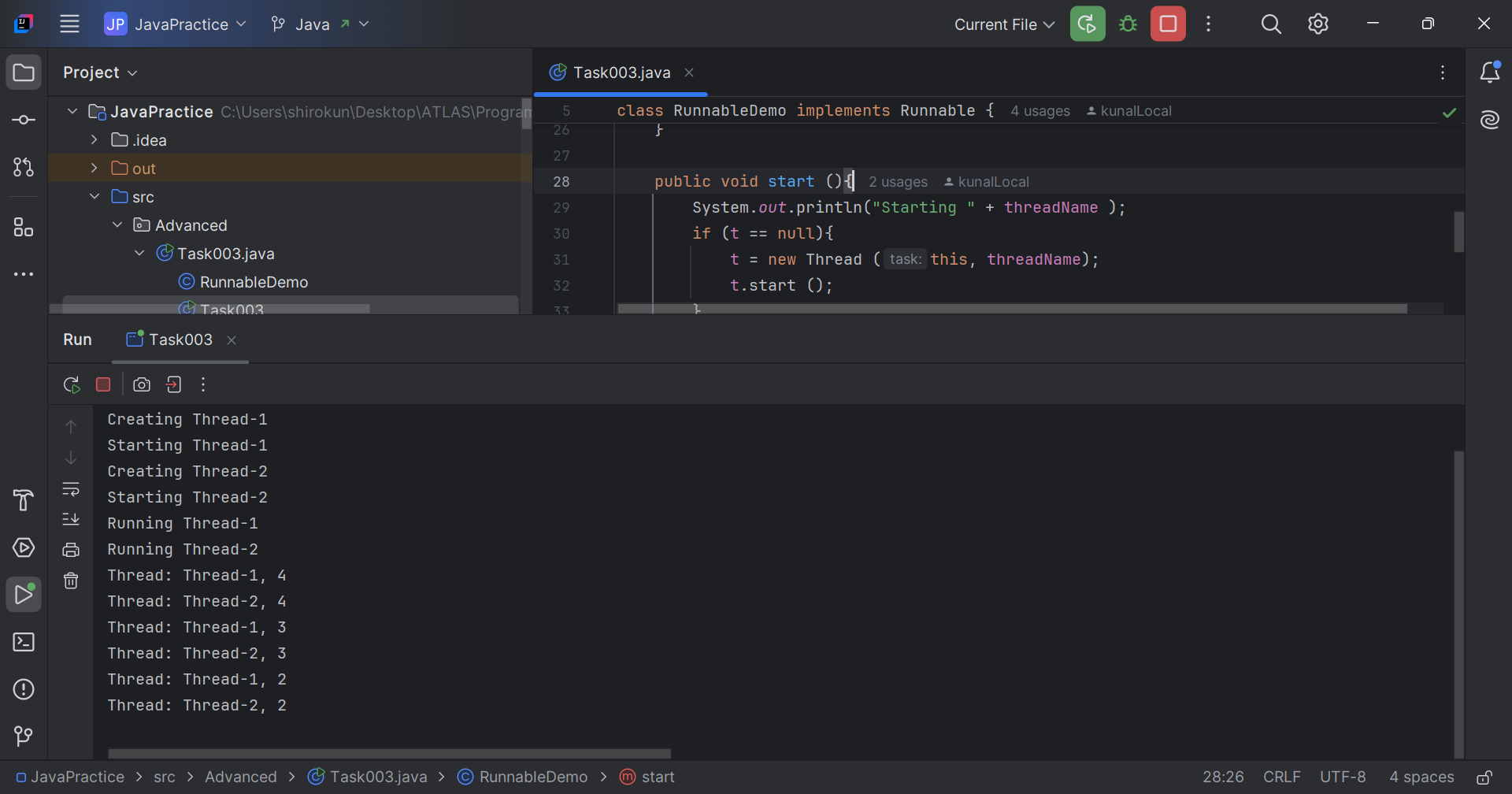
Day 10 – 20/06/2025

Q1. What is a process?  
Ans. Process is an instance of a program running in memory, executing a sequence of instructions.

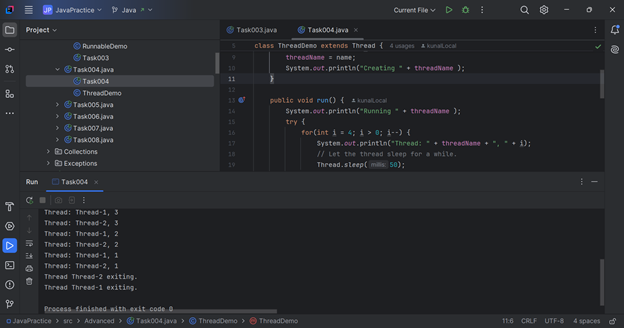
Q2. What is a thread?  
Ans. A thread is a lightweight process that runs concurrently with other thread within the same process. Threads share same memory space and resources as the parent process.

// Task003: Multithreading  
  
package Advanced;  
  
class RunnableDemo implements Runnable {  
 private Thread t;  
 private final String threadName;  
  
 RunnableDemo( String name){ //constructor with 1 parameter  
 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*(5000);  
 }  
 } 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 Task003 {  
 public static void main(String[] args) {  
 RunnableDemo R1 = new RunnableDemo( "Thread-1");  
 R1.start();  
 RunnableDemo R2 = new RunnableDemo( "Thread-2");  
 R2.start();  
 }  
}



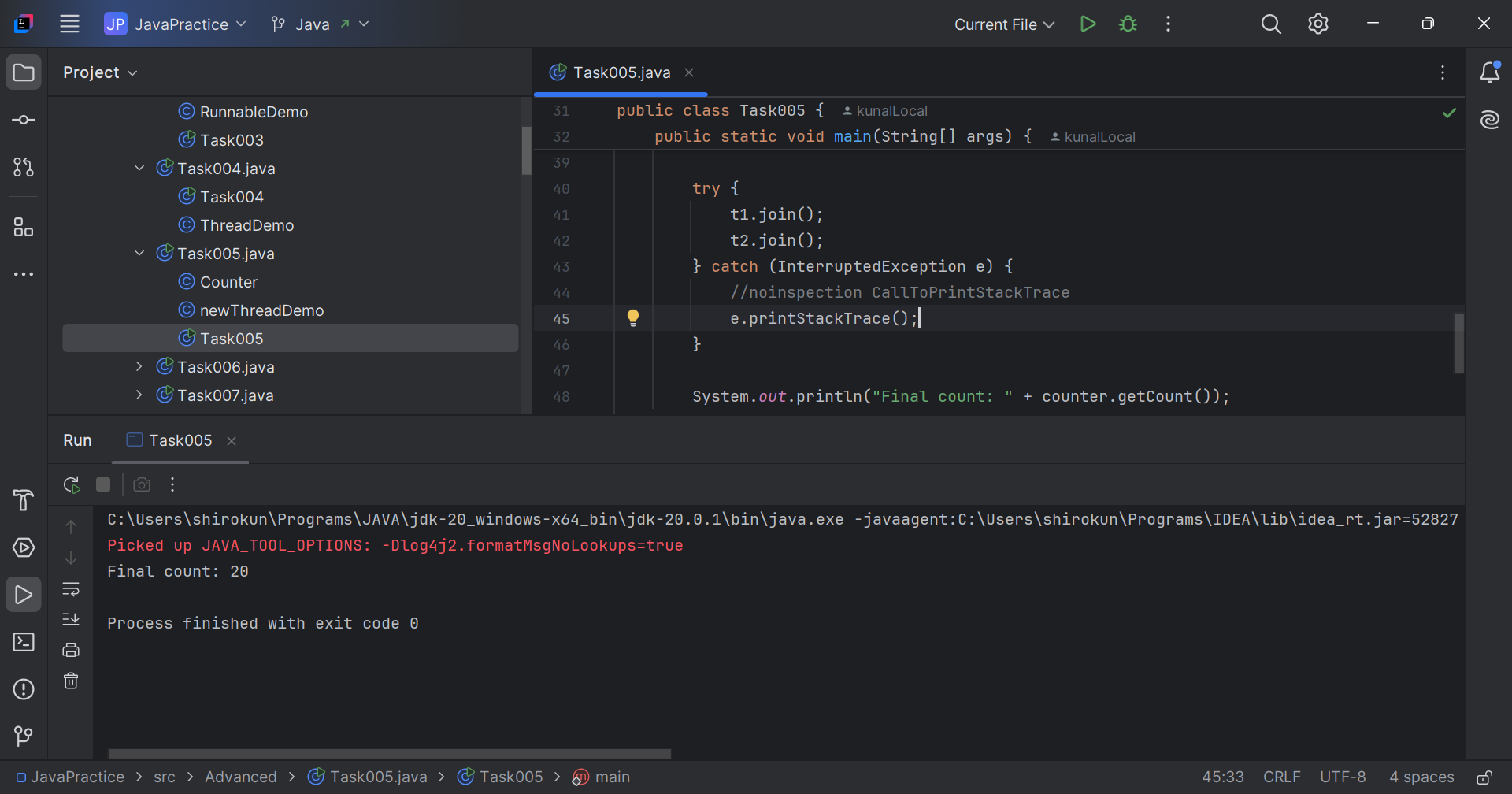
// Task004: Extending the thread class  
package Advanced;  
  
class ThreadDemo extends Thread {  
 private Thread t;  
 private final String threadName;  
 ThreadDemo( String name){  
 threadName = name;  
 System.*out*.println("Creating " + threadName );  
 }

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 Task004 {  
 public static void main(String[] args) {  
 ThreadDemo T1 = new ThreadDemo( "Thread-1");  
 T1.start();  
 ThreadDemo T2 = new ThreadDemo( "Thread-2");  
 T2.start();  
 }  
}

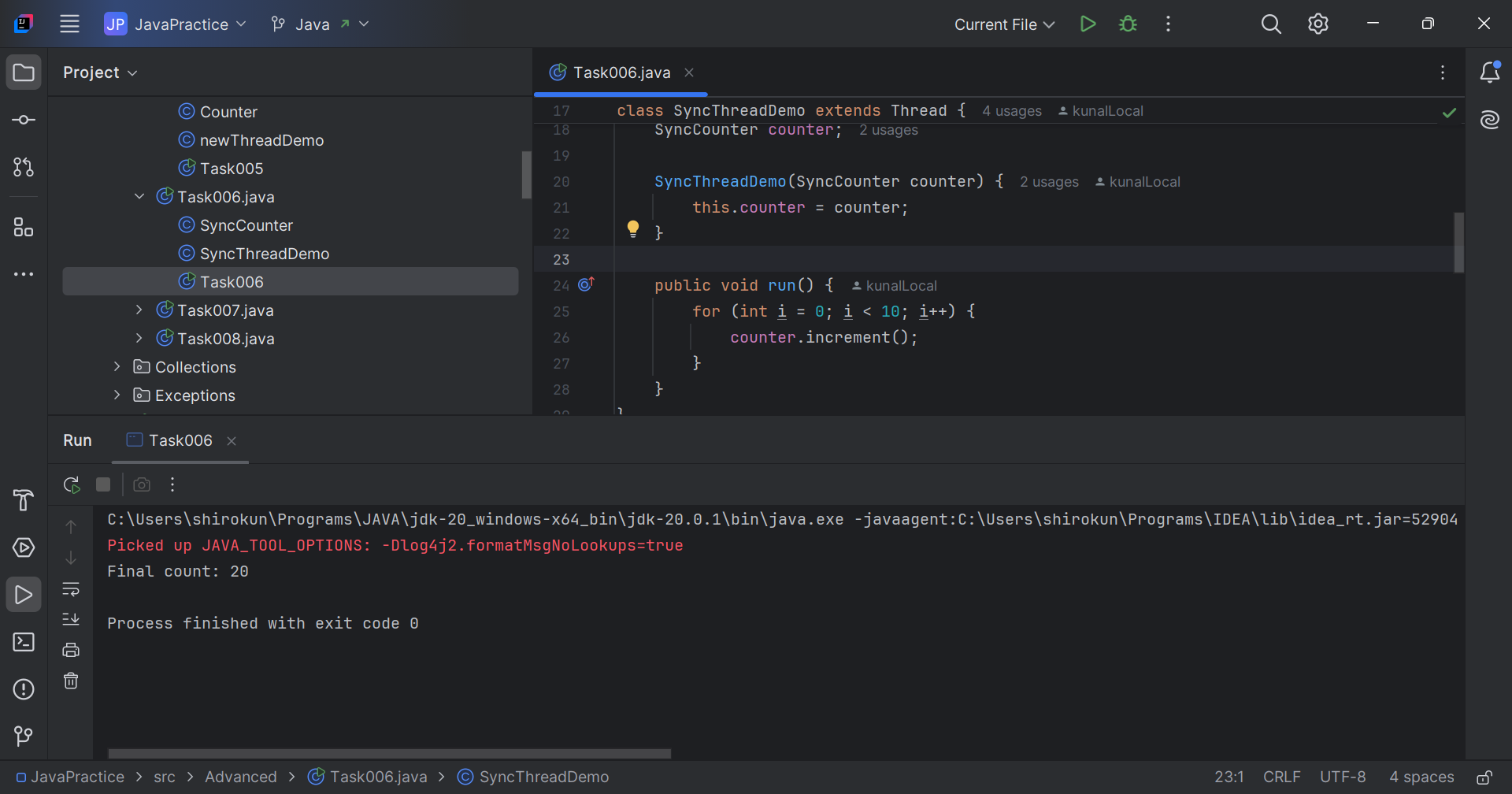


// Task005: Threads  
  
package Advanced;  
  
class Counter {  
 private int count = 0;  
  
 public void increment() {  
 count++;  
 }  
  
 public int getCount() {  
 return count;  
 }  
}

class newThreadDemo extends Thread {  
 Counter counter;  
  
 newThreadDemo(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();  
 newThreadDemo t1 = new newThreadDemo(counter);  
 newThreadDemo t2 = new newThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + counter.getCount());  
 }  
}

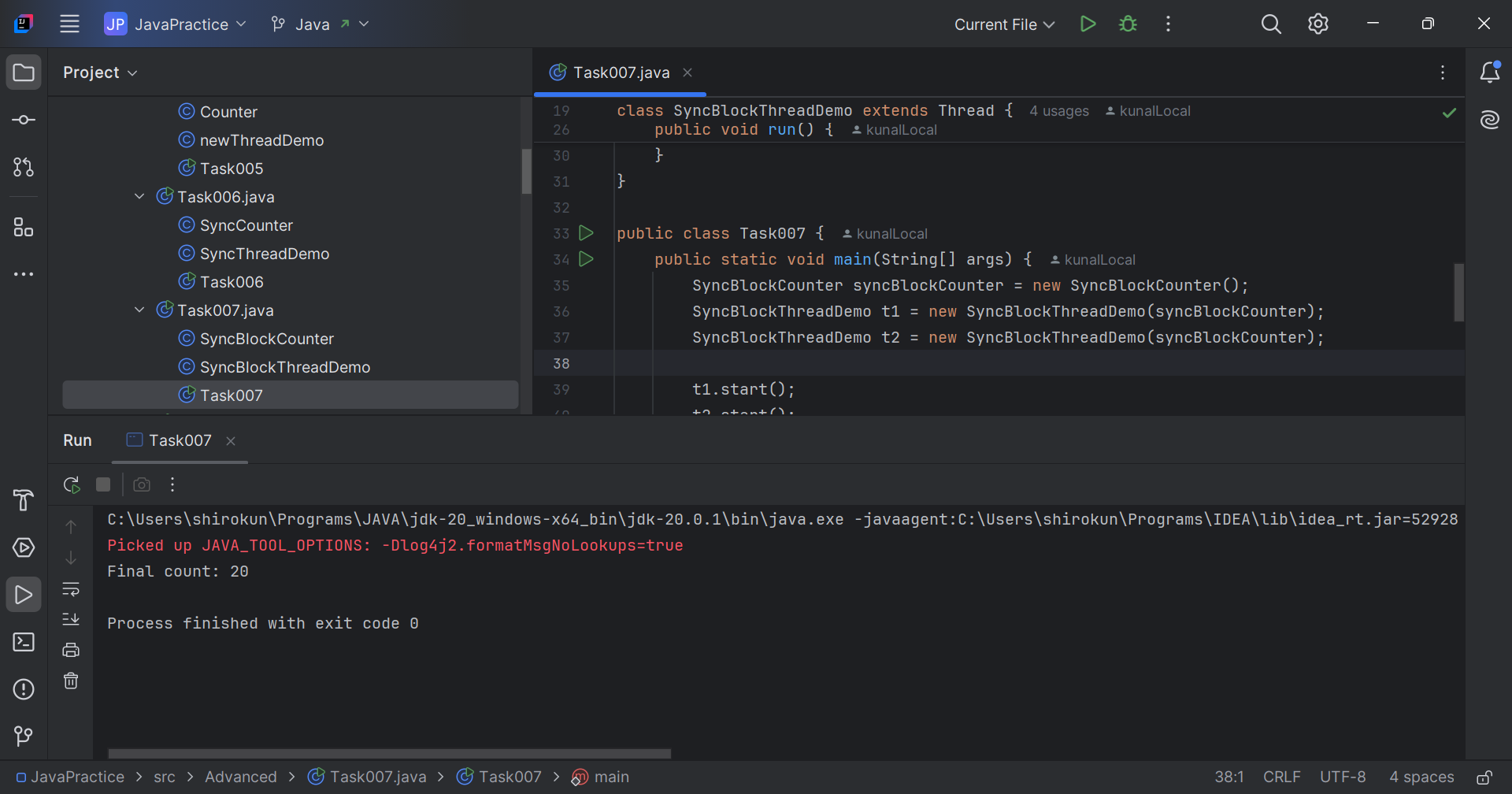


// Task006: Threads Synchronization  
  
package Advanced;  
  
class SyncCounter {  
 private int count = 0;  
  
 public synchronized void increment() {  
 count++;  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
class SyncThreadDemo extends Thread {  
 SyncCounter counter;  
  
 SyncThreadDemo(SyncCounter counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 counter.increment();  
 }  
 }  
}  
  
public class Task006 {  
 public static void main(String[] args) {  
 SyncCounter counter = new SyncCounter();  
 SyncThreadDemo t1 = new SyncThreadDemo(counter);  
 SyncThreadDemo t2 = new SyncThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + counter.getCount());  
 }  
}

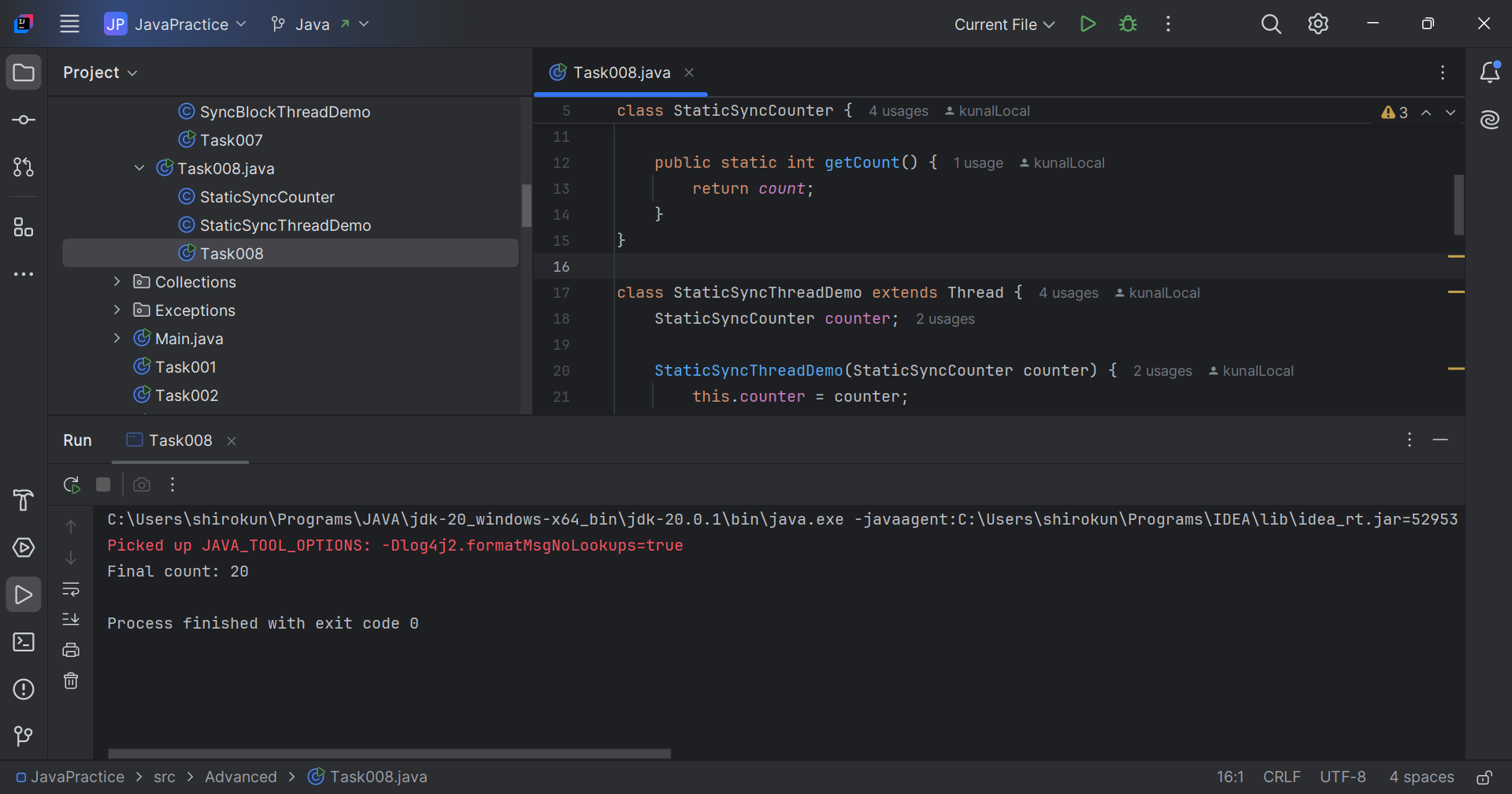


// Task007: Threads Synchronization block  
  
package Advanced;  
  
class SyncBlockCounter {  
 private int count = 0;  
  
 public void increment() {  
 synchronized (this) {  
 count++;  
 }  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
class SyncBlockThreadDemo extends Thread {  
 SyncBlockCounter counter;  
  
 SyncBlockThreadDemo(SyncBlockCounter 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) {  
 SyncBlockCounter syncBlockCounter = new SyncBlockCounter();  
 SyncBlockThreadDemo t1 = new SyncBlockThreadDemo(syncBlockCounter);  
 SyncBlockThreadDemo t2 = new SyncBlockThreadDemo(syncBlockCounter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + syncBlockCounter.getCount());  
 }  
}



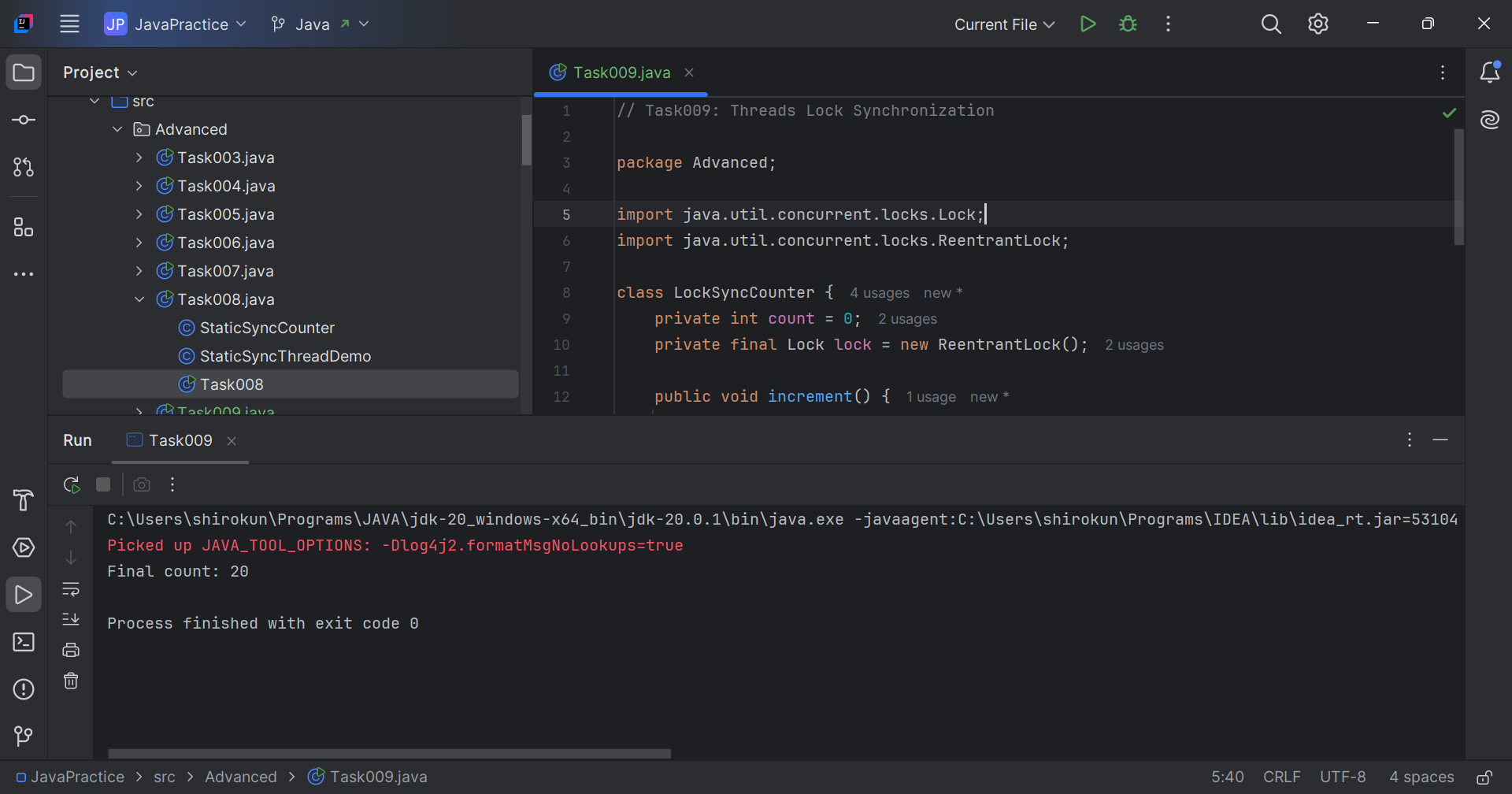
// Task008: Threads Static Synchronization  
  
package Advanced;  
  
class StaticSyncCounter {  
 private static int *count* = 0;  
  
 public static synchronized void increment() {  
 *count*++;  
 }  
  
 public static int getCount() {  
 return *count*;  
 }  
}  
  
class StaticSyncThreadDemo extends Thread {  
 StaticSyncCounter counter;  
  
 StaticSyncThreadDemo(StaticSyncCounter counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 counter.*increment*();  
 }  
 }  
}  
  
public class Task008 {  
 public static void main(String[] args) {  
 StaticSyncCounter staticSyncCounter = new StaticSyncCounter();  
 StaticSyncThreadDemo t1 = new StaticSyncThreadDemo(staticSyncCounter);  
 StaticSyncThreadDemo t2 = new StaticSyncThreadDemo(staticSyncCounter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + staticSyncCounter.*getCount*());  
 }  
}



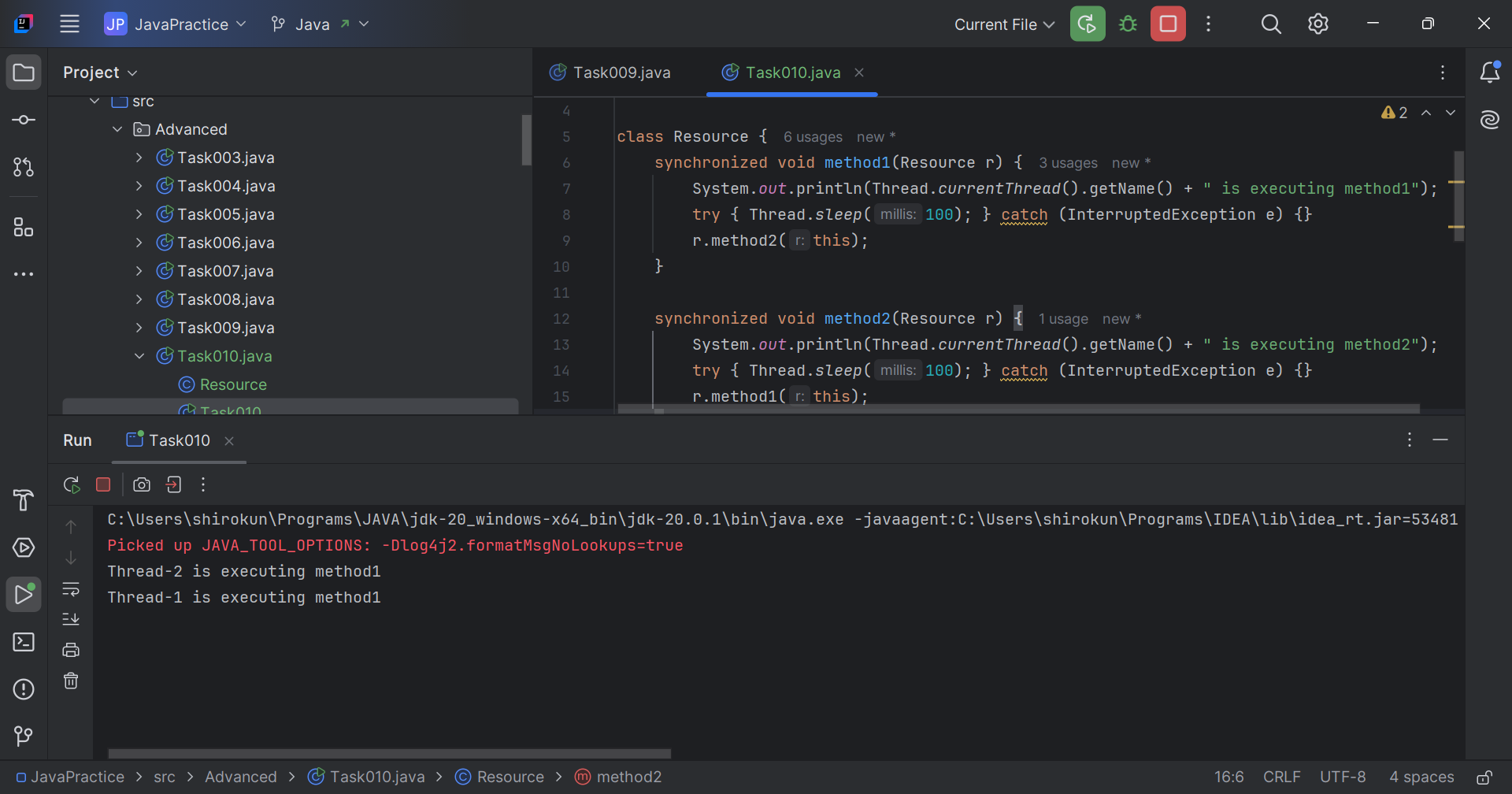
// Task009: Threads Lock Synchronization  
  
package Advanced;  
  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
class LockSyncCounter {  
 private int count = 0;  
 private final Lock lock = new ReentrantLock();  
  
 public void increment() {  
 lock.lock();  
 try {  
 count++;  
 } finally {  
 lock.unlock();  
 }  
 }

public int getCount() {  
 return count;  
 }  
  
}

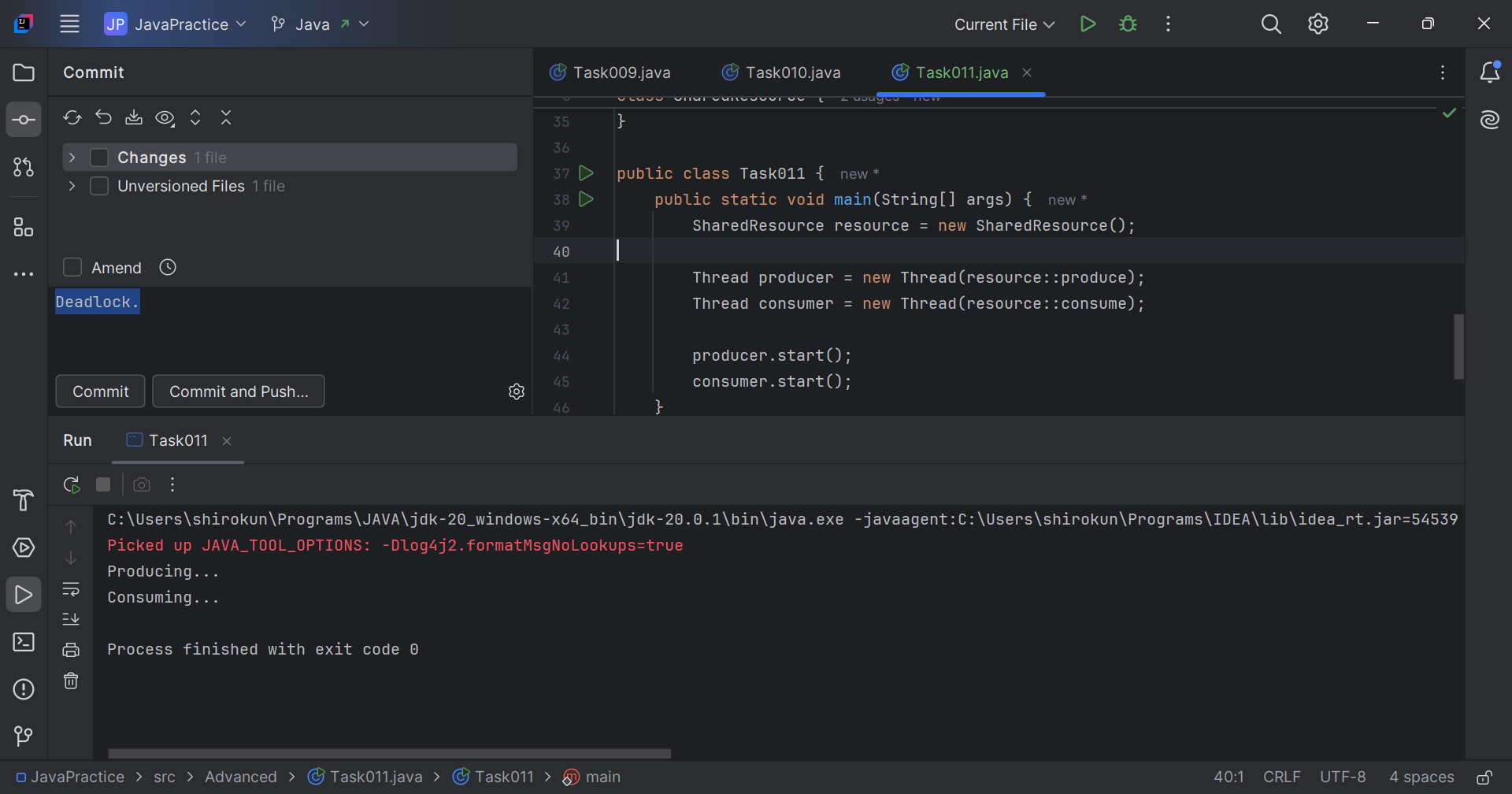
class LockSyncThreadDemo extends Thread {  
 LockSyncCounter counter;  
  
 LockSyncThreadDemo(LockSyncCounter counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 counter.increment();  
 }  
 }  
}  
  
public class Task009 {  
 public static void main(String[] args) {  
 LockSyncCounter lockSyncCounter = new LockSyncCounter();  
 LockSyncThreadDemo t1 = new LockSyncThreadDemo(lockSyncCounter);  
 LockSyncThreadDemo t2 = new LockSyncThreadDemo(lockSyncCounter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final count: " + lockSyncCounter.getCount());  
 }  
}



// Task010: Deadlocks  
  
package Advanced;  
  
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();  
 }  
}

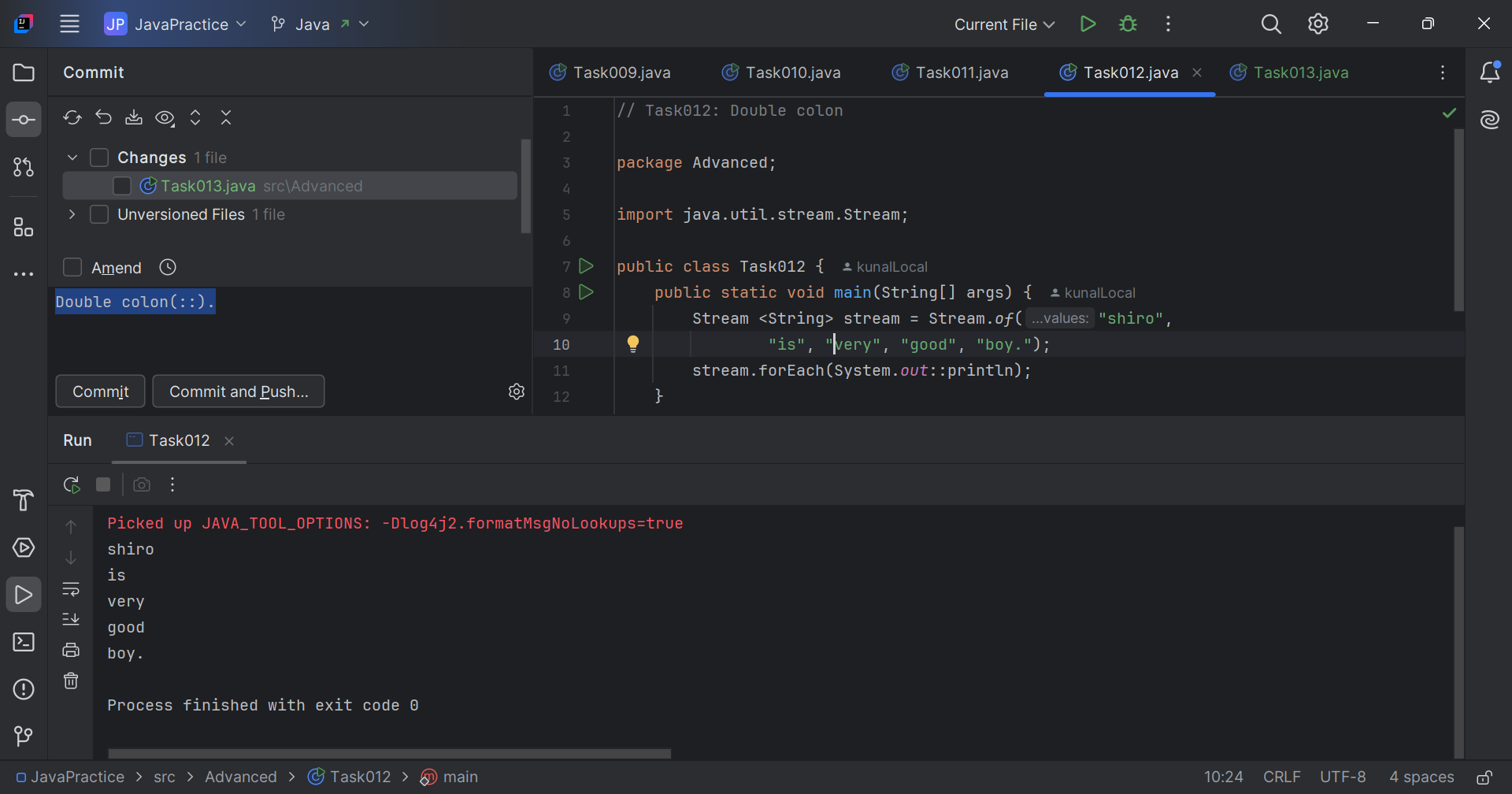


// Task011: Inter Thread Communication  
  
package Advanced;  
  
class SharedResource {  
 private boolean ready = false;  
  
 synchronized void produce() {  
 try {  
 while (ready) {  
 wait();  
 }  
 System.*out*.println("Producing...");  
 ready = true;  
 notify();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
 }  
  
 synchronized void consume() {  
 try {  
 while (!ready) {  
 wait();  
 }  
 System.*out*.println("Consuming...");  
 ready = false;  
 notify();  
 } catch (InterruptedException e) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
 }  
}

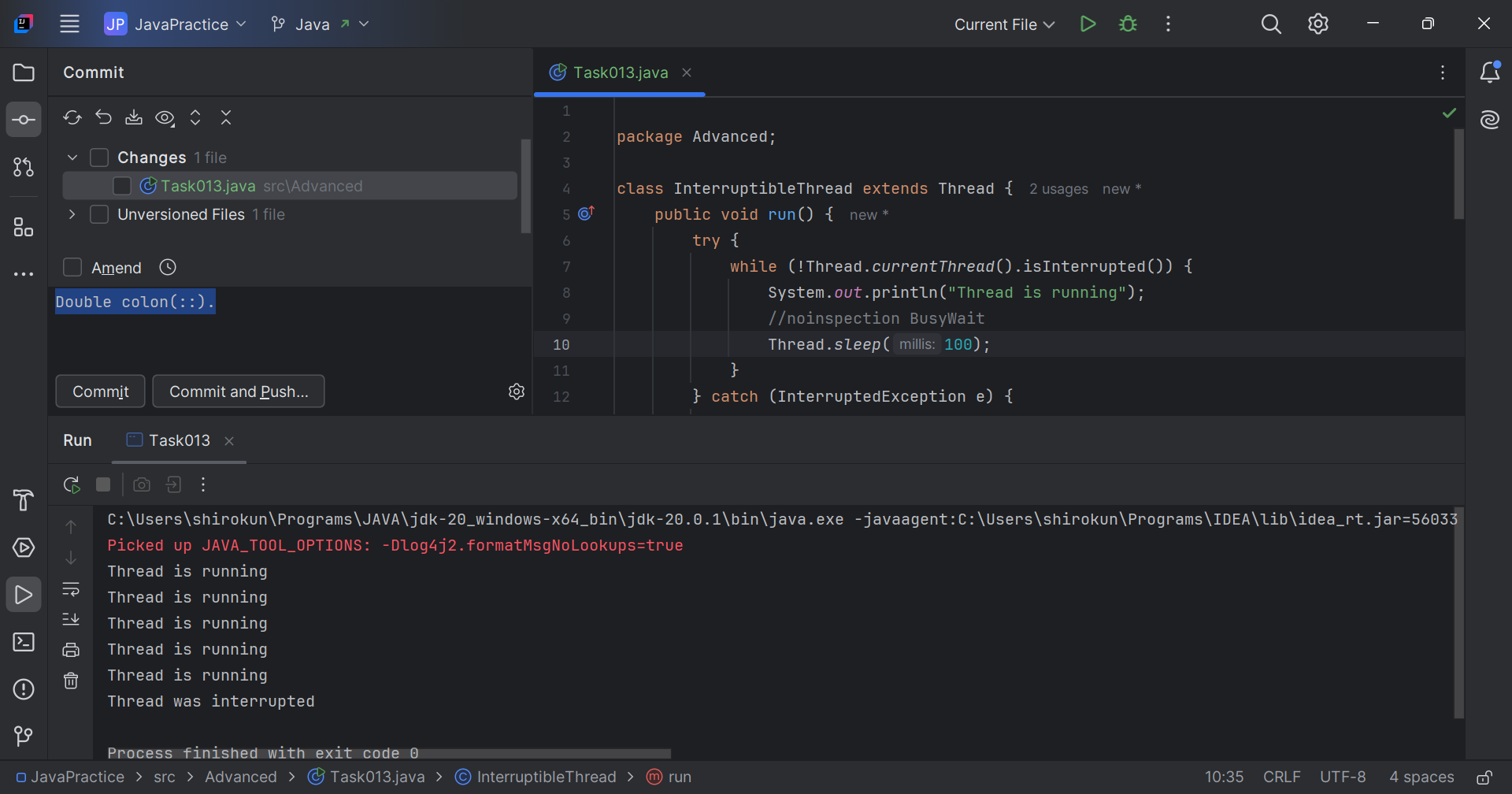


public class Task011 {  
 public static void main(String[] args) {  
 SharedResource resource = new SharedResource();  
  
 Thread producer = new Thread(resource::produce);  
 Thread consumer = new Thread(resource::consume);  
  
 producer.start();  
 consumer.start();  
 }  
}

// Task012: Double colon  
  
package Advanced;  
  
import java.util.stream.Stream;  
  
public class Task012 {  
 public static void main(String[] args) {  
 Stream <String> stream = Stream.*of*("shiro",  
 "is", "very", "good", "boy.");  
 stream.forEach(System.*out*::println);  
 }  
}



// Task013: Interruptions.  
  
package Advanced;  
  
class InterruptibleThread extends Thread {  
 public void run() {  
 try {  
 while (!Thread.*currentThread*().isInterrupted()) {  
 System.*out*.println("Thread is running");  
 //noinspection BusyWait  
 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) {  
 //noinspection CallToPrintStackTrace  
 e.printStackTrace();  
 }  
 }  
}



Q14. What are Daemon threads? Explain.  
Ans. Daemon threads are threads that run in the background, performing tasks that are not critical to the main program’s execution. Daemon threads typically have a low priority. Garbage collection is an example of daemon thread.

Q15. What are the debugging tools in Java? List down a few.  
Ans. Some of the popular Java debugging tools:

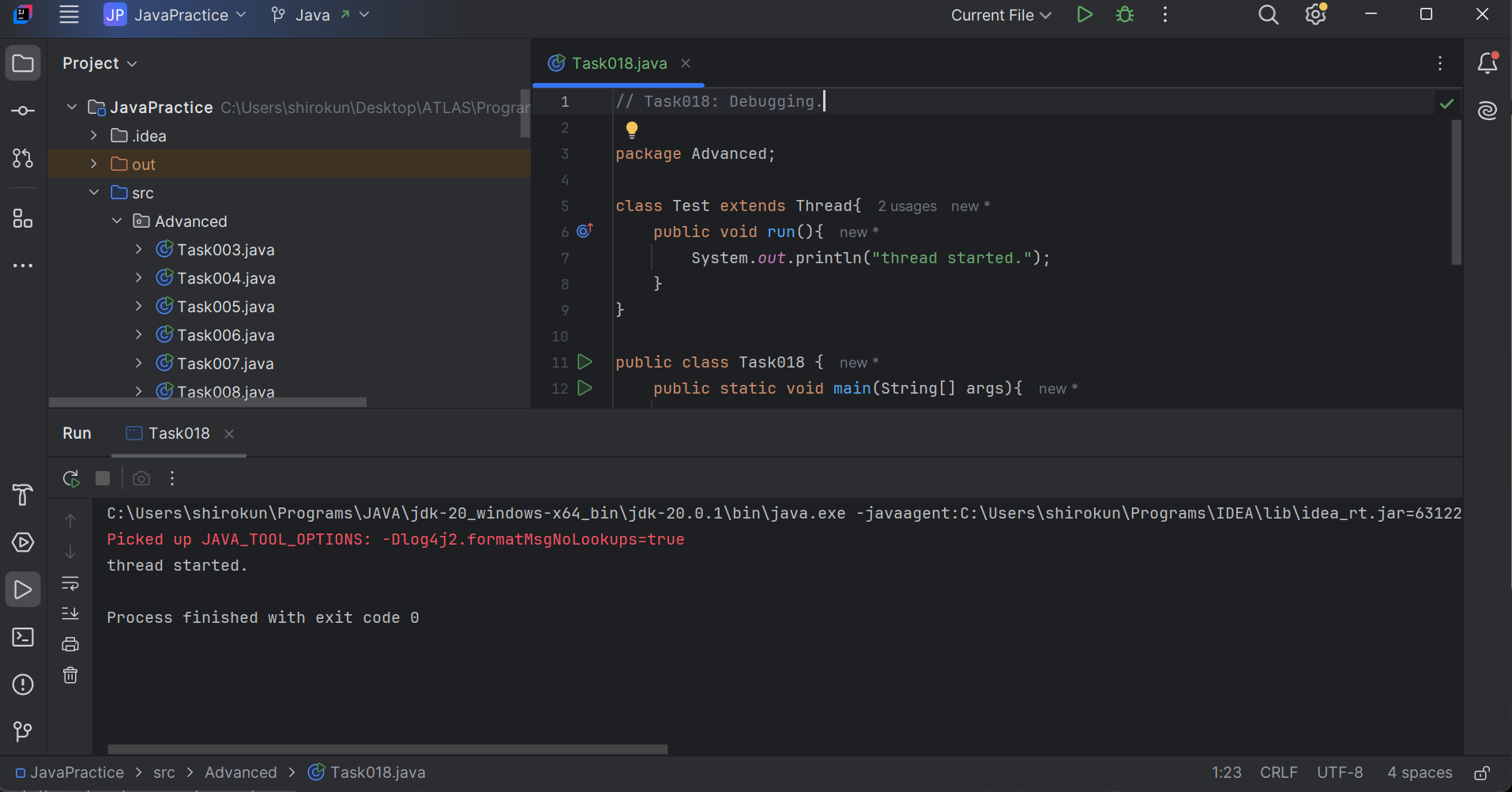
1. Eclipse Debugger: Built-in debugger in Eclipse IDE.  
2. IntelliJ IDEA Debugger: Built-in debugger in IntelliJ IDEA.  
3. VisualVM: A visual tool for monitoring and debugging Java applications.  
4. jdb (Java Debugger): A command-line debugger for Java.

Q16. Try to understand the error messages.  
Ans: Compile time error messages – grammatical mistakes, missing characters like – , ; . etc.

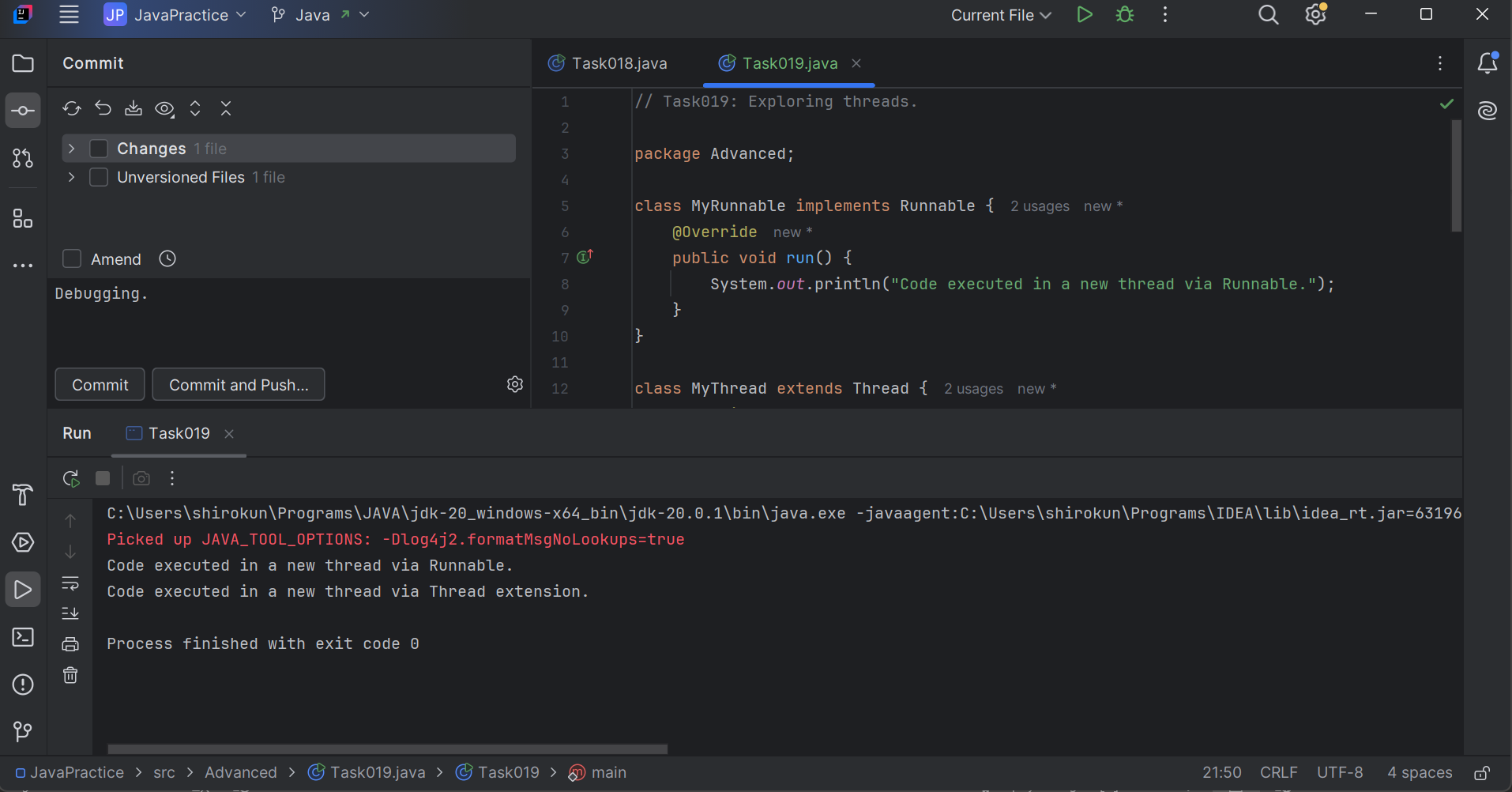
* Runtime error or exceptions – an error that occurs during the execution of a program, while an exception can be caught and handled by the program.
* Stack overflow error – occurs when a program attempts to use more memory than is available on the call stack, often due to infinite recursion.
* Array index of out bounds – occurs when trying to access an element with an index that is outside the bounds of the array usually less than 0 or greater than or equal to the array's length.
* IO exception – occurs when an input/output operation fails, such as reading or writing to a file, network connection, or other I/O device.
* Null pointer exception – occurs when you try to access or manipulate a null object reference, such as calling a method or accessing a field on a null object.

Q17. What is stack trace?  
Ans. Stack trace is a report of the error or exception that occurred in a program, showing the sequence of method calls that led to the error.

// Task018: Debugging.  
  
package Advanced;  
  
class Test extends Thread{  
 public void run(){  
 System.*out*.println("thread started.");  
 }  
}  
  
public class Task018 {  
 public static void main(String[] args){  
  
 Test t1 = new Test();  
 t1.start();  
 }  
}



// Task019: Exploring threads.  
  
package Advanced;  
  
class MyRunnable implements Runnable {  
 @Override  
 public void run() {  
 System.*out*.println("Code executed in a new thread via Runnable.");  
 }  
}  
  
class MyThread extends Thread {  
 @Override  
 public void run() {  
 System.*out*.println("Code executed in a new thread via Thread extension.");  
 }  
}  
class Task019 {  
 public static void main(String[] args) {  
 MyRunnable runnableInstance = new MyRunnable();  
 MyThread threadInstance = new MyThread();  
 Thread t1 = new Thread(runnableInstance);  
 t1.start(); // For MyRunnable  
 threadInstance.start(); // For MyThread  
 }  
}



// Task020: Thread Stack Trace.  
  
package Advanced;  
  
public class Task020 {  
 public static void main(String[] args) {  
 *method1*();  
 }  
  
 public static void method1() {  
 *method2*();  
 }  
  
 public static void method2() {  
 *method3*();  
 }  
  
 public static void method3() {  
 StackTraceElement[] stackTrace = Thread.*currentThread*().getStackTrace();  
 System.*out*.println("Thread Stack Trace:");  
 // Iterate through the StackTraceElement array and print details  
 for (StackTraceElement element : stackTrace) {  
 System.*out*.println(" Class: " + element.getClassName() +  
 ", Method: " + element.getMethodName() +  
 ", Line: " + element.getLineNumber());  
 }  
 }  
}

