Day 10 - 20th June 2025

|  |  |
| --- | --- |
| Advance concepts | Collections Framework intro, Streams, File I/O, Multithreading overview |
| Trobuleshooting | Debugging Tools, Error Messages and Stack Traces, Breakpoints and Code Stepping, Logging for Debugging, Common Bug Patterns, Debugging Strategies, Hands-on Debugging Practice |

Multi Threading:

Task 1:

What is a Process?

Processes are basically the programs that are dispatched from the ready state and are scheduled in the CPU for execution. PCB ( [Process Control Block](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/) ) holds the context of process. A process can create other processes which are known as Child Processes. The process takes more time to terminate, and it is isolated means it does not share the memory with any other process. The process can have the following [states](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/) new, ready, running, waiting, terminated and suspended.

Task 2:

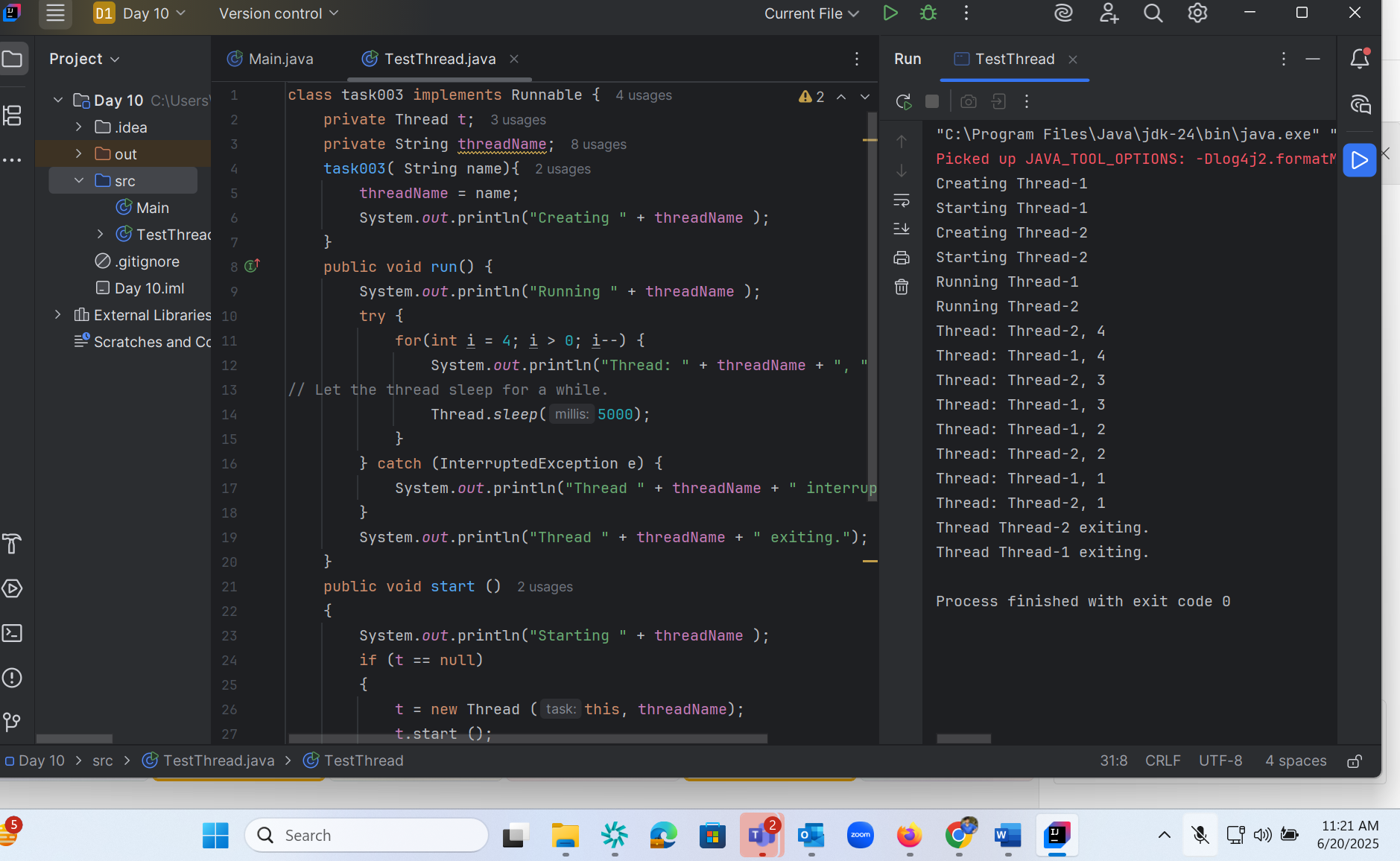
What is a Thread?

A thread is the smallest unit of execution within a process.

Lightweight unit within a process Shares memory with other threads Example Multiple tabs in Chrome

When you open a text editor like Notepad or run a Python script, the operating system creates a process for that program to execute.

Task 3:

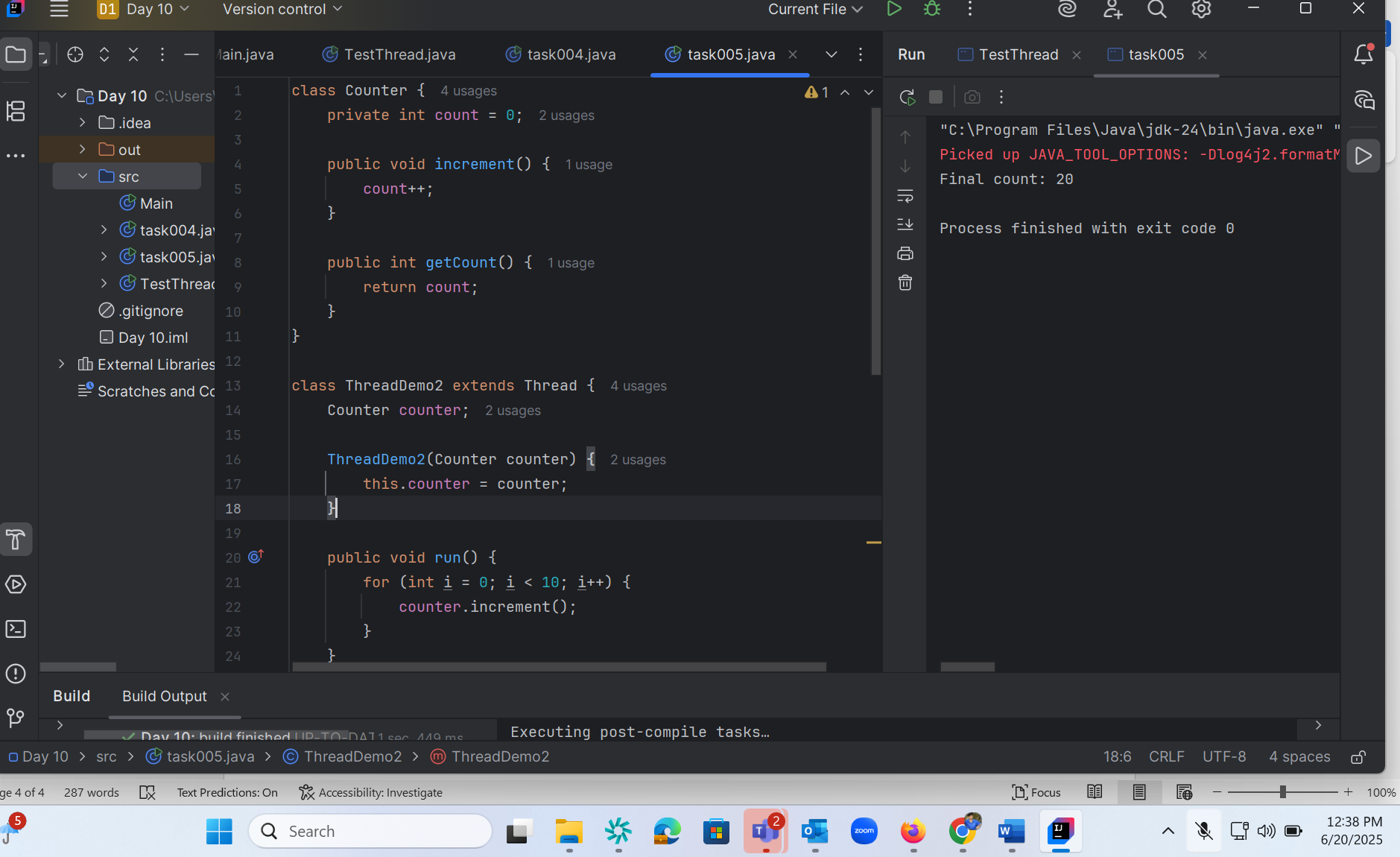


Task 4:



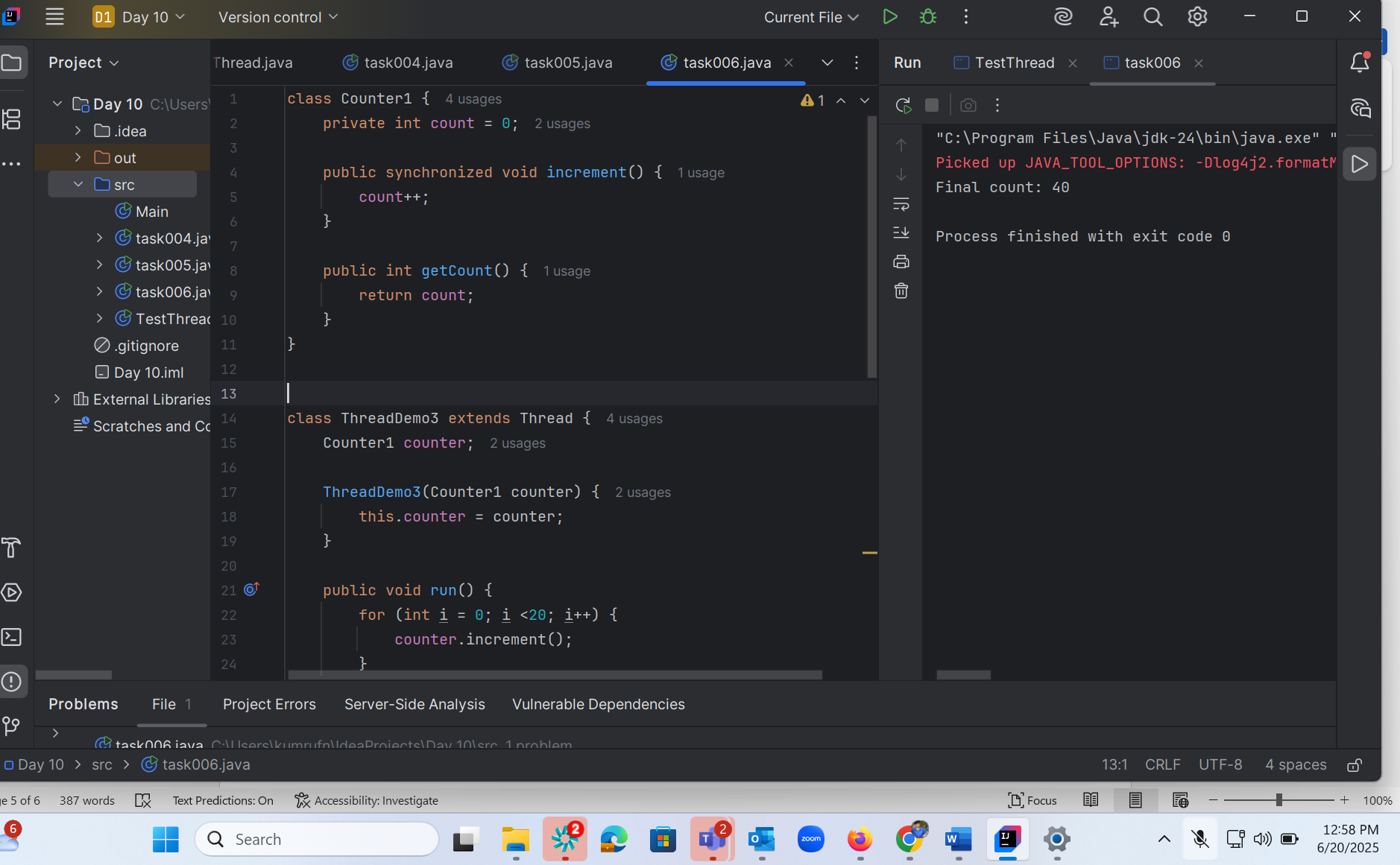
Task 5:

class Counter {  
 private int count = 0;  
  
 public void increment() {  
 count++;  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
class ThreadDemo2 extends Thread {  
 Counter counter;  
  
 ThreadDemo2(Counter counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 counter.increment();  
 }  
 }  
}  
  
public class task005 {  
 public static void main(String[] args) {  
 Counter counter = new Counter();  
 ThreadDemo2 t1 = new ThreadDemo2(counter);  
 ThreadDemo2 t2 = new ThreadDemo2(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + counter.getCount());  
 }  
}



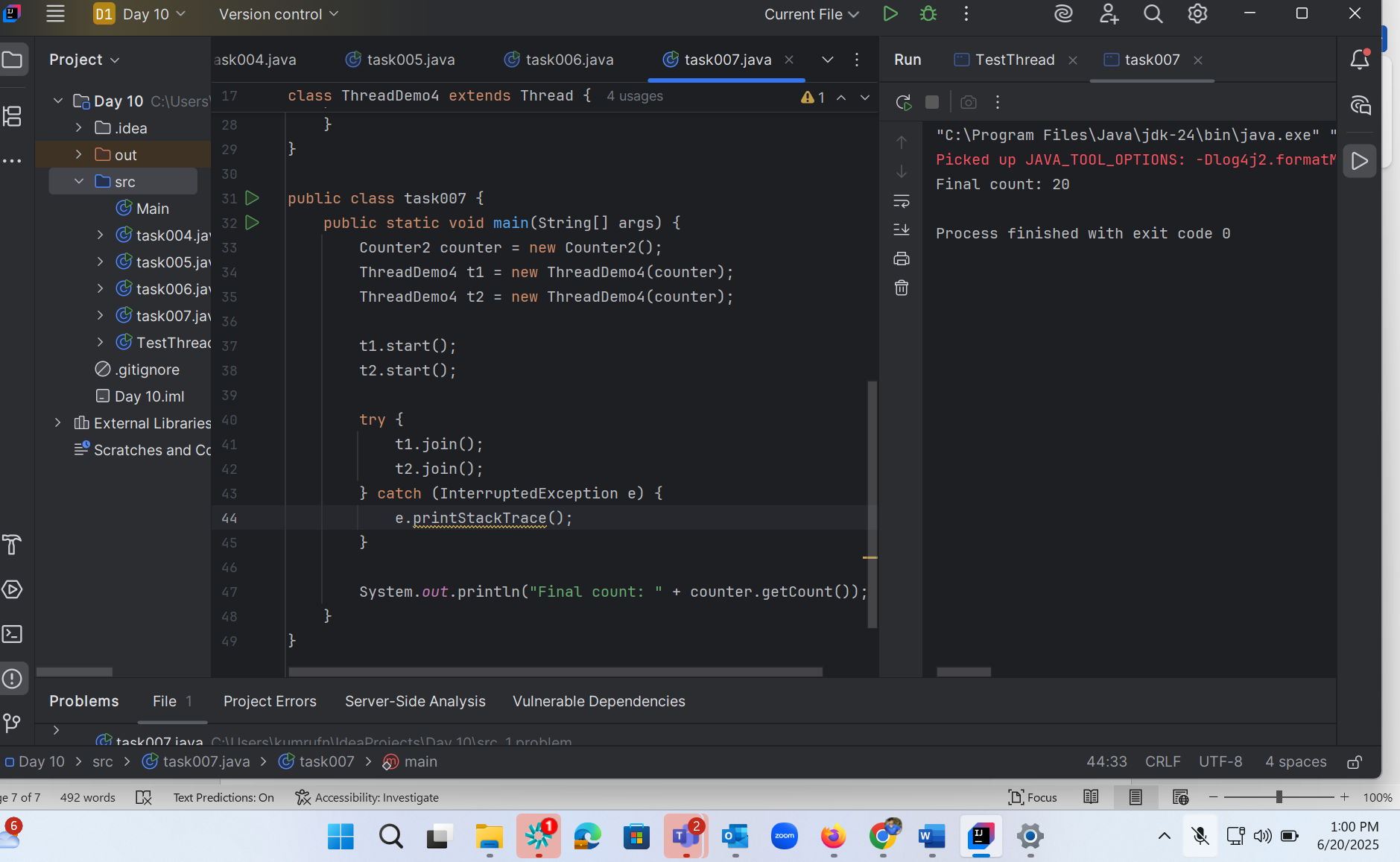
Task6:

class Counter1 {  
 private int count = 0;  
  
 public synchronized void increment() {  
 count++;  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
  
class ThreadDemo3 extends Thread {  
 Counter1 counter;  
  
 ThreadDemo3(Counter1 counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i <20; i++) {  
 counter.increment();  
 }  
 }  
}  
  
public class task006 {  
 public static void main(String[] args) {  
 Counter1 counter = new Counter1();  
 ThreadDemo3 t1 = new ThreadDemo3(counter);  
 ThreadDemo3 t2 = new ThreadDemo3(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + counter.getCount());  
 }  
}



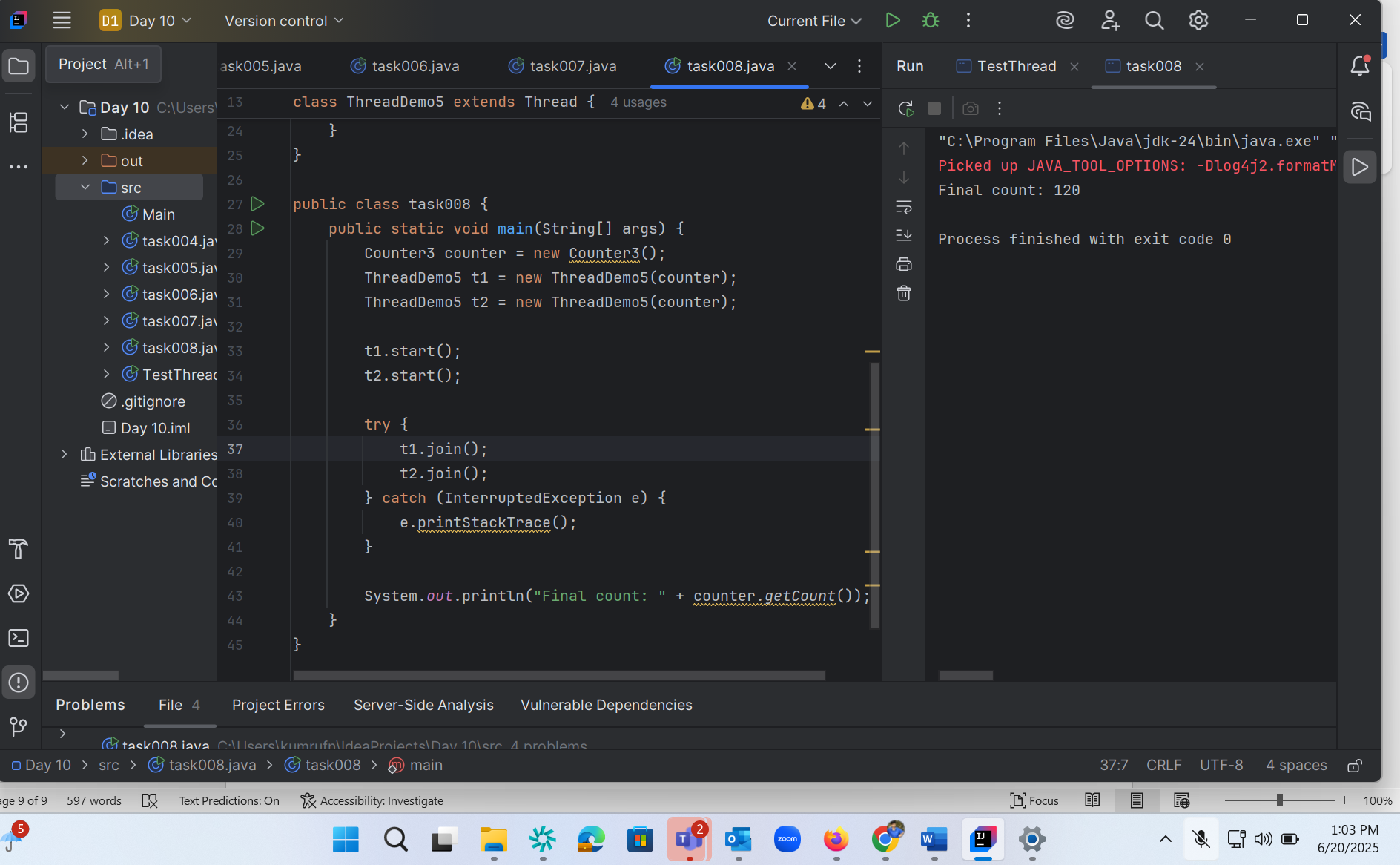
Task 7 :

class Counter2 {  
 private int count = 0;  
  
 public void increment() {  
 synchronized (this) {  
 count++;  
 }  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
  
  
class ThreadDemo4 extends Thread {  
 Counter2 counter;  
  
 ThreadDemo4(Counter2 counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i <10; i++) {  
 counter.increment();  
 }  
 }  
}  
  
public class task007 {  
 public static void main(String[] args) {  
 Counter2 counter = new Counter2();  
 ThreadDemo4 t1 = new ThreadDemo4(counter);  
 ThreadDemo4 t2 = new ThreadDemo4(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + counter.getCount());  
 }  
}



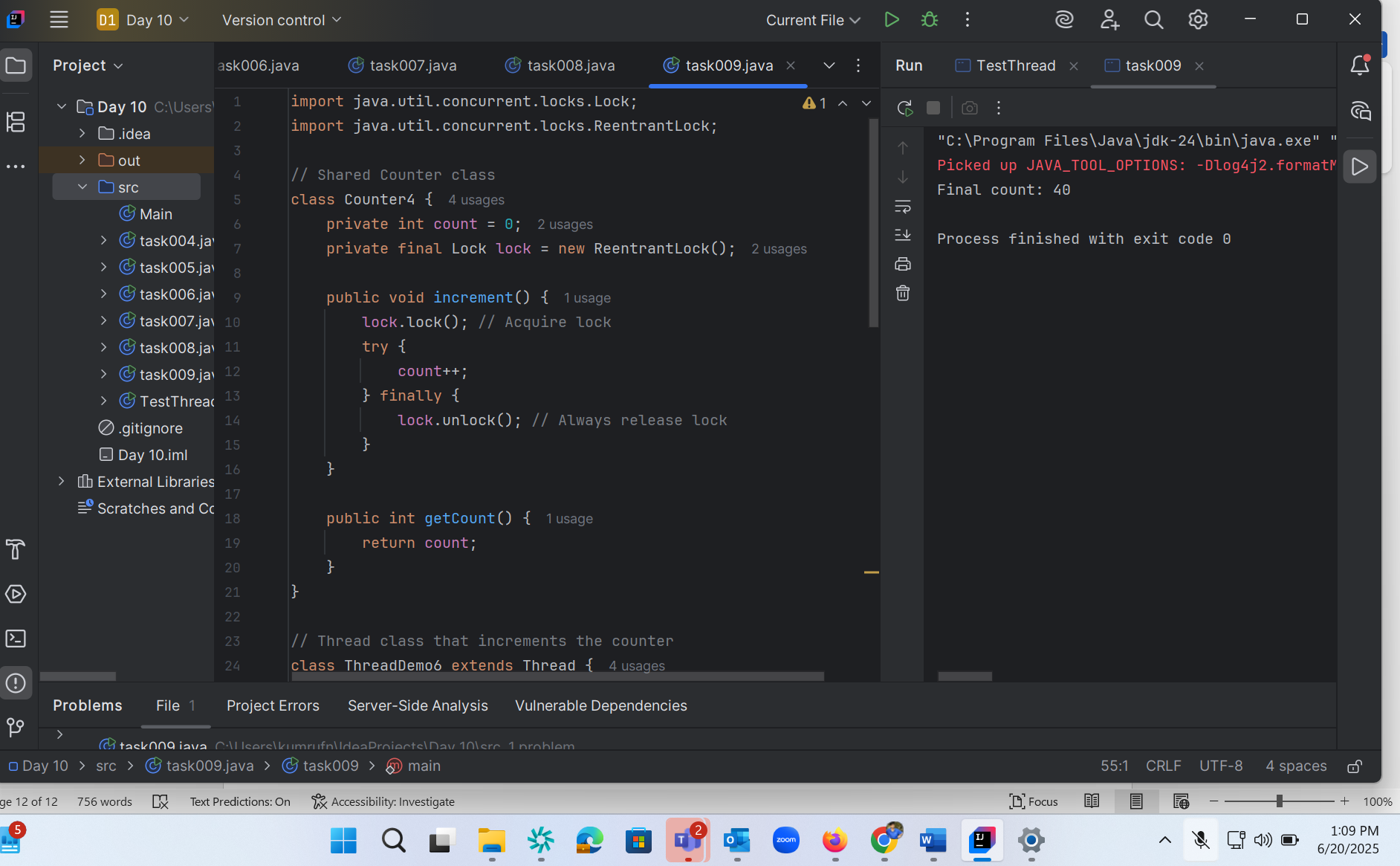
Task 8 :

class Counter3 {  
 private static int *count* = 0;  
  
 public static synchronized void increment() {  
 *count*++;  
 }  
  
 public static int getCount() {  
 return *count*;  
 }  
}  
  
class ThreadDemo5 extends Thread {  
 Counter3 counter;  
  
 ThreadDemo5(Counter3 counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i <60; i++) {  
 counter.*increment*();  
 }  
 }  
}  
  
public class task008 {  
 public static void main(String[] args) {  
 Counter3 counter = new Counter3();  
 ThreadDemo5 t1 = new ThreadDemo5(counter);  
 ThreadDemo5 t2 = new ThreadDemo5(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + counter.*getCount*());  
 }  
}



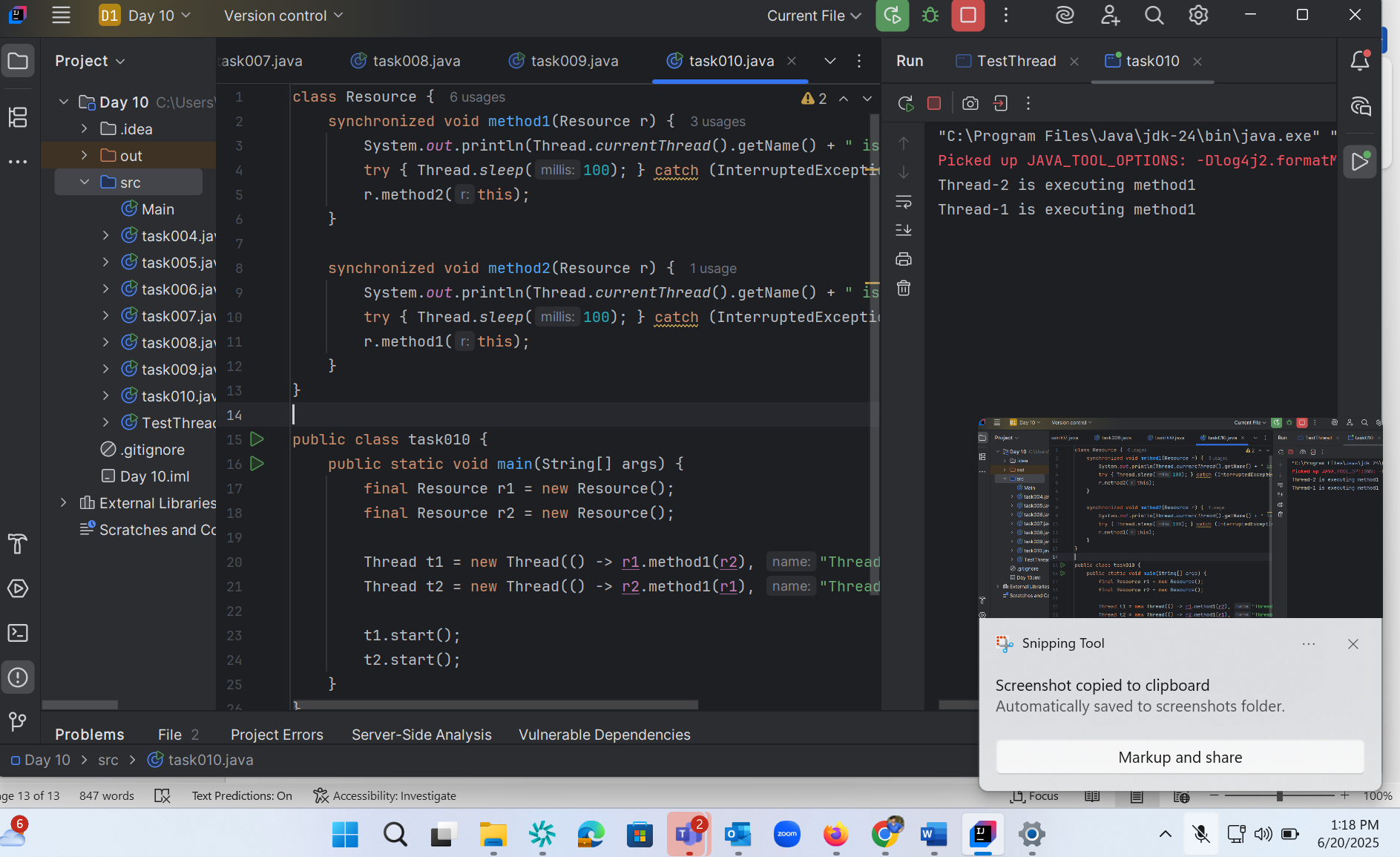
Task 9:

import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
// Shared Counter class  
class Counter4 {  
 private int count = 0;  
 private final Lock lock = new ReentrantLock();  
  
 public void increment() {  
 lock.lock(); // Acquire lock  
 try {  
 count++;  
 } finally {  
 lock.unlock(); // Always release lock  
 }  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
// Thread class that increments the counter  
class ThreadDemo6 extends Thread {  
 private final Counter4 counter;  
  
 ThreadDemo6(Counter4 counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 20; i++) {  
 counter.increment();  
 }  
 }  
}  
  
// Main class  
public class task009 {  
 public static void main(String[] args) {  
 Counter4 counter = new Counter4();  
  
 ThreadDemo6 t1 = new ThreadDemo6(counter);  
 ThreadDemo6 t2 = new ThreadDemo6(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();  
 }  
  
 // Expected output: 40  
 System.*out*.println("Final count: " + counter.getCount());  
 }  
}



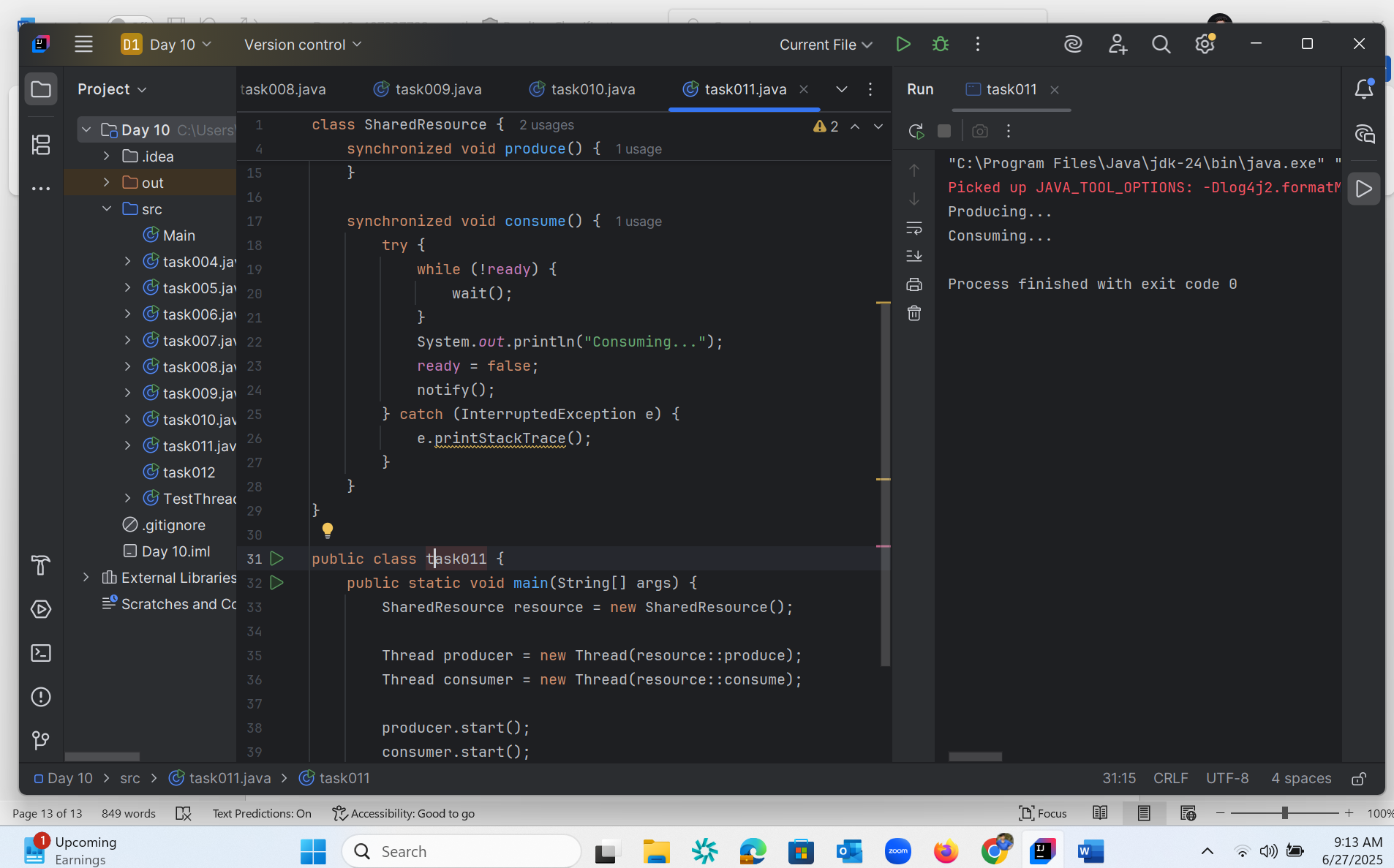
Task 10 :

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 task010 {  
 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();  
 }  
}



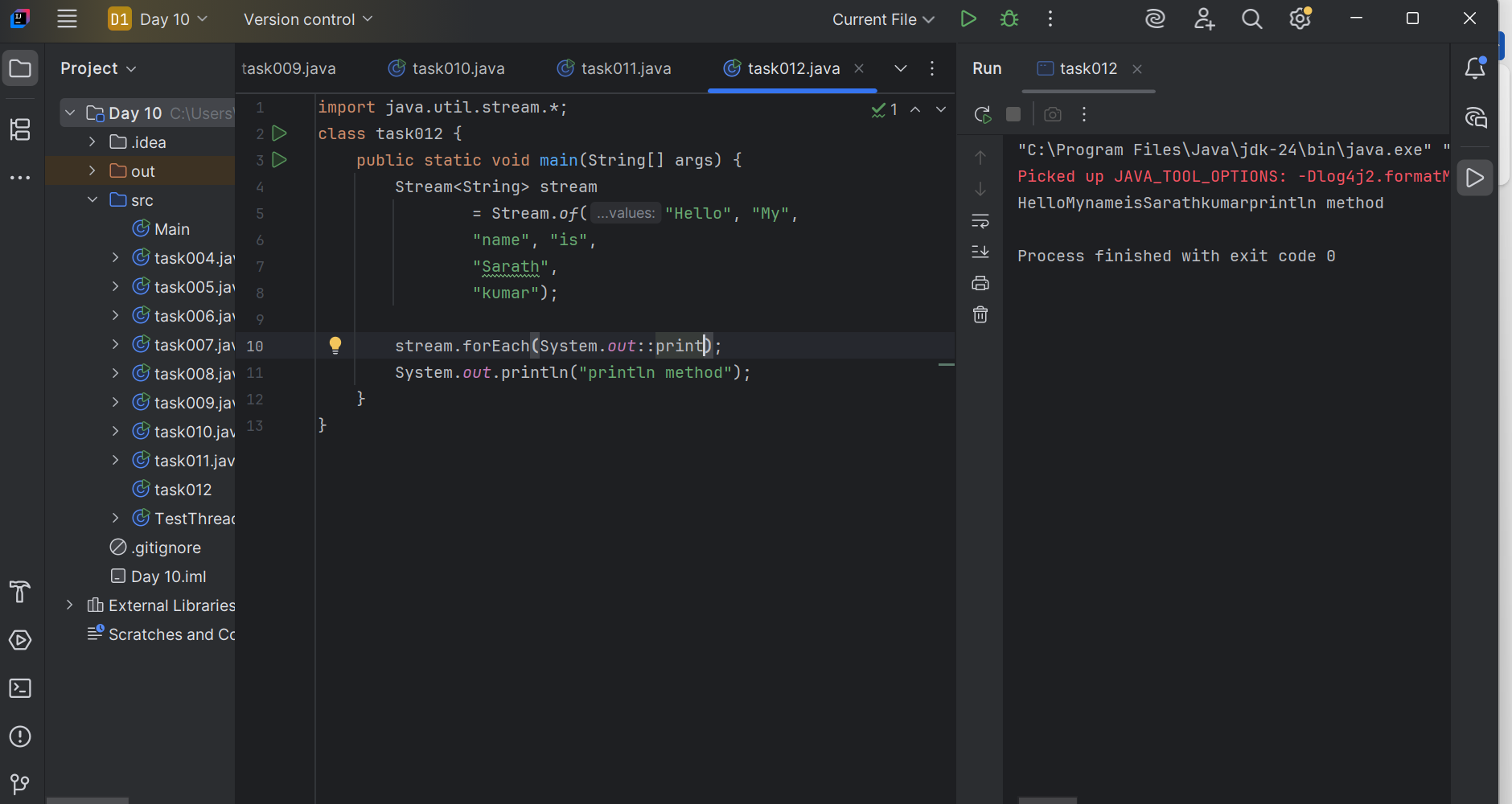
Task 11:

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 task011 {  
 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();  
 }  
}



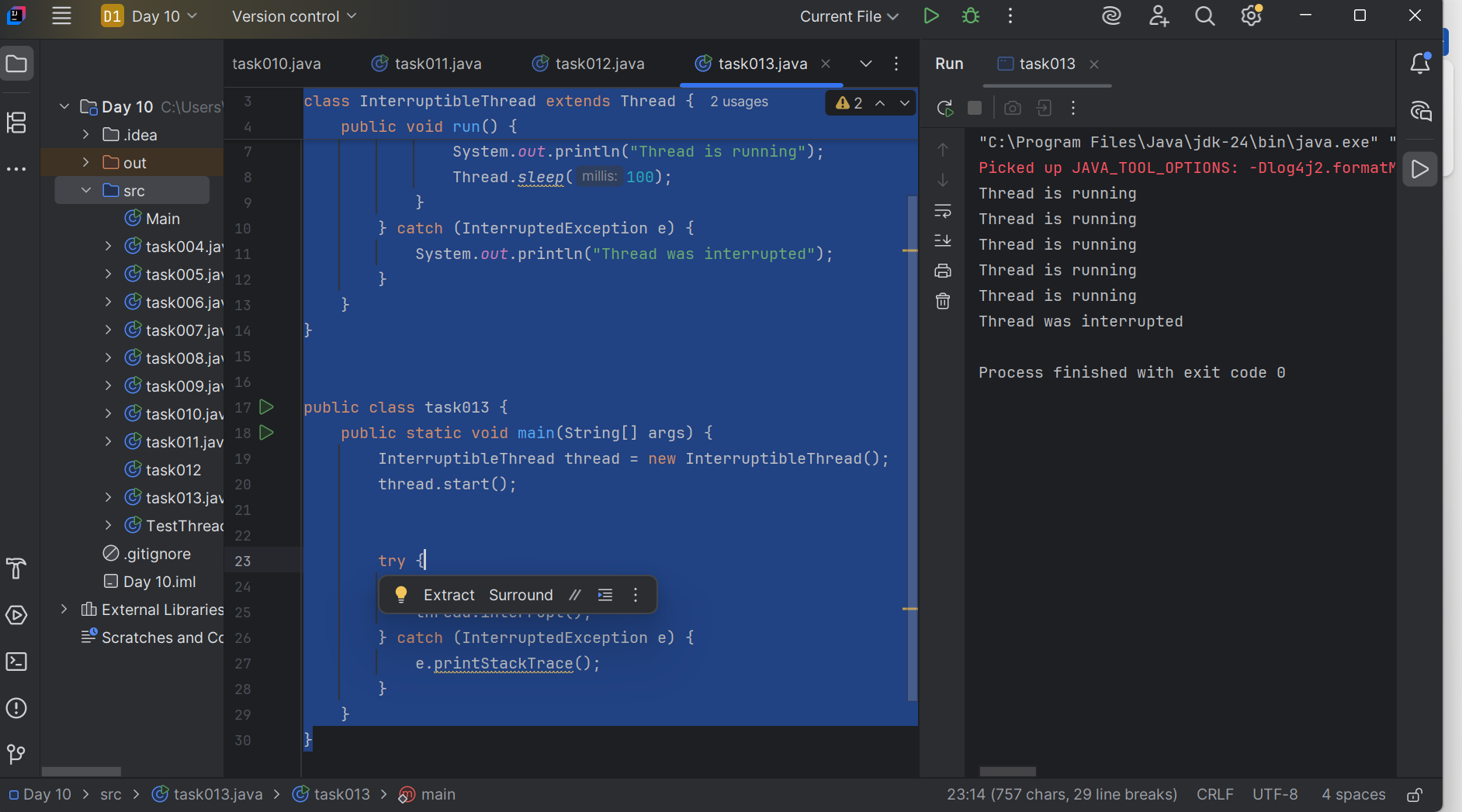
Task 12 :

import java.util.stream.\*;  
class task012 {  
 public static void main(String[] args) {  
 Stream<String> stream  
 = Stream.*of*("Hello", "My",  
 "name", "is",  
 "Sarath",  
 "kumar");  
  
 stream.forEach(System.*out*::print);  
 System.*out*.println("println method");  
 }  
}



Task 13:

//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 task013 {  
 public static void main(String[] args) {  
 InterruptibleThread thread = new InterruptibleThread();  
 thread.start();  
  
  
 try {  
 Thread.*sleep*(500);  
 thread.interrupt();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}



Task 14 :

Task 14: What are Daemon threads? Explain… Daemon threa is a background thread that runs in the background to perform tasks like garbage collection,handling background I/O operations without blocking main program from exiting Example : Auto-Save Feature in a Text Editor

Task 15: