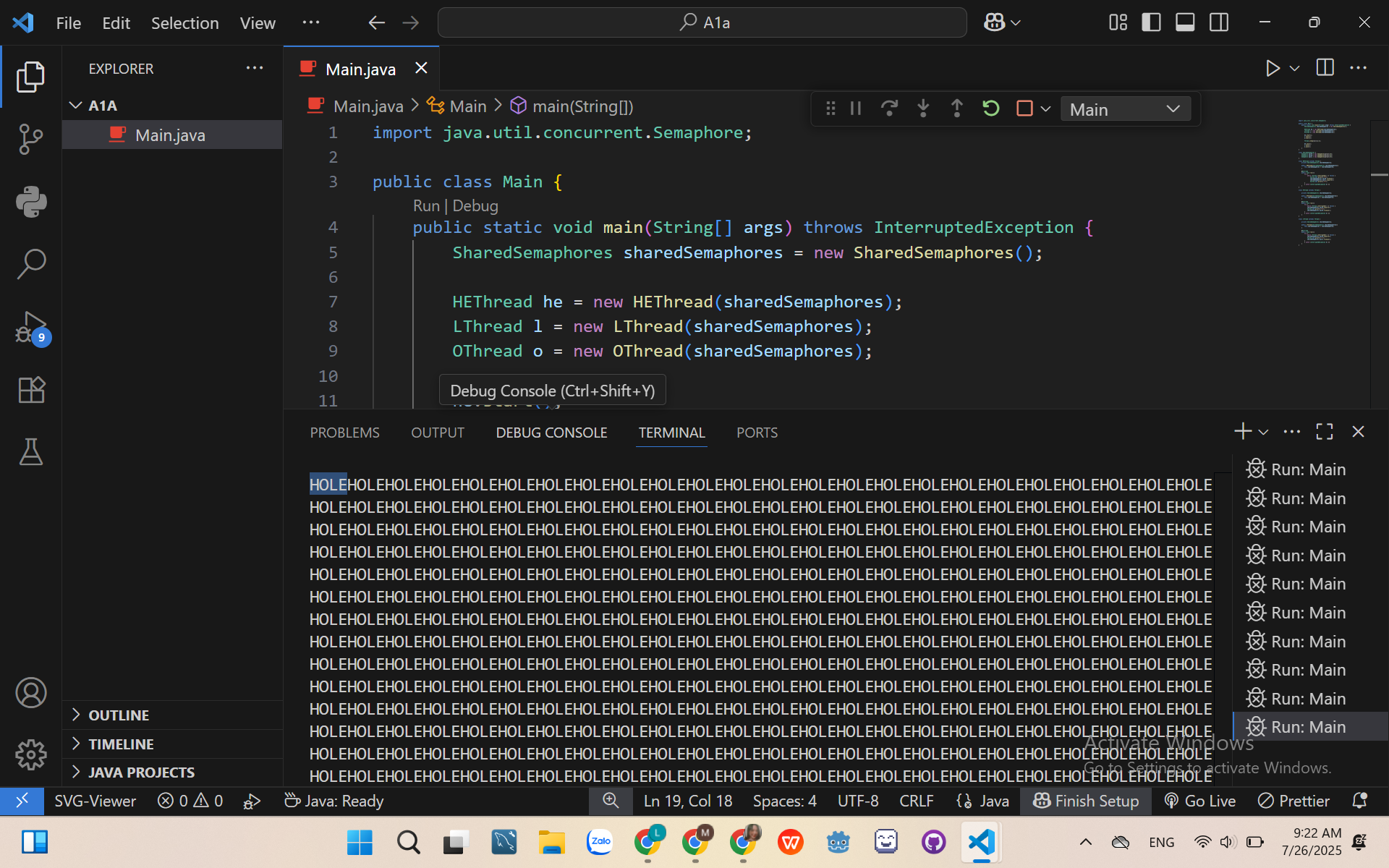
A1>

a>



Explanation:

1. Constraint:

* Must print in the order: H -> O -> L -> E
* The first thread repeadtedly prints the character 'H' and then the character 'E'.
* The cycle repeats indefinitely.

+ HEThread: print H right away; wait L to be printed then print E

+ OThread: wait for H to be printed

+ LThread: wait for O to be printed

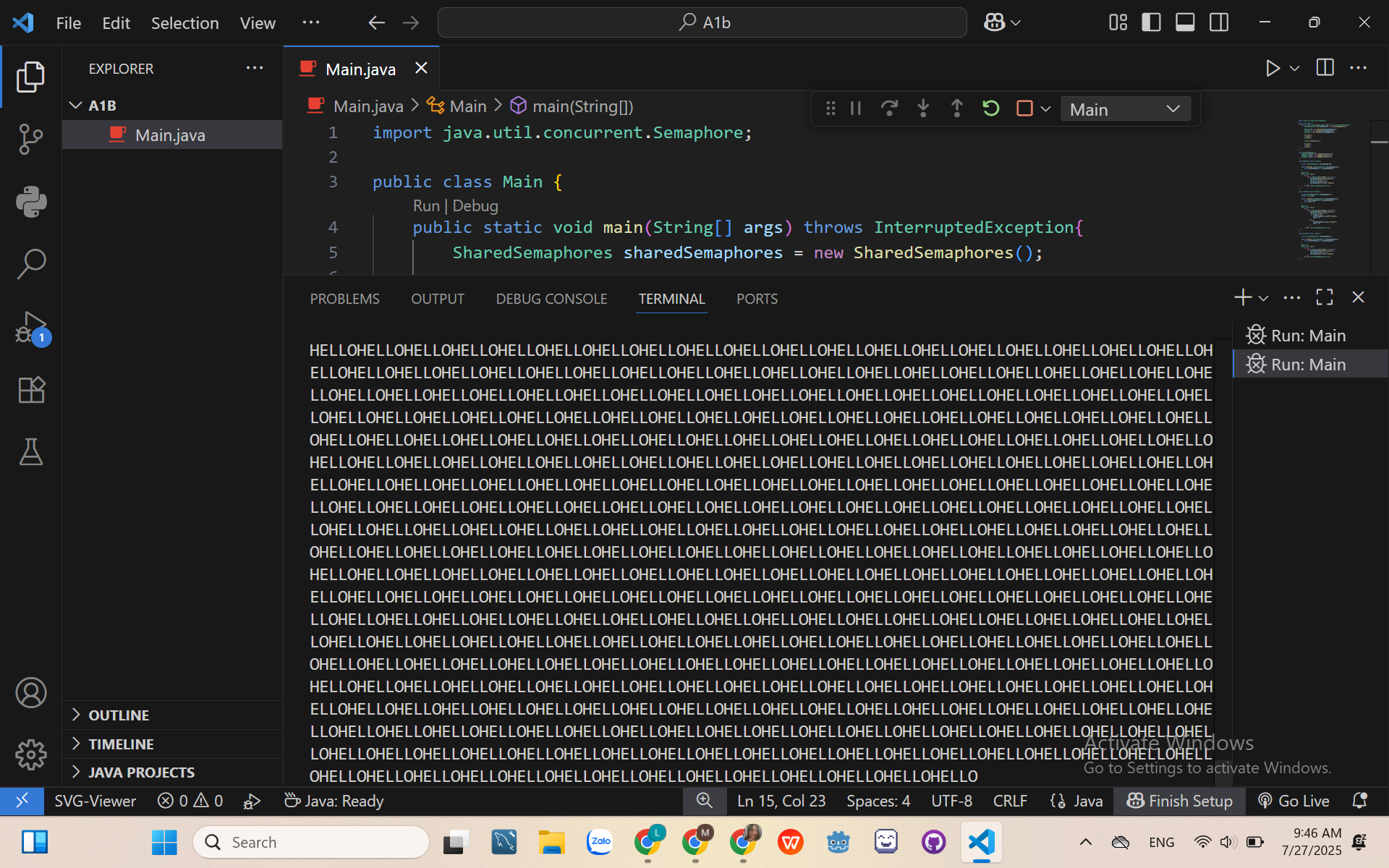
1. Shared State:

+ Semaphore waitH = new Semaphore(0);

+ Semaphore waitO = new Semaphore(0);

+ Semaphore waitL = new Semaphore(0);

b>



Explanation:

1. Constraint:

* Must print in the order: H -> E -> L -> L -> O
* The first thread repeadtedly prints the character 'H' and then the character 'E'.
* The ‘L’ need to be printed twice without using a loop
* The cycle repeats indefinitely.

+ HEThread: print HE right away

+ OThread: wait for L to be printed

+ LThread: if (times == 0), then wait for HE to be printed; if (times = 1), then still wait for HE to be printed

1. Shared State:

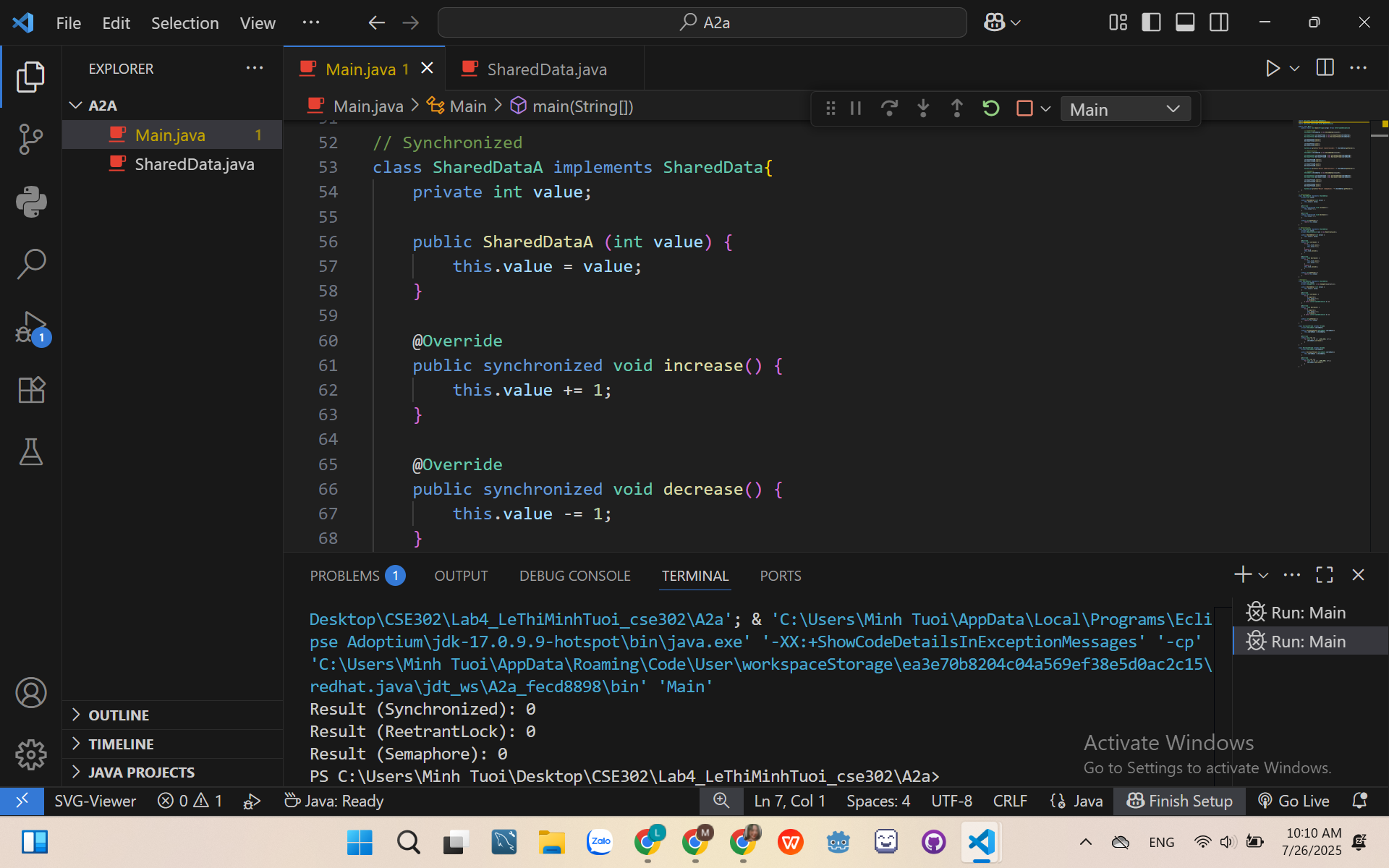
+ Semaphore waitHE = new Semaphore(0);

+ Semaphore waitO = new Semaphore(1);

+ Semaphore waitL = new Semaphore(0);

A2>

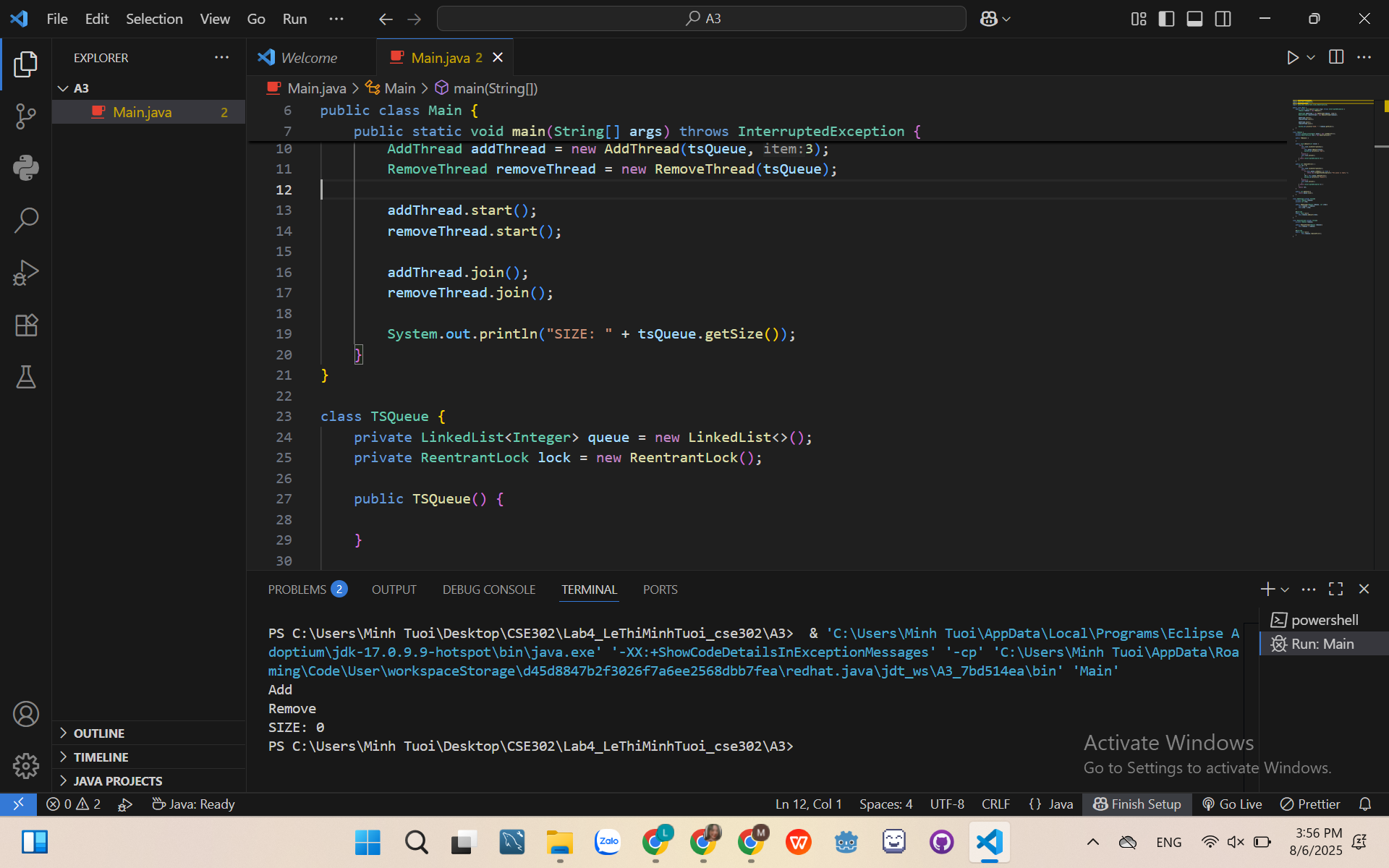
A, b, c>



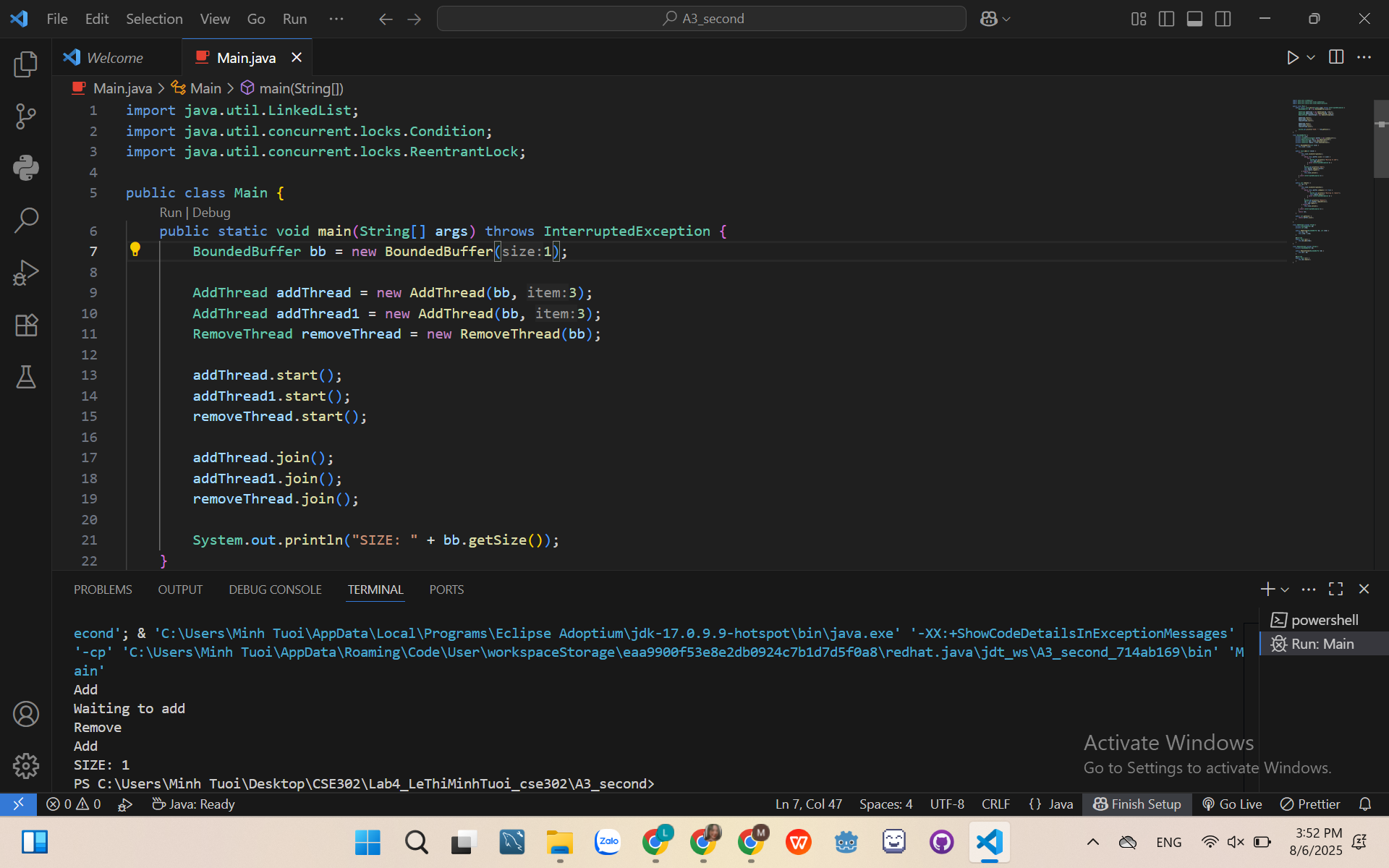
Explanation:

* Using Synchronized:
* Only one thread can be inside either increase() or decrease() at any given time.
* This ensures that the read–modify–write operation is atomic.
* The final output should be 0.
* Using ReentrantLock:
* Works like synchronized, but with more flexibility:
* Can try to acquire the lock with a timeout.
* Can interrupt while waiting for the lock.
* The try/finally ensures the lock is released even if an exception occurs.
* Using semaphore:
* A binary semaphore (new Semaphore(1)) works like a lock. Only one permit is available → only one thread at a time can enter the critical section.

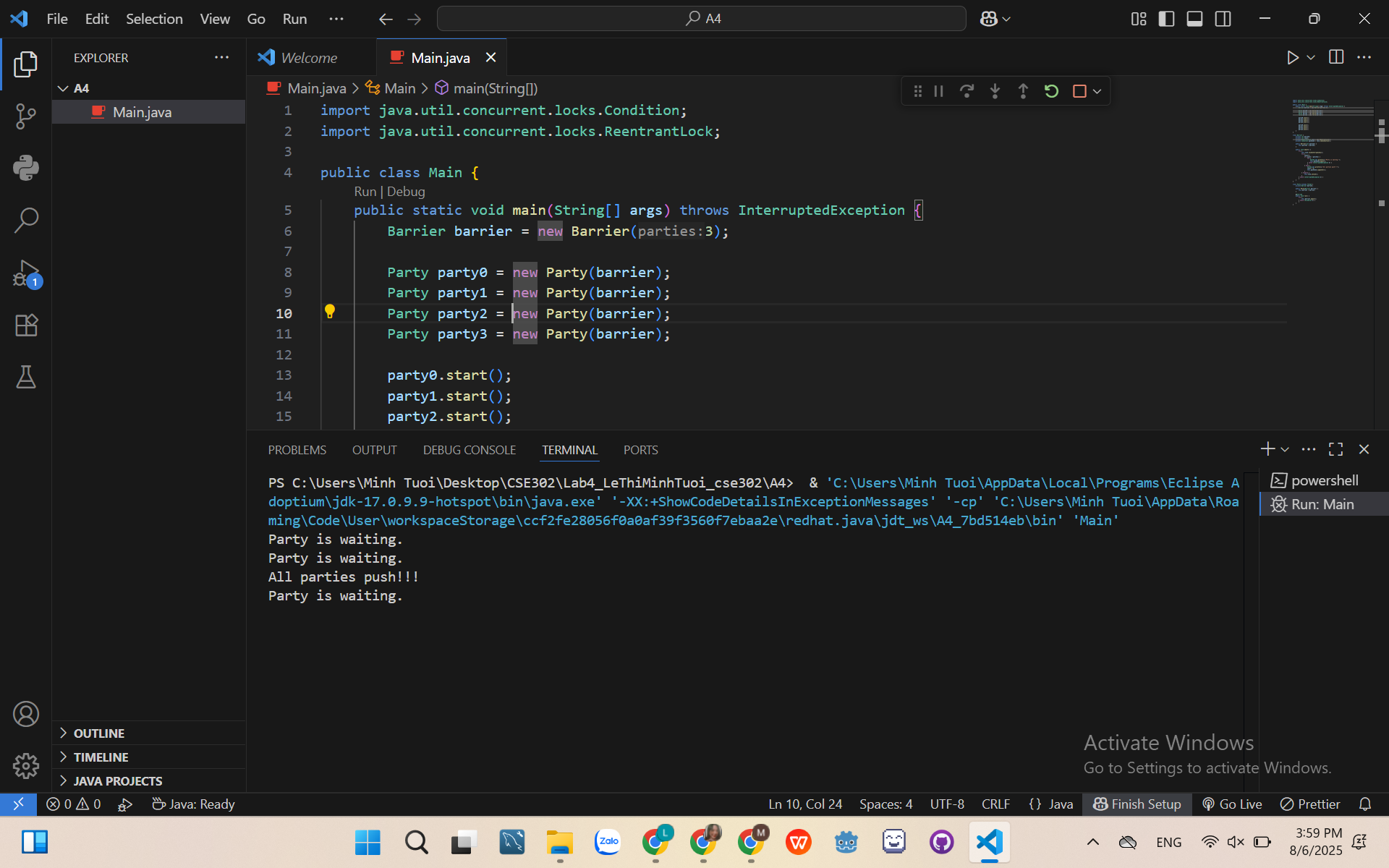
A3>



A3\_second>



A4>



A5>

