**Multithreading**

**Multitasking:**

It is based on two types:

1. Process-Based

2. Thread-Based

1. Process-Based: Like we’re working on ms-office while listening to music. But from ms-office we can only work on any one doc such as ms word and out of many songs, we can only listen only one song. It is called Multitasking.

It has multiple separate programmes.   
It has dependency.  
It is slower than thread based.

1. Thread-Based: Like, a programme takes 20 min to complete. We can divide this programme into different part as threads. These threads are known for independent execution.

So, now, each thread would take 10 min. Since thread is known for independent execution, so the programme will execute completely in 10 mins only. It’s called **Multithreading.**

It has one single programme.  
It doesn’t have any dependency.  
It is much faster than process based.

Definition:

Multithreading is a process to execute multiple threads at the same time without dependency of other threads called multithreading.

**Note:**   
- JVM never execute a programme directly but by using threads. Because There is always a single thread present in every java programme called ‘main’ thread, which we doesn’t need to start by using start() method. It is executed by JVM automatically because main thread is a part of JVM.  
  
- Nobody can guess the exact output.  
- We can’t throw exception using throws keyword from override methods.

- When we’ve multiple jobs then we use multithreading.

- We can’t start a thread twice. If we try to start a single thread twice, we’ll encounter with an exception called ‘IllegalThreadStateException’

**What is Thread?**Thread is a pre-defined class which is available in java.lang package.

Thread is a basic unit of CPU and it is well known for independent execution.

It has a separate path of execution.

There is always a thread in any java programme called main thread which is used by JVM to call main method.

**How to create Thread in Java?**

1. By **extending Thread** class.  
   Here, Thread will be the super class having a predefined overridden method called run().  
   To, start this run method, we’ve a predefined method called start(). Start() start the thread and internally execute the run method.   
     
   Although, We can also pass the object to an extended Thread class as an argument to the constructor of pre-defined Thread Class in extending Thread class method.
2. By **implementing** **Runnable** Interface (Best Approach).  
   It having a predefined overridden method called run().  
   Since, start() method is a pre-defined method in Thread class. Hence, we can only call this by the instance of Thread class. And to execute the run() method of runnable interface, we’ve to pass the object reference of that class implementing runnable interface as argument to the constructor of Thread class.

**Multitasking Vs Multithreading**

Multitasking:   
It has more than one separate programme.  
Upcoming process is dependent to the previous process till it complete its execution. So, it has dependency.

Multithreading:  
It has only one programme.  
Processes are not dependent to each other. So, it has no dependency. Because here processes are completed using thread which execute separately at the same time.

Note: Completion of threads in multithreading is totally random.

**Thread Scheduler:**

Thread Scheduler is the part of JVM which executes multiple threads on a single processor randomly.

It is basically responsible for scheduling the threads.

It randomly picks the thread for execution.

Thread scheduler based on the following Scheduling algorithm:

1. FCFS (If execution of thread is happening line by line)
2. SJF (Short Job First)
3. Round Robin (Randomly pick the thread)

It is completely unknow which algorithm is going to pick by thread scheduler. Hence it’s not possible to guess the exact output in multithreading.

**Note:**If two threads have same priority, then it depends on Operating System which threads will going to executed.

**Thread Life Cycle:**

As we know a thread is well know for independent execution. During the life cycle a thread can move from different states.

1. New State (Born)
2. Runnable State (Ready)
3. Running State (Execution)
4. Waiting State (Blocked)
5. Dead State (Exit)

Let say ‘t’ is a thread.

1. So, we’ve to used t.start() to start a thread and move to the born/new state.
2. In ready state, thread is ready for the execution by wait for its selection by Thread Scheduler. As Thread Scheculer select thread form Ready/runnable state it moves to the running state.
3. In Execution/running state, thread is under execution process means executing run() method. After complete execution of run() method it transfer to the exit state.
4. If tread is partially, executed and we want in between transfer to the dead state, we’ve to use t.stop() method.
5. Any thread in blocked/waiting state came from running state by using any of these thread method: t.suspend() or t.join() or t.sleep() or t.wait() method.
6. After entering to the blocked state, thread will then transfer back to the ready state. In case of t.suspend() method we use t.resume() method to transfer back to the ready state. All the rest methods will automatically transfer it back to the ready state.

Note: Each and every thread in java is executed by JVM.

**Sleept() Method:**

Sleep is a static method of thread class which throws checked exception i.e., Interrupted Exception (Because it move thread from running to waiting state which means this is a kind of interruption in the normal flow of thread cycle).

The main purpose of sleep method to put a thread into temporary waiting state.

**Note:**

Run() method is an overridden method. Hence, we cannot throw exception to overridden method using throws keyword.

**Join() Method:**

The main purpose of Join() method is to put the thread into temporary waiting state, if we want to execute complete any particular thread among all the treads available in the thread pool.

Join method also throw a checked exception i.e., Interrupted Exception.

**Thread Priority:**

In java it is possible to assign the priority of thread. To set these priority java thread class has provided two predefined methods.

1. setPriority(): Set Priority to the thread
2. getPriority(): Get Priority from the thread as number.

The thread class has also provided Three pre-defined final static variable and its value lie between 1 to 10.

Thread.MIN\_PRIORITY 🡪 1 (Minimum value)

Thread.NORM\_PRIORITY 🡪 5 (Default Priority value set by JVM)

Thread.MAX\_PRIORITY 🡪 10 ( Maximum value)

Highest Priority Thread would execute first.

We will encounter with IllegalArgumentException if we cross the Priority limit.

**Synchronization**

Synchronization is a technique through which we can control multiple threads or among the number of threads only one thread will enter inside the synchronized area.

**Note:-**

The main purpose of Synchronization is to overcome the problem of multithreading, when multiple threads are trying to access the same resource at same time on that situation it may provide some wrong result.

Synchronization is broadly classified into two categories:-

1. Method Level Synchronization
2. Block Level Synchronization (Recommended)
3. Method Level Synchronization:

When multiple thread trying to enter into the method, then because of synchronized keyword associated with the method will stop all the threads out of the method and allow only one method to enter into the method. And doesn’t allow next method to enter till the previous method doesn’t complete its job.

**Note:**Every object has a lock in java and this lock can be given to only one thread at all the time.   
Ex:  
Table obj = new Table  
Thread1 t1 = new Thread1();  
Thread t2 = new Thread2();

If a method is synchronized then only that thread will able to enter into the method having lock given by the object. As it complete its job, then that lock will take by another thread to enter into the synchronized method.

1. Block Level Synchronization: