***Java - Multithreading***

Java provides built-in support for *multithreaded programming*. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

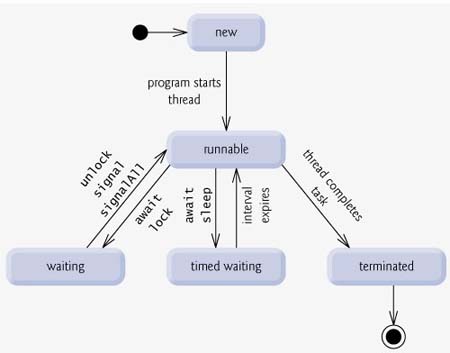
A multithreading is a specialized form of multitasking. Multitasking threads require less overhead than multitasking processes.

I need to define another term related to threads: **process:** A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

## Life Cycle of a Thread:

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



Above mentioned stages are explained here:

* **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
* **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
* **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
* **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
* **Terminated:**A runnable thread enters the terminated state when it completes its task or otherwise terminates.

## Thread Priorities:

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java priorities are in the range between MIN\_PRIORITY (a constant of 1) and MAX\_PRIORITY (a constant of 10). By default, every thread is given priority NORM\_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependentant.

## Creating a Thread:

Java defines two ways in which this can be accomplished:

* You can implement the Runnable interface.
* You can extend the Thread class, itself.

## Create Thread by Implementing Runnable:

The easiest way to create a thread is to create a class that implements the **Runnable** interface.

To implement Runnable, a class need only implement a single method called **run( )**, which is declared like this:

public void run( )

You will define the code that constitutes the new thread inside run() method. It is important to understand that run() can call other methods, use other classes, and declare variables, just like the main thread can.

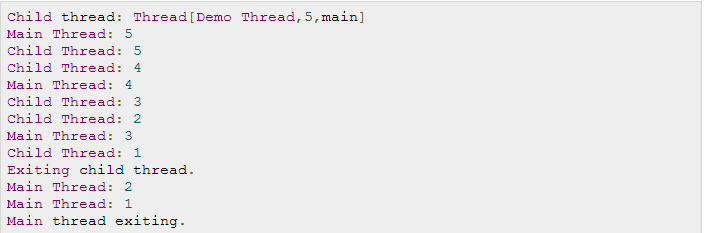
After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

Thread(Runnable threadOb, String threadName);

Here *threadOb* is an instance of a class that implements the Runnable interface and the name of the new thread is specified by *threadName*.

After the new thread is created, it will not start running until you call its **start( )** method, which is declared within Thread. The start( ) method is shown here:

void start( );



## Create Thread by Extending Thread:

The second way to create a thread is to create a new class that extends **Thread**, and then to create an instance of that class.

The extending class must override the **run( )** method, which is the entry point for the new thread. It must also call **start( )** to begin execution of the new thread.

