import java.lang.management.\*;

public class DeadlockSimulation {

private static final Object Resource1 = new Object();

private static final Object Resource2 = new Object();

static class Thread1 extends Thread {

@Override

public void run() {

synchronized (Resource1) {

System.out.println("Thread 1: Locked Resource 1");

try { Thread.sleep(100); } catch (InterruptedException e) {}

synchronized (Resource2) {

System.out.println("Thread 1: Locked Resource 2");

}

}

}

}

static class Thread2 extends Thread {

@Override

public void run() {

synchronized (Resource2) {

System.out.println("Thread 2: Locked Resource 2");

try { Thread.sleep(100); } catch (InterruptedException e) {}

synchronized (Resource1) {

System.out.println("Thread 2: Locked Resource 1");

}

}

}

}

public static void detectDeadlock() {

ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();

while (true) {

long[] deadlockedThreadIds = threadMXBean.findDeadlockedThreads();

if (deadlockedThreadIds != null && deadlockedThreadIds.length > 0) {

System.out.println("Deadlock detected! Terminating program.");

for (long threadId : deadlockedThreadIds) {

ThreadInfo threadInfo = threadMXBean.getThreadInfo(threadId);

System.out.println("Deadlocked thread: " + threadInfo.getThreadName());

}

System.exit(0);

}

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

Thread1 thread1 = new Thread1();

Thread2 thread2 = new Thread2();

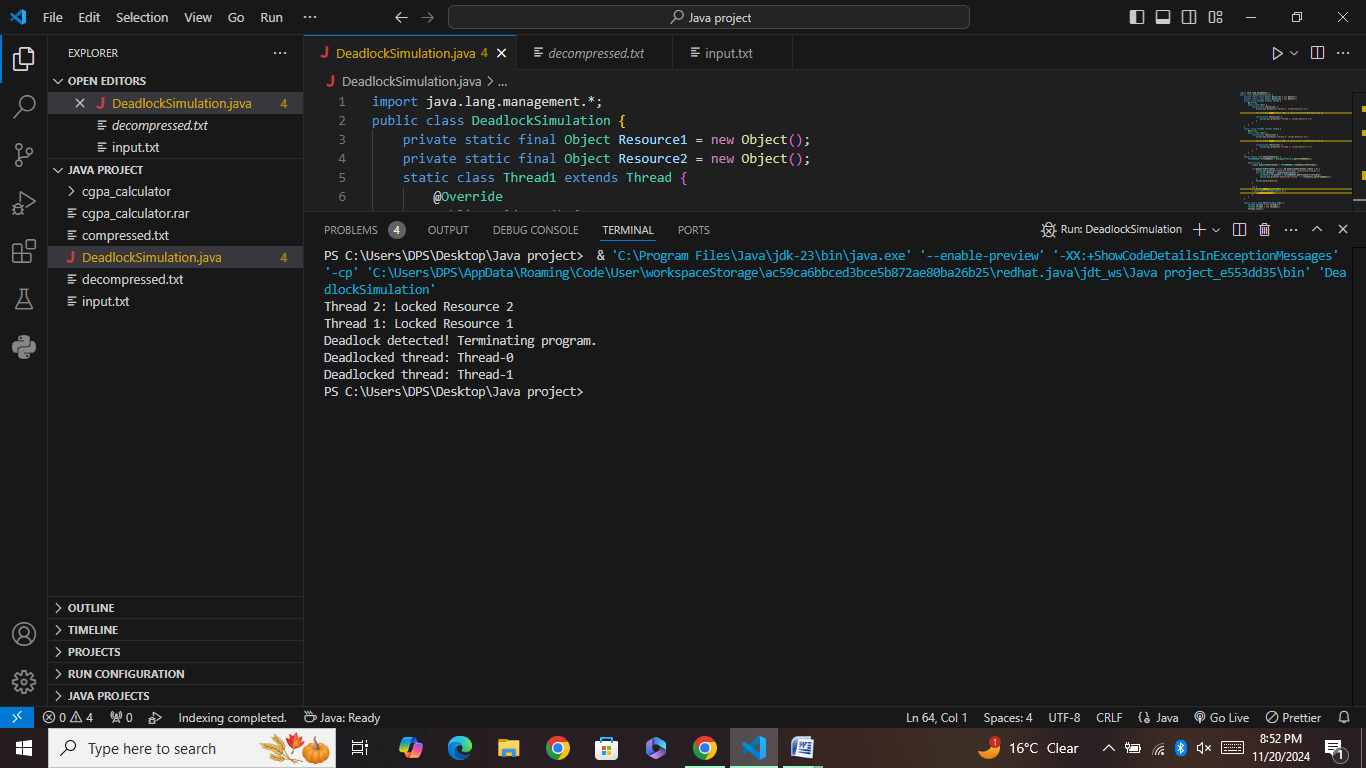
thread1.start();

thread2.start();

new Thread(() -> detectDeadlock()).start();

}

}



Explanation ;

The DeadlockSimulation program creates a deadlock scenario with two threads. Thread1 locks Resource1 and waits for Resource2, while Thread2 locks Resource2 and waits for Resource1, causing a circular wait (deadlock). The detectDeadlock method, using Java's ThreadMXBean, monitors and detects deadlocks every 500 milliseconds. If a deadlock is detected, the program prints the deadlocked threads' information and terminates.