

A description of semaphore and how it works:

semaphores coordinating resource to prevent conflict on resources (synchronisation techniques). They are used to implement synchronisation between processes, impose mutual exclusion, and prevent race situations.

Wait (acquire) and signal (release) are the two operations offered by semaphores. The semaphore's value is decreased by the wait operation while it is increased by the signal operation. Any process that executes a wait action will be blocked until another process executes a signal operation when the semaphore's value is zero.

Critical sections—areas of code that can only be performed by one process at a time—are implemented using semaphores. Processes can coordinate access to shared resources, like shared memory or I/O devices, by utilizing semaphores.

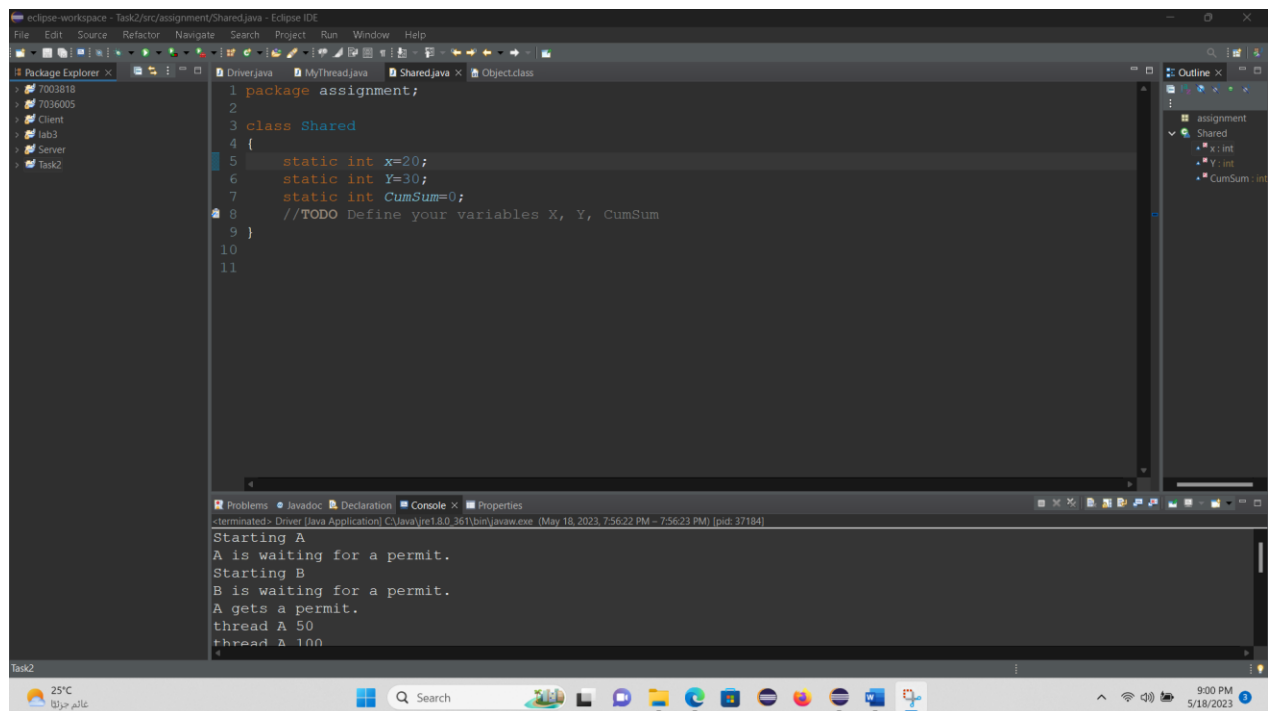
A discussion about what you used in the assignment:

The program uses a semaphore to control access to the CumSum,X,Y variables, which is a static variable within the Shared class. Shared.CumSum performing operation $CumSum = ((x + y) + CumSum)$ five times by thread A and performing operation $CumSum = (CumSum - (x + y))$ five times by thread B .To prevent these two threads from accessing Shared.CumSum and Shared.X and Shared.Y at the same time, access is allowed only after a permit is acquired from the controlling semaphore. After access is complete, the permit is released. In this way, only one thread at a time will access Shared.CumSum and Shared.X and Shared.Y , as the output shows.

Notice the call to sleep() within run() method inside MyThread class. It is used to prove that accesses to Shared.CumSum and Shared.X and Shared.Y are synchronized by the semaphore. In run(), the call to sleep() causes the invoking thread to pause between each access to Shared.CumSum and Shared.X and Shared.Y . This would normally enable the second thread to run. However, because of the semaphore, the second thread must wait until the first has released the permit, which happens only after all accesses by the first thread are complete. Thus, Shared.CumSum is first incremented by value $= (Shared.X + Shared.Y)$ five times by thread A and then decremented by value $= (Shared.X + Shared.Y)$ five times by thread B.

Step 1

We started by initializing the variables X=20 , Y=30 , CumSum = 0 in Shared Class .



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows a project named 'Task2' with sub-packages 'Client', 'Server', and 'Task2'. The main editor displays the 'Shared.java' file with the following code:

```
1 package assignment;
2
3 class Shared
4 {
5     static int x=20;
6     static int Y=30;
7     static int CumSum=0;
8     //TODO Define your variables X, Y, CumSum
9 }
10
11
```

The Outline view on the right shows the class structure: 'assignment' containing 'Shared', which has static variables 'x : int', 'Y : int', and 'CumSum : int'.

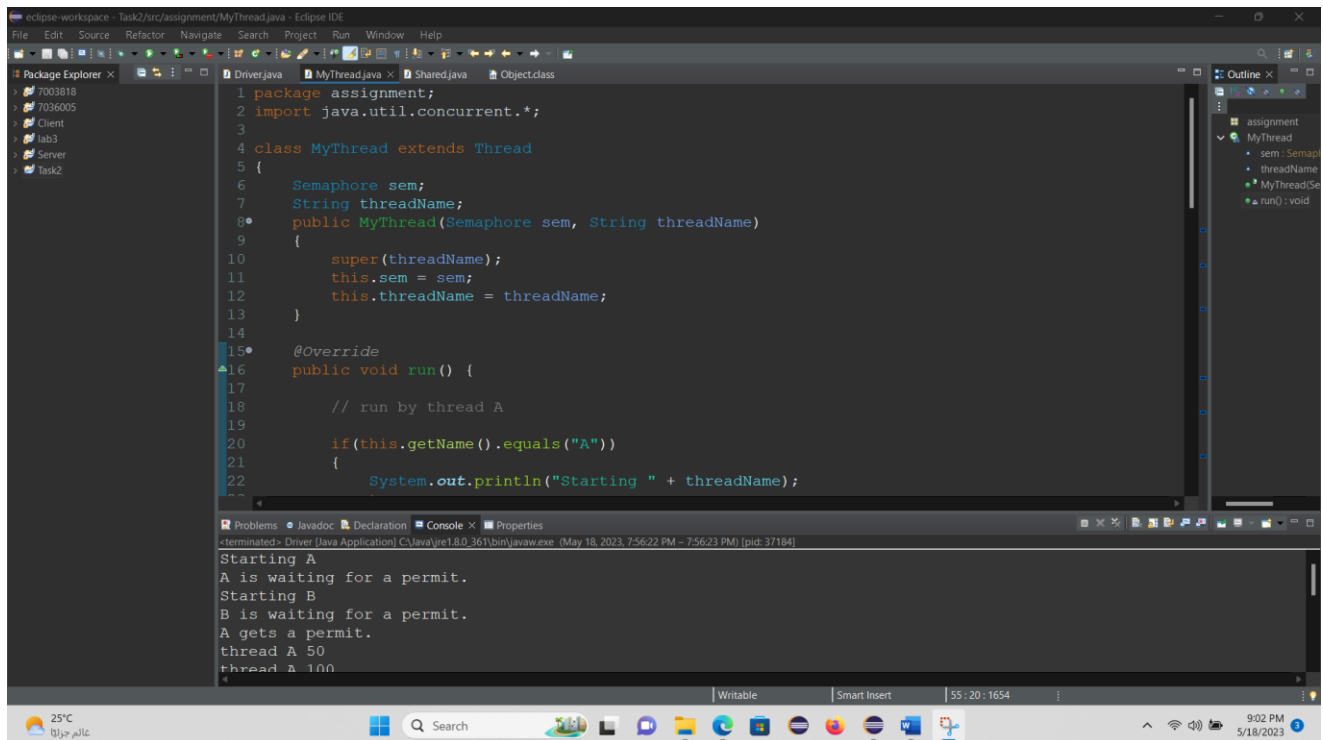
The Console view at the bottom shows the output of the application:

```
Starting A
A is waiting for a permit.
Starting B
B is waiting for a permit.
A gets a permit.
thread A 50
thread A 100
```

The status bar at the bottom indicates the temperature is 25°C and the date is 5/18/2023.

Step 2

Then we extended our MyThread class to thread class to allow threads run in parallel and overriding run method .



The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows a project named 'Task2' with sub-packages 'Client', 'lab3', 'Server', and 'Task2'.
- MyThread.java:** The main file being edited, containing the following code:

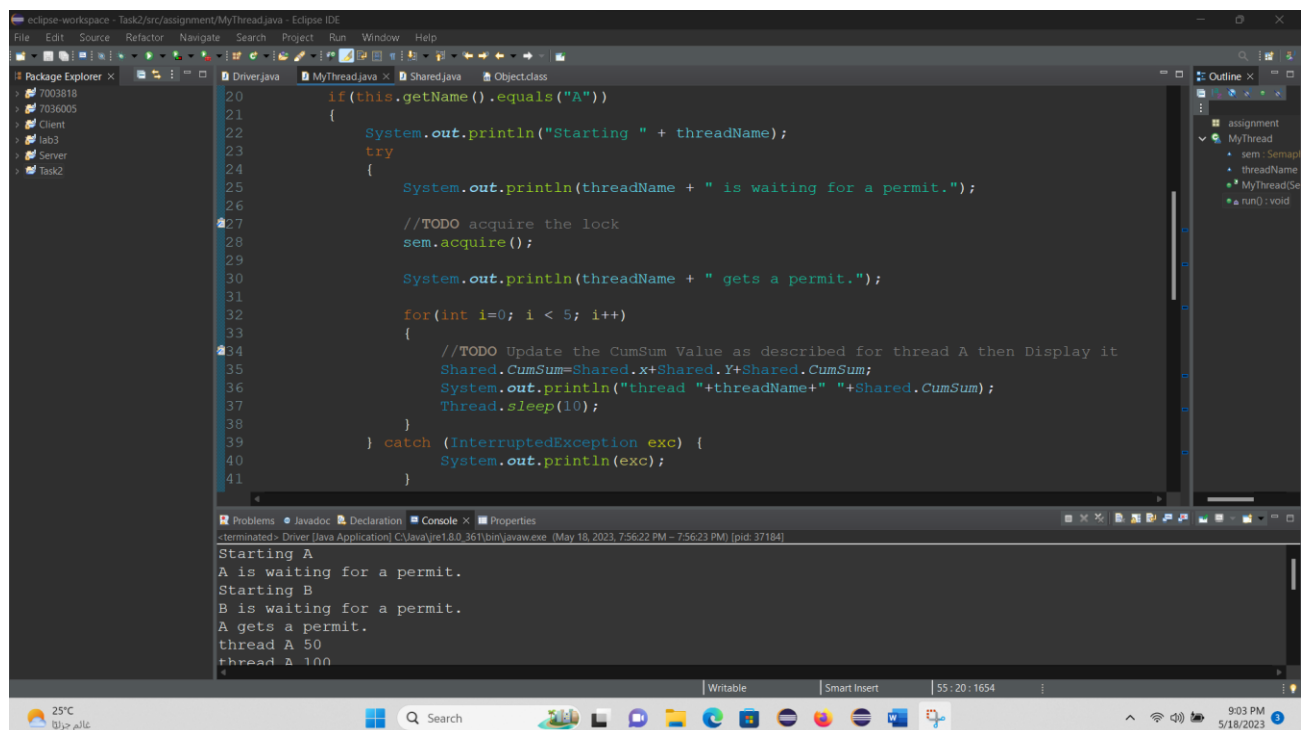
```
1 package assignment;
2 import java.util.concurrent.*;
3
4 class MyThread extends Thread
5 {
6     Semaphore sem;
7     String threadName;
8     public MyThread(Semaphore sem, String threadName)
9     {
10         super(threadName);
11         this.sem = sem;
12         this.threadName = threadName;
13     }
14
15     @Override
16     public void run() {
17
18         // run by thread A
19
20         if(this.getName().equals("A"))
21         {
22             System.out.println("Starting " + threadName);
```

- Outline:** Shows the class structure with members: 'sem: Semaphore', 'threadName: String', 'MyThread(Semaphore sem, String threadName): void', and 'run(): void'.
- Console:** Displays the output of the program:

```
<terminated> Driver (Java Application) C:\Java\jre1.8.0_361\bin\javaw.exe (May 18, 2023, 7:56:22 PM) [pid: 37184]
Starting A
A is waiting for a permit.
Starting B
B is waiting for a permit.
A gets a permit.
thread A 50
thread A 100
```

Step 3

We check Name of the thread if it equal to 'A' then we print process Name and surround body with try and catch statements as it may throws InterruptedException then in body of try statement printing the Thread name and locking the resource then printing that the thread get a permit then we will keep incrementing the CumSum by value equal to (X+Y) for 5 times and at each time we will print the new value of CumSum in each time and we will make the thread sleep for 10 milli seconds in each iteration and in case of exception has been thrown we will print the exception

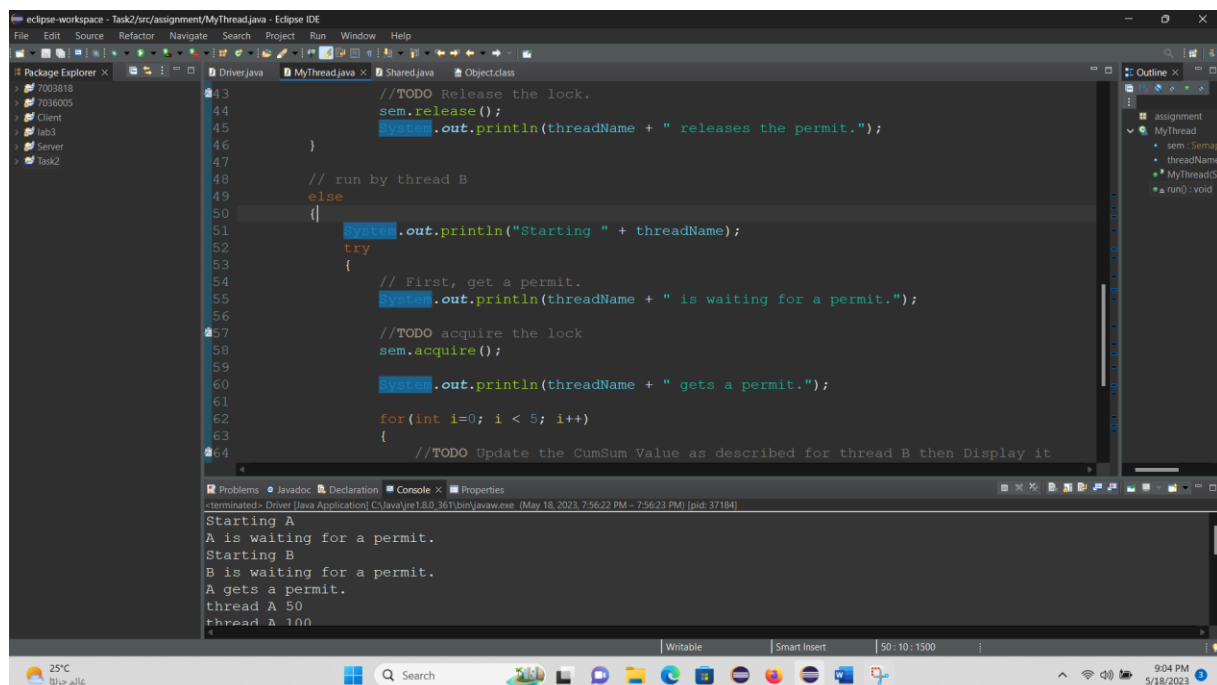


```
20     if(this.getName().equals("A"))
21     {
22         System.out.println("Starting " + threadName);
23         try
24         {
25             System.out.println(threadName + " is waiting for a permit.");
26
27             //TODO acquire the lock
28             sem.acquire();
29
30             System.out.println(threadName + " gets a permit.");
31
32             for(int i=0; i < 5; i++)
33             {
34                 //TODO Update the CumSum Value as described for thread A then Display it
35                 Shared.CumSum=Shared.x+Shared.Y+Shared.CumSum;
36                 System.out.println("thread "+threadName+" "+Shared.CumSum);
37                 Thread.sleep(10);
38             }
39         } catch (InterruptedException exc) {
40             System.out.println(exc);
41         }
42     }
```

Starting A
A is waiting for a permit.
Starting B
B is waiting for a permit.
A gets a permit.
thread A 50
thread A 100

Step 4

We will release the resource and printing that thread released the resource.
In case if it is not 'A' then it should be 'B' then we print process Name and
surround body with try and catch statements as it may throws
InterruptedException then in body of try statement printing the Thread name
then we wait to get a permit and printing that thread is waiting untill it accuire
the resource after it gets hold on the resource , we lock the resource then
printing that the thread get a permit



The screenshot shows the Eclipse IDE with a Java project named 'Task2'. The main editor displays the code for 'MyThread.java'. The code implements a thread that can be run by either thread A or thread B. Thread A releases a semaphore, and thread B acquires it, performing a task (printing thread name and a loop) while holding the lock. The console output shows the execution flow: 'Starting A', 'A is waiting for a permit.', 'Starting B', 'B is waiting for a permit.', 'A gets a permit.', 'thread A 50', and 'thread A 100'.

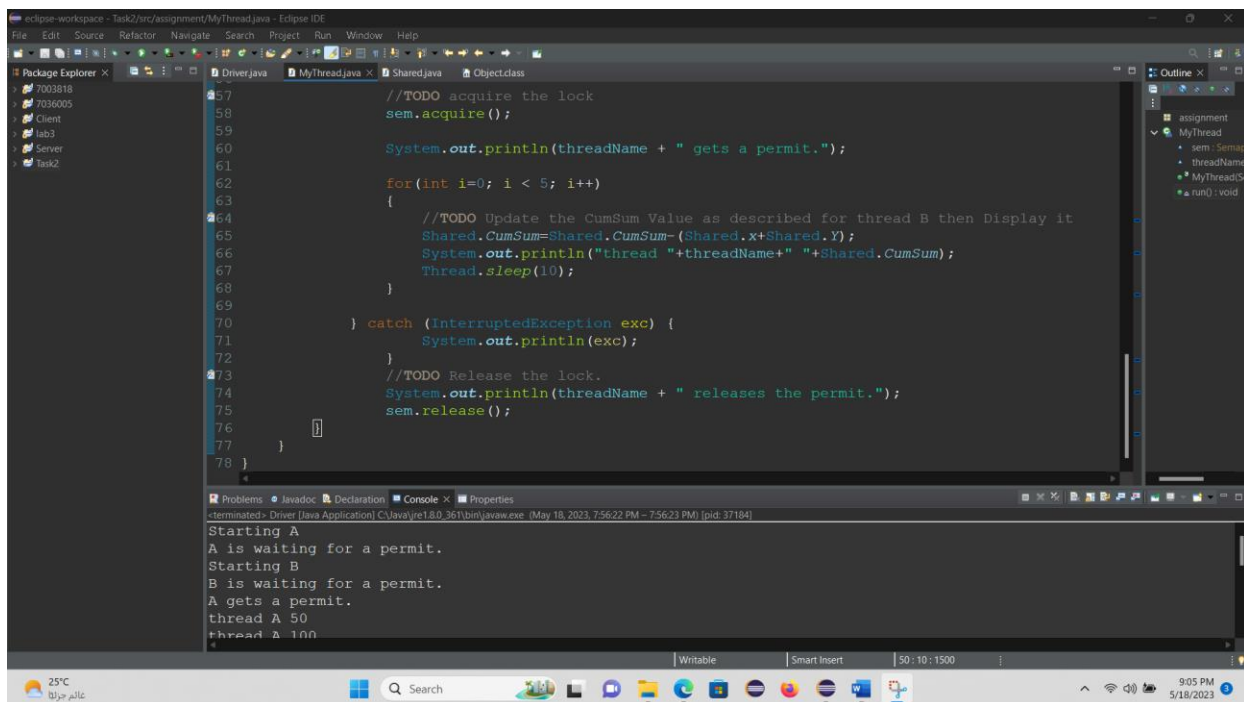
```
43 //TODO Release the lock.  
44 sem.release();  
45 System.out.println(threadName + " releases the permit.");  
46 }  
47  
48 // run by thread B  
49 else  
50 {  
51     System.out.println("Starting " + threadName);  
52     try  
53     {  
54         // First, get a permit.  
55         System.out.println(threadName + " is waiting for a permit.");  
56  
57         //TODO acquire the lock  
58         sem.acquire();  
59  
60         System.out.println(threadName + " gets a permit.");  
61  
62         for(int i=0; i < 5; i++)  
63         {  
64             //TODO Update the CumSum Value as described for thread B then Display it  
65         }  
66     }  
67     catch (InterruptedException e)  
68     {  
69         System.out.println(threadName + " interrupted while waiting for a permit.");  
70     }  
71 }
```

Console Output:

```
<terminated> Driver [Java Application] C:\Java\jre1.8.0_361\bin\javaw.exe (May 18, 2023, 7:56:22 PM - 7:56:23 PM) [pid: 37184]  
Starting A  
A is waiting for a permit.  
Starting B  
B is waiting for a permit.  
A gets a permit.  
thread A 50  
thread A 100
```

Step 5

then we will keep decrementing the CumSum by value equal to (X+Y) for 5 times and at each time we will print the new value of CumSum in each time and we will make the thread sleep for 10 milli seconds in each iteration and in case of exception has been thrown we will print the exception and then We will release the resource and printing that thread released the resource.

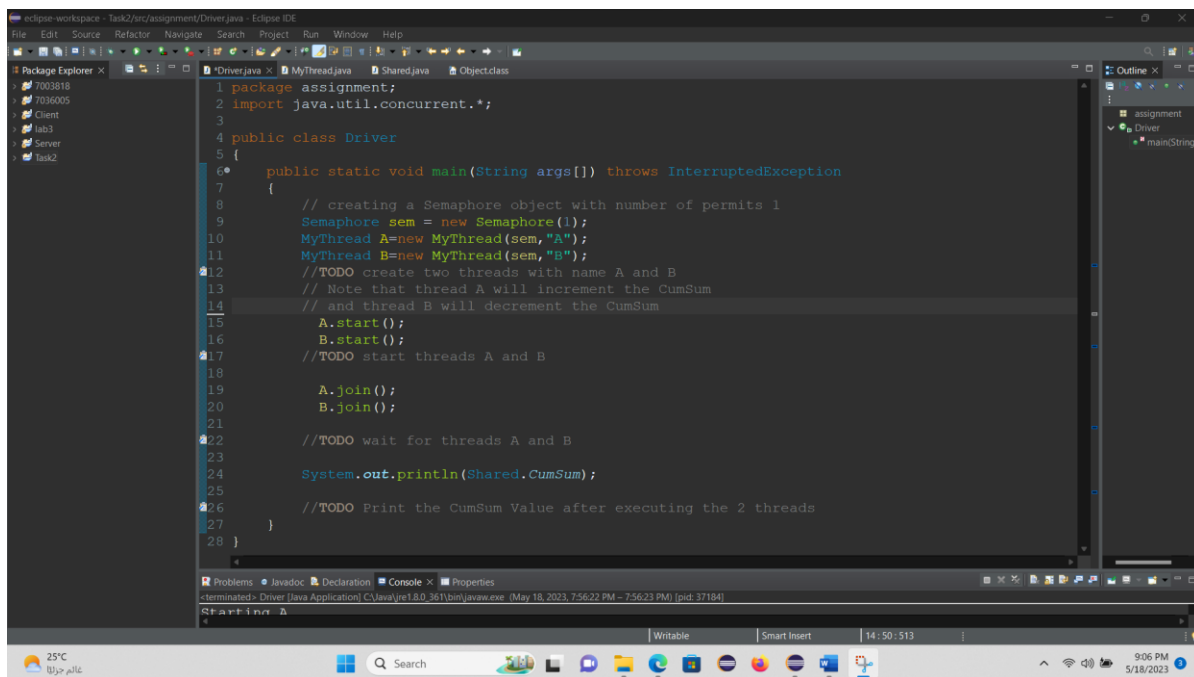


```
57 //TODO acquire the lock
58 sem.acquire();
59
60 System.out.println(threadName + " gets a permit.");
61
62 for(int i=0; i < 5; i++)
63 {
64 //TODO Update the CumSum Value as described for thread B then Display it
65 Shared.CumSum=Shared.CumSum- (Shared.x+Shared.Y);
66 System.out.println("thread "+threadName+" "+Shared.CumSum);
67 Thread.sleep(10);
68 }
69
70 } catch (InterruptedException exc) {
71 System.out.println(exc);
72 }
73 //TODO Release the lock.
74 System.out.println(threadName + " releases the permit.");
75 sem.release();
76
77 }
78 }
```

Starting A
A is waiting for a permit.
Starting B
B is waiting for a permit.
A gets a permit.
thread A 50
thread A 100

Step 6

In driver class we will create a semaphore (sem) and initialize it with 1 which is initial permit count and we will create 2 Threads from MyThread , MyThread A will take a value (sem, 'A') , and MyThread B will take a value (sem, 'B') then will start the 2 threads to run in parallel Then we will wait for thread A and B to finish running without conflict on resource each one at a time , one will wait till the other running till death then the waiting one will start running till death also Then printing the final value for the CumSum

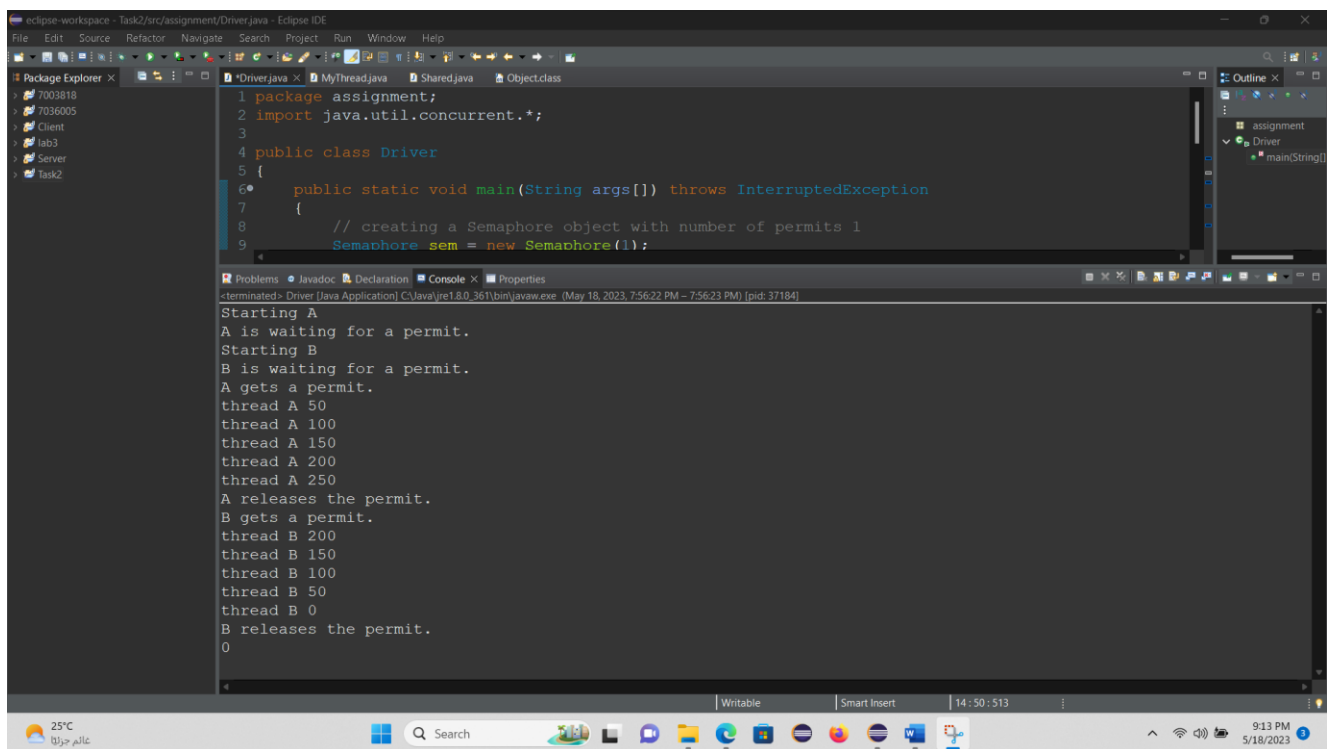


```
1 package assignment;
2 import java.util.concurrent.*;
3
4 public class Driver
5 {
6     public static void main(String args[]) throws InterruptedException
7     {
8         // creating a Semaphore object with number of permits 1
9         Semaphore sem = new Semaphore(1);
10        MyThread A=new MyThread(sem,"A");
11        MyThread B=new MyThread(sem,"B");
12        //TODO create two threads with name A and B
13        // Note that thread A will increment the CumSum
14        // and thread B will decrement the CumSum
15        A.start();
16        B.start();
17        //TODO start threads A and B
18
19        A.join();
20        B.join();
21
22        //TODO wait for threads A and B
23
24        System.out.println(Shared.CumSum);
25
26        //TODO Print the CumSum Value after executing the 2 threads
27    }
28 }
```

Problems • Javadoc • Declaration • Console x • Properties
<terminated> Driver [Java Application] C:\Java\jdk1.8.0_361\bin\java.exe (May 18, 2023, 7:56:22 PM - 7:56:23 PM) [pid: 37184]
Starting A

Step 7

In the output we can see that B started running then A , then A waited for B, then B waited for A , then A get the permit and we kept incrementing the CumSum in Thread A by value X+Y which is 50 for 5 times (50,100,150,200,250) then A thread died ,then B thread gets a permit to use the resource as A finished and ,then A released it , then kept decrementing the CumSum in Thread B by value X+Y which is 50 for 5 times (200,150,100,50,0) then A thread died released the resource



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'Task2' with a package 'assignment' containing a 'Driver' class. The main editor shows the source code for 'Driver.java', which uses a Semaphore to control two threads, A and B. The console output at the bottom shows the execution flow: Thread A starts and waits for a permit, then increments its cumulative sum (CumSum) by 50 five times (50, 100, 150, 200, 250) before releasing the permit. Thread B then starts, waits for a permit, and decrements its CumSum by 50 five times (200, 150, 100, 50, 0) before releasing the permit. The final output is '0'.

```
1 package assignment;
2 import java.util.concurrent.*;
3
4 public class Driver
5 {
6     public static void main(String args[]) throws InterruptedException
7     {
8         // creating a Semaphore object with number of permits 1
9         Semaphore sem = new Semaphore(1);
```

Starting A
A is waiting for a permit.
Starting B
B is waiting for a permit.
A gets a permit.
thread A 50
thread A 100
thread A 150
thread A 200
thread A 250
A releases the permit.
B gets a permit.
thread B 200
thread B 150
thread B 100
thread B 50
thread B 0
B releases the permit.
0