**LAB-7**

1. Write the programme to open a text file named input 2, and copy its contents to an output text file output 2.

Program-

package demo;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.BufferedReader;

import java.io.BufferedWriter;

public class FileCopy {

public static void main(String[] args) {

String inputFileName = "input2.txt";

String outputFileName = "output2.txt";

try (BufferedReader reader = new BufferedReader(new FileReader(inputFileName));

BufferedWriter writer = new BufferedWriter(new FileWriter(outputFileName))) {

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine();

}

System.out.println("Contents copied successfully from " + inputFileName + " to " + outputFileName);

} catch (IOException e) {

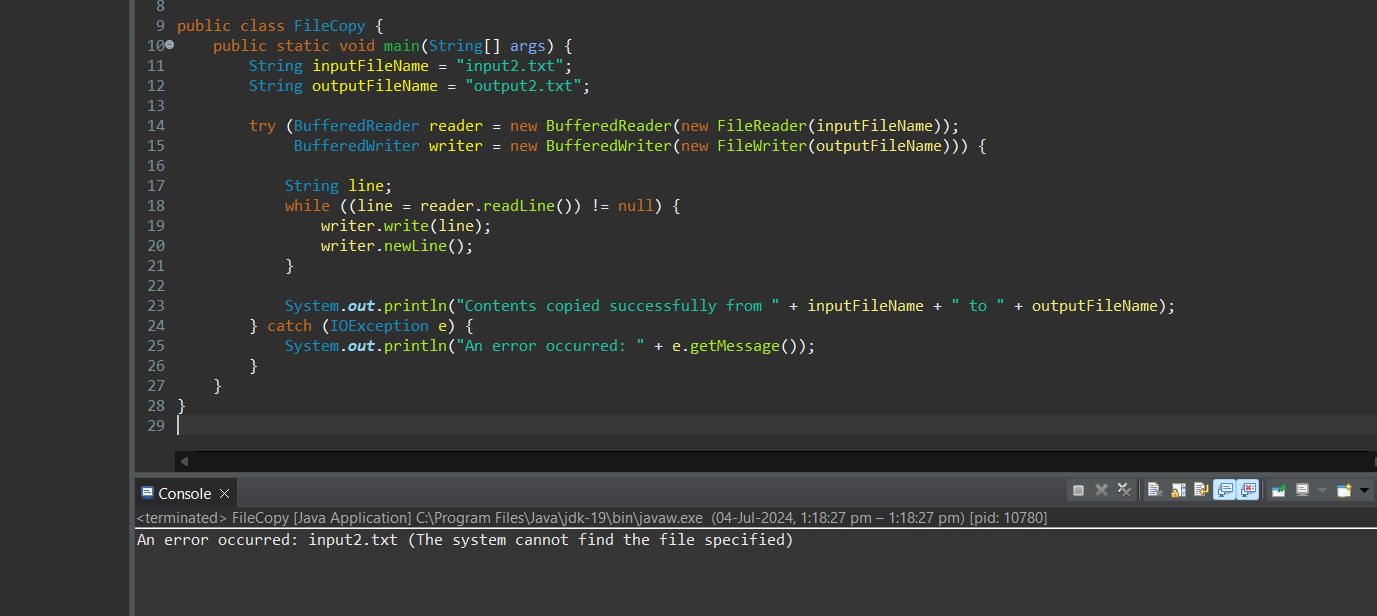
System.out.println("An error occurred: " + e.getMessage());

}

}

}

Output:



1. Write the programme to show multithreading for the string “multi threads”. Show the resulting output.

Program-

package demo;

class MultiThreadDemo implements Runnable {

private String part;

public MultiThreadDemo(String part) {

this.part = part;

}

public void run() {

System.out.println(Thread.currentThread().getName() + ": " + part);

}

public static void main(String[] args) {

String str = "multi threads";

String part1 = str.substring(0, 5); // "multi"

String part2 = str.substring(6); // "threads"

Thread thread1 = new Thread(new MultiThreadDemo(part1), "Thread 1");

Thread thread2 = new Thread(new MultiThreadDemo(part2), "Thread 2");

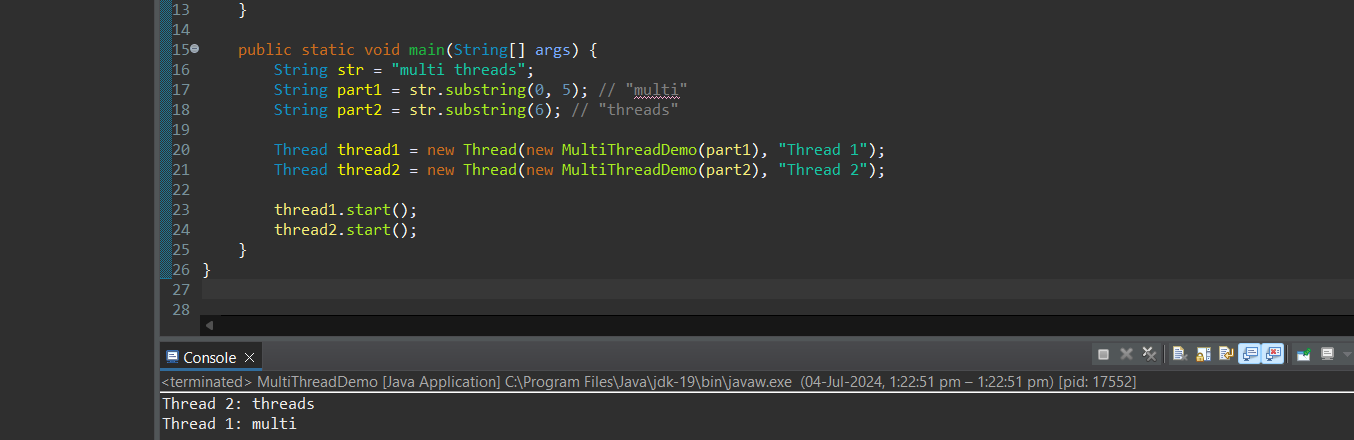
thread1.start();

thread2.start();

}

}

Output:



1. Implement a Java program that creates a thread using the Runnable interface. The thread should print numbers from 1 to 10 with a delay of 1 second between each number.

Program-

package demo;

class NumberPrinter implements Runnable {

public void run() {

try {

for (int i = 1; i <= 10; i++) {

System.out.println(i);

Thread.sleep(1000); // Sleep for 1 second (1000 milliseconds)

}

} catch (InterruptedException e) {

System.out.println("Thread interrupted: " + e.getMessage());

}

}

public static void main(String[] args) {

NumberPrinter numberPrinter = new NumberPrinter();

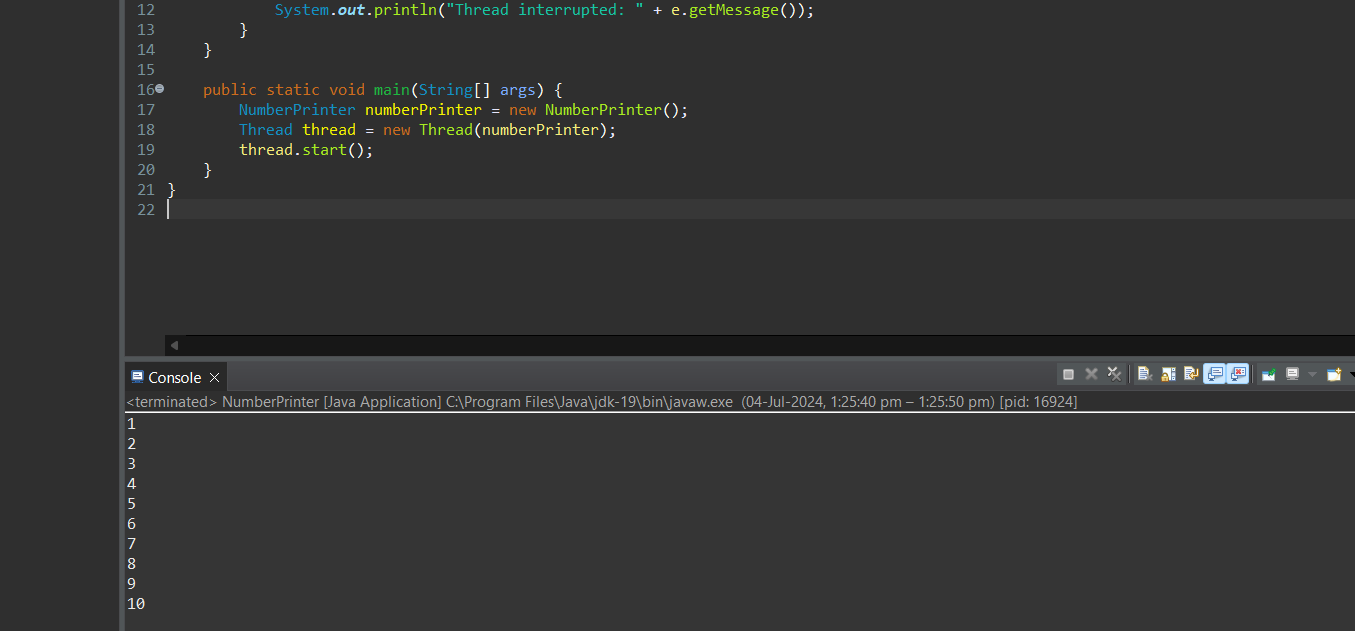
Thread thread = new Thread(numberPrinter);

thread.start();

}

}

Output:



1. Write a Java program that creates and starts three threads. Each thread should print its name and count from 1 to 5 with a delay of 500 milliseconds between each count.

Program-

package demo;

class CountingThread implements Runnable {

private String threadName;

public CountingThread(String threadName) {

this.threadName = threadName;

}

public void run() {

try {

for (int i = 1; i <= 5; i++) {

System.out.println(threadName + ": " + i);

Thread.sleep(500); // Sleep for 500 milliseconds

}

} catch (InterruptedException e) {

System.out.println(threadName + " interrupted: " + e.getMessage());

}

}

public static void main(String[] args) {

Thread thread1 = new Thread(new CountingThread("Thread 1"));

Thread thread2 = new Thread(new CountingThread("Thread 2"));

Thread thread3 = new Thread(new CountingThread("Thread 3"));

thread1.start();

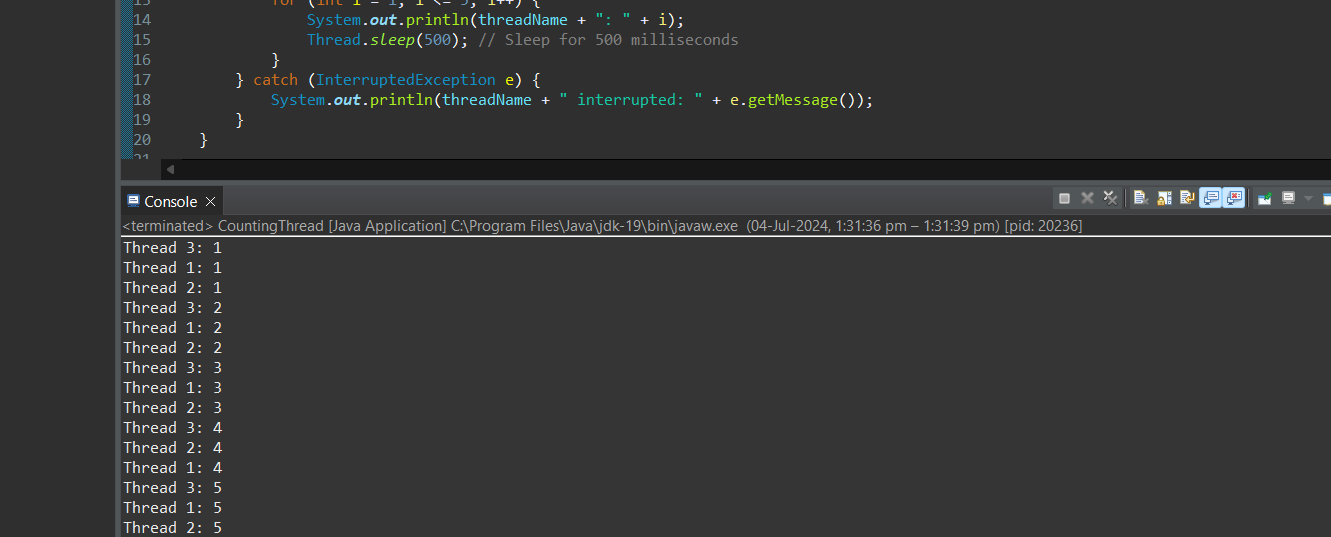
thread2.start();

thread3.start();

}

}

Output:



1. Create a Java program that demonstrates thread priorities. Create three threads with different priorities and observe the order in which they execute.

Program-

package demo;

class PriorityThread extends Thread {

public PriorityThread(String name) {

super(name);

}

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println(getName() + ": " + i);

try {

Thread.sleep(500); // Sleep for 500 milliseconds

} catch (InterruptedException e) {

System.out.println(getName() + " interrupted: " + e.getMessage());

}

}

}

public static void main(String[] args) {

PriorityThread highPriorityThread = new PriorityThread("High Priority Thread");

PriorityThread mediumPriorityThread = new PriorityThread("Medium Priority Thread");

PriorityThread lowPriorityThread = new PriorityThread("Low Priority Thread");

highPriorityThread.setPriority(Thread.MAX\_PRIORITY); // Highest priority (10)

mediumPriorityThread.setPriority(Thread.NORM\_PRIORITY); // Normal priority (5)

lowPriorityThread.setPriority(Thread.MIN\_PRIORITY); // Lowest priority (1)

lowPriorityThread.start();

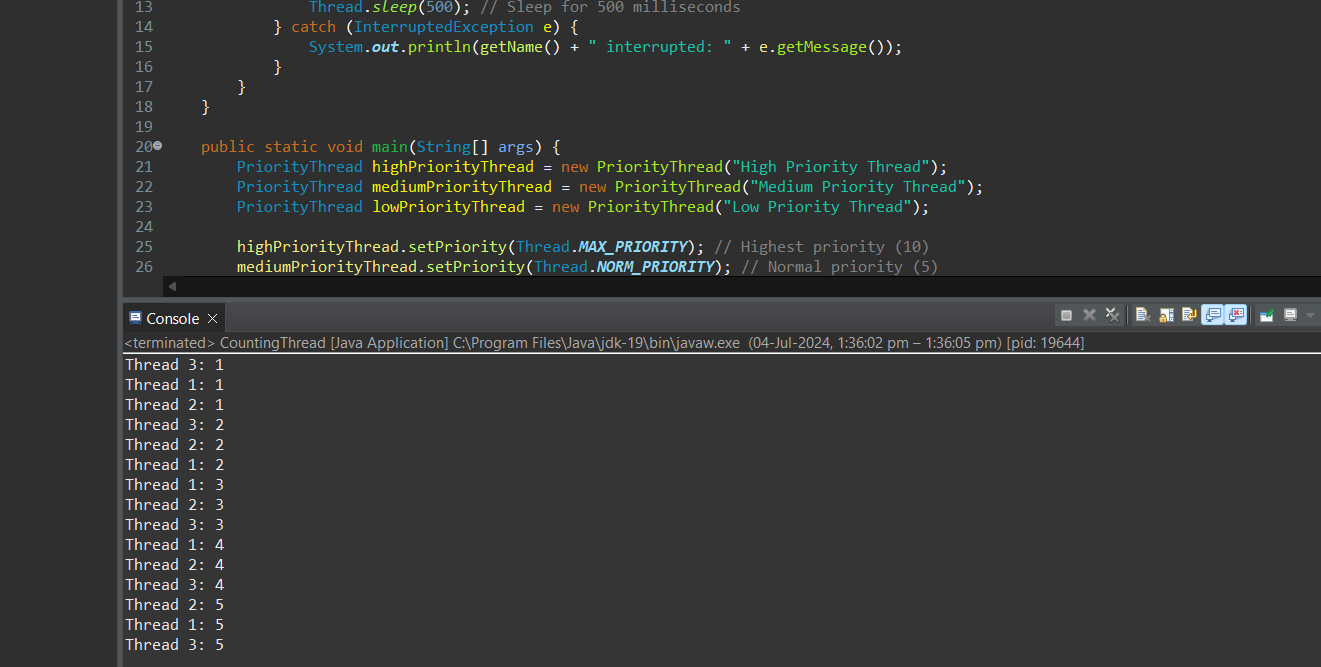
mediumPriorityThread.start();

highPriorityThread.start();

}

}

Output:



1. Write a Java program that creates a deadlock scenario with two threads and two resources.

Program-

package demo;

class Resource {

private final String name;

public Resource(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

class DeadlockThread extends Thread {

private final Resource resource1;

private final Resource resource2;

public DeadlockThread(String name, Resource resource1, Resource resource2) {

super(name);

this.resource1 = resource1;

this.resource2 = resource2;

}

public void run() {

synchronized (resource1) {

System.out.println(getName() + " locked " + resource1.getName());

try {

Thread.sleep(50); // Sleep for 50 milliseconds to make deadlock more likely

} catch (InterruptedException e) {

System.out.println(getName() + " interrupted: " + e.getMessage());

}

synchronized (resource2) {

System.out.println(getName() + " locked " + resource2.getName());

}

}

}

public static void main(String[] args) {

Resource resource1 = new Resource("Resource 1");

Resource resource2 = new Resource("Resource 2");

DeadlockThread thread1 = new DeadlockThread("Thread 1", resource1, resource2);

DeadlockThread thread2 = new DeadlockThread("Thread 2", resource2, resource1);

thread1.start();

thread2.start();

}

}

Output:

