Thread : - A thread is said to be a single path of execution within a program. It is a part of a program. So it can be said that if a program is having more than one parts that can run independently of each other, the program is having multiple threads.

The thread Life Cycle

The Start () method :

The start () method puts a particular thread into the ready state or makes the thread eligible to run. So if there is no other threads that is running or the current thread is a higher priority thread, then this thread will be executed. But if it is not so, the current thread may have to wait for some period of time before it actually starts running. So calling the start method on a thread doesn't mean that the thread will start running. It depends on the scheduling algorithm that the thread scheduler follows. The start () method invokes the run () method that we are going to discuss next.

The signature of the start () method is:

```
void start ()
```

The run () Method:

This defines the code that constitutes the new thread. This method can call other methods, use other classes and declare variables, just like the main thread can. The only difference is that run () establishes the entry point for another concurrent thread of execution within your program. The new thread ends when the run () returns.

The signature of the run () method is:

```
public void run ()
```

The isAlive () Method:

The method isAlive () when called on a thread determines whether the thread has finished its execution or not. If the thread on which it was called has finished, this method returns false. Otherwise it returns true.

Signature of the isAlive () method is:

```
final boolean isAlive()
```

The stop () method:

This method is used to terminate the thread explicitly. Once terminated, the thread cannot be resumed once again.

==========

Thread States

- **Running State**
- **Waiting State**
  - **Suspended State**
  - **Sleeping State**
- **Ready State**
- **Dead State**

Running state:

A thread is said to be in the running state when the CPU is executing it. It means that the code attached with the thread is being executed.

Waiting state:

A thread is said to be in the waiting state if it was given a chance of execution but it did not complete its execution. It may choose to go into sleep. The other possibility may be that some other thread suspends the thread. Suspending the execution of a thread is now deprecated in the new version.

Ready state:

When the thread is waiting for the CPU to execute it.

Dead state:

When the thread has finished its execution.

==========
==========

Other methods of the Thread class

sleep () :-The **sleep** () puts the currently active thread to sleep for the specified number of milliseconds that is passed to it as an argument. This method is static and may be accessed through the Thread class name.

setName ():-Sets the name of the thread to that specified by **newname**.

getName () : -Returns a string that specifies the thread name.

join () **:-** This method is used more frequently than the **isAlive** () method to wait for a thread. It provides more functionality than the **isAlive** () method. This method waits for the called thread to terminate. So the calling thread has to wait till the called thread finishes and joins the calling thread. The name of the method comes from this concept that the called thread joins the calling thread and till this happens the calling thread has to wait for the called thread.

currentThread () **: -**This method returns a reference to the thread on which it is called. The main thread of a Java program is created as soon as the execution of the program is started. However, in order to control this thread with the help of the Thread object, we have to acquire a reference to it. This can be achieved with the **currentThread** () method. This method returns a reference to the thread on which it is called. Once a reference to the main thread is obtained with the help of the said method, it can be controlled like any other thread.

There are two methods of creating a thread. Either they can extend from the Thread class or implement the Runnable interface. Both the Thread class and the Runnable interface are available in the package java.lang.

The thread scheduler decides when each thread should be allowed

to run using thread priorities. Usually higher-priority threads get more CPU time than lower-priority threads.

MIN_PRIORITY and **MAX_PRIORITY**. These values are **1 to 10**, respectively.To return a thread to default priority, specify **NORM_PRIORITY**, which is currently **5**.