Multithreaded Programming:

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

1. Extending the Thread class

We can make our class runnable as thread by extending the class **java.lang.Thread.** It includes the following steps:

i) Declare the class as extending the **Thread** class.

The Thread class can be extended as follows:

```
Class Mythread extends Thread
{
        …………
        …………
        …………
}
```

Now we have a new type of thread Mythread.

ii) Implement the **run()** method.
The **run()** method has been inherited by the class **MyThread.** The basic implementation of **run()** is as follows:

```
Public void run()
{
    ……………
    ……………// Thread code here
    ……………
}
```

When we start the new thread, java calls the thread's **run()** method.

iii) Create a thread object and call the **start()** method to initiate the thread execution.

This step is performed as follows:

```
MyThread a= new MyThread();
a.start();
```

The first line instantiates a new object of class MyThread. This statement creates the object. The thread is in a newborn state.
The second line calls the start() method causing the thread to move into the runnable state. Now, the thread is said to be in the running state.

2. Implementing the Runnable Interface

The **Runnable** interface declares the **run()** method that is required for implementing threads in our program. To do this, we perform the following steps:

i) Declare the class as implementing the **Runnable** interface.

```
Class Mythread implements Runnable
{
        …………
        …………
        …………
}
```

Now we have a new type of thread Mythread.

ii) Implement the **run()** method.

The basic implementation of **run()** is as follows:

```
Public void run()
{
    ……………
    ……………// Thread code here
    ……………
}
```

When we start the new thread, java calls the thread's **run()** method.

iii) Create a thread by defining an object that is instantiated from this "runnable" class as the target of the thread.

This step is performed as follows:

```
MyThread a= new MyThread();
```

iv) Call the thread's **start()** method to run the thread.

```
a.start();
```

isAlive() and join() method

Sometimes one thread needs to know when other thread is terminating. In java, **isAlive()** and **join()** are two different methods that are used to check whether a thread has finished its execution or not.

1. **isAlive() :** It tests if this thread is alive. A thread is alive if it has been started and has not yet died. There is a transitional period from when a thread is running to when a thread is not running. This method is used to find out if a thread has actually been started and has yet not terminated.

   General Syntax:
   final boolean isAlive( )

   Return Value: returns true if the thread upon which it is called is still running. It returns false otherwise.

2. **join():** This method waits until the thread on which it is called terminates. Using **join()** method, we tell our thread to wait until the specified thread completes its execution.
   General **Syntax:**
   final void join( ) throws InterruptedException