Synchronisation

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

**Why use Synchronization?**

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

**There are two types of synchronization**

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

**Thread Synchronization**

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

**Mutual Exclusive**

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

**Concept of Lock in Java**

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

**Java Synchronized Method**

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

**Synchronized Block in Java**

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.

If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.

**Points to Remember**

* Synchronized block is used to lock an object for any shared resource.
* Scope of synchronized block is smaller than the method.
* A Java synchronized block doesn't allow more than one JVM, to provide access control to a shared resource.
* The system performance may degrade because of the slower working of synchronized keyword.
* Java synchronized block is more efficient than Java synchronized method.

**Static Synchronization**

If you make any static method as synchronized, the lock will be on the class not on object.

Diagram

Description automatically generated

**Problem without static synchronization**

Suppose there are two objects of a shared class (e.g. Table) named object1 and object2. In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock. But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock. We don't want interference between t1 and t3 or t2 and t4. Static synchronization solves this problem.

**Deadlock in Java**

Deadlock in Java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock

**Inter-thread Communication in Java**

**Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

* wait()
* notify()
* notifyAll()

**1) wait() method**

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

**2) notify() method**

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

**3) notifyAll() method**

Wakes up all threads that are waiting on this object's monitor.

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| The wait() method releases the lock. | The sleep() method doesn't release the lock. |
| It is a method of Object class | It is a method of Thread class |
| It is the non-static method | It is the static method |
| It should be notified by notify() or notifyAll() methods | After the specified amount of time, sleep is completed. |

**Interrupting a Thread:**

If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException. If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behaviour and doesn't interrupt the thread but sets the interrupt flag to true. Let's first see the methods provided by the Thread class for thread interruption.

**The 3 methods provided by the Thread class for interrupting a thread**

* **public void interrupt()**
* **public static boolean interrupted()**
* **public boolean isInterrupted()**