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

import java.io.\*;

public class FileCopy {

public static void main(String[] args) {

// Names of the input and output files

String inputFileName = "input2.txt";

String outputFileName = "output2.txt";

// Use try-with-resources to ensure streams are closed after use

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

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

String line;

// Read each line from the input file and write it to the output file

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

writer.write(line);

writer.newLine(); // Write a newline character after each line

}

System.out.println("File copied successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

OUTPUT:

Q.2 Write the programme to show multithreading for the string “multi threads”. Show the resulting output.  
  
CODE:

public class MultiThreadingExample {

public static void main(String[] args) {

String text = "multi threads";

// Create an array to hold the threads

Thread[] threads = new Thread[text.length()];

// Loop through each character in the string

for (int i = 0; i < text.length(); i++) {

final int index = i;

threads[i] = new Thread(() -> {

System.out.print(text.charAt(index) + " ");

});

// Start each thread

threads[i].start();

}

// Wait for all threads to finish

for (Thread thread : threads) {

try {

thread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

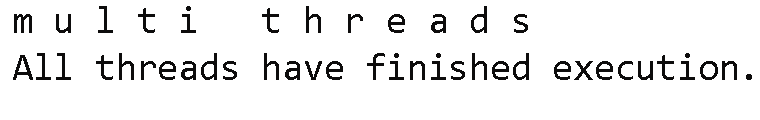
}

System.out.println("\nAll threads have finished execution.");

}

}

OUTPUT:



Q.3 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.

CODE:

public class NumberPrinter implements Runnable {

@Override

public void run() {

try {

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

System.out.println(i);

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

}

} catch (InterruptedException e) {

System.out.println("Thread was interrupted.");

}

}

public static void main(String[] args) {

// Create an instance of the NumberPrinter

NumberPrinter numberPrinter = new NumberPrinter();

// Create a Thread object and pass the Runnable instance

Thread thread = new Thread(numberPrinter);

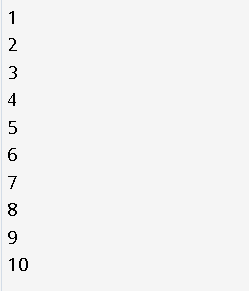
// Start the thread

thread.start();

}

}

OUTPUT:



Q.4 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.

CODE:

class CounterThread implements Runnable {

private String threadName;

public CounterThread(String threadName) {

this.threadName = threadName;

}

@Override

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 + " was interrupted.");

}

}

public static void main(String[] args) {

// Create instances of CounterThread

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

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

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

// Create Thread objects and pass the Runnable instances

Thread t1 = new Thread(thread1);

Thread t2 = new Thread(thread2);

Thread t3 = new Thread(thread3);

// Start the threads

t1.start();

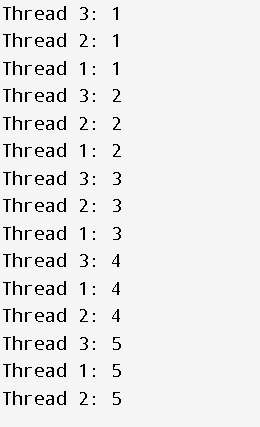
t2.start();

t3.start();

}

}

OUTPUT:



Q.5 Create a Java program that demonstrates thread priorities. Create three threads with different priorities and observe the order in which they execute.

CODE:

class PriorityThread extends Thread {

public PriorityThread(String name) {

super(name);

}

@Override

public void run() {

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

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

try {

Thread.sleep(200); // Sleep for 200 milliseconds

} catch (InterruptedException e) {

System.out.println(getName() + " was interrupted.");

}

}

}

public static void main(String[] args) {

// Create three threads with different priorities

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

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

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

// Set the priorities

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

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

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

// Start the threads

lowPriorityThread.start();

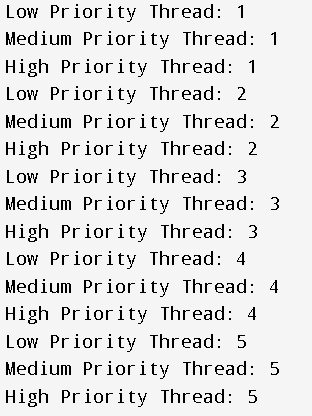
mediumPriorityThread.start();

highPriorityThread.start();

}

}

OUTPUT:



Q.6 Write a Java program that creates a deadlock scenario with two threads and two resources.

CODE:

public class DeadlockExample {

public static void main(String[] args) {

final Object resource1 = "Resource 1";

final Object resource2 = "Resource 2";

// Thread 1 tries to lock resource1 then resource2

Thread thread1 = new Thread(() -> {

synchronized (resource1) {

System.out.println("Thread 1: locked resource 1");

// Adding delay so that Thread 2 can lock resource2

try { Thread.sleep(50); } catch (InterruptedException e) {}

synchronized (resource2) {

System.out.println("Thread 1: locked resource 2");

}

}

});

// Thread 2 tries to lock resource2 then resource1

Thread thread2 = new Thread(() -> {

synchronized (resource2) {

System.out.println("Thread 2: locked resource 2");

// Adding delay so that Thread 1 can lock resource1

try { Thread.sleep(50); } catch (InterruptedException e) {}

synchronized (resource1) {

System.out.println("Thread 2: locked resource 1");

}

}

});

// Start both threads

thread1.start();

thread2.start();

}

}

OUTPUT:

