- **Semaphore Program:**

```
package Meenu_a4;

import java.util.*;
import java.util.concurrent.*;

//A shared resource/class.

class SharedResource
{
static int count = 0;
}

class MyThread extends Thread
{

Semaphore semaphore;

String threadName;

public MyThread(Semaphore semaphore, String threadName)
{
```

```
super(threadName);
this.semaphore = semaphore;
this.threadName = threadName;
}

public void run()
{
if (this.getName().equals("Thread1"))
{
        System.out.println("Starting " + threadName);

        try
        {
                // First, get a permit.
                System.out.println(threadName + " is waiting for a permit.");

                //Acquiring the lock
                semaphore.acquire();

                System.out.println(threadName + " gets a permit.");

                // Accessing the shared resource.
                // other waiting threads will wait, until this thread releases the lock
                for (int i = 0; i < 5; i++)
                {
                        SharedResource.count++;
```

```
                        System.out.println(threadName + ": " + SharedResource.count);

                        //Allowing a context switch if possible.for thread B to execute
                        Thread.sleep(10);
        }
}

catch(InterruptedException exc)
{
        System.out.println(exc);
}

// Release the permit.

System.out.println(threadName + " releases the permit.");
semaphore.release();
}

// Run by thread B
else
{
System.out.println("Starting " + threadName);

try
{
        // First, get a permit.
```

```
            System.out.println(threadName + " is waiting for a permit.");

            // acquiring the lock
            semaphore.acquire();

            System.out.println(threadName + " gets a permit.");

            // Now, accessing the shared resource.
            // other waiting threads will wait, until this thread release the lock
            for (int i = 0; i < 5; i++)
            {
                  SharedResource.count--;

                  System.out.println(threadName + ": " + SharedResource.count);

                  Thread.sleep(10);
            }
}

catch(InterruptedException exc)
{
            System.out.println(exc);
}
// Release the permit.

System.out.println(threadName + " releases the permit.");
```

```
semaphore.release();
}
}
}

//Main class

public class Meenu_a4
{
public static void main(String args[]) throws InterruptedException
{
        // Creating a Semaphore object with number of permits = 1
        Semaphore semaphore = new Semaphore(1);

        // Creating two threads with name t1 and t2
        // Note that thread A will increment the count and thread B will decrement the
count

        MyThread t1 = new MyThread(semaphore, "Thread1");

        MyThread t2 = new MyThread(semaphore, "Thread2");

        t1.start();

        t2.start();
        // waiting for threads t1 and t2
```

```
        t1.join();

        t2.join();

        //count will always be 0 after both threads complete their execution

        System.out.println("count: " + SharedResource.count);
}
}
```

- **OUTPUT:**

```
<terminated> Meenu_a4 [Java Application] C:\Program Files\Java\jdk1.8.0_261\bin\javaw.exe  (01-Nov-2022, 10:27:11 pm – 10:27:11 pm)
Starting Thread2
Starting Thread1
Thread2 is waiting for a permit.
Thread1 is waiting for a permit.
Thread2 gets a permit.
Thread2: -1
Thread2: -2
Thread2: -3
Thread2: -4
Thread2: -5
Thread2 releases the permit.
Thread1 gets a permit.
Thread1: -4
Thread1: -3
Thread1: -2
Thread1: -1
Thread1: 0
Thread1 releases the permit.
count: 0
```

- **Mutex Program:**

```
package Meenu_a4;

import java.lang.*;

public class Meenu_a4
{
public final static int NUMTHREADS = 3;

public static int sharedData = 0;

public static int sharedData2 = 0;

/* Any real java object or array would suit for
synchronization */
/* We invent one here since we have two unique data items to
synchronize */
/* An in this simple example, they're not in an object */

static class theLock extends Object
{
}

static public theLock lockObject = new theLock();
```

```
static class theThread extends Thread
{
	public void run()
	{
		System.out.print("Thread " + getName() + ": Entered\n");

		synchronized (lockObject)
		{
			/********** Critical Section ******************/

			System.out.print("Thread " + getName() +": Start critical section, in synchronized block\n");

			++sharedData;

			--sharedData2;

			System.out.print("Thread " + getName() +": End critical section, leave synchronized block\n");

			/********** Critical Section ******************/
		}
	}
}

public static void main(String argv[])
{
```

```
theThread threads[] = new theThread[NUMTHREADS];

System.out.print("Entered the testcase\n");

System.out.print("Synchronize to prevent access to shared data\n");

synchronized (lockObject)
{
        System.out.print("Create/start the thread\n");

        for (int i=0; i<NUMTHREADS; ++i)
        {
                threads[i] = new theThread();

                threads[i].start();
        }

        System.out.print("Wait a bit until we're 'done' with the shared data\n");

        try
        {
                Thread.sleep(3000);
        }

        catch(InterruptedException e)
        {
```

```
                    System.out.print("sleep interrupted\n");
                }
                System.out.print("Unlock shared data\n");
            }

            System.out.print("Wait for the threads to complete\n");

            try {for(int i=0; i <NUMTHREADS; ++i)
            {
                threads[i].join();

                System.out.print("Testcase completed\n");

                System.exit(0);
            }
        }
            catch (InterruptedException e)
            {
                System.out.print("Join interrupted\n");
            }
        }
        }
```

- **OUTPUT:**

```
<terminated> Meenu_a4 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_261\bin\javaw.exe  (01-Nov-2022, 10:24:53 pm – 10:24:56 pm)
Entered the testcase
Synchronize to prevent access to shared data
Create/start the thread
Wait a bit until we're 'done' with the shared data
Thread Thread-0: Entered
Thread Thread-2: Entered
Thread Thread-1: Entered
Unlock shared data
Wait for the threads to complete
Thread Thread-1: Start critical section, in synchronized block
Thread Thread-1: End critical section, leave synchronized block
Thread Thread-0: Start critical section, in synchronized block
Thread Thread-0: End critical section, leave synchronized block
Testcase completed
Thread Thread-2: Start critical section, in synchronized block
Thread Thread-2: End critical section, leave synchronized block
```