Multi-Threading in Java

Multitasking:

- Multitasking is a process of performing multiple tasks simultaneously using single processor.
- We use multitasking to optimize the utilization of CPU.
- Multitasking can be achieved by two ways:
    - Process-based Multitasking(Multiprocessing)
    - Thread-based Multitasking(Multithreading)

Process-based Multitasking (Multiprocessing):

- Each process have its very own location in memory for example each process designates separate memory zone
- Process is heavy weight.
- Cost of communication between the process is high.
- Switching from one process to another (Context-Switching) consumes lot of time.

Thread-based Multitasking (Multithreading):

- Threads share the same address space.
- Thread is lightweight, a smallest unit of processing.
- Cost of communication between the thread is low.
- They don't allocate separate memory area so context-switching between the threads takes less time than processes.

  Note:
- At least one process is required for each thread.
- Multithreading is mostly used in games, animation etc.

How to create thread ?

- There are two ways to create a thread:
  - ➢ By extending Thread class
  - ➢ By implementing Runnable interface

Thread class:

- Thread class is the sub class of 'Object' class and it implements Runnable interface (by default).
- Thread class will be having constructors and methods to perform operations on thread.
- When a class is extending the Thread class, it overrides the run() method from the Thread class to define the code executed by the thread.

Runnable interface:

- Runnable interface will have only one method named run().
- It is mostly recommended to use when creating thread.
- **public void run():** is used to perform action for a thread.

Steps for creating a thread

1) Write a class that extends Thread class or implements Runnable interface this is available in lang package.
2) Write public void run () method in that class, this is the method by default executed by any thread.
3) Create an object to that class (Inside main()).
4) Create a Thread Class Object and attach it to your class object.
5) Start running the thread.

Creating Thread by implementing Runnable interface

```
public class ClassA implements Runnable
{
    public void run()
      {
         for(int i=0;i<5;i++)
         System.out.println("Run method");
      }
    public static void main(String[] args)
     {
       ClassA a=new ClassA();
       Thread t1=new Thread(a);
       Thread t2=new Thread();
        System.out.println("Java is awesome");
     }
}
```

```
t1.start();
t1.run();
t2.start();
t2.run();
```

t1.start()

New Thread will be generated which is responsible for the execution of **ClassA** run() method.

t1.run()

No new Thread will be generated but **ClassA** run() method will be called just like a normal method call.

t2.start()

A new Thread will be generated which is responsible for the implementation of **Thread class** run()method

t2.run()

No new Thread will be generated but **Thread class** run() method will be called just like a normal method call.

Creating Thread by extending Thread class

```
public class ClassA extends Thread
        {
                public void run()
                {
                        for(int i=0;i<5;i++)
                        System.out.println("Run method");
                }
                public static void main(String[] args)
                {
                        ClassA a=new ClassA();
                        a.start();
                        System.out.println("Java is awesome");
                }
        }
```

Life Cycle of a Thread

New	Thread is created but not yet started.
Runnable	A thread in the Runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as processor
Blocked	A thread in the blocked state is waiting to enter a synchronized block/method or reenter a synchronized block/method.
Waiting	A thread will be in waiting state for a unspecified period of time, due to calling one of the methods like **wait(),join()** etc
Timed_waiting	A thread will be in waiting state for another thread for a specified waiting time is in this state
Terminated	The thread has completed execution

A thread can be in only one state at a given point in time. **Thread.getState()**