**Multithreading**

1. What is multithreading, multitasking?
2. How many ways we can start a thread?
   1. Thread scheduler
   2. Difference between start() and run()
   3. Importance of start()
      1. Register Thread with Thread Scheduler
      2. All other mandatory low level activities.
      3. Invoke or calling run() method.
   4. What if not overriding run()
   5. Override start()
   6. Overloading run()
3. Life cycle of thread
   1. **New/born**: when we create a thread object, MyThread t=new MyThreaad();
   2. **Ready/runnable**: when we call t.start()
   3. **Running**: when Thread scheduler allocates CPU
   4. **Dead**: when run() execution completes
4. Can we start thread twice?
5. Thread priority
6. Getting setting thread name
7. Methods to prevent thread execution **yield(), join(), sleep()**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Yield()** | **Join()** | **Sleep()** |
|  | To pause current executing thread, and give chance to other thread of same priority. | If a Thread wants to wait until completing some other Thread then we should go for join. | If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep |
| Static method | Yes | No | Yes |
| Final | No | Yes | No |
| Overloaded | No | Yes | yes |
| Throws InterruptedException? | No | Yes | yes |

**Synchronization**

* Applicable for method and block, not to class and variable.

Object level lock:

Every java object has a lock. If thread wants to execute any synchronized method on object. Then thread should get lock of that object. Once it got the lock, then that thread can execute any synchronized method and other waiting threads are not allowed to execute any synchronized method but they are allowed to execute non-synchronized method of that class. Once the execution is completed, then thread releases the lock of that object.

Class level lock:

Every class in java has a unique lock. If a Thread wants to execute a static synchronized method then it required class level lock. Once a Thread got class level lock then it is allow executing any static synchronized method of that class. While a Thread executing any static synchronized method the remaining Threads are not allow to execute any static synchronized method of that class simultaneously. But remaining Threads are allowed to execute normal synchronized methods, normal static methods, and normal instance methods simultaneously.

**Getting lock of various object:**

Current object lock:

Synchronized(this){}

Specific object lock for employee object

Synchronized(employee){}

Class level lock for Employee

Synchronized(Employee.class){}

**Inter thread communication**

* We use wait(), notify(), notifyAll() method to communicate within thread.
* To call above method, thread should get lock of the object firstly.
* Above method must be called in synchronized areas only.
* Hence we can call wait(), notify() and notifyAll() methods only from synchronized area otherwise we will get runtime exception saying ***IllegalMonitorStateException***.
* Once a Thread calls **wait()** method on the given object 1st it releases the lock of that object immediately and entered into waiting state.
* Once a Thread calls **notify()** (or) **notifyAll()** methods it releases the lock of that object but may not immediately.