when each state change occurs. For example: Thread 1: NEW Thread 1: RUNNABLE Thread 1: BLOCKED (waiting for Thread 2) Thread 2: NEW Thread 2: RUNNABLE Thread 2: enters synchronized block Thread 1: RUNNABLE Thread 1: WAITING (waiting for notification) Thread 2: sends notification Thread 1: RUNNABLE Thread 1: TERMINATED Thread 2: exits synchronized block Thread 2: TERMINATED  
   PROGRAM:



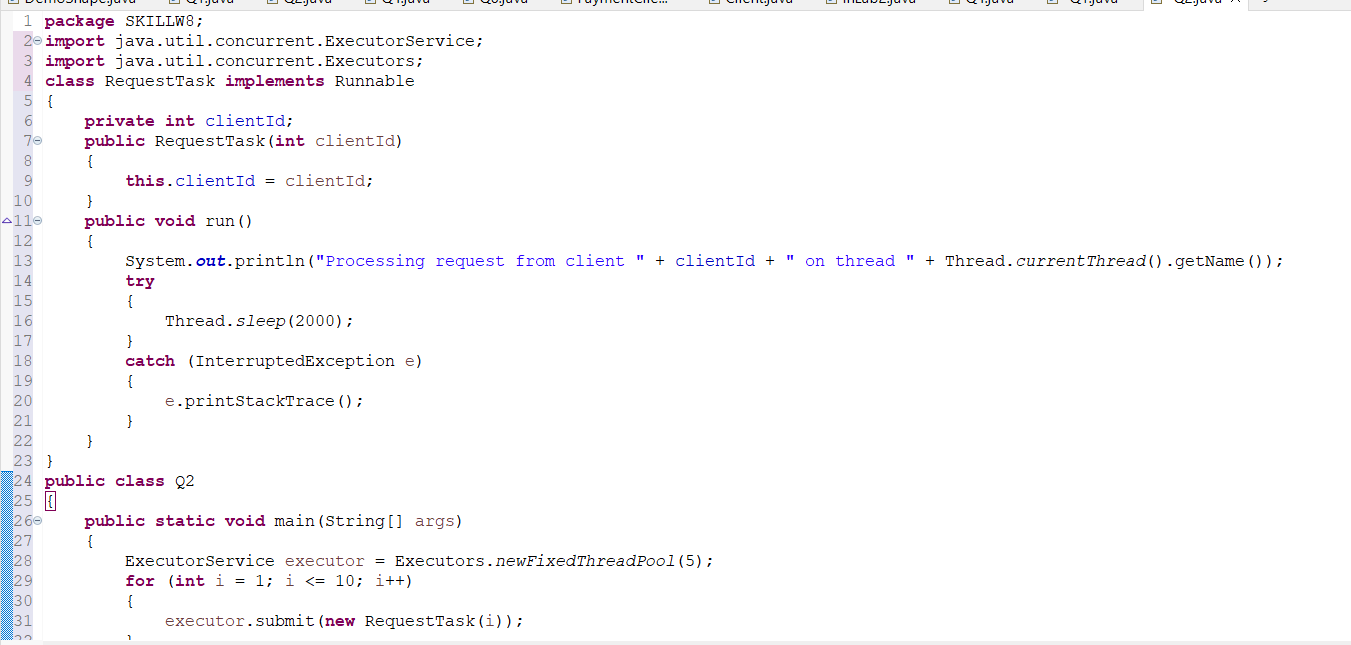
OUTPUT:

A screenshot of a computer

Description automatically generated

1. Thread Pools for Efficient Task Execution You are working on a web server that handles multiple client requests concurrently. Instead of creating a new thread for every request, you want to manage the threads efficiently using a thread pool. Implementation Steps: 1. Define the Request Processing Task: o Create a class that implements the Runnable interface to simulate request processing. Each instance of this class will represent a single client request. o 2. Create the Thread Pool: o Utilize Executors.newFixedThreadPool(int nThreads) to create a thread pool with a fixed number of threads. This will control the maximum number of concurrent requests being processed. 3. Submit Tasks to the Thread Pool: o In the main method, simulate incoming requests by creating instances of the request processing task and submitting them to the thread pool for execution. 4. Shutdown the Thread Pool: o Use the shutdown() method on the ExecutorService instance to initiate an orderly shutdown, allowing previously submitted tasks to execute before terminating the pool.

PROGRAM:



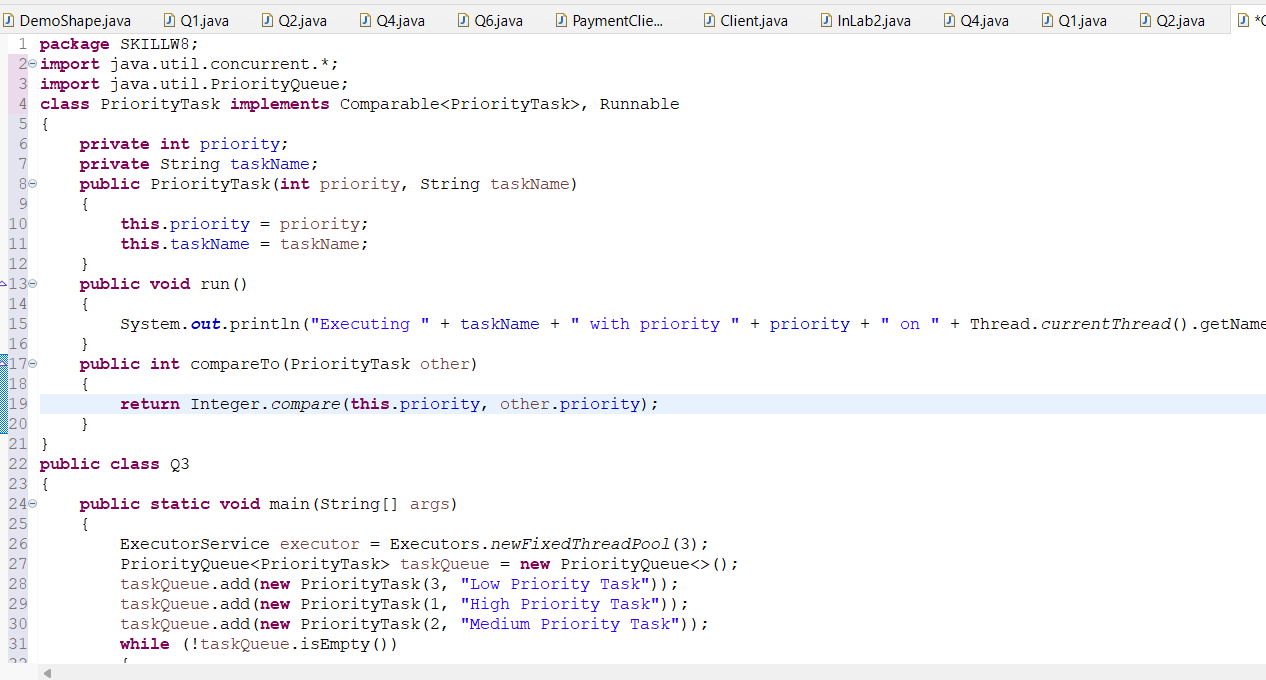
OUTPUT:

A screenshot of a computer program

Description automatically generated

1. You are developing a task scheduling system where tasks are assigned different priorities. Use a Priority Queue to store tasks based on their priority and assign threads from a thread pool to process the highest-priority tasks first. Ensure that lower-priority tasks are still processed, but only after higher-priority tasks have been completed. Key Components: 1. Task Class: Each task has a priority and a name or some other identifier. 2. Priority Queue: A PriorityQueue is used to store the tasks, where tasks with the highest priority are processed first (lower number indicates higher priority). 3. Thread Pool: A ThreadPoolExecutor is used to manage a pool of worker threads that will execute the tasks concurrently. 4. Task Submission: Tasks are submitted to the priority queue and processed by threads in priority order

PROGRAM:



OUTPUT:

A screenshot of a computer program

Description automatically generated