Day 10, Emp\_ID : 112916146

Task 1 : What is a thread and a Process?

A **process** is like the actual running version of a program (e.g., Chrome, Word) — it's not just the file on disk, but the active thing doing work

Example : An independent workspace: it has its own memory, files, and settings. It runs its own tasks, and if it crashes, it doesn’t directly affect others

A **thread** is a smaller worker inside a process. A process can have one or many. Each thread is a sequence of instructions being executed.

Example : A colleagues sharing the same office (process)—they share memory and files. This makes it easy and fast to communicate

### **Why They Matter**

* **Processes** keep apps isolated (cool for security and stability).
* **Threads** make apps fast and responsive by multitasking (e.g., listening to music while browsing).
* Threads are efficient but need careful coordination to avoid conflicts.

Task2

package MultiThreading;

class RunnableDemo implements Runnable {

private Thread t;

private String threadName;

RunnableDemo( String name){

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

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

// Let the thread sleep for a while.

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start ()

{

System.out.println("Starting " + threadName );

if (t == null)

{

t = new Thread (this, threadName);

t.start ();

}

}

}

public class Task2 {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}



Task3

package MultiThreading;

class RunnableDemo extends Thread {

private String threadName;

RunnableDemo(String name) {

super(name);

threadName = name;

System.out.println("Creating " + threadName);

}

@Override

public void run() {

System.out.println("Running " + threadName);

try {

for (int i = 4; i > 0; i--) {

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

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

}

public class Task3 {

public static void main(String[] args) {

RunnableDemo t1 = new RunnableDemo("Thread-1");

t1.start(); // directly starts the thread

RunnableDemo t2 = new RunnableDemo("Thread-2");

t2.start();

}

}



Task4

package MultiThreading;

class Counter {

private int count = 0;

public void increment() {

count++;

}

public int getCount() {

return count;

}

}

class ThreadDemo extends Thread {

Counter counter;

ThreadDemo(Counter counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 10; i++) {

counter.increment();

}

}

}

public class Task4 {

public static void main(String[] args) {

Counter counter = new Counter();

ThreadDemo t1 = new ThreadDemo(counter);

ThreadDemo t2 = new ThreadDemo(counter);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Final count: " + counter.getCount());

}

}



Task 5

package MultiThreading;

class Counter {

private int count = 0;

public synchronized void increment() {

count++;

}

public int getCount() {

return count;

}

}

class ThreadDemo extends Thread {

private Counter counter;

public ThreadDemo(Counter c) { counter = c; }

@Override

public void run() {

for (int i = 0; i < 10; i++) {

counter.increment();

}

}

}

public class Task5 {

public static void main(String[] args) throws InterruptedException {

Counter counter = new Counter();

ThreadDemo t1 = new ThreadDemo(counter);

ThreadDemo t2 = new ThreadDemo(counter);

t1.start(); t2.start();

t1.join(); t2.join();

System.out.println("Final count: " + counter.getCount()); // Always 20

}

}



Task 6

package MultiThreading;

class Counter {

private int count = 0;

public void increment() {

// Synchronized block ensures only one thread modifies 'count' at a time

synchronized (this) {

count++;

}

}

public int getCount() {

return count;

}

}

class ThreadDemo extends Thread {

private final Counter counter;

ThreadDemo(Counter counter) {

this.counter = counter;

}

@Override

public void run() {

for (int i = 0; i < 10; i++) {

counter.increment();

}

}

}

public class Task6 {

public static void main(String[] args) {

Counter counter = new Counter();

ThreadDemo t1 = new ThreadDemo(counter);

ThreadDemo t2 = new ThreadDemo(counter);

t1.start();

t2.start();

try {

t1.join(); // Wait for t1 to finish

t2.join(); // Wait for t2 to finish

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Final count: " + counter.getCount()); // Expected: 20

}

}



Task7

package MultiThreading;

class Counter {

private static int count = 0;

public static synchronized void increment() {

count++;

}

public static int getCount() {

return count;

}

}

class ThreadDemo extends Thread {

// No-argument constructor

ThreadDemo() {}

public void run() {

for (int i = 0; i < 10; i++) {

Counter.increment();

}

}

}

public class Task7 {

public static void main(String[] args) {

ThreadDemo t1 = new ThreadDemo();

ThreadDemo t2 = new ThreadDemo();

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Final count: " + Counter.getCount());

}

}



Task8

package MultiThreading;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

class Counter {

private int count = 0;

private final Lock lock = new ReentrantLock(); // Lock object

public void increment() {

lock.lock(); // Acquire the lock

try {

count++; // Critical section

} finally {

lock.unlock(); // Always release the lock

}

}

public int getCount() {

return count;

}

}

class ThreadDemo extends Thread {

private final Counter counter;

ThreadDemo(Counter counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 10; i++) {

counter.increment();

}

}

}

public class Task8 {

public static void main(String[] args) {

Counter counter = new Counter(); // Shared object

ThreadDemo t1 = new ThreadDemo(counter);

ThreadDemo t2 = new ThreadDemo(counter);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Final count: " + counter.getCount()); // Expected: 20

}

}



Task 9 : oral done

Task 10

package MultiThreading;

//Deadlock

class Resource {

synchronized void method1(Resource r) {

System.out.println(Thread.currentThread().getName() + " is executing method1");

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

r.method2(this);

}

synchronized void method2(Resource r) {

System.out.println(Thread.currentThread().getName() + " is executing method2");

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

r.method1(this);

}

}

public class Task10{

public static void main(String[] args) {

final Resource r1 = new Resource();

final Resource r2 = new Resource();

Thread t1 = new Thread(() -> r1.method1(r2), "Thread-1"); //-> are called lambda expression

Thread t2 = new Thread(() -> r2.method1(r1), "Thread-2"); // 2 parameters are passes here,

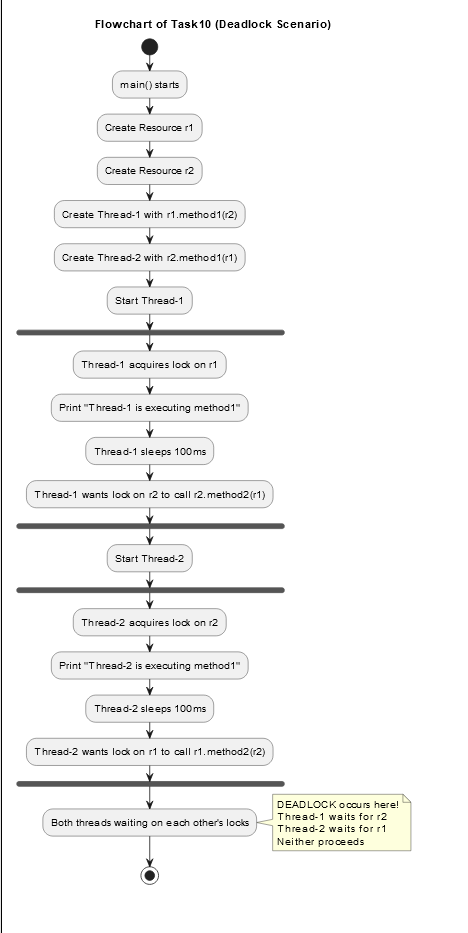
t1.start();

t2.start();

}

}





Task11

package MultiThreading;

//interThreadCommunication

class SharedResource {

private boolean ready = false;

synchronized void produce() {

try {

while (ready) { // true so producer will wait

wait();

}

System.out.println("Producing..."); // when its producer turn it will print this

ready = true; //sets

notify();// give a wake up call to consume

} catch (InterruptedException e) {

e.printStackTrace();

}

}

synchronized void consume() {

try {

while (!ready) { // if asle, it will wait and when its turn comes it will print this

wait();

}

System.out.println("Consuming...");

ready = false; // since the previous produced is over, nothing left to be consumed

notify(); // wake call to produce

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Task11 {

public static void main(String[] args) {

SharedResource resource = new SharedResource();

Thread producer = new Thread(resource::produce);

Thread consumer = new Thread(resource::consume);

producer.start();

consumer.start();

}

}



Task12

package MultiThreading;

class InterruptibleThread extends Thread {

public void run() {

try {

while (!Thread.currentThread().isInterrupted()) {

System.out.println("Thread is running");

Thread.sleep(100);

}

} catch (InterruptedException e) {

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

}

}

}

public class Task12 {

public static void main(String[] args) {

InterruptibleThread thread = new InterruptibleThread();

thread.start();

try {

Thread.sleep(500);

thread.interrupt();

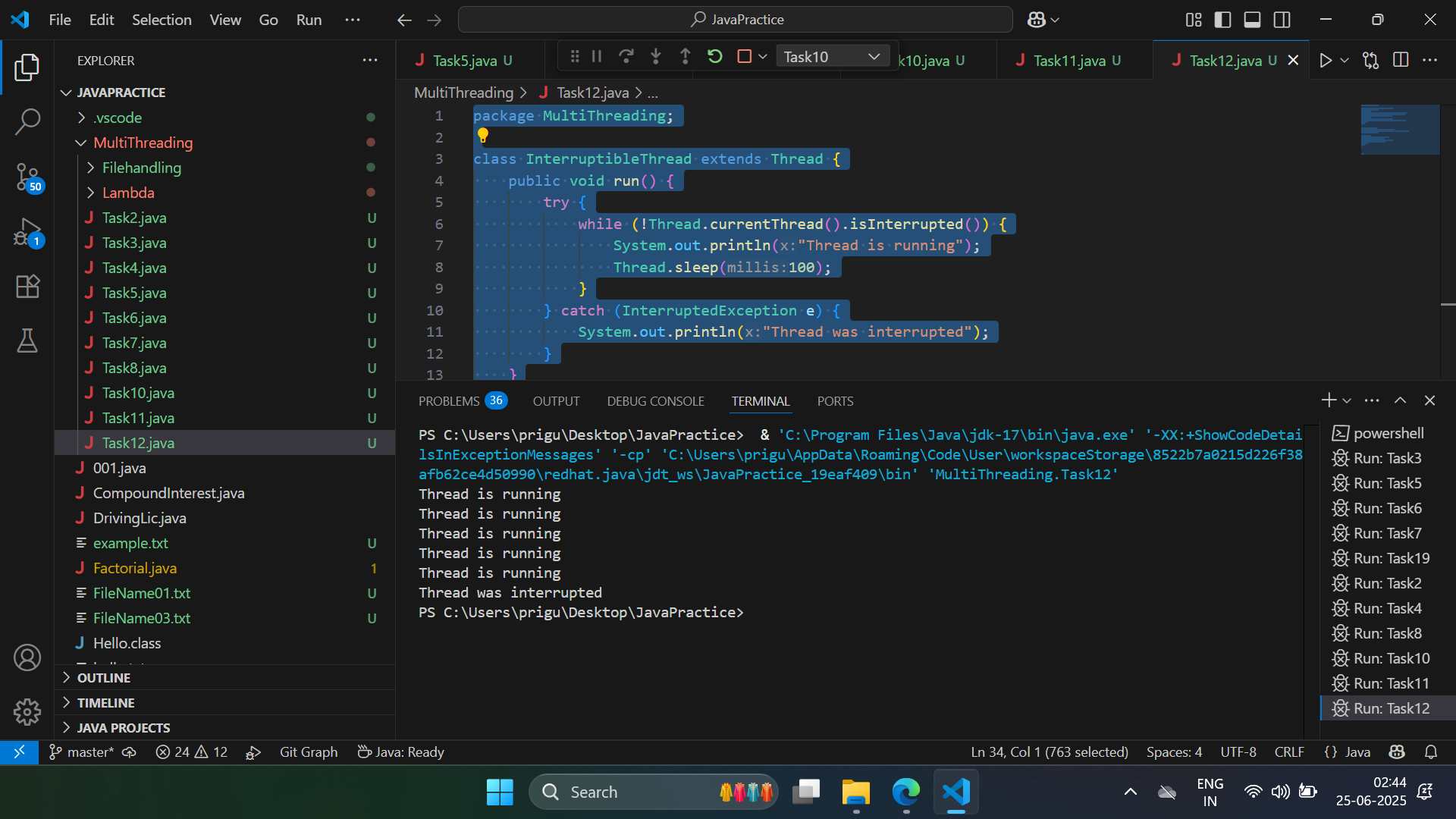
} catch (InterruptedException e) {

e.printStackTrace();

}

}

}



Task 13

Oral was done

Task 14

What is a Thread Pool

A **Thread Pool** in Java is a pool (group) of pre-created worker threads that are reused to perform tasks. Instead of creating a new thread for every task (which is costly), the thread pool reuses existing threads to improve performance and resource management.

### **Real-Life Example:**

Imagine you have:

* 5 workers in a restaurant (threads)
* 100 customer orders (tasks)

Instead of hiring 100 temporary workers, you let the 5 workers handle all 100 orders **one by one**. That’s what a **thread pool** does.

Task 15

package MultiThreading.Filehandling;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

public class Task15 {

public static void main(String[] args) {

File f1 = new File("FileName01.txt"); // create a File object

FileOutputStream outfile = null;

// Byte array representing "I LOVE INDIA"

byte[] Text = {'I', ' ', 'L', 'I', 'K', 'E', ' ', 'I', 'N', 'D', 'I', 'A'};

// byte[] Text = {'I', ' ', 'L', 'O', 'V', 'E', ' ', 'I', 'N', 'D', 'I', 'A'};

try {

outfile = new FileOutputStream(f1); // create file and output stream

outfile.write(Text); // write byte array to file

} catch (IOException e) {

System.out.println(e);

System.exit(-1);

} finally {

try {

if (outfile != null)

outfile.close(); // always close the stream

} catch (IOException e) {

System.out.println("Error while closing file: " + e);

}

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

}

}



Task16

package MultiThreading.Filehandling;

import java.io.\*;

public class Task16 {

public static void main(String args[]) {

FileInputStream infile = null;

int b;

try {

infile = new FileInputStream("FileName01.txt");

while ((b = infile.read()) != -1) {

System.out.println((char)b);

}

infile.close();

}

catch(IOException e) {

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}



Task17

package MultiThreading.Filehandling;

import java.io.\*;

import java.util.\*;

public class Task17 {

public static void main(String[] args) {

FileOutputStream outfile = null;

FileInputStream infile = null;

Scanner sc = new Scanner(System.in);

try {

// Step 1: Take input from the user

System.out.print("Enter a line to write into the file: ");

String s = sc.nextLine();

// Step 2: Convert string to byte array

byte[] b1 = s.getBytes();

// Step 3: Write the byte array to file

outfile = new FileOutputStream("in.txt"); // this will overwrite if file exists

outfile.write(b1);

outfile.close(); // Always close after writing

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

// Step 4: Read back the file content

infile = new FileInputStream("in.txt");

int byteData;

System.out.println("\nContent of the file:");

while ((byteData = infile.read()) != -1) {

System.out.print((char) byteData); // convert byte to char and print

}

infile.close(); // Always close after reading

} catch (IOException e) {

System.out.println("Error: " + e);

System.exit(-1);

} finally {

sc.close();

try {

if (outfile != null) outfile.close();

if (infile != null) infile.close();

} catch (IOException e) {

System.out.println("Error while closing file: " + e);

}

}

}

}



Task 18

package MultiThreading.Filehandling;

import java.io.\*;

public class Task18 {

public static void main(String[] args) {

File f1 = new File("FileName03.txt");

FileWriter fw = null;

try {

fw = new FileWriter(f1); // creates or overwrites FileName03.txt

fw.write("ahmedabad\n");

fw.write("baroda\n");

fw.close(); // always close the writer

}

catch (FileNotFoundException e) {

System.out.println("Sorry..!! File Not Found...!!!");

}

catch (IOException e) {

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

}

System.out.println("Write operation done!!");

}

}



Task19

package MultiThreading.Filehandling;

import java.io.\*;

public class Task19 {

public static void main(String[] args) {

FileReader fr = null;

try {

fr = new FileReader("FileName03.txt"); // file must exist

int ch;

while ((ch = fr.read()) != -1) {

System.out.print((char) ch); // convert and print each character

}

System.out.println("\nReading complete");

fr.close(); // close the file reader

}

catch (FileNotFoundException e) {

System.out.println("Sorry..!! File Not Found...!!!");

}

catch (IOException e) {

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

}

}

}



Task20

package MultiThreading.Filehandling;

import java.io.\*;

// CopyByte

public class Task20 {

public static void main(String[] args) {

FileInputStream infile = null;

FileOutputStream outfile = null;

try {

// Open input and output files

infile = new FileInputStream("FileName01.txt");

outfile = new FileOutputStream("NewFile06.txt");

int byteRead;

// Read and write byte-by-byte until end of file

while ((byteRead = infile.read()) != -1) {

outfile.write(byteRead);

}

System.out.println("Byte copied from FieName01.txt to NewFile06.txt file.");

}

catch (FileNotFoundException e) {

System.out.println("Sorry..!! File Not Found...!!!");

}

catch (IOException e) {

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

}

finally {

// Close files in finally block to ensure they close even if error occurs

try {

if (infile != null) infile.close();

if (outfile != null) outfile.close();

} catch (IOException e) {

System.out.println("Error while closing files: " + e.getMessage());

}

}

}

}



Task21

package MultiThreading.Filehandling;

import java.io.\*;

//File Merge

public class Task21 {

public static void main(String[] args) {

// Name of the output (merged) file

String mergedFile = "MergedFile.txt";

try (

// Input streams for both source files

FileInputStream file1 = new FileInputStream("NewFile05.txt");

FileInputStream file2 = new FileInputStream("in.txt");

// Merge both input streams

SequenceInputStream sequenceStream = new SequenceInputStream(file1, file2);

// Buffered input stream for efficient reading

BufferedInputStream bufferedIn = new BufferedInputStream(sequenceStream);

// Buffered output stream to the new merged file

BufferedOutputStream bufferedOut = new BufferedOutputStream(new FileOutputStream(mergedFile))

) {

int byteData;

// Read from the combined input stream and write to the output file

while ((byteData = bufferedIn.read()) != -1) {

bufferedOut.write(byteData);

}

// Flush data to ensure it's written to disk

bufferedOut.flush();

System.out.println(" Merged content written to: " + mergedFile);

} catch (IOException e) {

System.out.println(" Error during file merging: " + e.getMessage());

}

}

}



Task22

package MultiThreading.Filehandling;

import java.io.\*;

public class Task22 {

public static void main(String args[]) {

// Check if two command line arguments are passed

if (args.length != 2) {

System.out.println("Usage: java FileRenameDemo <oldfilename> <newfilename>");

return;

}

// Create File objects

File f1 = new File(args[0]); // old file name

File f2 = new File(args[1]); // new file name

// Attempt to rename

if (f1.renameTo(f2)) {

System.out.println("File renamed from " + f1.getName() + " to " + f2.getName() + " successfully.");

} else {

System.out.println("Failed to rename file. Make sure the source file exists.");

}

}

}



Task23

package MultiThreading.Filehandling;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class Task23 {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("in.txt"))) {

String line;

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

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}



Task24

package MultiThreading.Filehandling;

import java.io.FileReader;

import java.io.IOException;

public class Task24 {

public static void main(String[] args) {

FileReader reader = null;

try {

// Open the file for reading

reader = new FileReader("in.txt");

int character;

// Read one character at a time until end of file

while ((character = reader.read()) != -1) {

System.out.print((char) character);

}

System.out.println("\nFile reading completed.");

} catch (IOException e) {

System.out.println("An error occurred while reading the file.");

e.printStackTrace();

} finally {

// Ensure the file is closed

try {

if (reader != null) {

reader.close();

}

} catch (IOException e) {

System.out.println("Failed to close the file.");

e.printStackTrace();

}

}

}

}



Task25

package MultiThreading.Filehandling;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

public class Task25 {

public static void main(String[] args) {

// File to write to

String filePath = "example.txt";

// Content to write

String content = "Hello, World!\nThis is a BufferedWriter example.";

// BufferedWriter writes text efficiently to files

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {

writer.write(content); // Writing the string to the file

System.out.println("Content written to file.");

} catch (IOException e) {

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

}

}

}

