**day10\_107856406\_dsdipt\_sudipto\_18june2025**

**Employee Code:** 107856406

**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 18 June 2025 (Day 10)

### ***Task 1: What is a thread and a Process?***

A **process** is an independent program in execution. It has its **own memory space**, system resources, and at least one thread (usually the main thread).

**Example:** When you open **Google Chrome**, the OS creates a **new process** for it.

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

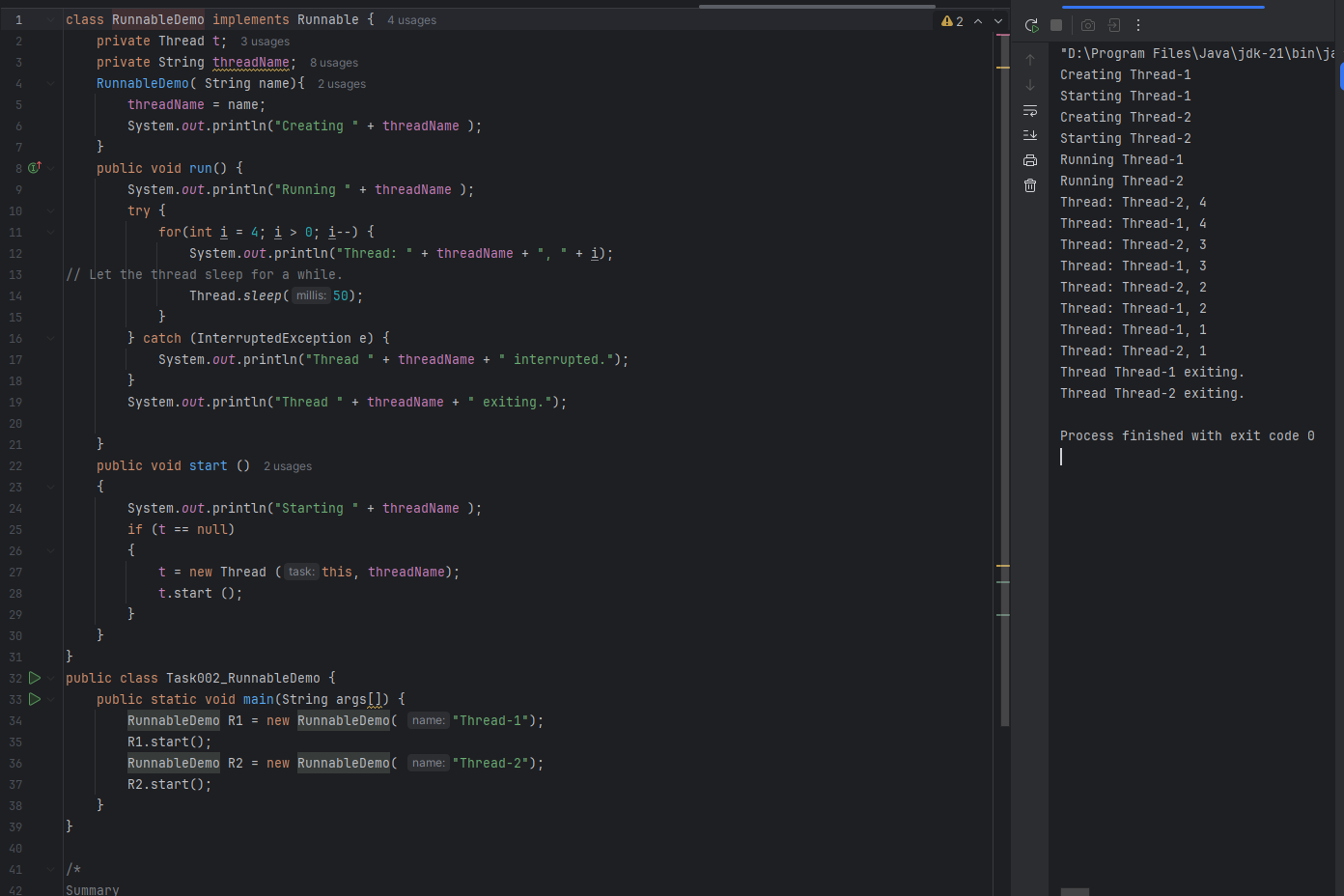
All threads in the same process share the same memory and resources.

**Example:** Inside Chrome, one thread might handle the webpage, another thread might handle a video you're watching.

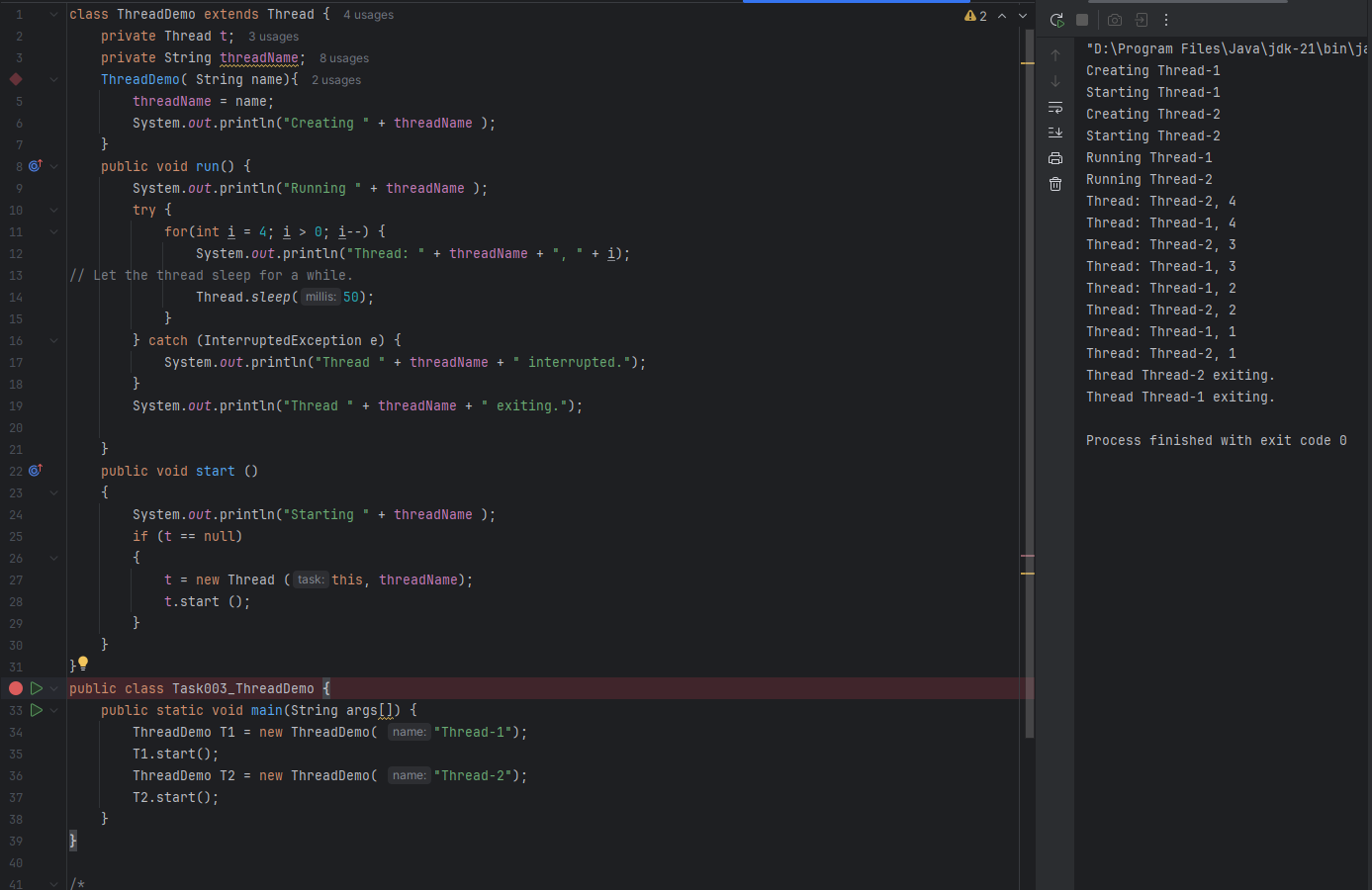
🔄 **Analogy:**

* A **process** is like a **house** .
* **Threads** are the **people inside the house** .
  + They share rooms (memory) but can do different tasks at the same time.

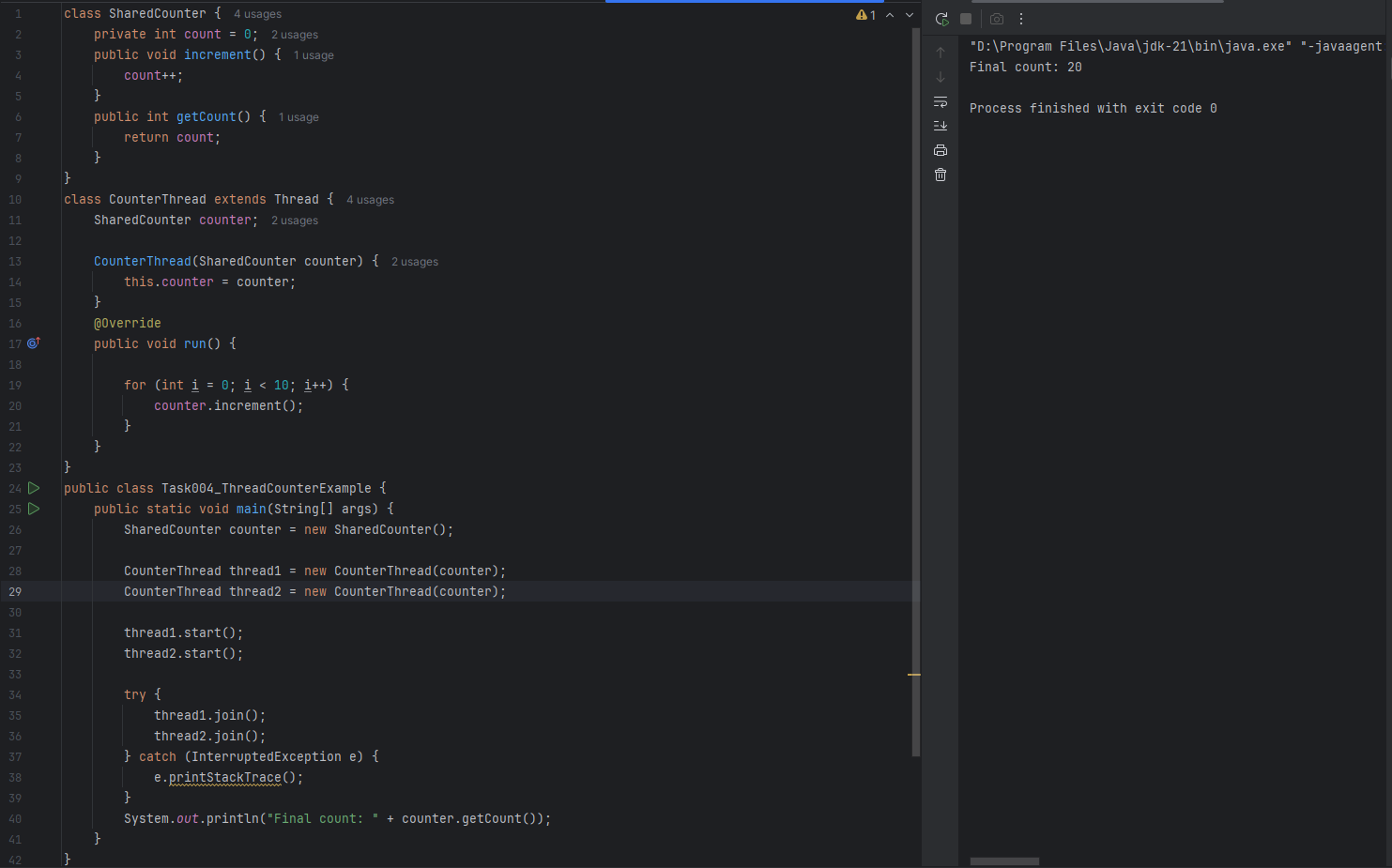
### ***Task 2: Implement and Observe Java Runnable Thread Execution***



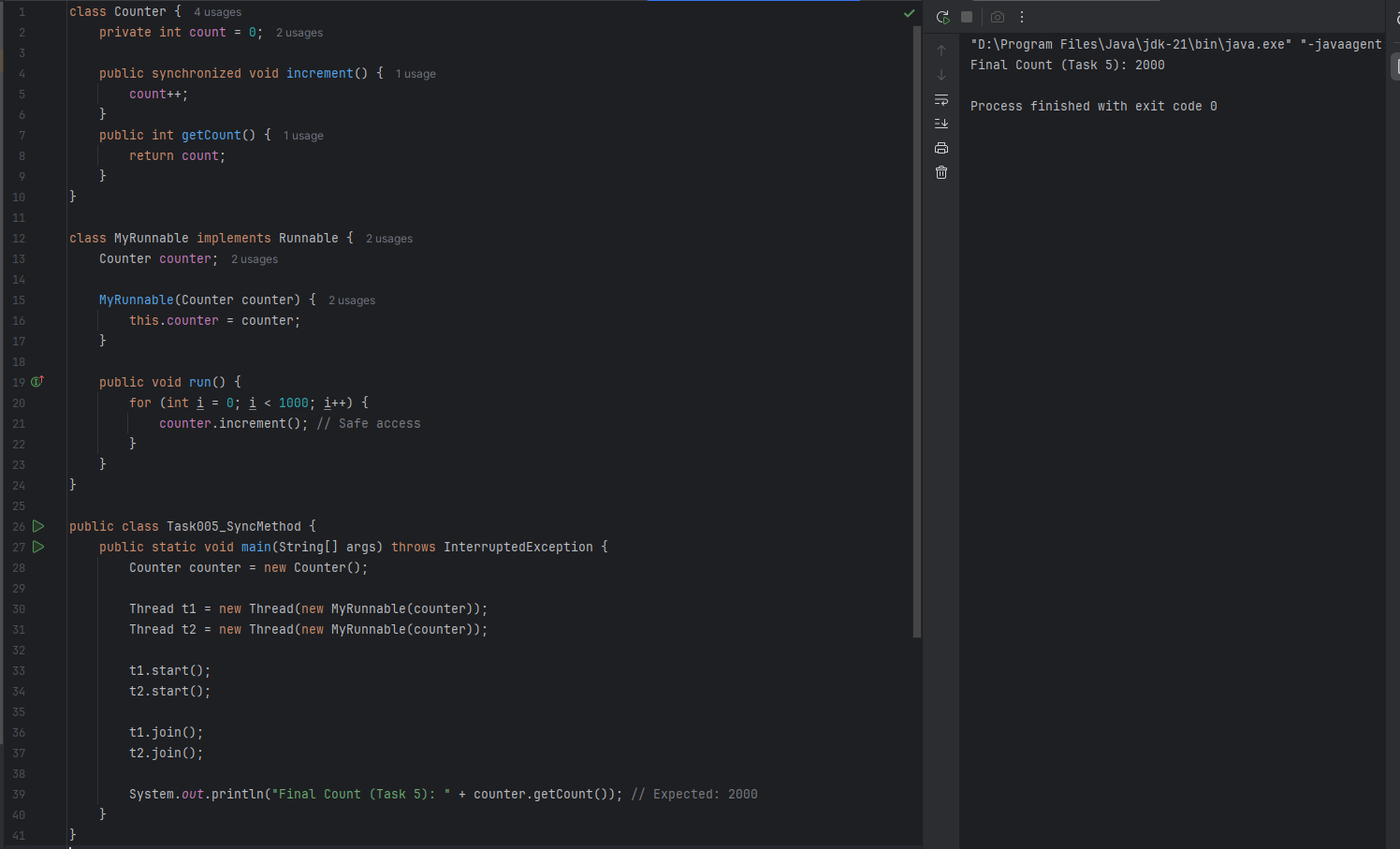
### ***Task 3: Convert Runnable Example to Extend Thread***



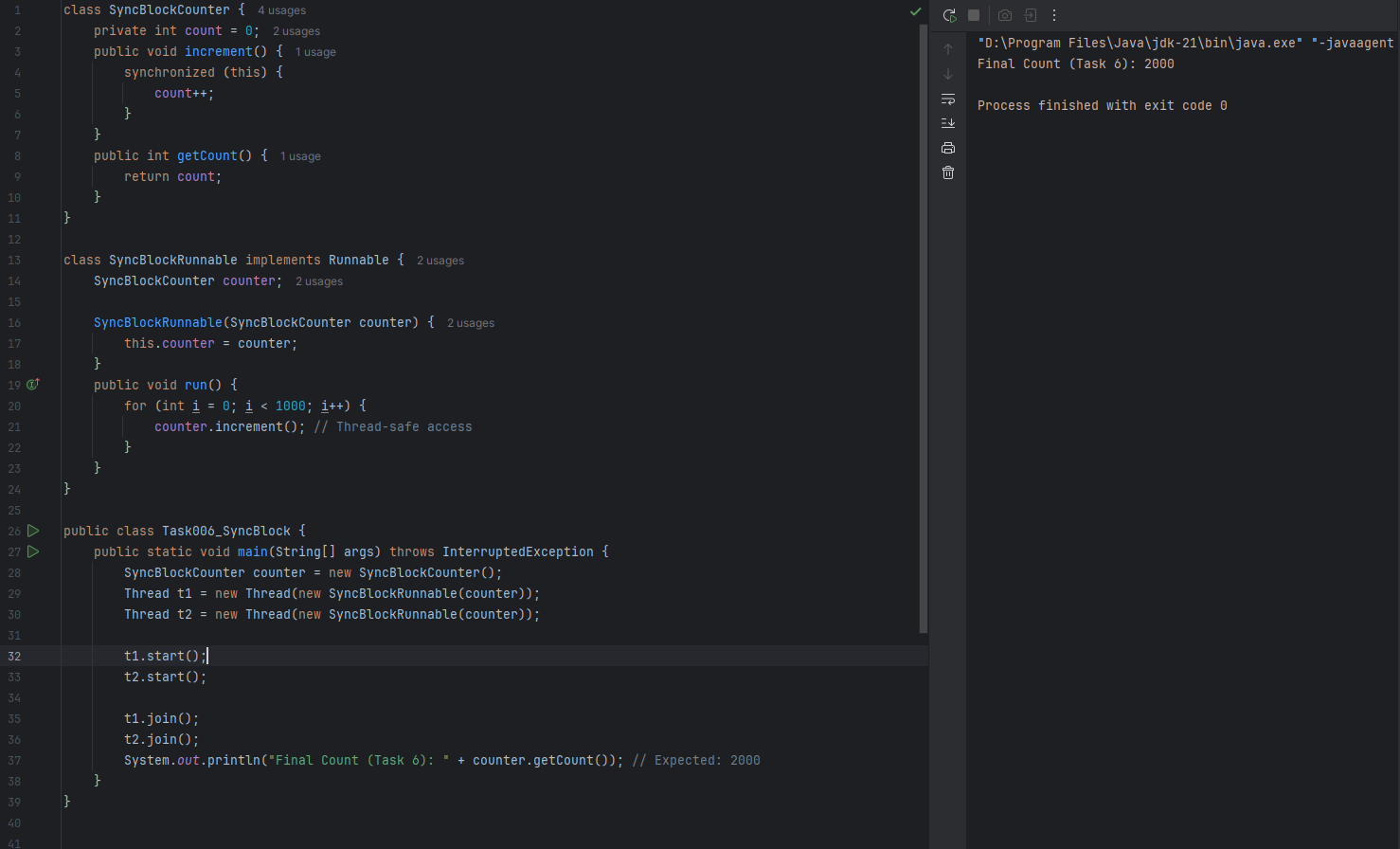
### ***Task 4: Thread interference using a shared counter without synchronization***



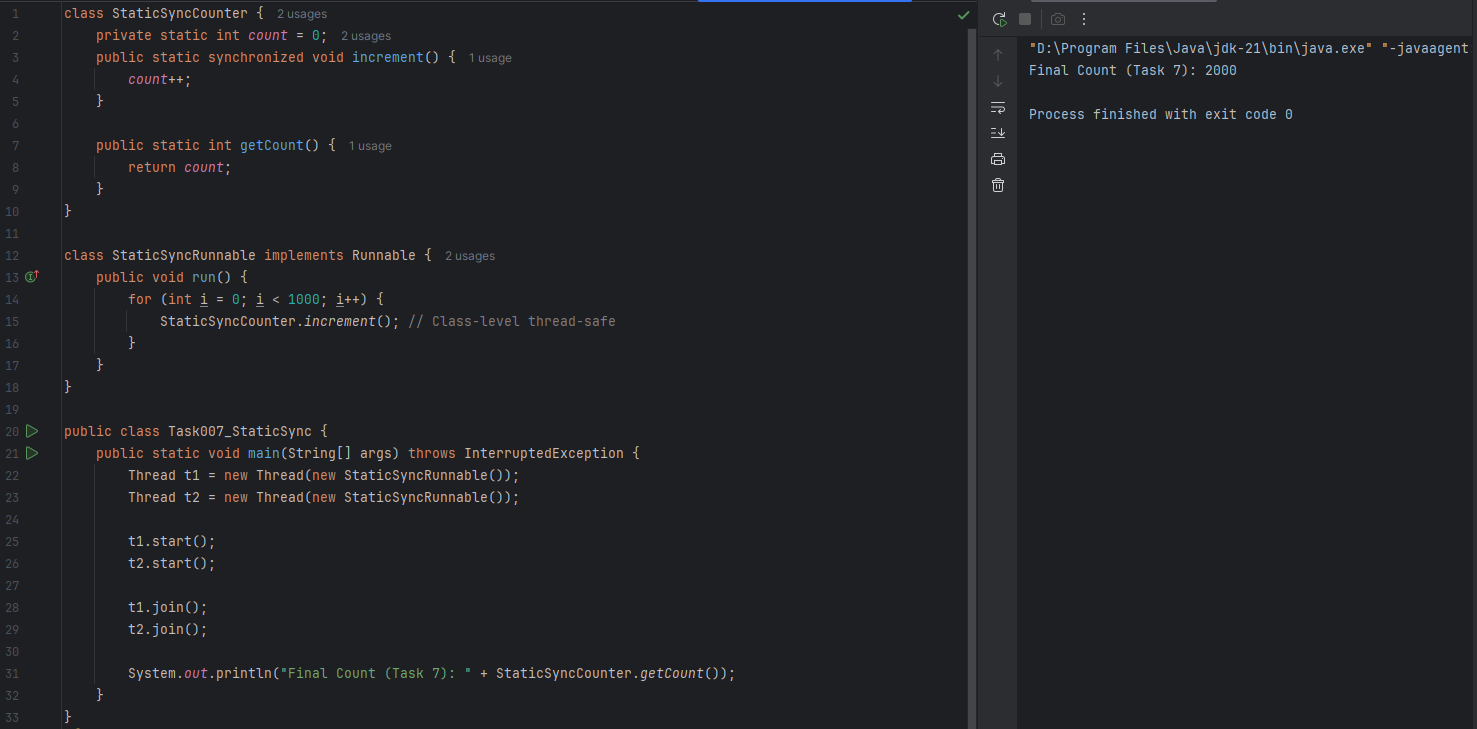
### ***Task 5: Thread-safe increment using a synchronized method in a shared counter***



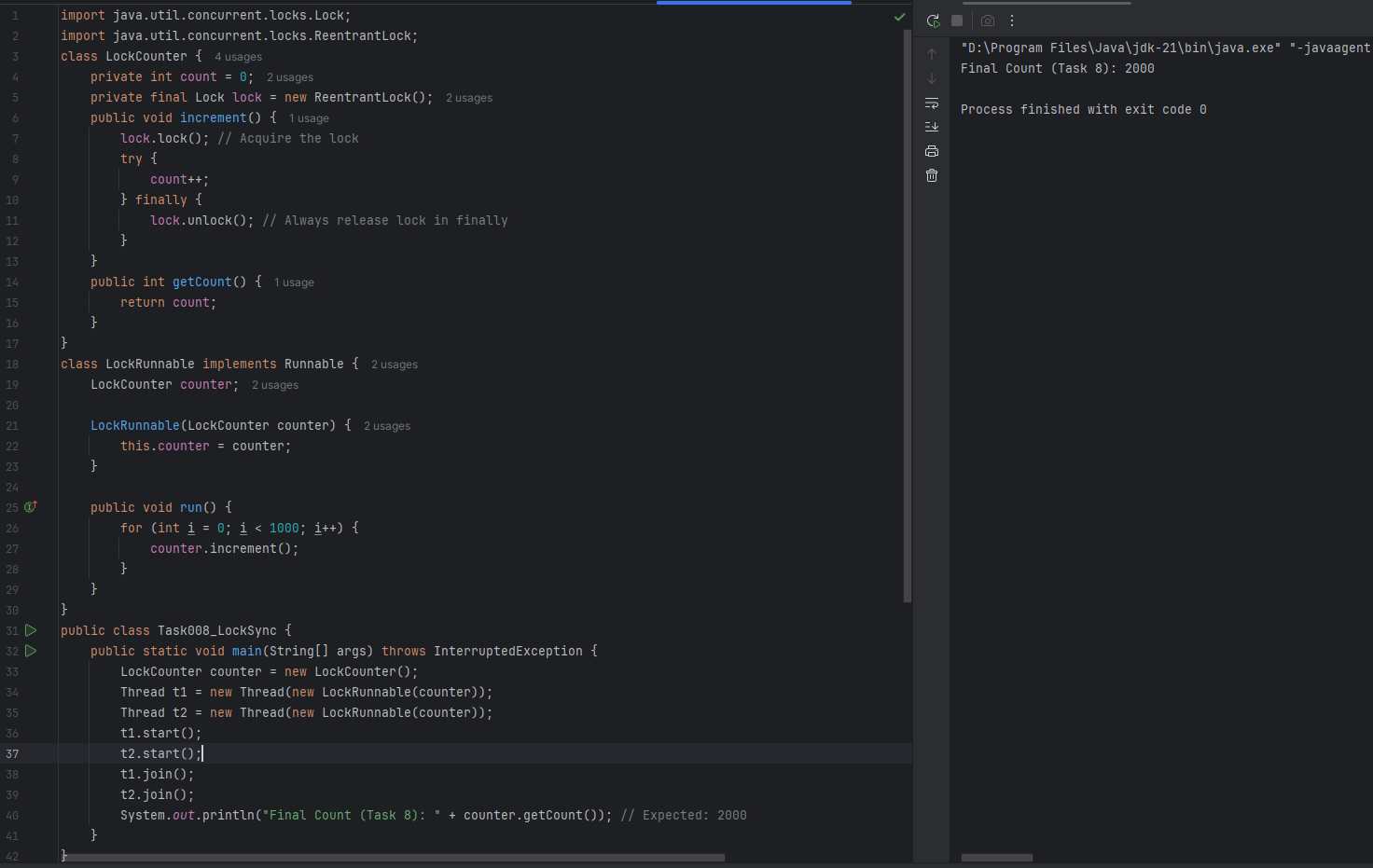
### ***Task 6: Demonstrate thread synchronization using a synchronized block***



### ***Task 7: Demonstrate thread-safe static counter using class-level synchronization***



### ***Task 8: Thread-safe counter using ReentrantLock for explicit synchronization***



### ***Task 9: What is synchronization***

**Synchronization** in Java is a technique to control **access of multiple threads to shared resources** (like variables, methods, or objects) to **prevent data inconsistency and race conditions**.

**Why is it needed?**

When **two or more threads** try to access a **shared resource (e.g., a counter variable)** simultaneously, they can interfere with each other, causing **unexpected results**.

**Example (Without Synchronization):**

class Counter {

int count = 0;

public void increment() {

count++;

}

}

Two threads calling increment() at the same time can cause incorrect count values due to a **race condition**.

**Without Synchronization:**

* Data inconsistency
* Race conditions
* Bugs that are hard to reproduce and fix

**Example with Synchronized Method**

class Counter {

int count = 0;

public synchronized void increment() {

count++;

}

}

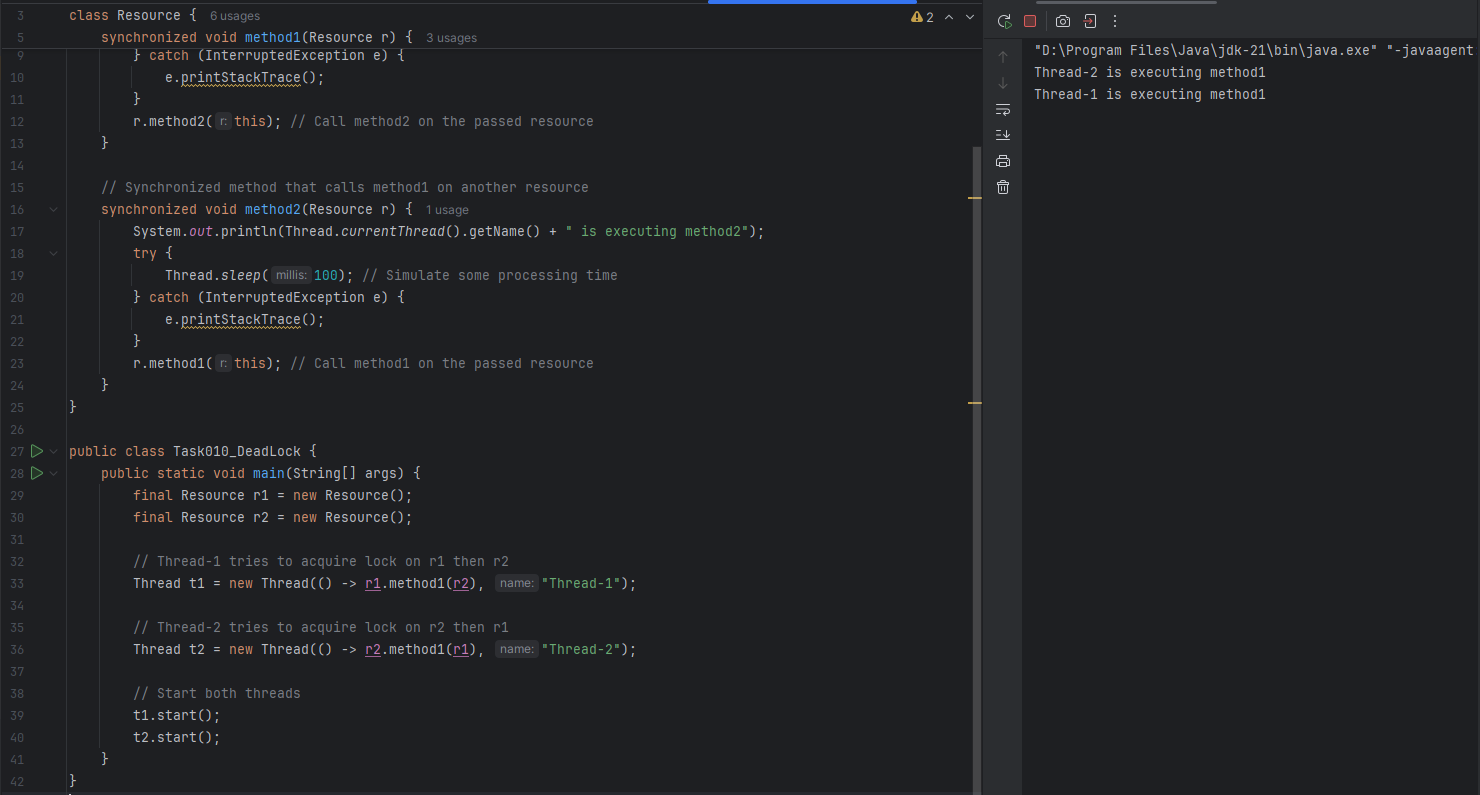
* synchronized ensures that only one thread at a time can call increment().

**Synchronization ensures only one thread accesses the critical section at a time**, preventing problems in multithreaded environments.

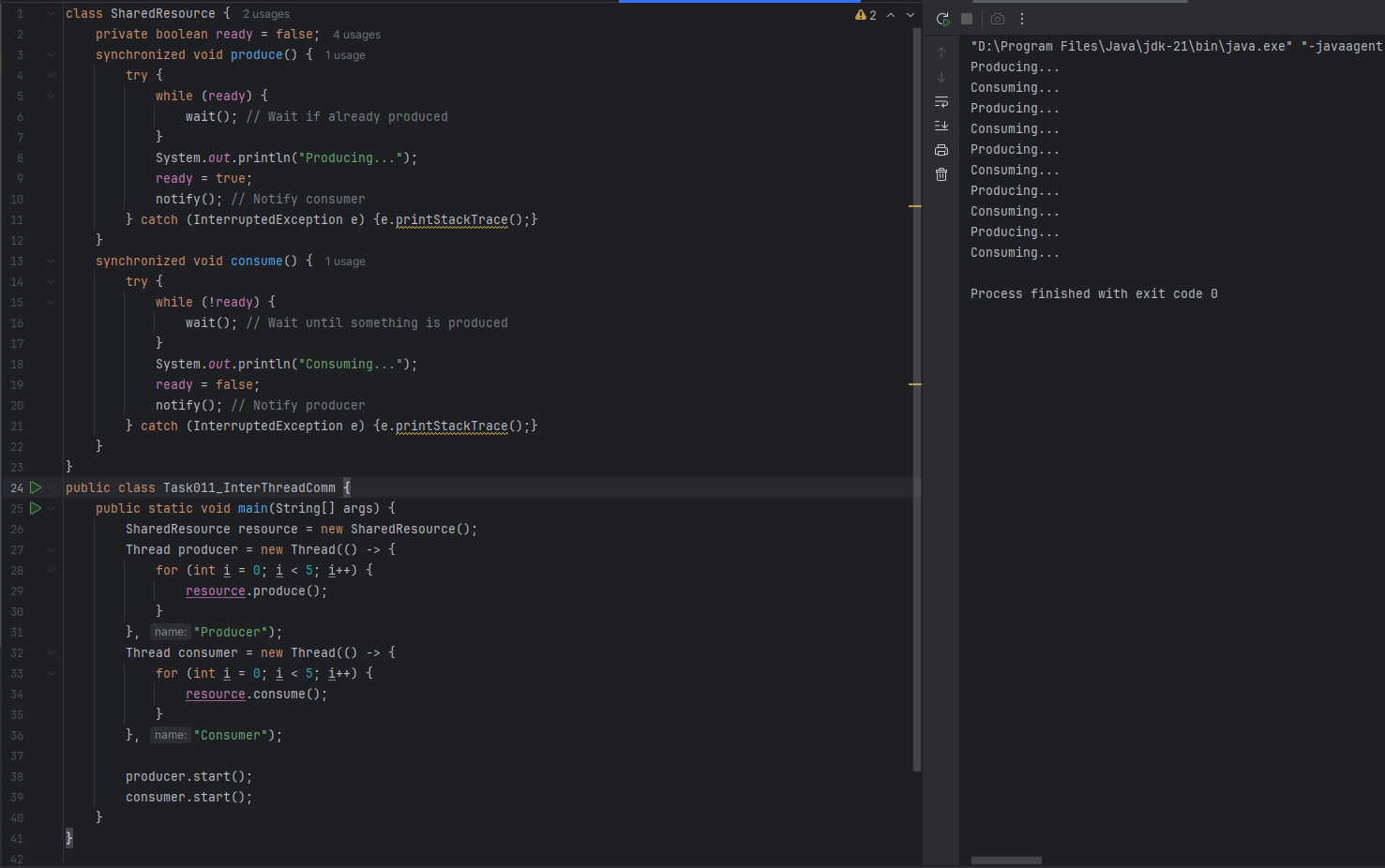
**How Synchronization Helps**

By using **synchronized blocks or methods**, Java ensures that only **one thread at a time** can access the critical section (shared resource), thus avoiding conflicts.

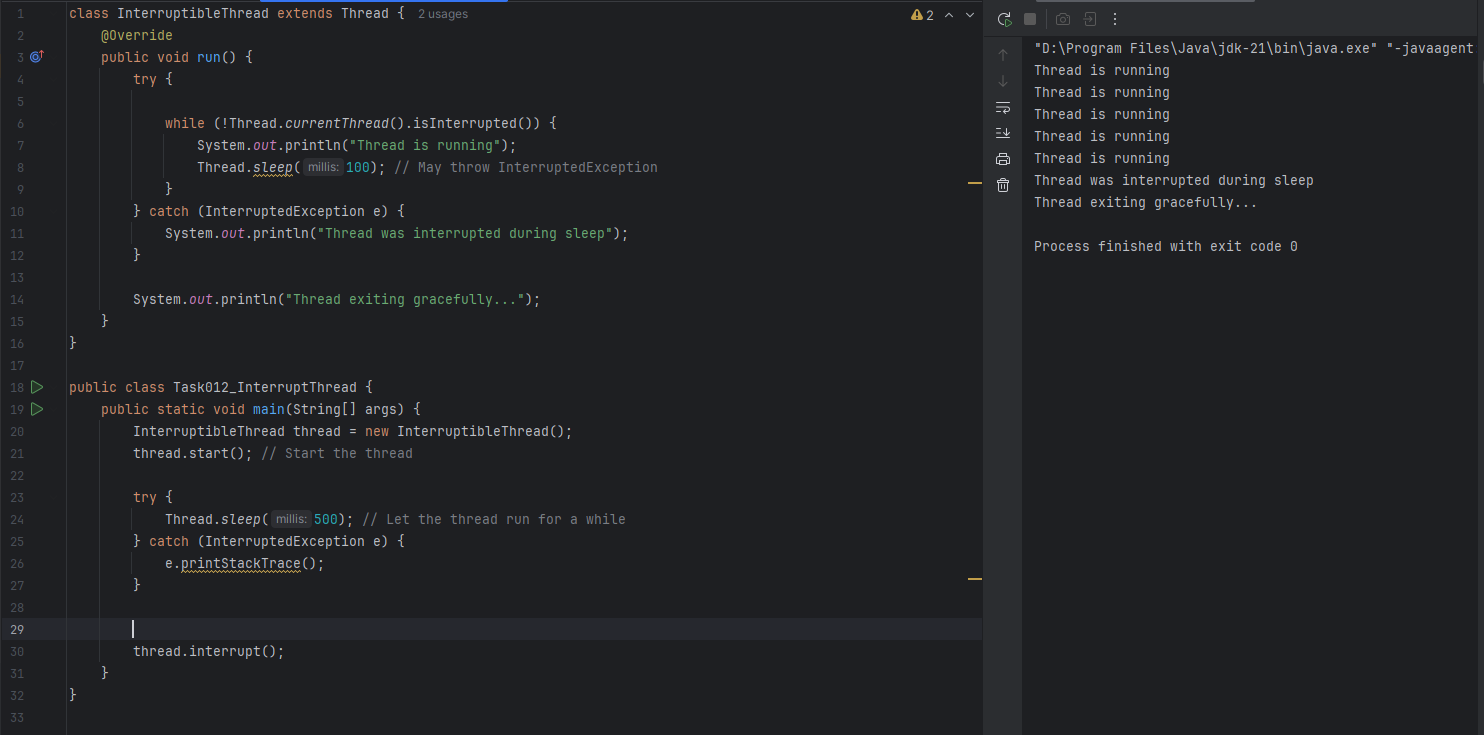
### ***Task 10: Demonstrate deadlock scenario using nested synchronized method calls***



### ***Task 11: Use wait and notify to manage thread interaction on shared resource***



### ***Task 12: Demonstrate thread interruption handling in Java***



### ***Task 13: Types of Synchronization in Java***

**A. Synchronized Method**

* **Declared with the synchronized keyword.**
* **Locks the entire method so that only one thread can access it at a time.**
* **Lock is on the object (instance).**

Example:

**public synchronized void increment() {**

**count++;**

**}**

**B. Synchronized Block**

* **More granular than synchronized method.**
* **Used to lock only a portion of the code.**
* **Useful for better performance (only critical section is locked).**

Example:

**public void increment() {**

**synchronized(this) {**

**count++;**

**}**

**}**

**We can also lock a specific object:**

**synchronized(someObject) {**

**// critical section**

**}**

**C. Static Synchronization**

* **When you synchronize a static method.**
* **Lock is on the class object, not instance.**

Example:

**public static synchronized void staticIncrement() {**

**// thread-safe code**

**}**

**Equivalent block version:**

**synchronized(MyClass.class) {**

**// critical section**

**}**

**D. Explicit Locking (ReentrantLock)**

* **From java.util.concurrent.locks.**
* **Provides manual locking/unlocking.**
* **More flexible than synchronized, supports:**
  + **Try-locking**
  + **Timed locks**
  + **Interruptible locks**

Example:

**Lock lock = new ReentrantLock();**

**lock.lock();**

**try {**

**// critical section**

**} finally {**

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

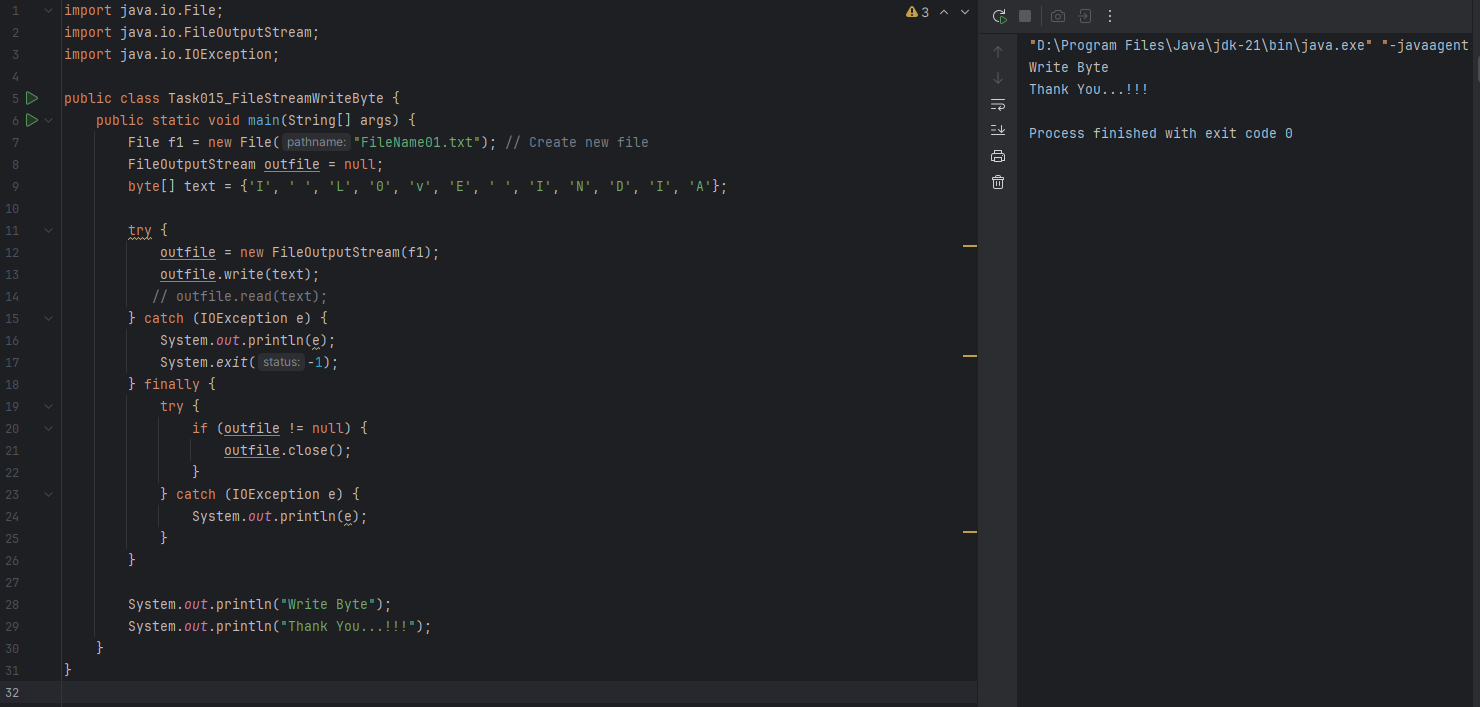
**}**

### ***Task 14: What is a Thread pool?***

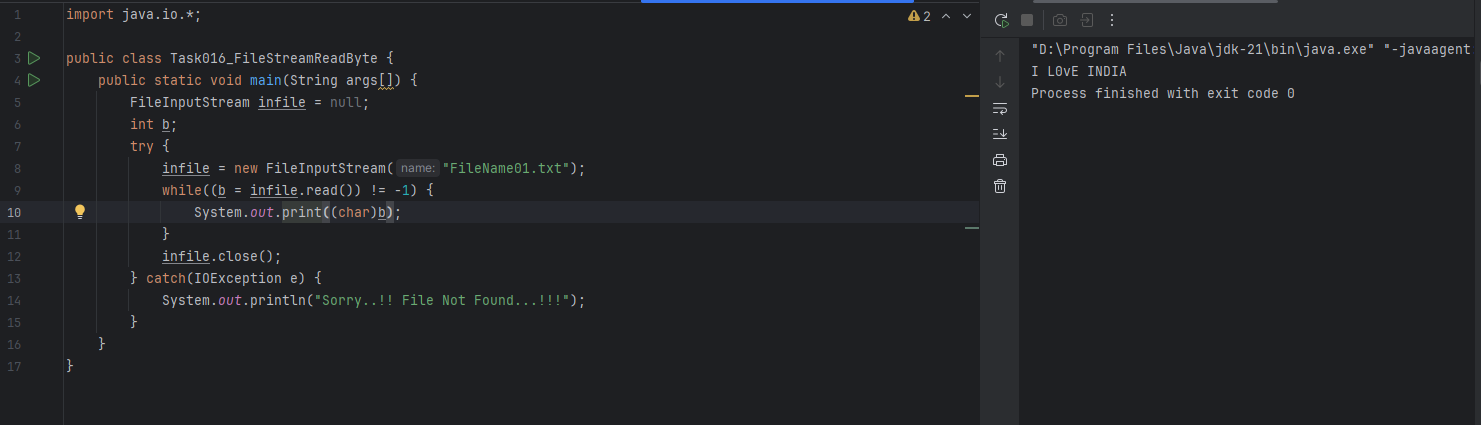
A **Thread Pool** is a collection (or pool) of pre-created, reusable threads that are managed by a system or application to perform multiple tasks efficiently.

Instead of creating a new thread for every task (which is slow and resource-heavy), a **thread pool reuses existing threads**, improving performance and resource management.

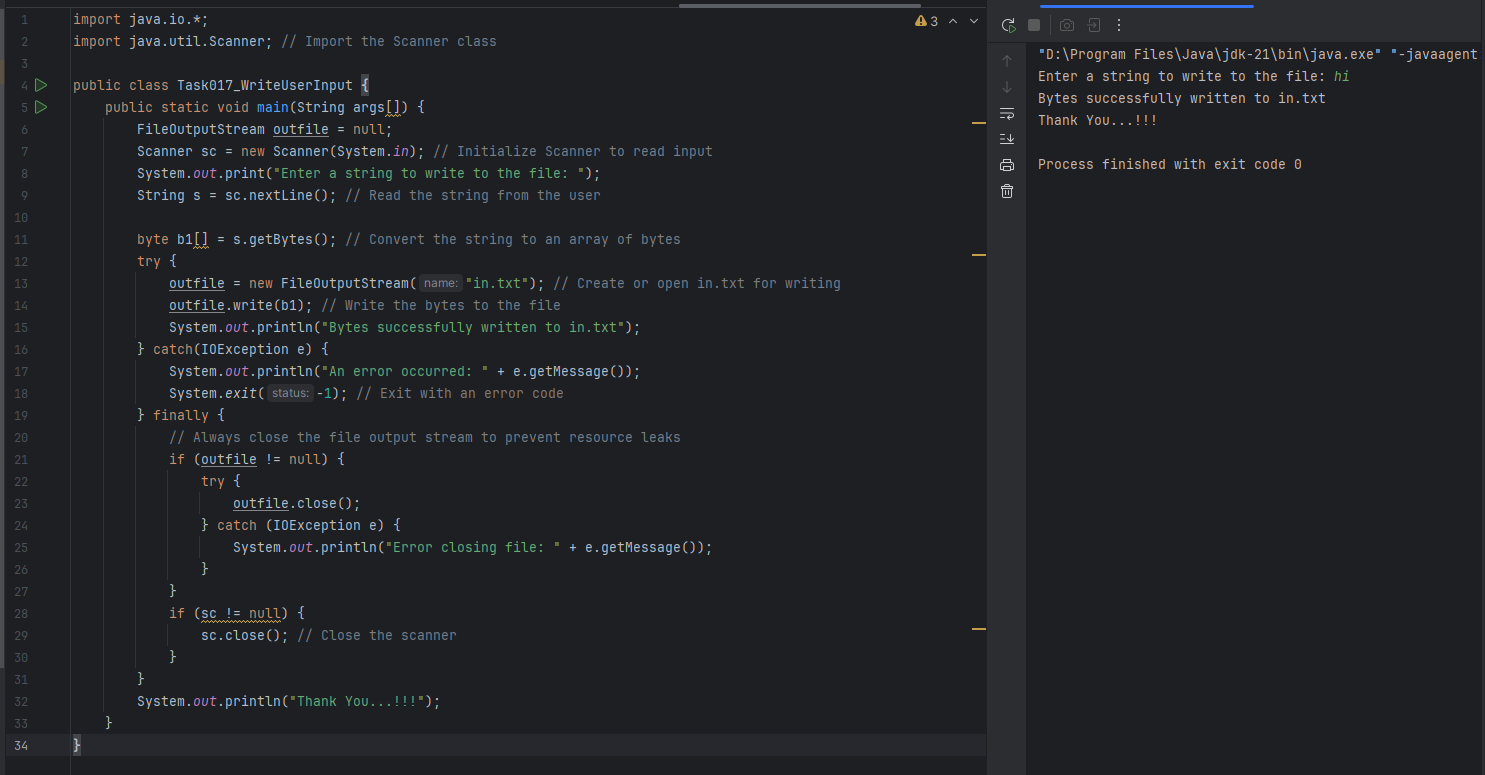
### ***Task 15: Write byte data to a file using FileOutputStream in Java***



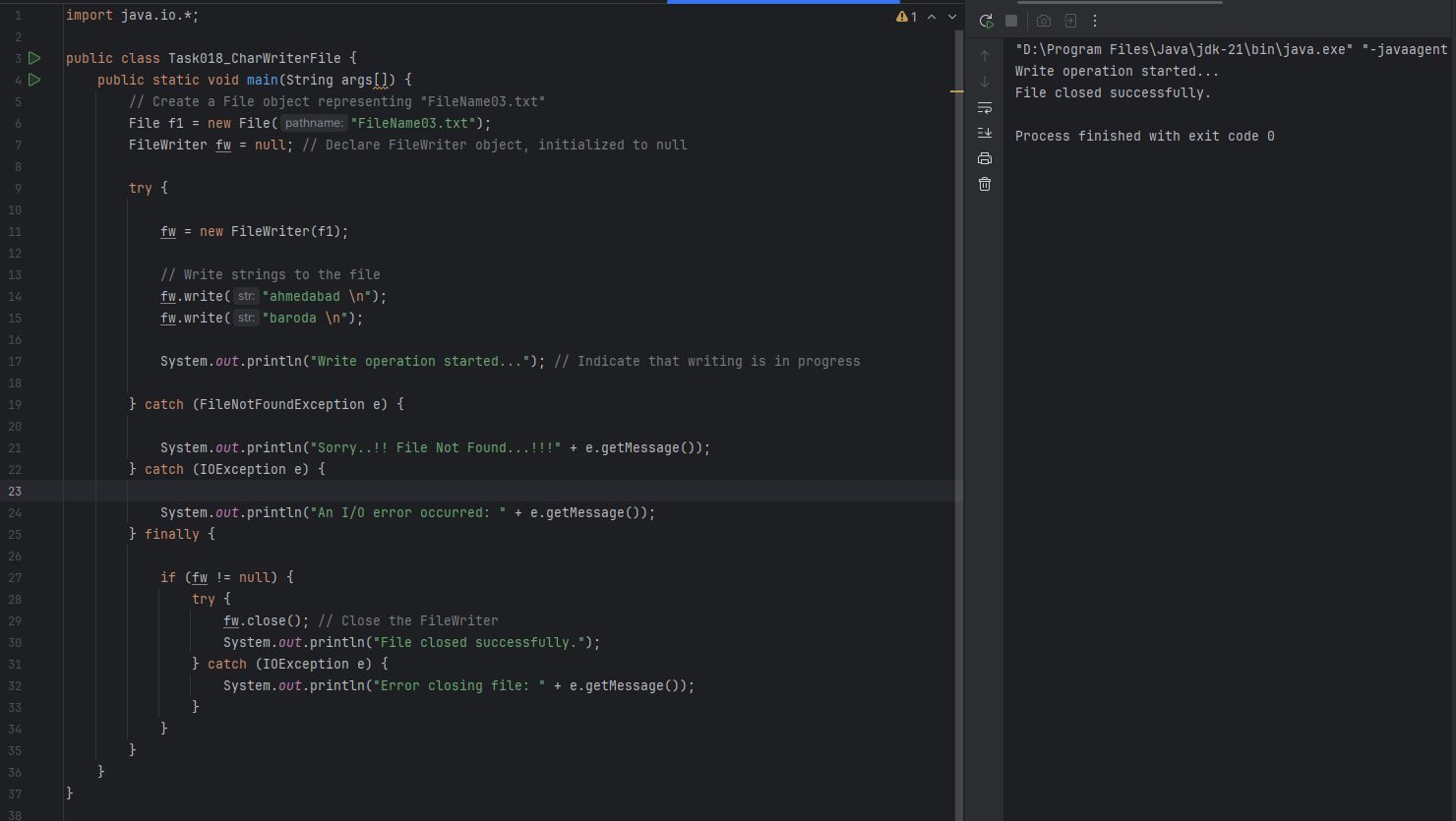
### ***Task 16: Read byte data from a file using FileInputStream in Java.***



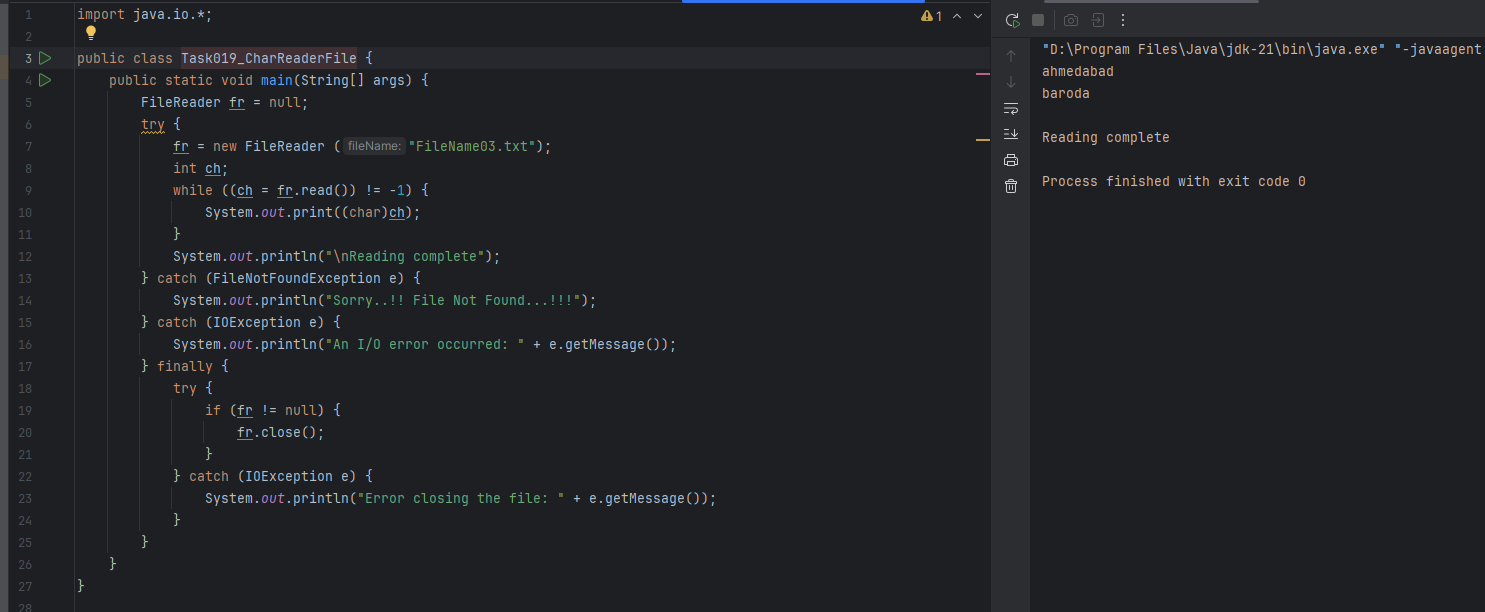
### ***Task 17: Accept user input and write it to a file using FileOutputStream.***



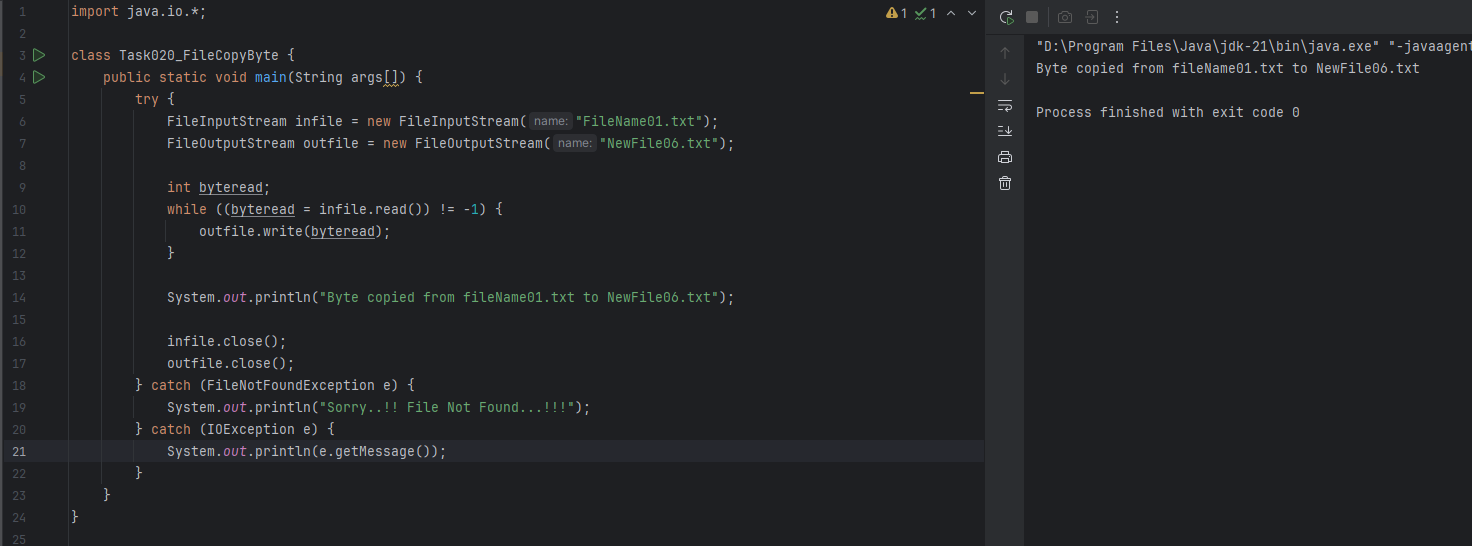
### ***Task 18: Write character data to a file using FileWriter in Java***



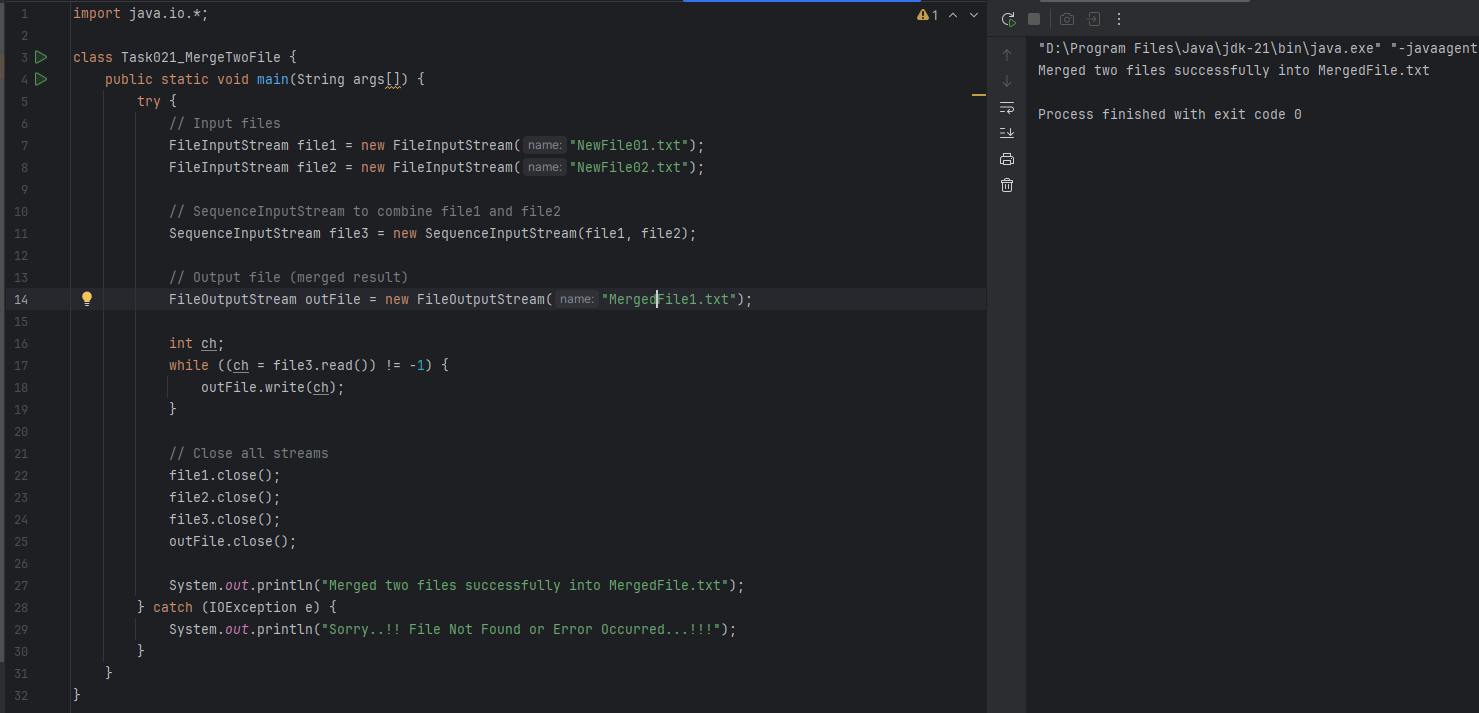
### ***Task 19: Read character data from a file using FileReader in Java***



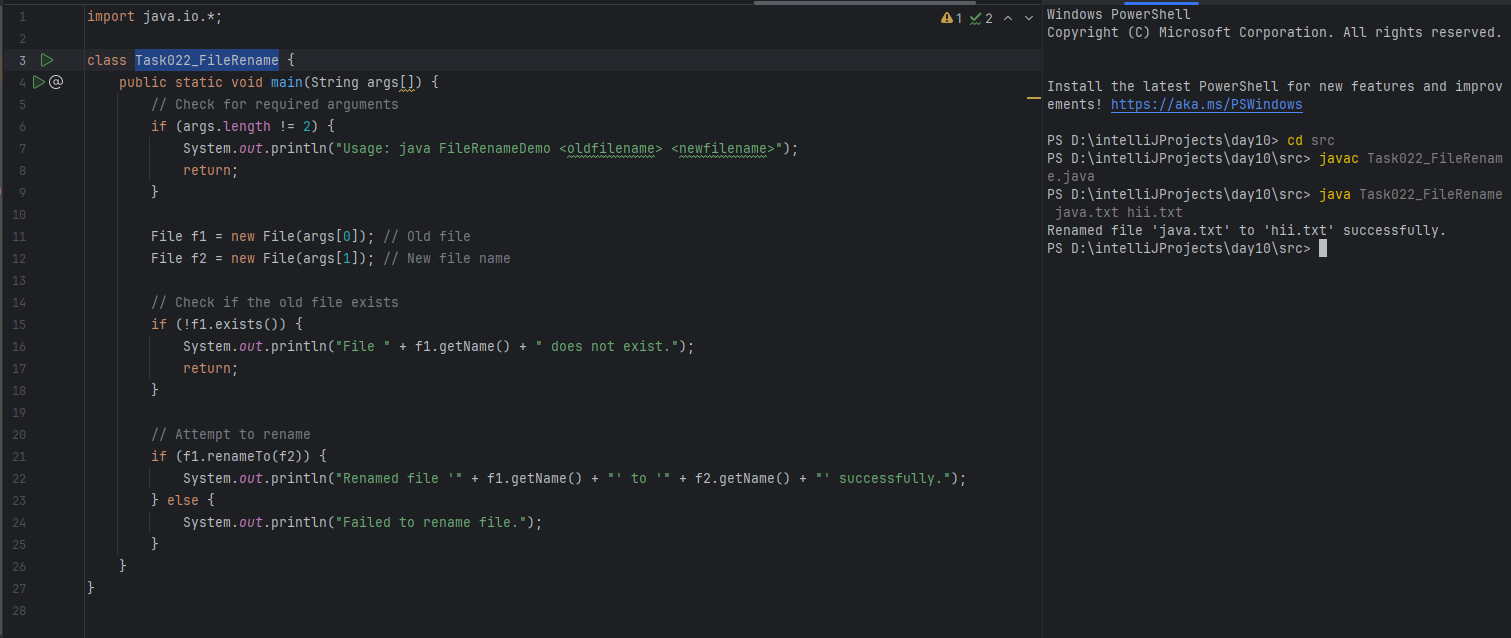
### ***Task 20: Copy using byte streams with FileInputStream and FileOutputStream***



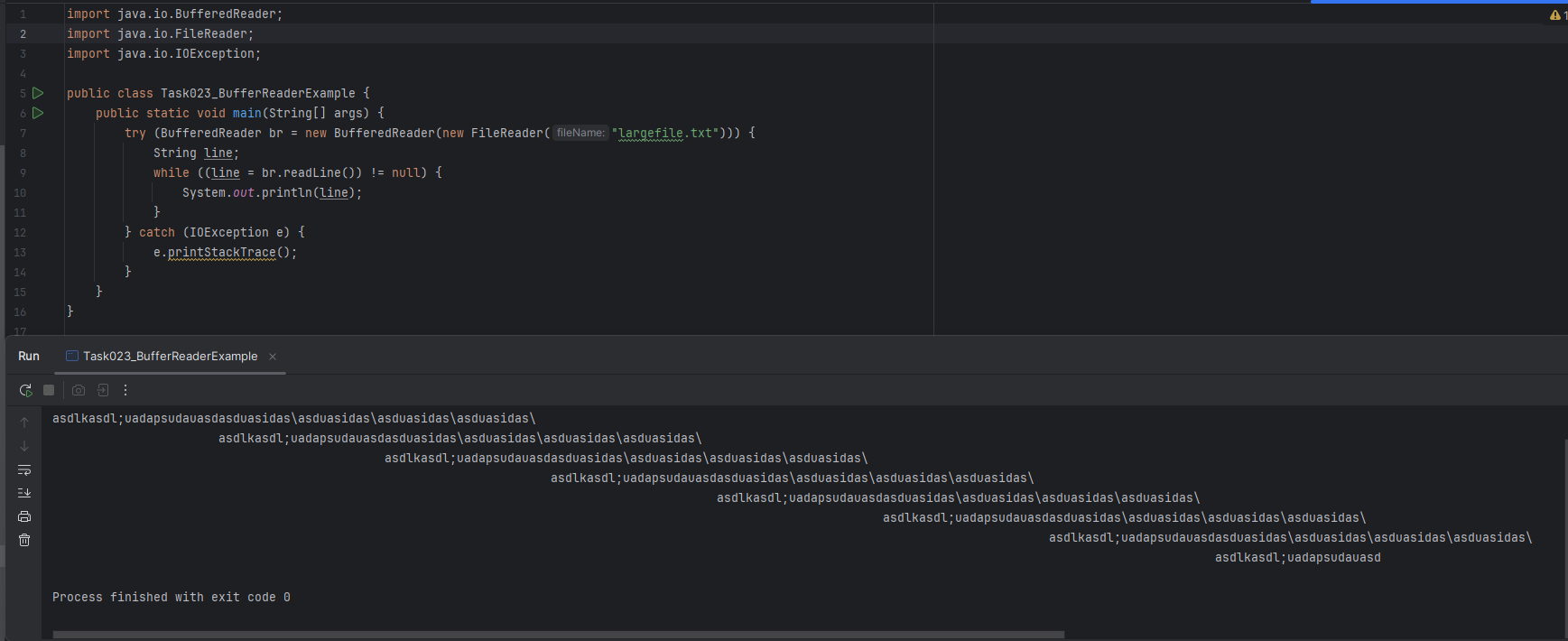
### ***Task 21: Merge two input files into one using SequenceInputStream in Java***



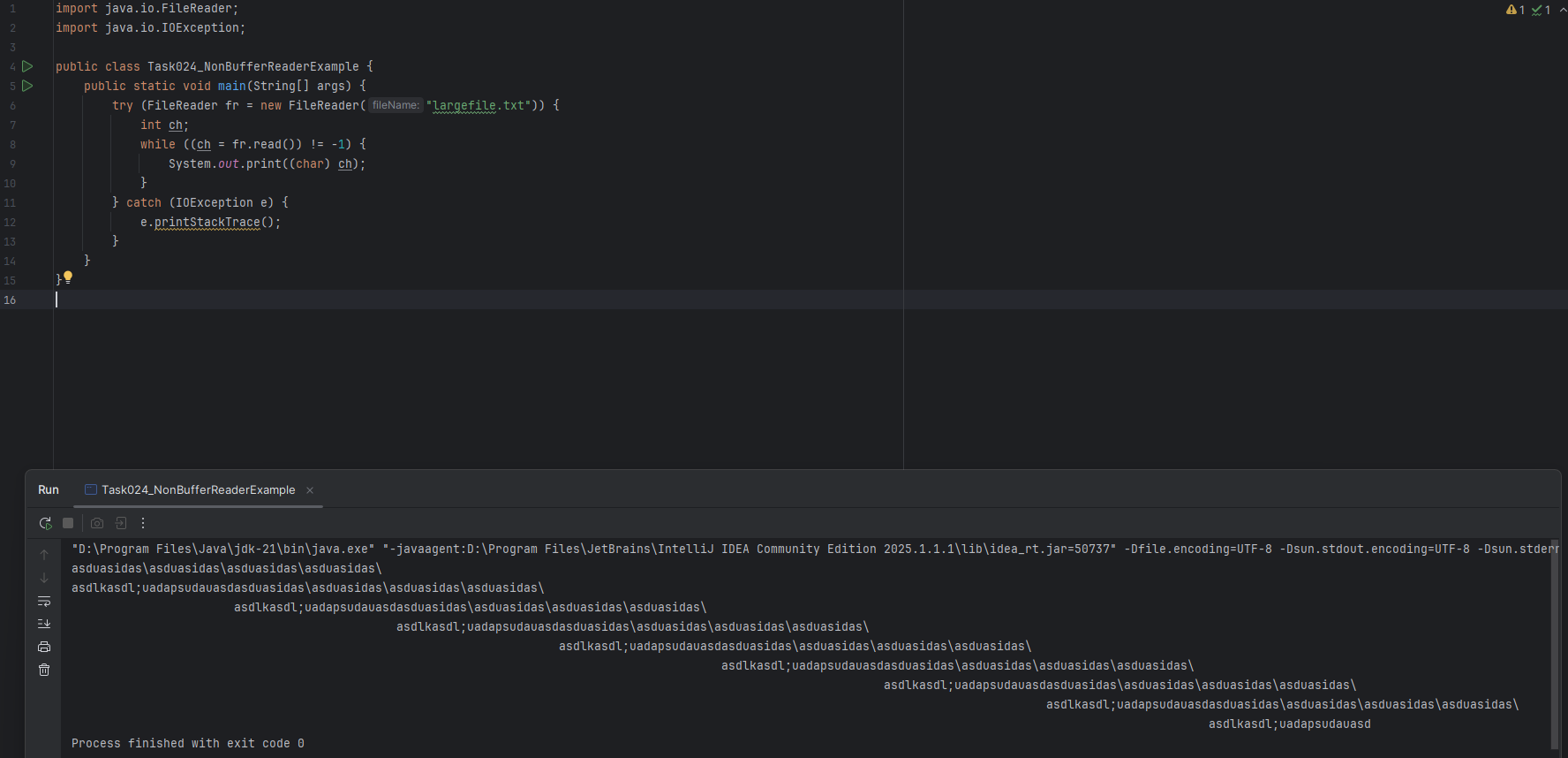
### ***Task 22: Rename a file using File class and command-line arguments in Java***



### ***Task 23: Read a text file line by line using BufferedReader in Java***



### ***Task 24: Read a file character by character using FileReader in Java***



### ***Task 25: Write text to a file using BufferedWriter***

