**ID: hrajranj**

**Day 10 – 18th June 2025**

Multi Threading:

**Task 1:**

What is a thread and a Process?

Process is as an instance of a program that is being executed by our computer's CPU. When we run a program, the operating system creates a new process and assigns it a unique process ID. This process receives a set of resources, such as memory space and CPU time, to execute the program. The process has its own stack, data, and heap. Each process runs in its own virtual address space and is isolated from other processes. If a process crashes, it doesn't affect any other running process.

Thread is the smallest unit of execution. It is basically a segment of a process, and shares the same memory space and resources as the process it belongs to. Threads are considered 'lightweight' because they use fewer resources than processes. Each thread runs in the context of the process, and multiple threads in the same process share the same data and code. Threads make tasks run faster by allowing for simultaneous execution of operations.

**Task 2:**

**Understand the below code and run it to see the output.. Need to explain…**

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 TestThread {

public static void main(String args[]) {

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

R1.start();

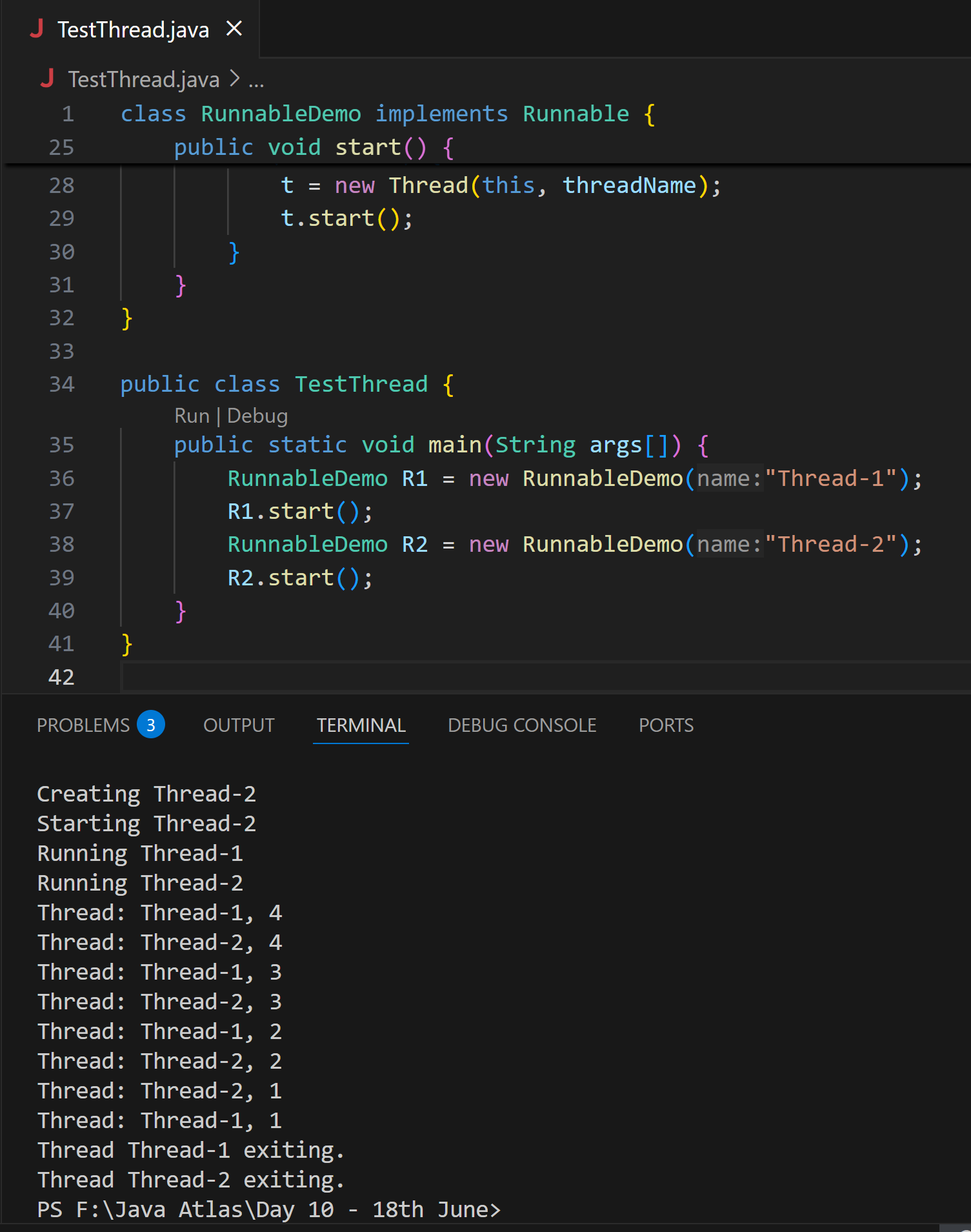
RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}

**Ans:** Done



**Task 3**

In the above code … try extending Thread class… and observe the output..

Understand:

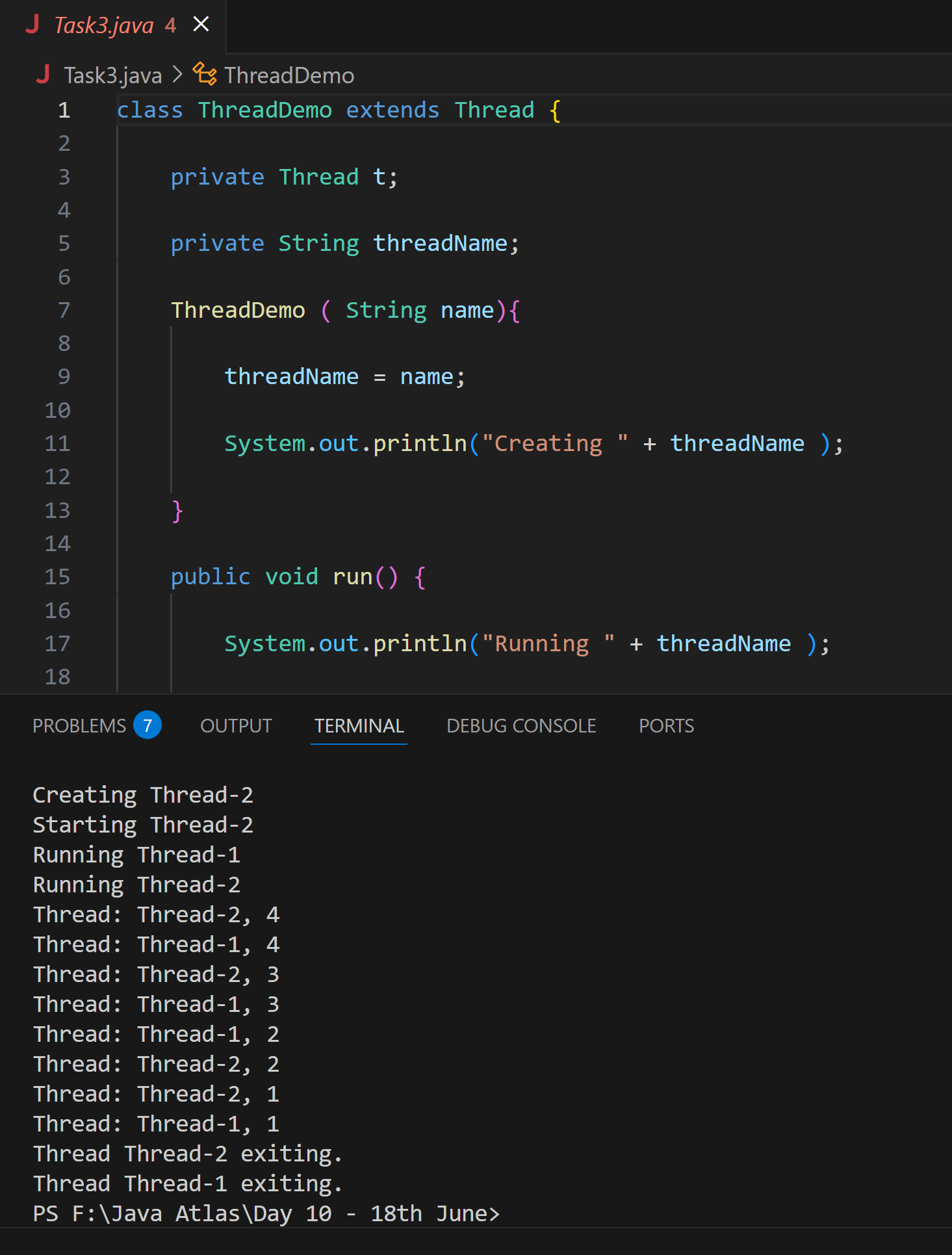
ex:

Public Class1 extends Thread{   }

Or

4public Class1 extends  Class2 implements Runnable (     )

**Ans:** Done



**Task 4:**

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 Main {

    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();

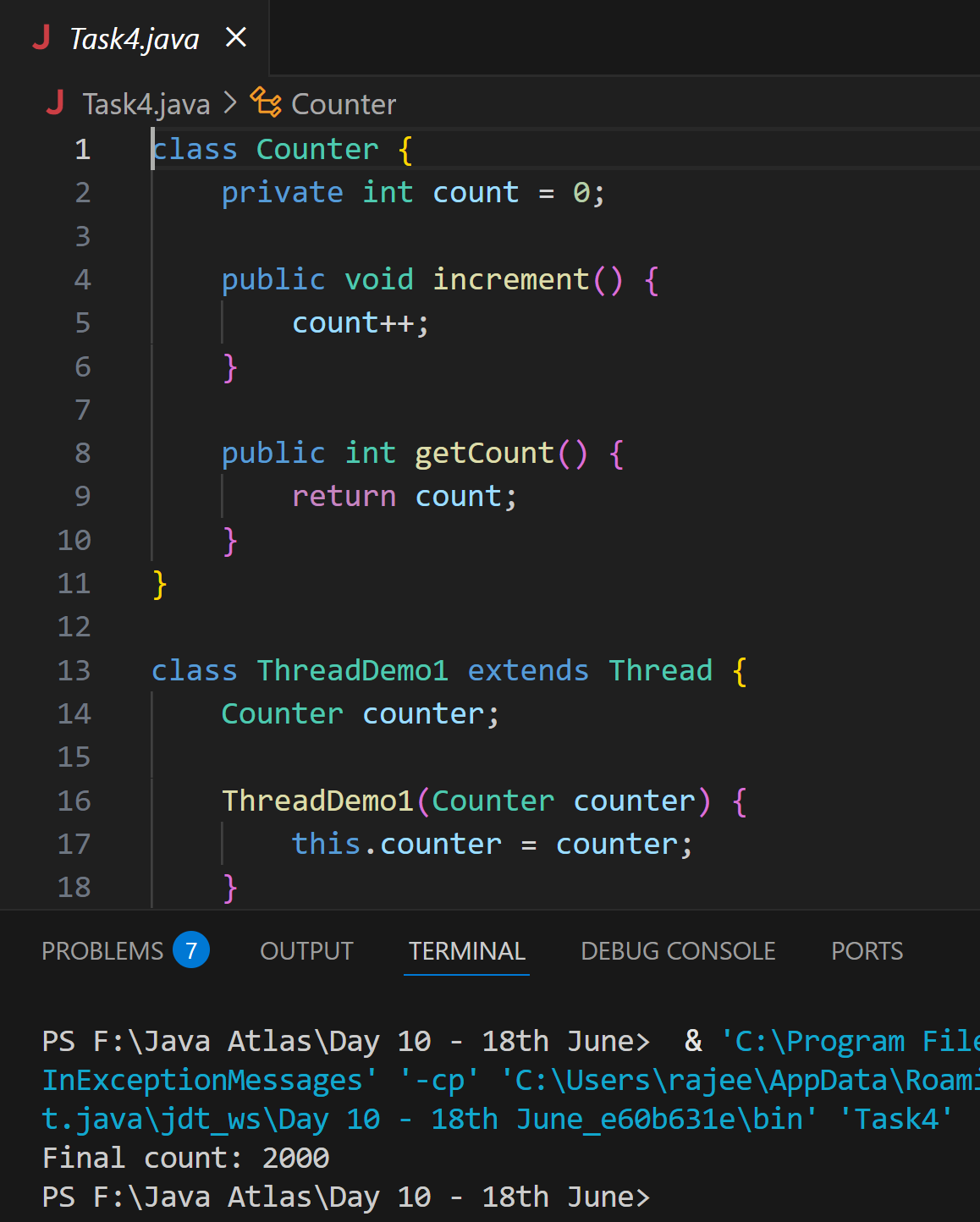
        }

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

    }

}

**Ans:** Done



**Task 5:**

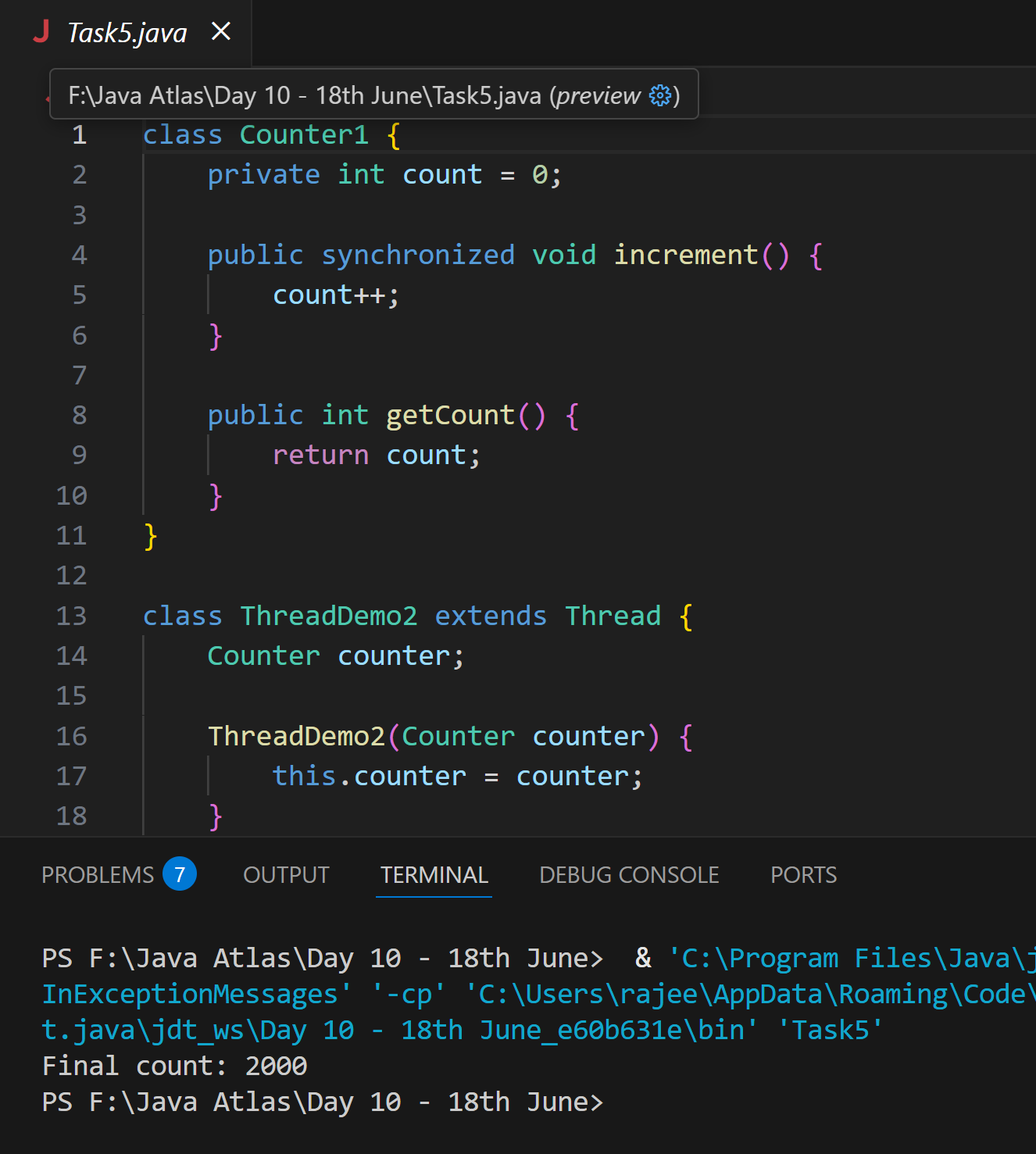
Use synchronized method:

Hint:

1. Synchronized Method:  
Synchronize the entire method to ensure only one thread can execute it at a time.

class Counter {  
    private int count = 0;  
  
    public synchronized void increment() {  
        count++;  
    }  
  
    public int getCount() {  
        return count;  
    }  
}

**Ans:** Done



**Task 6:**

Using Sync Block

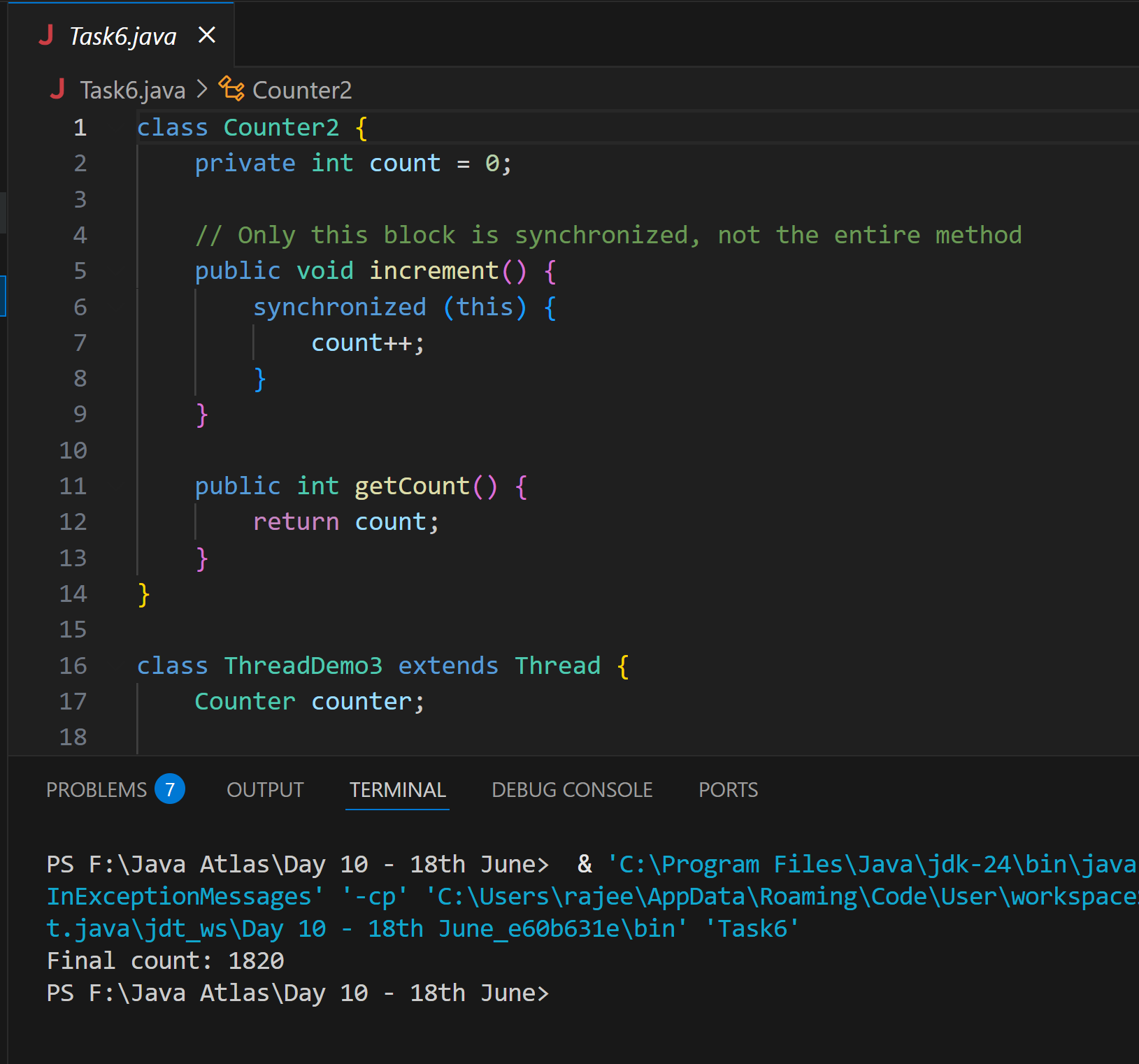
12.00 to 12.05

Hint:

2. Synchronized Block:  
Synchronize a block of code instead of the entire method, providing more control and efficiency.

class Counter {  
    private int count = 0;  
  
    public void increment() {  
        synchronized (this) {  
            count++;  
        }  
    }  
  
    public int getCount() {  
        return count;  
    }  
}

**Ans:** Done



**Task 7:**

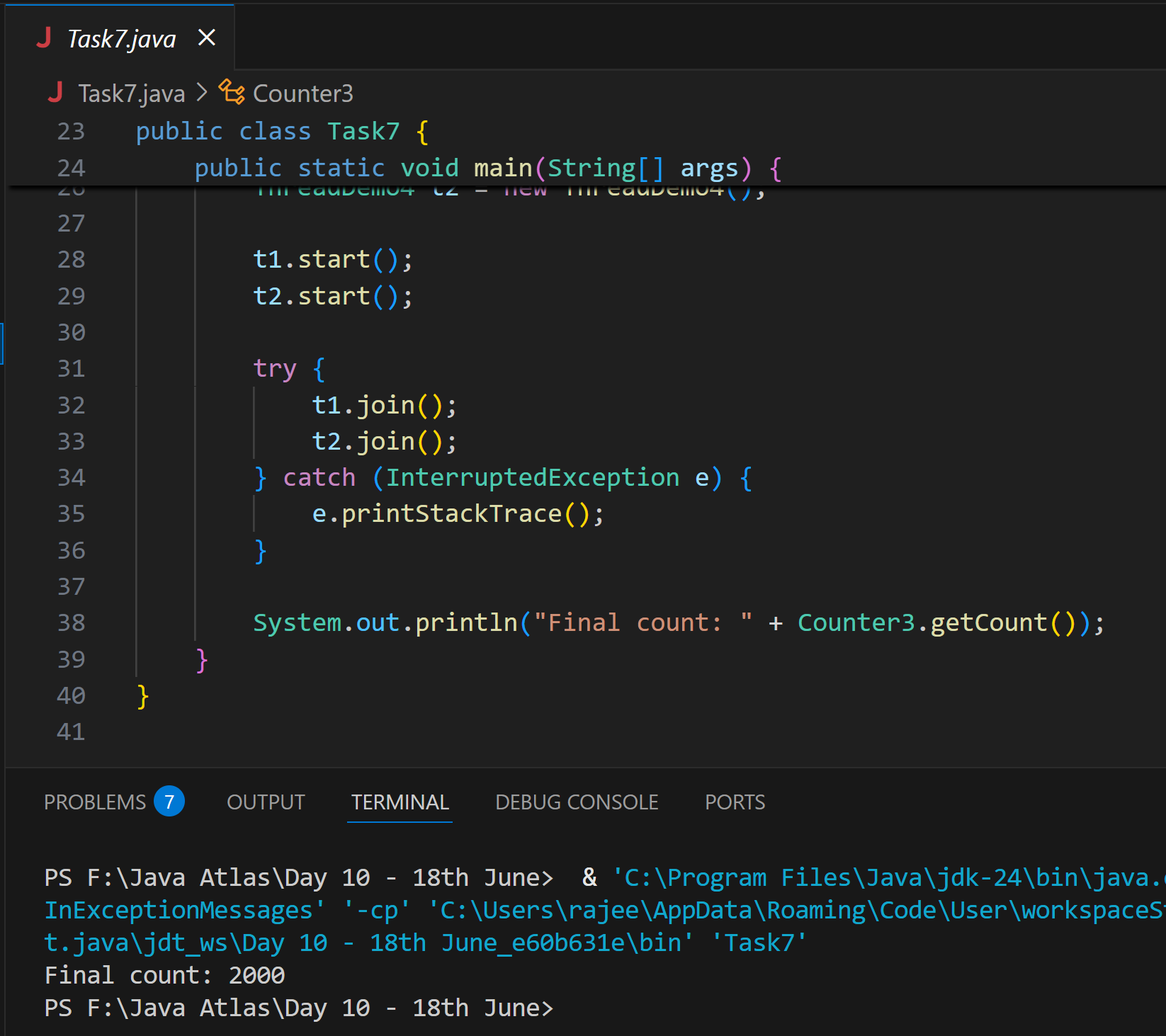
Using Static Sync

Hint:

3. Static Synchronization:  
Synchronize static methods to ensure only one thread can execute them for the class, not the instance.

class Counter {  
    private static int count = 0;  
  
    public static synchronized void increment() {  
        count++;  
    }  
  
    public static int getCount() {  
        return count;  
    }  
}

**Ans:** Done



**Task 8:**

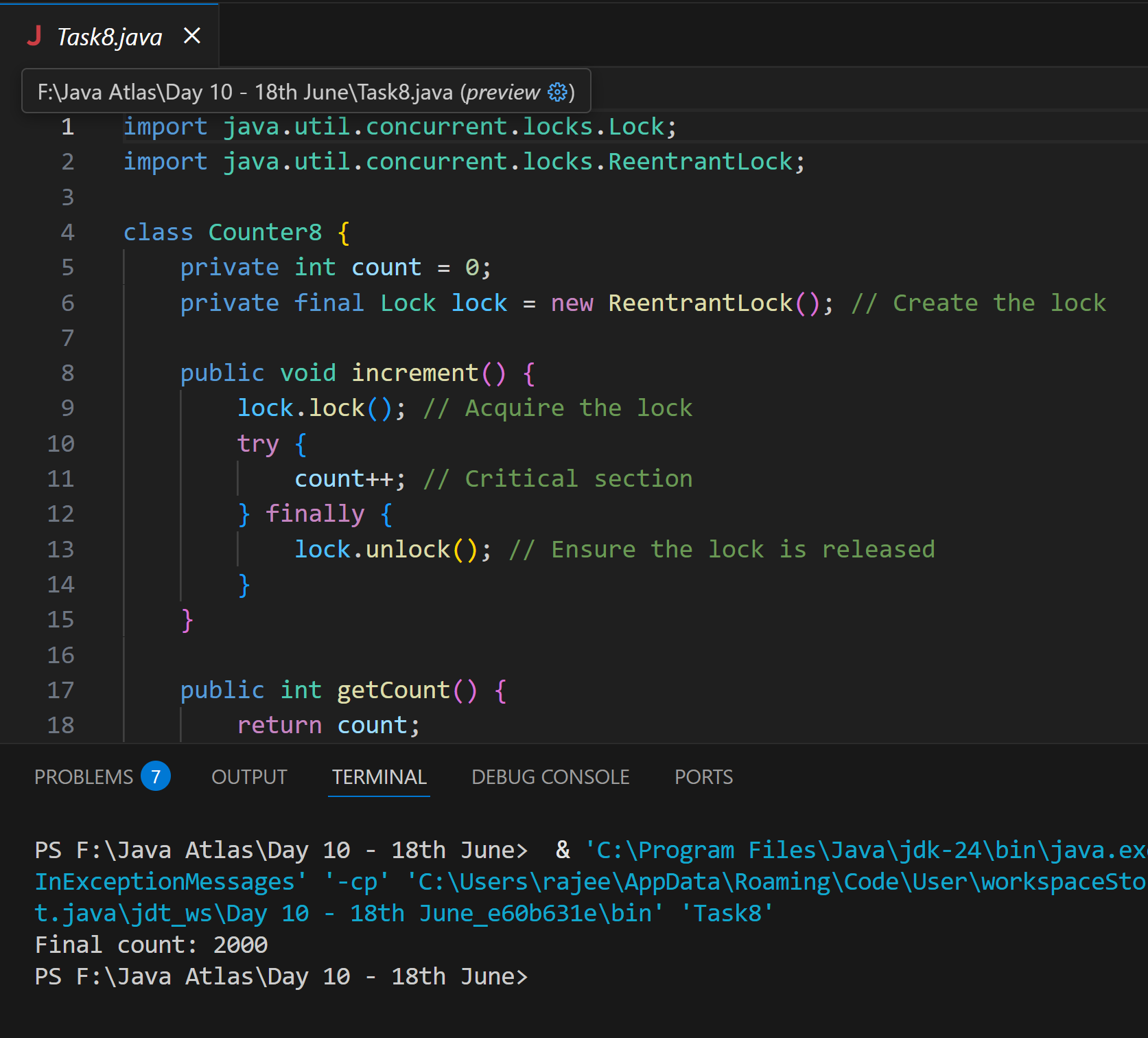
Using Locks

Hint:

4. Locks:  
Use `java.util.concurrent.locks.Lock` for more sophisticated thread synchronization.

import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
class Counter {  
    private int count = 0;  
    private final Lock lock = new ReentrantLock();  
  
    public void increment() {  
        lock.lock();  
        try {  
            count++;  
        } finally {  
            lock.unlock();  
        }  
    }  
  
    public int getCount() {  
        return count;  
    }  
}

**Ans:** Done



**Task 9:**

Sync Topic Rating will be done…

Answer: Done

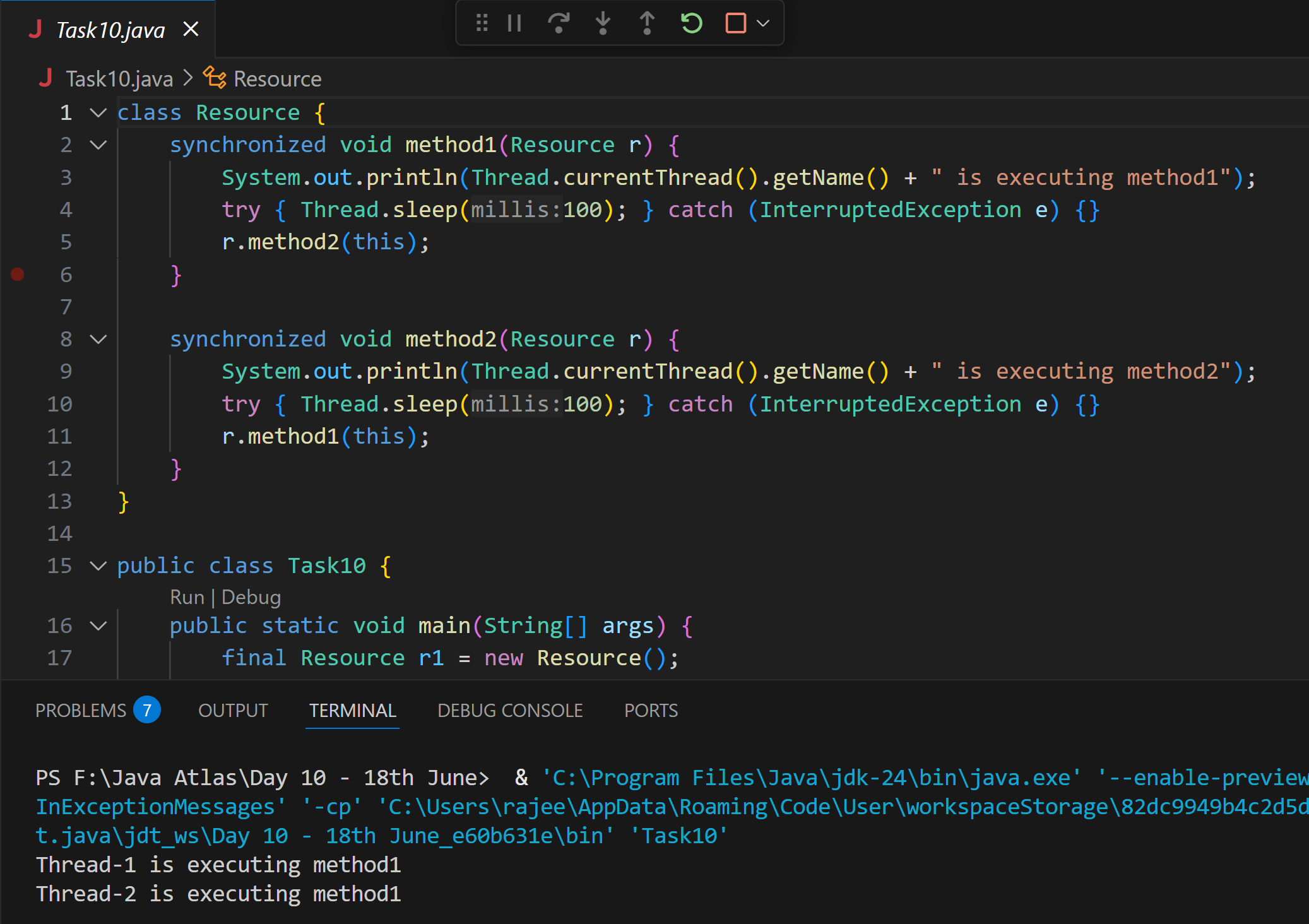
**Task 10:**

Dead Lock 👍

**Example of 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 DeadlockExample {  
    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");  
        Thread t2 = new Thread(() -> r2.method1(r1), "Thread-2");  
  
        t1.start();  
        t2.start();  
    }  
}

**Ans:** Done



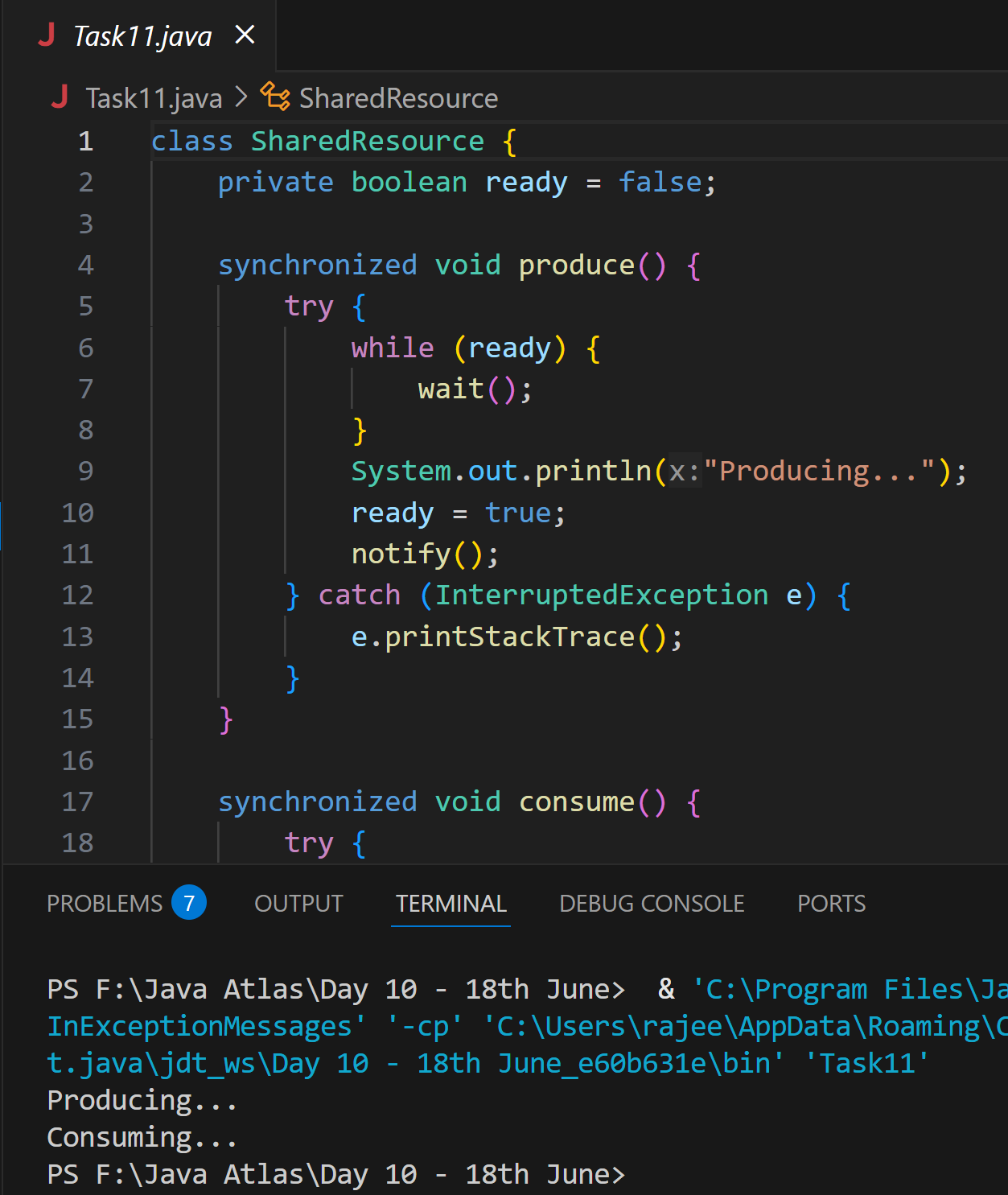
**Task 11:**

Inter- thread communication…

Example of Inter-thread Communication

class SharedResource {  
    private boolean ready = false;  
  
    synchronized void produce() {  
        try {  
            while (ready) {  
                wait();  
            }  
            System.out.println("Producing...");  
            ready = true;  
            notify();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
  
    synchronized void consume() {  
        try {  
            while (!ready) {  
                wait();  
            }  
            System.out.println("Consuming...");  
            ready = false;  
            notify();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
public class InterThreadCommunicationExample {  
    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();  
    }  
}

**Ans:** Done



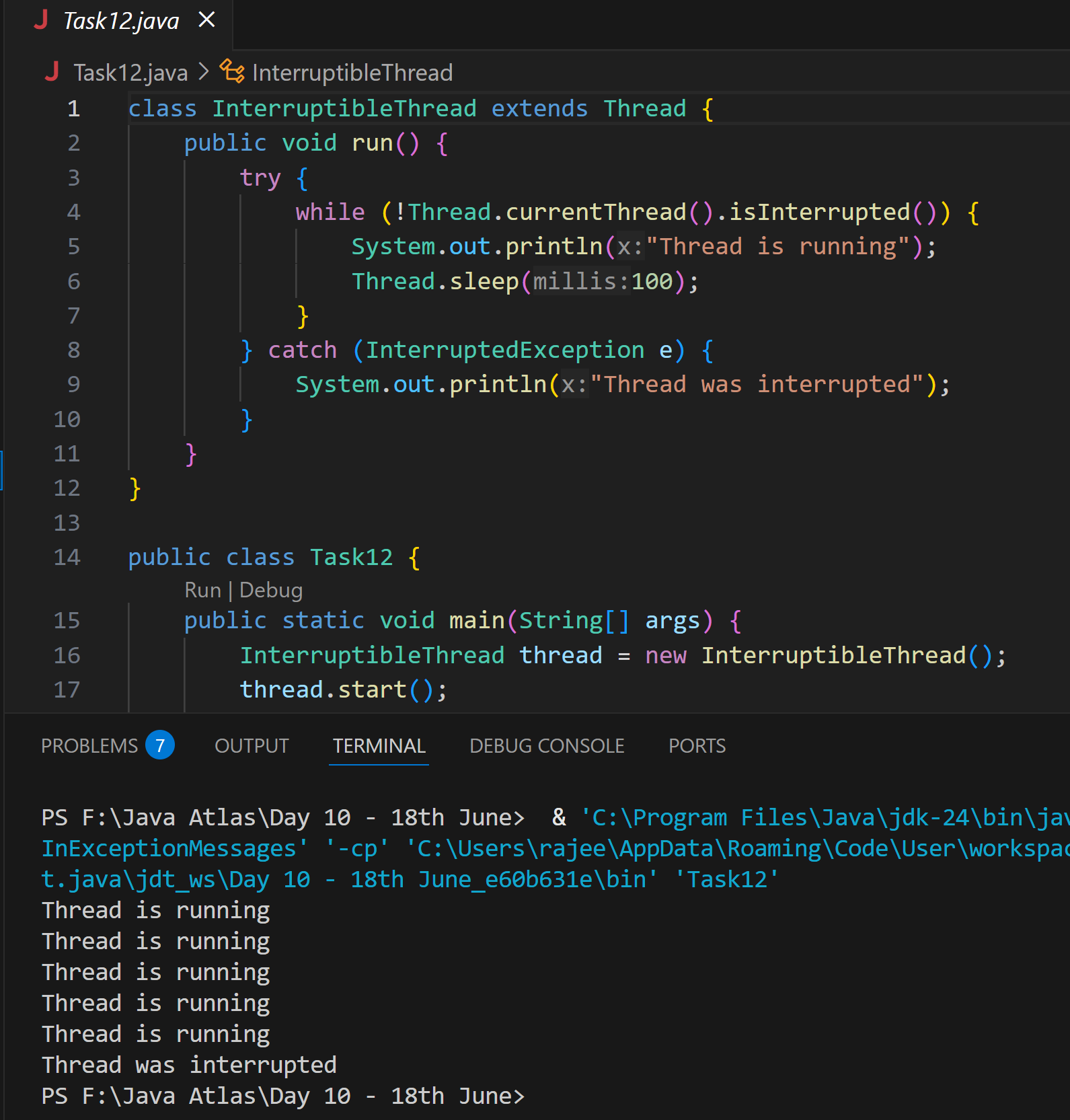
**Task 12:**

Interrupting a thread

**Example of Interrupting a Thread**

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 InterruptExample {  
    public static void main(String[] args) {  
        InterruptibleThread thread = new InterruptibleThread();  
        thread.start();  
  
        try {  
            Thread.sleep(500);  
            thread.interrupt();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}

**Ans:** Done



**Task 13**

Rating qn regarding Synchronisation…

**Task 14:**

**What is Thread pool?**

**Answer:**

A Thread Pool is a pool (or group) of worker threads that are reused to execute multiple tasks. Instead of creating a new thread every time a task is needed, you submit tasks to a thread pool, which assigns them to an available thread from the pool.

Creating and destroying threads repeatedly is expensive (in terms of system resources).

A thread pool improves performance by:

* Reusing threads
* Managing thread lifecycle efficiently
* Limiting the number of concurrent threads (avoiding resource exhaustion)
* Providing better control over task execution

**Task 15:**

Run the below code and see the file with the given name created or not..

Run it again with I like India instead of I love India.. And see the file …

public class WriteByte

{

public static void main(String args[])

{

File f1=new File(“FileName01.txt”); \\ to create new file FileOutputStream outfile = null;

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

try

{

outfile = new FileOutputStream(f1);

outfile.write(Text);

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

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

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

}

}

**Ans: Done**

****

**Task 16:**

Try this code to see the output …

**Write a program which reads byte from file.**

import java.io.\*;

public class ReadingByte

{

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)

{

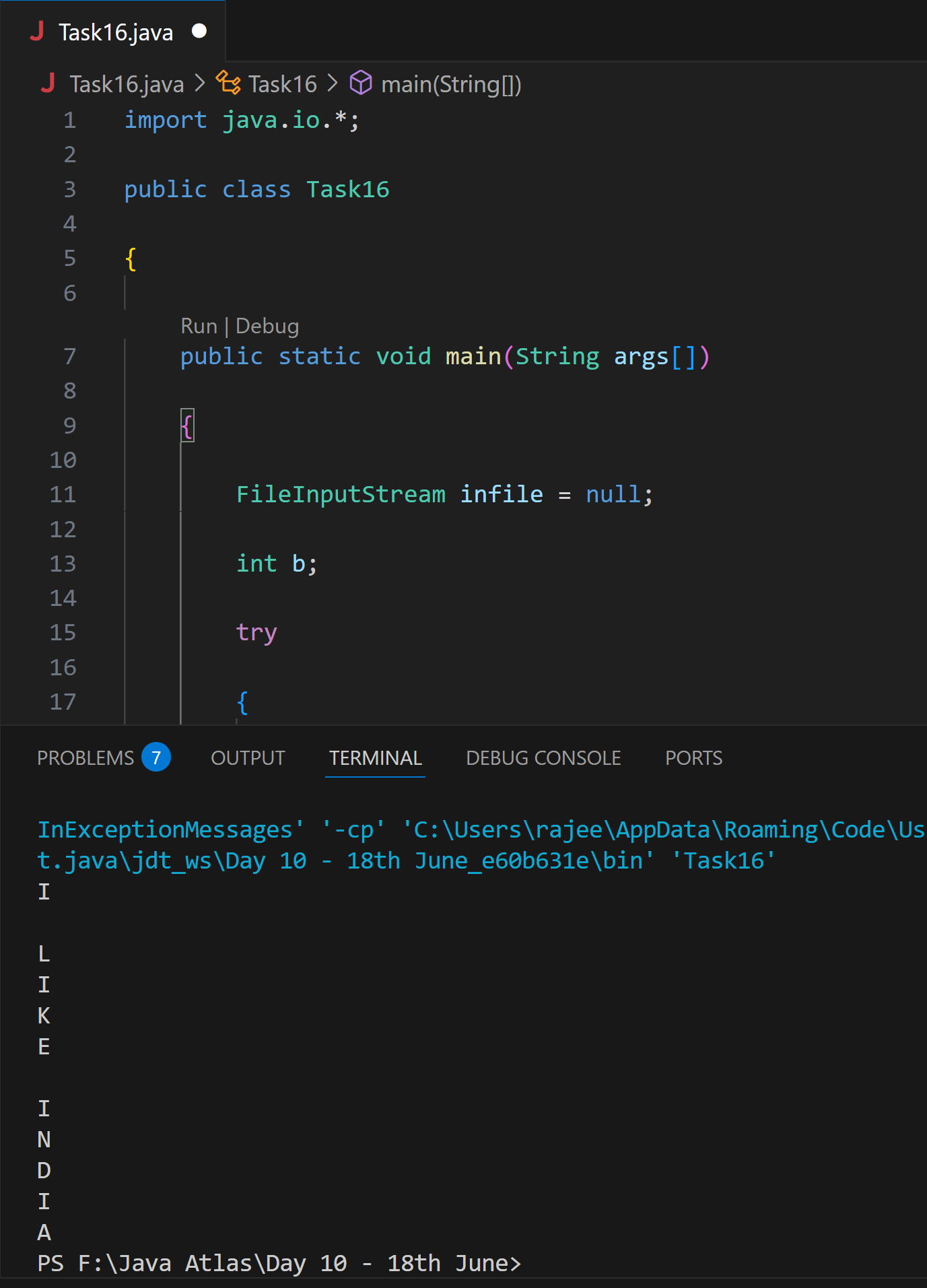
System.out.println("Sorry..!! File Not Found...!!!");

}

}

}

**Ans: Done**



**Task 17:**

Create  a file and see the output…

import java.io.\*;

import java.util.\*;

public class WriteByte\_1

{

public static void main(String args[]) {

FileOutputStream outfile = null;

//String s=args[0]; // to input string from command line Scanner sc=new Scanner(System.in);

String s=sc.nextLine();

byte b1[] = s.getBytes();

try

{

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

outfile.write(b1);

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

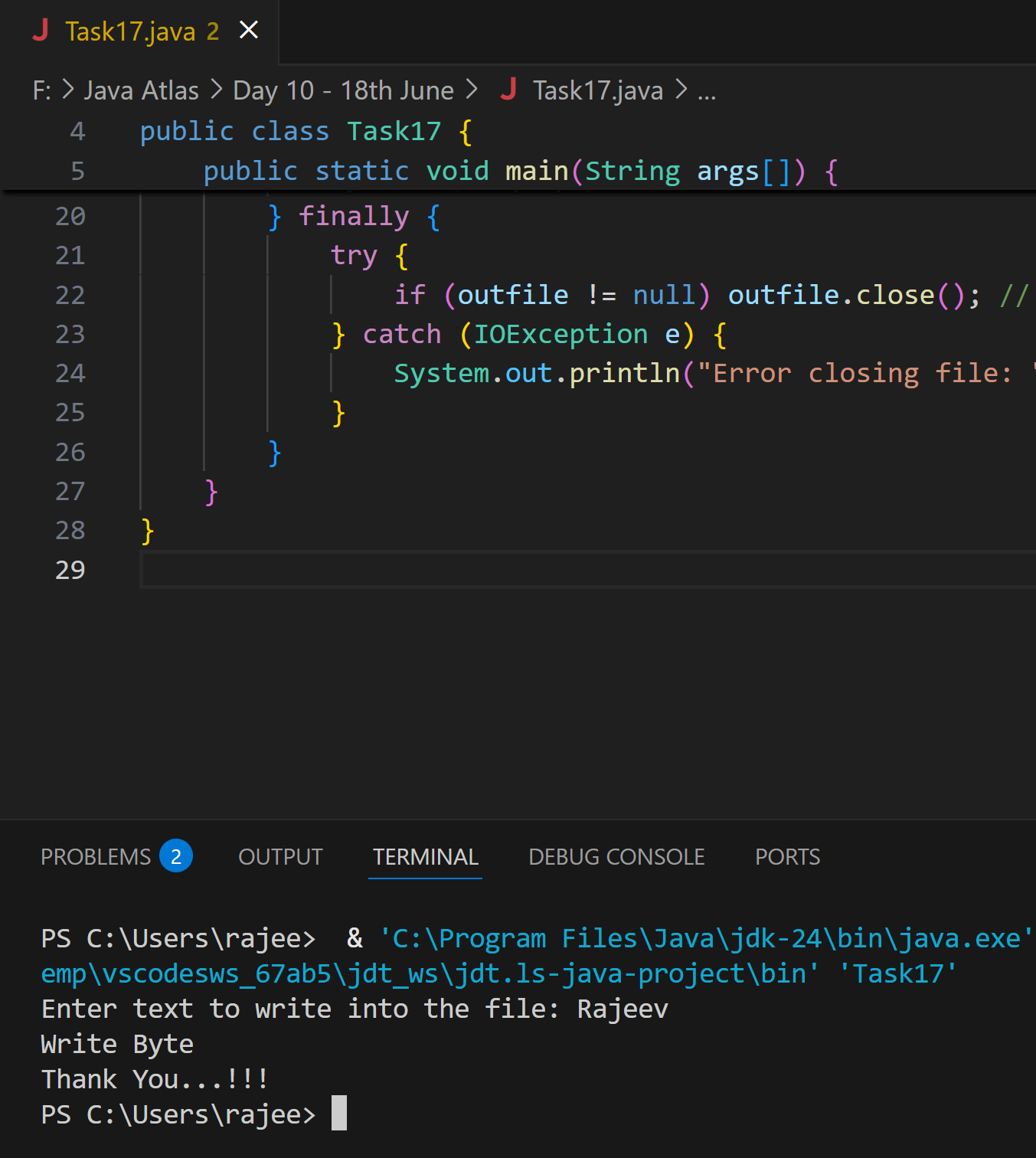
System.out.println("Write Byte");

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

}

}

**Ans: Done**



**Task 18:**

**Write a program which creates file and writes character into that file.**

import java.io.\*;

Class CharacterWrite {

public static void main(String args[]) {

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

FileWriterfw = null;

try {

fw=new FileWriter(f1);

fw.write("ahmedabad \n");

fw.write(" baroda \n");

fw.close();

}

catch(FileNotFoundException e)

{

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

}

catch(IOException e)

{

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

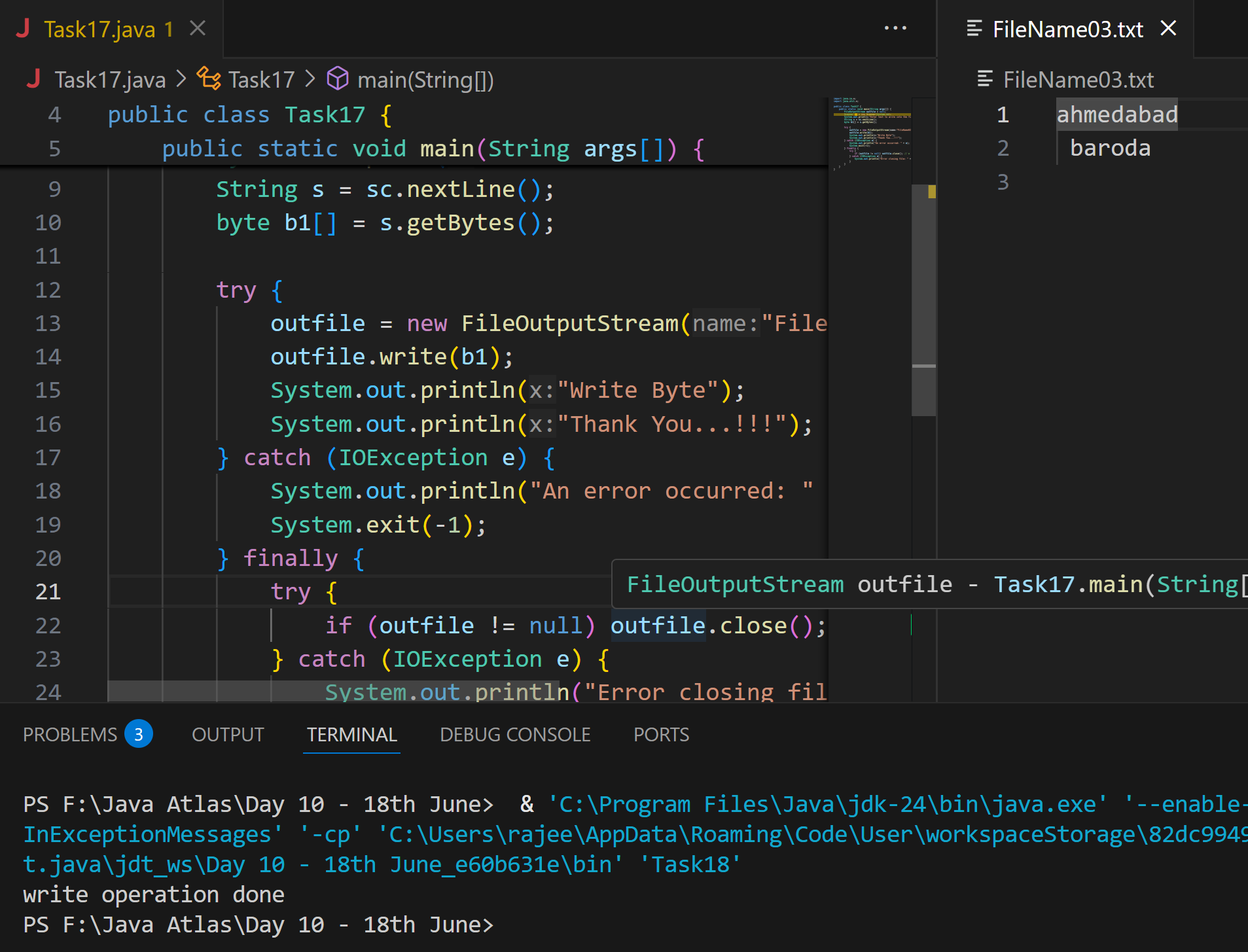
}

System.out.println(“ write operation done!!”);

}

}

**Ans: Done**



**Task 19**

**Write a program which reads character from file.**

import java.io.\*;

Class Readchar

{

public static void main(String args[])

{

FileReader fr =null;

try

{

fr = new FileReader("FileName03.txt");

int ch;

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

{

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

}

System.out.println("Reading complete");

fr.close();

}

catch(FileNotFoundException e)

{

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

}

catch(IOException e)

{

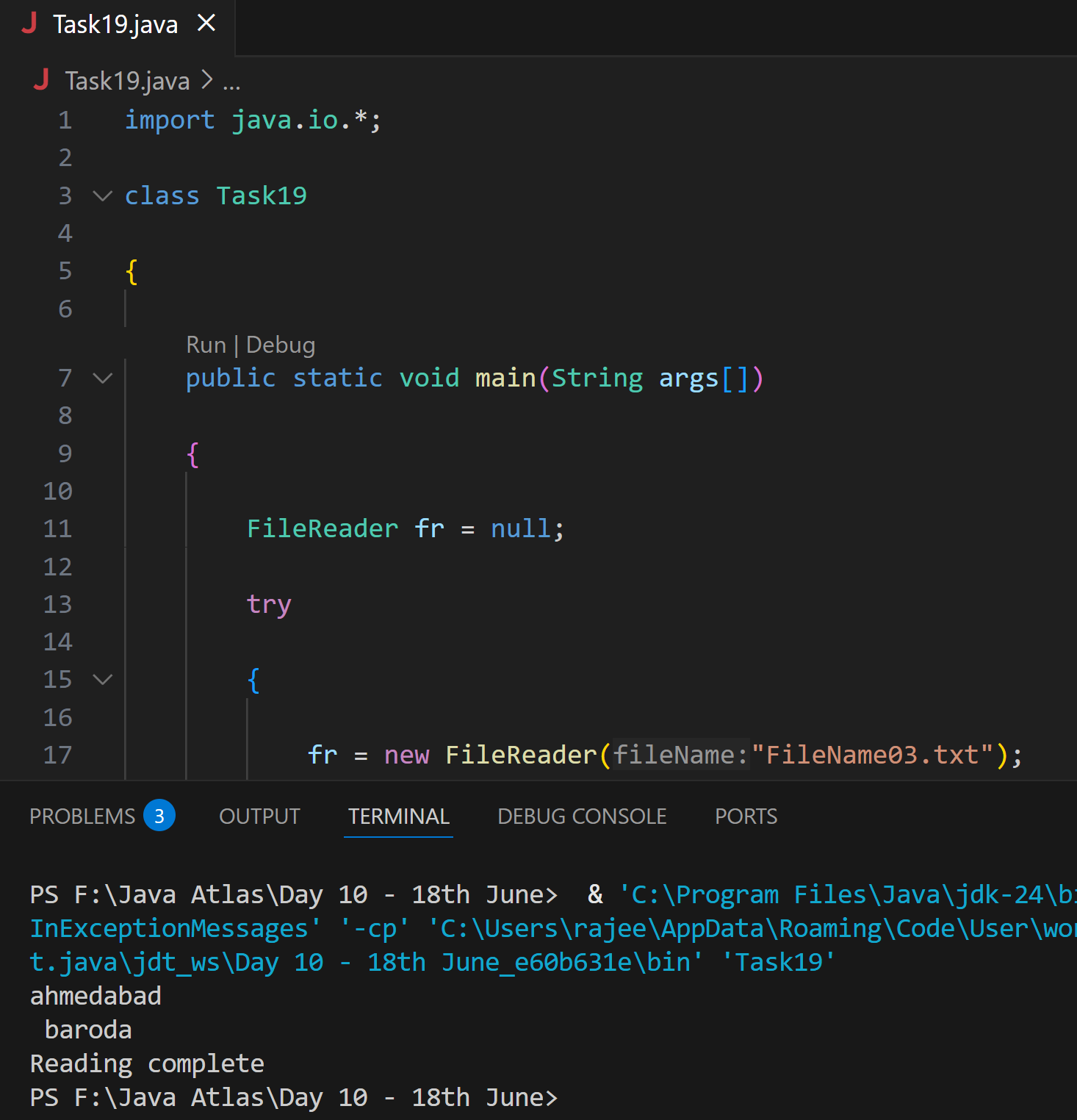
System.out.println(e.getMessage());

}

}

}

**Ans: Done**



**Task 20**

**Write a program to read one byte at a time from a file and copy it into another  file immediately**.

import java.io.\*;

classCopyByte

{

public static void main(String args[])

{

try

{

byte b=0;

FileInputStream infile = new FileInputStream("NewFile04.txt");

FileOutputStreamoutfile = new FileOutputStream("NewFile05.txt");

while(byteread != -1)

{

b = (byte)infile.read();

outfile.write(b);

}

System.out.println("Byte Copied From in.txt to out.txt FIle ");

}

catch(FileNotFoundException e)

{

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

}

catch(IOException e)

{

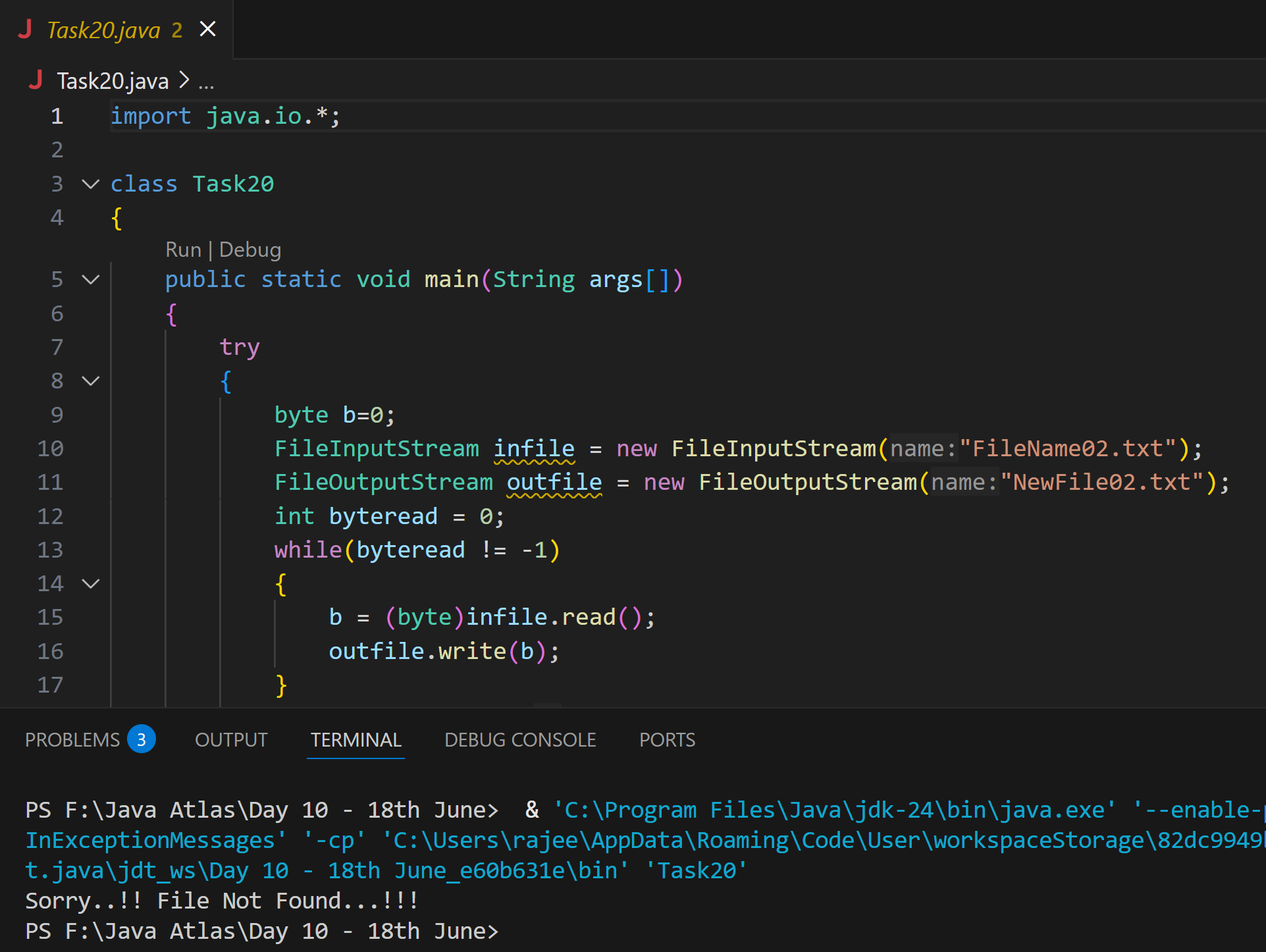
System.out.println(e.getMessage());

}

}

}

**Ans: Done**

****

**Task 21:**

Merging two files to 3rd file..

**Write a program to merge two files in third file.**

import java.io.\*;

classFileMergeDemo

{

public static void main(String args[])

{

try

{

FileInputStream file1 = new FileInputStream("File1.txt"); FileInputStream file2 = new FileInputStream("File2.txt"); SequenceInputStream file3 = new SequenceInputStream(file1, file2); BufferedInputStream br1 = new BufferedInputStream(file3); BufferedOutputStream br2 = new BufferedOutputStream(System.out); intch;

while((ch = br1.read())!=-1)

{

br2.write((char)ch);

}

br1.close();

br2.close();

file1.close();

file2.close();

System.out.println("Merge Two File Sucessfully ");

}

catch(IOException e)

{

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

}

}

}

**Ans: Done**



**Task 22:**

**Write an application to rename a file. Use the renameTo() method of File to  accomplish**

/\*this task. The first command line argument is the old filename and the second is  the newfilename.

\*/

import java.io.\*;

classFileRenameDemo

{

public static void main(String args[])

{

File f1 = new File(args[0]);

File f2 = new File(args[1]);

f1.renameTo(f2);

System.out.println("Rename File " +f1+" To "+f2+" Sucessfully "); }

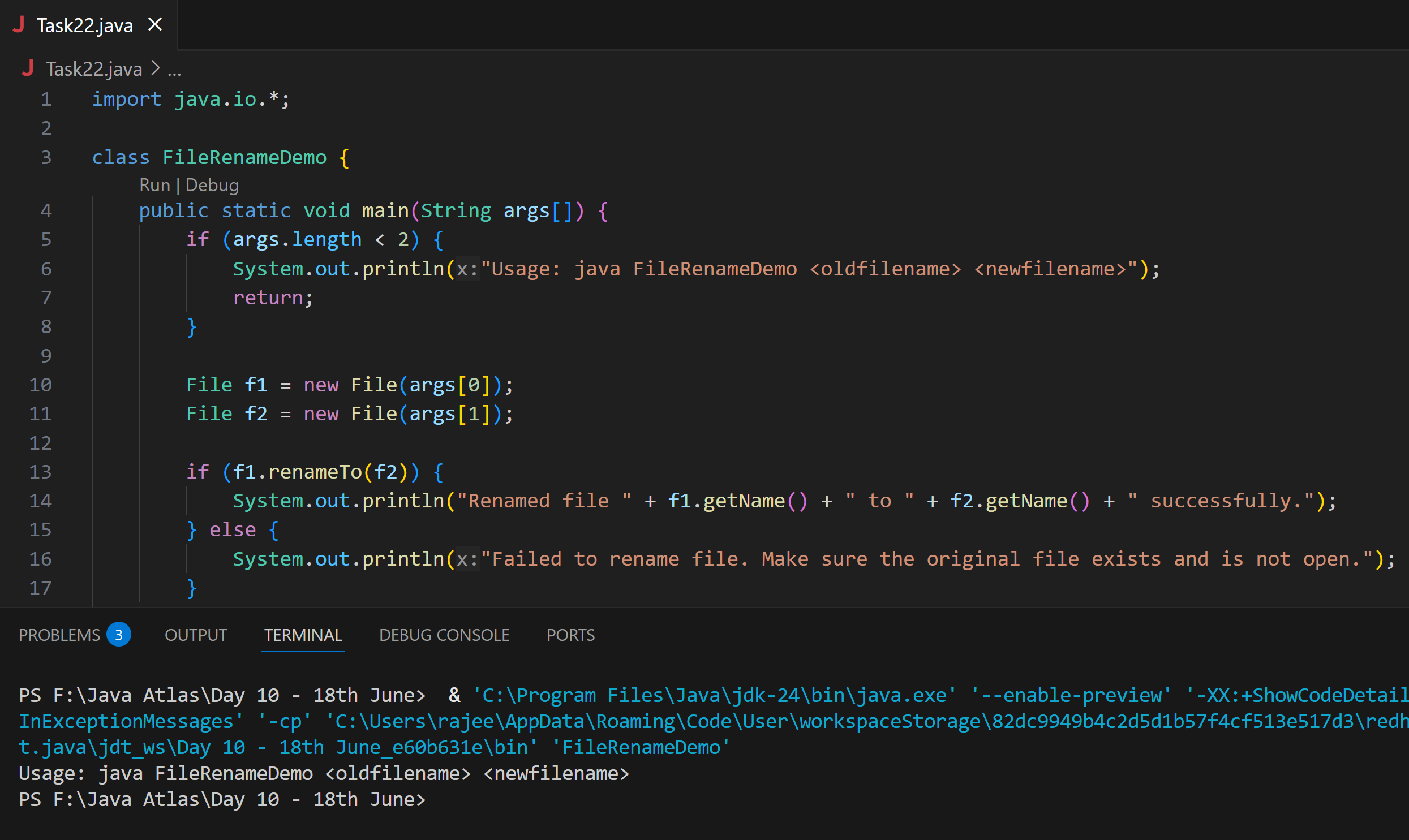
}

Output :

javacFileRenameDemo.java

javaFileRenameDemo input1.txt abc.txt

**Ans: Done**

****

**Task 23 👍**

==================================================

Buffered reader and writer — for large files to be read.

==================================================

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFileExample {

    public static void main(String[] args) {

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

            String line;

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

                System.out.println(line);

            }

        } catch (IOException e) {

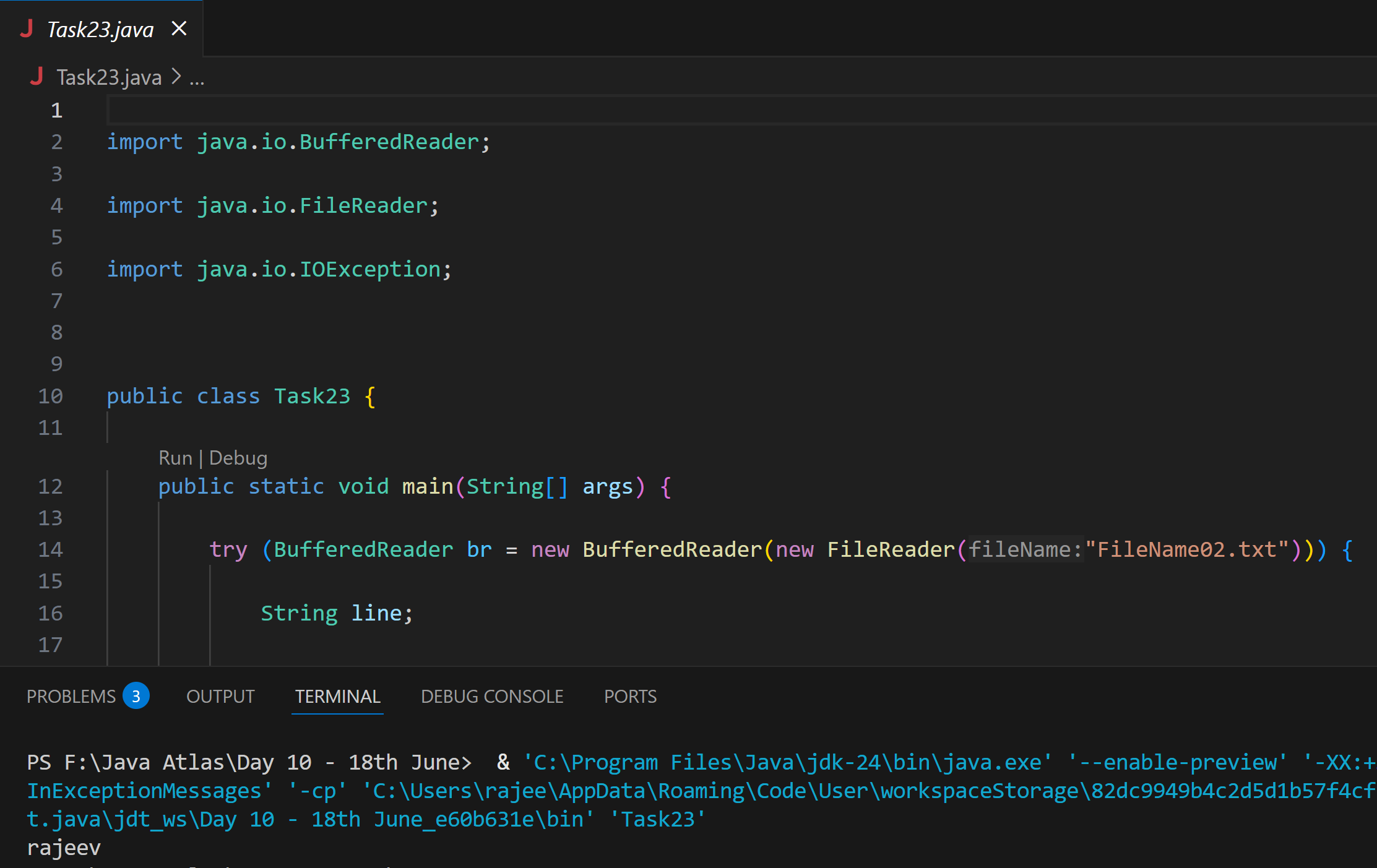
            e.printStackTrace();

        }

    }

}

**Ans: Done**



**Task 24** no n buffered

import java.io.FileReader;

import java.io.IOException;

public class NonBufferedReaderExample {

    public static void main(String[] args) {

        try (FileReader fr = new FileReader("largefile.txt")) {

            int ch;

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

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

            }

        } catch (IOException e) {

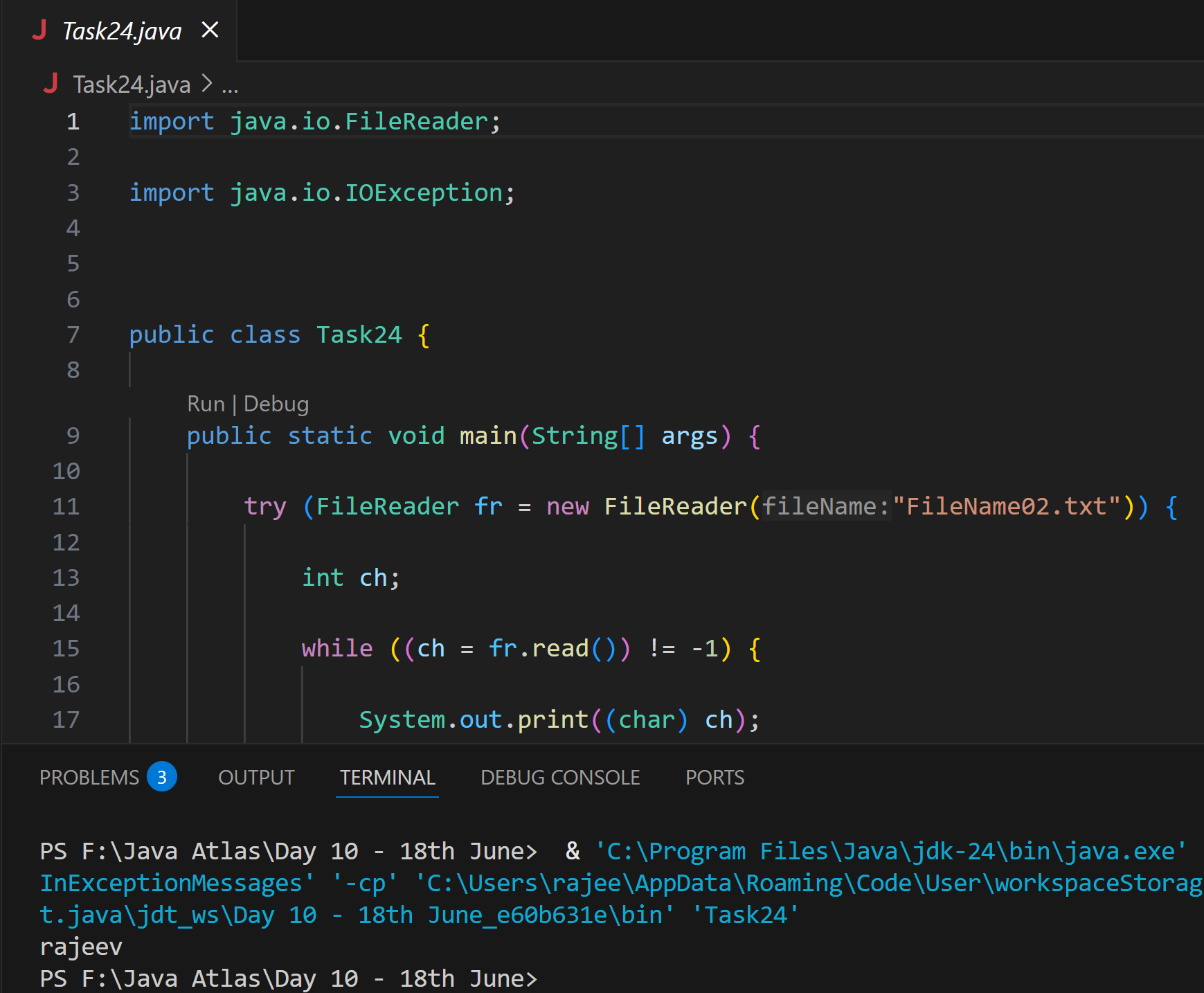
            e.printStackTrace();

        }

    }

}

**Ans: Done**



**Task 25:**

Buffered Writer

import java.io. BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

public class BufferedWriterDemo {

   public static void main(String[] args) {

      String filePath = "example.txt";

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

      // Initialize BufferedWriter with a FileWriter

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

         // Write content to the file

         writer.write(content);

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

      } catch (IOException e) {

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

      }

   }

}

**Ans: Done**

