**Introduction**

Most important topic after Collection for interview point of view.

**Multitasking**

* **Process based**
* **Thread Based**
  + Best suitable at programmatic level.
  + Context Switching is very cheap/easy.
  + Share Address spaces.
* **Objective of Multitasking**
  + To increase performance of system
  + To reduce processor ideal time of processor
* **The main important application areas of multithreading are:**
  + To develop multimedia graphics
  + To develop animations
  + To develop video games
  + To develop web servers & application servers(Tomcat provide support simultaneously for 60 threads by default)
  + A program to search SCJP in computer HardDisk memory.
* In Java, most of the work (90%) is done by the API only. (Thread, Runnable, ThreadGroup etc.)

**The ways to define a thread**

* By extending Thread class
* By implementing Runnable interface
* Which approach is better?

**By extending Thread class**

Thread Life Cycle

* new/born
* ready/runnable
* running
* dead/terminated

Standard Thread states

* [NEW](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#NEW)  
  A thread that has not yet started is in this state.
* [RUNNABLE](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#RUNNABLE)  
  A thread executing in the Java virtual machine is in this state.
* [BLOCKED](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#BLOCKED)  
  A thread that is blocked waiting for a monitor lock is in this state.
* [WAITING](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#WAITING)  
  A thread that is waiting indefinitely for another thread to perform a particular action is in this state.
* [TIMED\_WAITING](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#TIMED_WAITING)  
  A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
* [TERMINATED](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#TERMINATED)  
  A thread that has exited is in this state.

**Thread Schedular**

**Diff b/w output of t.start() & t.run()**

**t.start() & t.start()**

**By implementing Runnable interface**

**Getting and setting name of a thread**

Public final String getName();

Public final void setName();

**Getting current executing thread**

Thread.currentThread()- returns current executing thread

Thread class constructors

Thread t=new Thread();

Thread t=new Thread(Runnable r);

Thread t=new Thread(String name);

Thread t=new Thread(Runnable r, String name);

**Thread priorities**

**Default priority**

**Valid Range for priorities**

**Thread constants**

**Setting & getting priorities**

Public final int getPriority();

Public final void setPriority(int P);//valid range:1-10

**Methods to prevent thread execution-**

* Yield()
  + method causes to pause current executing thread to give the chance for remaining waiting threads of same priority.
  + public static native void yield();
* join
  + If a thread(t1) wants to wait until completing another thread(t2).
  + Then inside t1, t2.join();
  + Now t1 will be entered into waiting state, until t2 completes.
  + Now when t2 will complete, t1 can continue its execution.
  + Running state🡪t2.join🡪waiting state(thread got blocked for joining)🡪X🡪Ready/Runnable State Where X is:
    - 1)If t2 completes
    - 2)If time expires
    - 3)If waiting thread(t1 ) get interrupted
  + public final void join() throws InterruptedException;

public final void join(long ml) InterruptedException;

public final void join(long ml, int ns) InterruptedException;

* sleep
  + Sleeping makes man very poor.-Bible
  + Running state🡪X🡪Sleping state🡪Y🡪Ready/Runnable Sate Where
  + X- Thread.sleep(2000); or Thread.sleep(2000,234);
  + Y-If time expires or if sleeping thread got interrupted
  + public static native void sleep(long ms) throws InterruptedException;
  + public static void sleep(long ms, int ns) throws InterruptedException;
* **How a thread can interrupt another thread**
  + A thread can interrupt a sleeping thread or waiting thread by using interrupt() method of Thread class.
  + t.interrupt();
  + Whenever we are calling interrupt method, if the target thread is not in sleeping /waiting state ,then there is no immediate impact of interrupt call. Interupt call will be waited until, target thread entered into sleeping or waiting state.

If the target thread entered into sleeping or waiting state then immediately interrupt call will interrupt, the target thread.

**Comparison among yield(), join() & sleep()**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Yield() | Join() | Sleep() |
| Purpose | - | - | - |
| It is overloaded or not | No | Yes | Yes |
| Is it final? | No | Yes | No |
| Is it throws InterruptedException | No | Yes | Yes |
| Is it native or not? | Yes | No | Sleep(long ms)-native  Sleep(long ms, int ns)-not native |
| Is it static | Yes | No | Yes |

**Synchronization**

Advantage:Data inconsistency problem

Disadvantage: increases waiting time of threads and creates performance problems

Lock

Synchronized area

Non-Synchronized area

Normal Synchronization

Static synchronization

Synchronized method

Normal Synchronization

Static synchronization

Synchronized block

Normal Synchronization

On current object(this)

On other than current object

Static synchronization

**Interthread communication**

* Wait()
* Notify()
* Notify A ll()

**Deadlock**

A lock without key

**Deadlock vs Starvation**

**Deamonn thread**

The thread which is executing in the background.

Artists-makeupman

Usually daemon threads having low priority but based on requirement, daemon threads can run with high priority also.

**public boolean isDaemon()**

**public void setDaemon(boolean b)**

**Multithreading inhancement**

* + Lock concepts
  + Reentrant Lock
  + ThreadGroup
  + Thread Local
  + Executor Framework