Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 4

MULTITHREADING AND GENERIC PROGRAMMING

4.4 SYNCHRONIZING THREADS

COMPUTER SCIENCE & ENGINEERING

SYNCHRONIZATION

Synchronization is the capability to control the access of multiple threads to any shared resource. It is better option where we want to allow only one thread to access the shared resource.

Use of Synchronization

The synchronization is mainly used to

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

TYPES OF SYNCHRONIZATION

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Sairam

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)

MUTUAL EXCLUSIVE

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by 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.

RULES FOR USING SYNCHRONIZED KEYWORD

1. **Synchronized keyword in Java** is used to provide mutually exclusive access to a shared resource with multiple threads in Java. Synchronization in Java guarantees that no two threads can execute a synchronized method which requires the same lock simultaneously or concurrently.

2. You can use java synchronized keyword only on synchronized method or synchronized block.

3. Whenever a thread enters into java synchronized method or blocks it **acquires a lock** and whenever it leaves java synchronized method or block it releases the lock. The lock is released even if thread leaves synchronized method after completion or due to any Error or Exception.

4. Java Thread acquires an **object level lock** when it enters into an instance synchronized java method and acquires a class level lock when it enters into static synchronized java method.

5. **Java synchronized keyword is re-entrant in nature** it means if a java synchronized method calls another synchronized method which requires the same lock then the current thread which is holding lock can enter into that method without acquiring the lock.

RULES FOR USING SYNCHRONIZED KEYWORD

6. Java Synchronization will throw NullPointerException if object used in java synchronized block is null e.g. synchronized (myInstance) will throw java.lang.NullPointerException if myInstance is null.

7. One Major **disadvantage of Java synchronized keyword** is that it doesn't allow concurrent read, which can potentially limit scalability. By using the concept of lock stripping and using different locks for reading and writing, you can overcome this limitation of synchronized in Java. You will be glad to know that java.util.concurrent.locks.ReentrantReadWriteLock provides ready-made implementation of ReadWriteLock in Java.

8. One more **limitation of java synchronized keyword** is that it can only be used to control access to a shared object within the same JVM. If you have more than one JVM and need to synchronize access to a shared file system or database, the Java synchronized keyword is not at all sufficient. You need to implement a kind of global lock for that.

RULES FOR USING SYNCHRONIZED KEYWORD

9. **Java synchronized keyword incurs a performance cost.** A synchronized method in Java is very slow and can degrade performance. So use synchronization in java when it absolutely requires and consider using java synchronized block for synchronizing critical section only.

10. **Java synchronized block is better than java synchronized method** in Java because by using synchronized block you can only lock critical section of code and avoid locking the whole method which can possibly degrade performance. A good example of java synchronization around this concept is getting Instance() method Singleton class.

11. It's possible that both static synchronized and non-static synchronized method can run simultaneously or concurrently because they lock on the different object.

RULES FOR USING SYNCHRONIZED KEYWORD

12. From java 5 after a change in Java memory model **reads and writes are atomic for all variables declared using the volatile keyword** (including long and double variables) and simple atomic variable access is more efficient instead of accessing these variables via synchronized java code. But it requires more care and attention from the programmer to avoid memory consistency errors.

13. **Java synchronized code could result in deadlock or starvation** while accessing by multiple threads if synchronization is not implemented correctly. To know how to avoid deadlock in java see here.

14. According to the Java language specification **you can not use Java synchronized keyword with constructor** it's illegal and result in compilation error. So you can not synchronize constructor in Java which seems logical because other threads cannot see the object being created until the thread creating it has finished it.

15. **You cannot apply java synchronized keyword with variables** and can not use java volatile keyword with the method.

16. `Java.util.concurrent.locks` extends capability provided by java synchronized keyword for writing more sophisticated programs since they offer more capabilities e.g. **Reentrancy** and **interruptible locks**.

RULES FOR USING SYNCHRONIZED KEYWORD

17. Java synchronized keyword also synchronizes memory. In fact, java synchronized synchronizes the whole of thread memory with main memory.

18. Important method related to synchronization in Java are wait(), notify() and notifyAll() which is defined in Object class. Do you know, why they are defined in java.lang.object class instead of java.lang.Thread? You can find some reasons, which make sense.

19. **Do not synchronize on the non-final field on synchronized block in Java**. because the reference of the non-final field may change anytime and then different thread might synchronizing on different objects i.e. no synchronization at all. an example of synchronizing on the non-final field:

```
private String lock = new String("lock");
synchronized(lock){
    System.out.println("locking on :"  + lock);
}
```

Video Link

https://www.youtube.com/watch?v=_-nL6vljqzI