**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.

Code:-

package anudip;

import java.io.\*;

public class FileCopy {

public static void main(String[] args) {

// File paths

String inputFile = "C://Users//Naresh//eclipse-workspace//anudip//src//anudip/input2.txt";

String outputFile = "C://Users//Naresh//eclipse-workspace//anudip//src//anudip/output2.txt";

// Buffer size for copying

int bufferSize = 1024;

// Streams initialization

try (BufferedInputStream inputStream = new BufferedInputStream(new FileInputStream(inputFile));

BufferedOutputStream outputStream = new BufferedOutputStream(new FileOutputStream(outputFile))) {

byte[] buffer = new byte[bufferSize];

int bytesRead;

// Read from input file and write to output file

while ((bytesRead = inputStream.read(buffer)) != -1) {

outputStream.write(buffer, 0, bytesRead);

}

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

} catch (IOException e) {

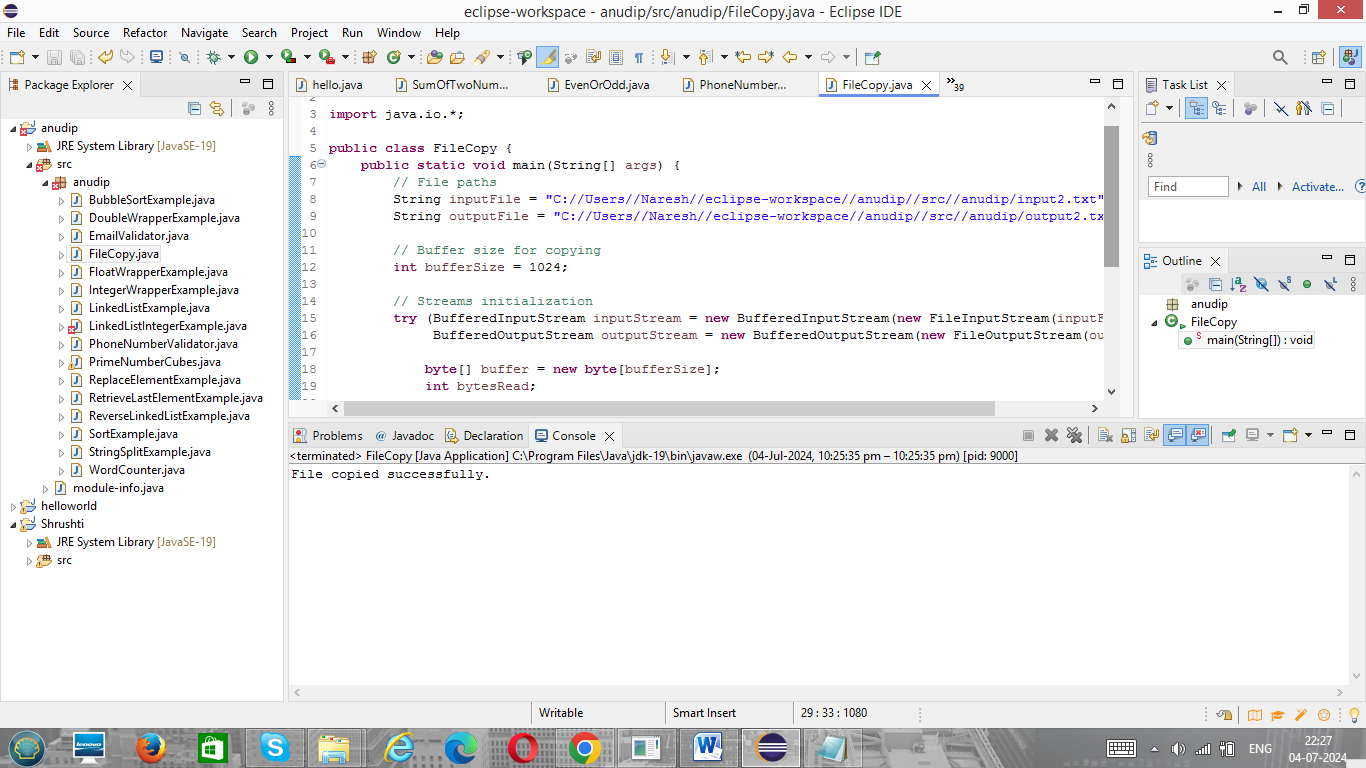
e.printStackTrace();

}

}

}

Output:-



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

Code:-

class PrintThread extends Thread {

private String text;

private int start;

private int end;

public PrintThread(String text, int start, int end) {

this.text = text;

this.start = start;

this.end = end;

}

@Override

public void run() {

for (int i = start; i < end; i++) {

System.out.print(text.charAt(i));

try {

Thread.sleep(100); // Adding some delay to simulate work

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

}

}

}

public class MultiThreadDemo {

public static void main(String[] args) {

String text = "multi threads";

// Create two threads to print different parts of the string

PrintThread thread1 = new PrintThread(text, 0, 5); // "multi"

PrintThread thread2 = new PrintThread(text, 6, text.length()); // "threads"

// Start the threads

thread1.start();

thread2.start();

// Wait for the threads to finish

try {

thread1.join();

thread2.join();

} catch (InterruptedException e) {

System.out.println(e.getMessage());

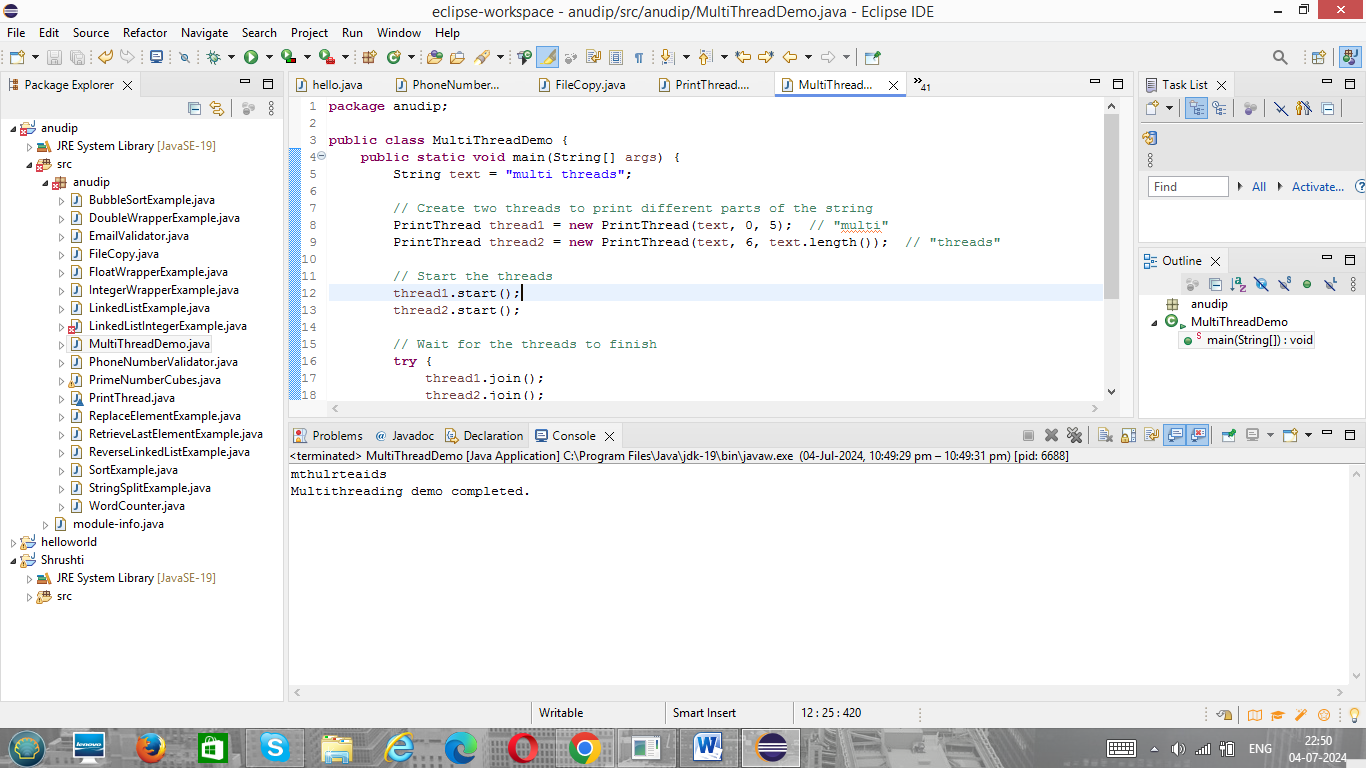
}

System.out.println("\nMultithreading demo completed.");

}

}

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.

Code:-

class NumberPrinter implements Runnable {

@Override

public void run() {

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

System.out.println(i);

try {

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

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

}

}

}

public class RunnableDemo {

public static void main(String[] args) {

// Create a Runnable object

Runnable numberPrinter = new NumberPrinter();

// Create a thread with the Runnable object

Thread thread = new Thread(numberPrinter);

// Start the thread

thread.start();

// Wait for the thread to finish

try {

thread.join();

} catch (InterruptedException e) {

System.out.println(e.getMessage());

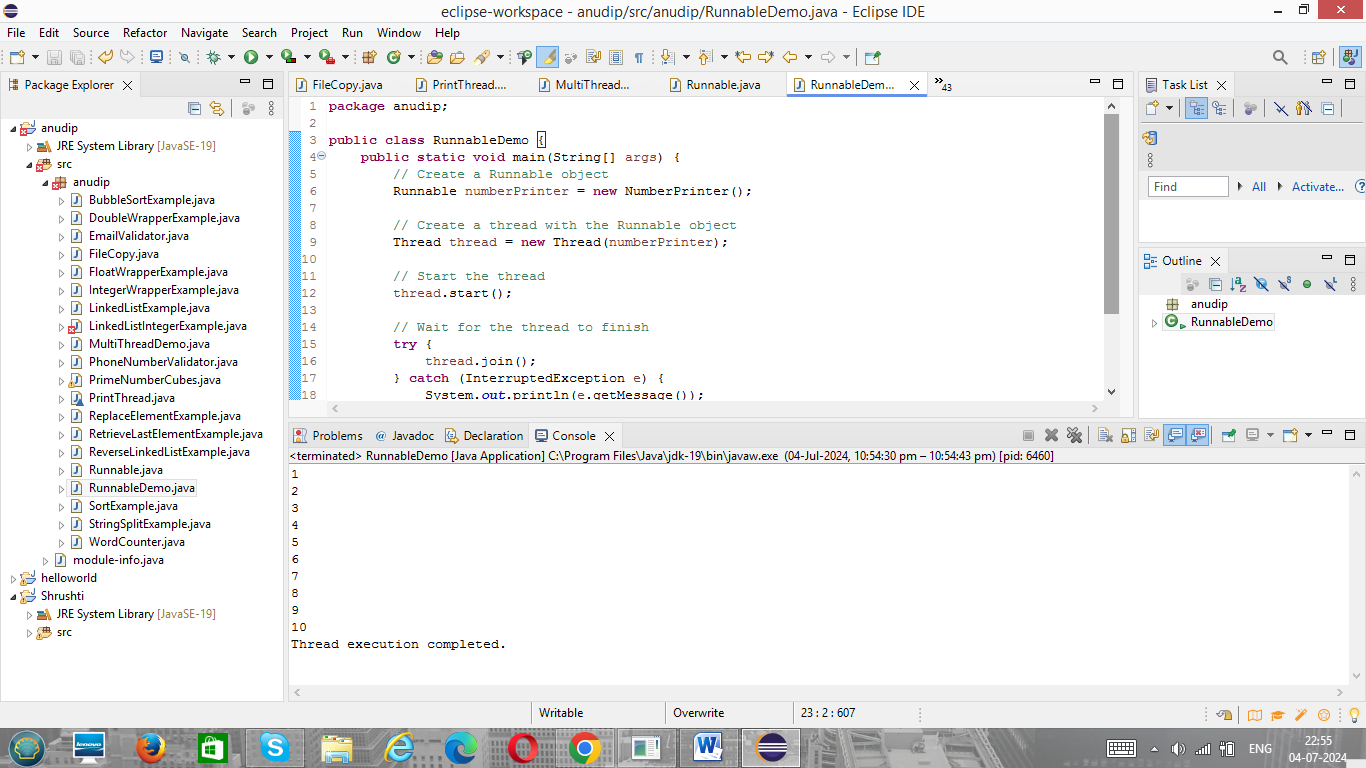
}

System.out.println("Thread execution completed.");

}

}

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.

Code:-

class CountPrinter implements Runnable {

private String threadName;

public CountPrinter(String threadName) {

this.threadName = threadName;

}

@Override

public void run() {

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

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

try {

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

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

}

}

}

public class MultiThreadDemo {

public static void main(String[] args) {

// Create Runnable objects with thread names

Runnable printer1 = new CountPrinter("Thread 1");

Runnable printer2 = new CountPrinter("Thread 2");

Runnable printer3 = new CountPrinter("Thread 3");

// Create Thread objects with Runnable objects

Thread thread1 = new Thread(printer1);

Thread thread2 = new Thread(printer2);

Thread thread3 = new Thread(printer3);

// Start the threads

thread1.start();

thread2.start();

thread3.start();

// Wait for all threads to finish

try {

thread1.join();

thread2.join();

thread3.join();

} catch (InterruptedException e) {

System.out.println(e.getMessage());

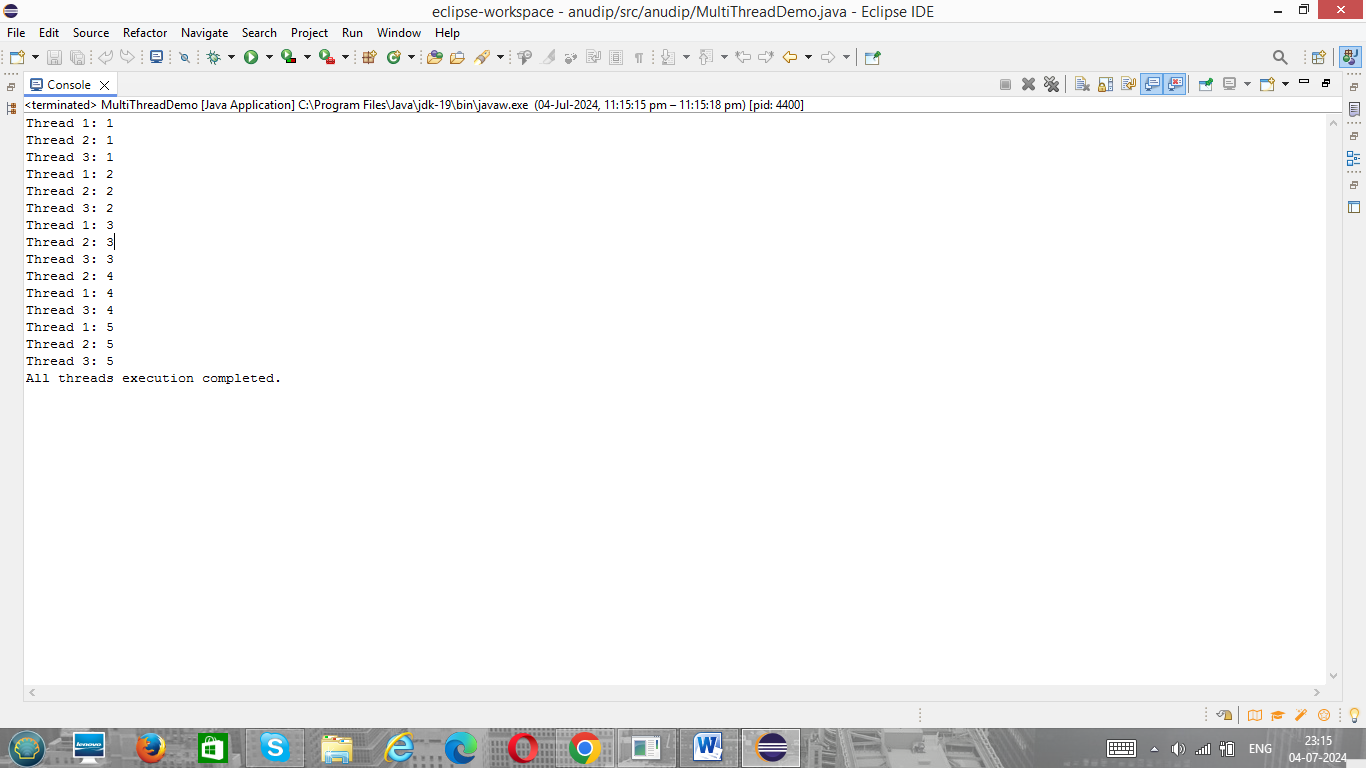
}

System.out.println("All threads execution completed.");

}

}

Output:-



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

Code:-

class PriorityPrinter implements Runnable {

private String threadName;

public PriorityPrinter(String threadName) {

this.threadName = threadName;

}

@Override

public void run() {

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

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

try {

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

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

}

}

}

public class ThreadPriorityDemo {

public static void main(String[] args) {

// Create Runnable objects with thread names

Runnable printer1 = new PriorityPrinter("Thread 1 (Priority 1)");

Runnable printer2 = new PriorityPrinter("Thread 2 (Priority 5)");

Runnable printer3 = new PriorityPrinter("Thread 3 (Priority 10)");

// Create Thread objects with Runnable objects

Thread thread1 = new Thread(printer1);

Thread thread2 = new Thread(printer2);

Thread thread3 = new Thread(printer3);

// Set different priorities

thread1.setPriority(Thread.MIN\_PRIORITY); // Priority 1

thread2.setPriority(Thread.NORM\_PRIORITY); // Priority 5

thread3.setPriority(Thread.MAX\_PRIORITY); // Priority 10

// Start the threads

thread1.start();

thread2.start();

thread3.start();

// Wait for all threads to finish

try {

thread1.join();

thread2.join();

thread3.join();

} catch (InterruptedException e) {

System.out.println(e.getMessage());

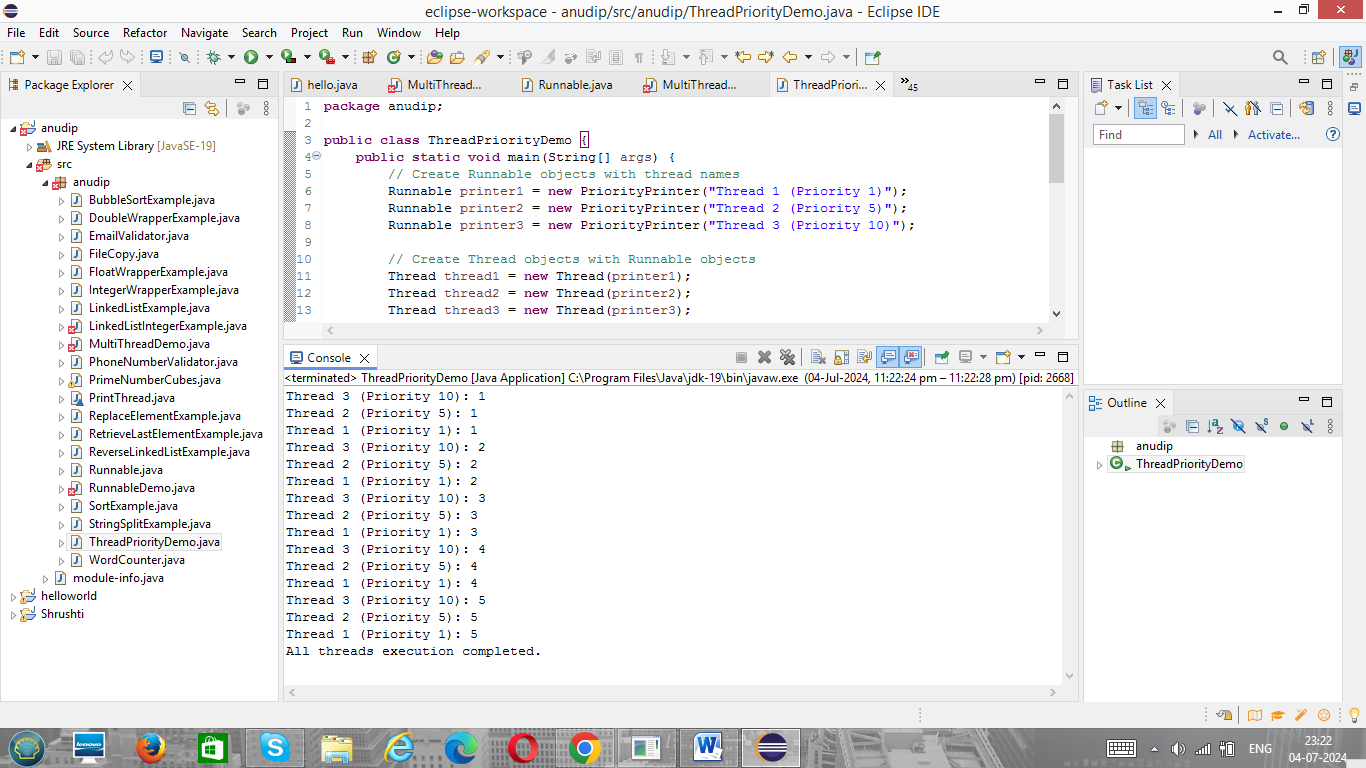
}

System.out.println("All threads execution completed.");

}

}

Output:-



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

Code:-

public class DeadlockDemo {

static class Resource {

private final String name;

Resource(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

static class Task implements Runnable {

private final Resource resource1;

private final Resource resource2;

Task(Resource resource1, Resource resource2) {

this.resource1 = resource1;

this.resource2 = resource2;

}

@Override

public void run() {

// Lock resource1

synchronized (resource1) {

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

// Adding some delay to make sure the other thread locks resource2

try {

Thread.sleep(50);

} catch (InterruptedException e) {

e.printStackTrace();

}

// Lock resource2

synchronized (resource2) {

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

}

}

}

}

public static void main(String[] args) {

// Create resources

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

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

// Create tasks

Task task1 = new Task(resource1, resource2);

Task task2 = new Task(resource2, resource1);

// Create threads

Thread thread1 = new Thread(task1, "Thread 1");

Thread thread2 = new Thread(task2, "Thread 2");

// Start threads

thread1.start();

thread2.start();

}

}

Output:-

