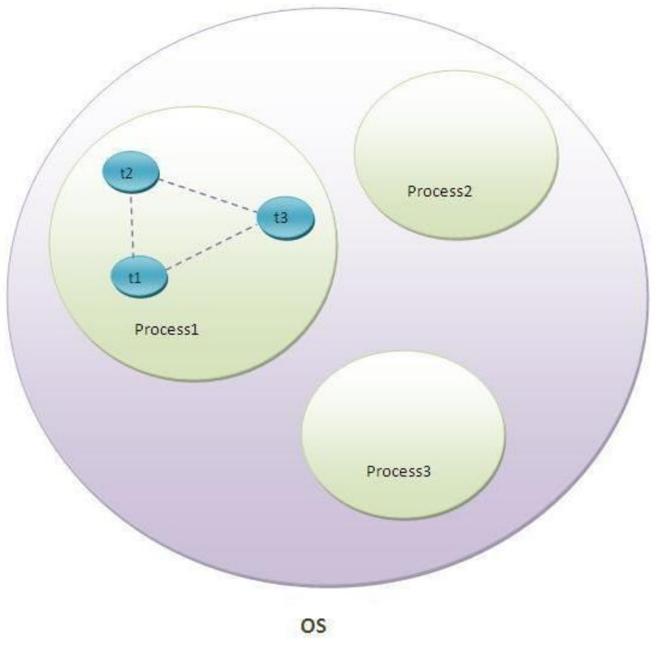
Thread in Java

❖ What is thread?

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



❖ What is multithreading?

Multithreading in java is a process of executing multiple threads simultaneously.

The aim of multithreading is to achieve the concurrent execution.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

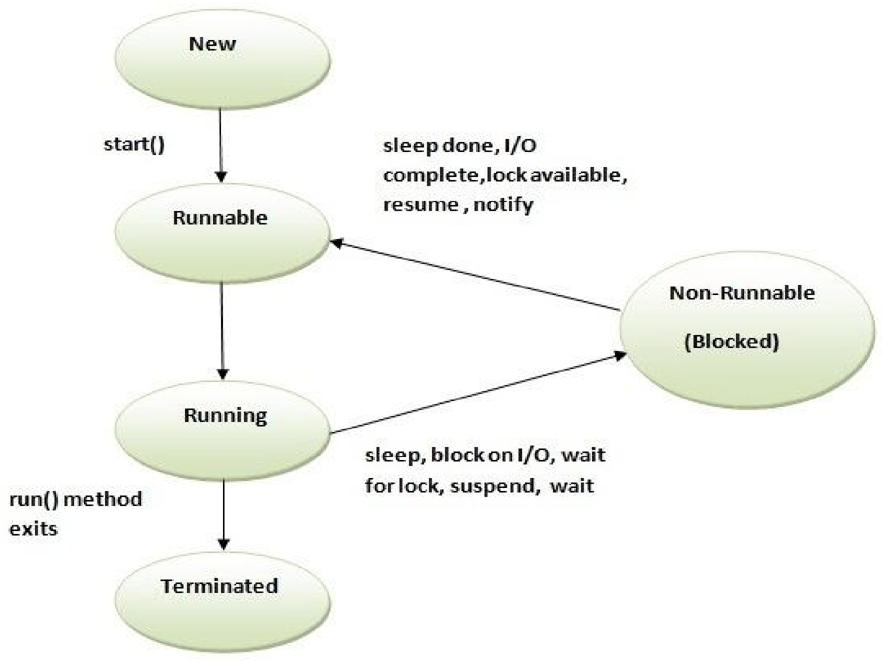
2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

❖ Life cycle of a Thread (Thread States) :-

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable(Ready)
3. Running
4. Non-Runnable (Blocked) (waiting)
5. Terminated


New
start()
sleep done, I/O complete, lock available, resume , notify
Runnable
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

New State :

If any new thread class is created that represent new state of a thread, In new state thread is created and about to enter into main memory. No memory is available if the thread is in new state.

Runnable (Ready) State :

In ready state thread will be entered into main memory, memory space is allocated for the thread and 1st time waiting for the CPU.

Running State :

Whenever the thread is under execution known as running state.

Non-Runnable (Blocked) :

This is the state when the thread is still alive, but is currently not eligible to run.

Terminated :

A thread is in terminated or dead state when its run() method exits.

❖ Define Thread :-

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

1) Using thread class :

Thread class provide constructors and methods to create and perform operations on a thread. Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Example :-

```
class Multi extends Thread
 {
   public void run()
   {
      System.out.println("Run method executed by child Thread");
    }
   public static void main(String args[])
  {
      Multi t1=new Multi();
      t1.start();
      System.out.println("Run method executed by main Thread");
   }
 }
```

Output :-

```
Run method executed by main thread

Run method executed by child Thread
```

Methods of Thread class

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.

11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

2) Using Runnable Interface :-

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Example:-

```
class Multi3 implements Runnable
{
public void run()
{
System.out.println("thread is running...");
}

public static void main(String args[])
{
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
```

❖ Interrupts:-

The **interrupt()** method of thread class is used to interrupt the thread. If any thread is in sleeping or waiting state then using the interrupt() method, we can interrupt the thread execution by throwing InterruptedException.

Syntax :

```
 public void interrupt()
```

Example :

```
public class Interrupt1 extends Thread
{
   public void run()
   {
      try
      {
         Thread.sleep(500);
         System.out.println("java");
      }
       catch(InterruptedException e)
       {
         System.out.println("Exception handled "+e);
      }
      System.out.println("thread is running...");
   }
   public static void main(String args[])
   {
      Interrupt1 t1=new Interrupt1();
       //Interrupt1 t2=new Interrupt1();
      // call run() method
      t1.start();
      // interrupt the thread
      t1.interrupt();
   }
}
```

Output :-

```
Exception handled java.lang.InterruptedException: sleep interrupted
thread is running...
```

❖ Join Thread :-

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

The join method allows one thread to wait for the completion of another. If t is a Thread object whose thread is currently executing,

t.join();

causes the current thread to pause execution until t's thread terminates

Example of thread with join() method

```
public class MyThread extends Thread
{
        public void run()
        {
                System.out.println("r1 ");
                try {
                Thread.sleep(500);
                }catch(InterruptedException ie){ }
                System.out.println("r2 ");
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();
                try{
                        t1.join();          //Waiting for t1 to finish
                }catch(InterruptedException ie){}

                t2.start();
        }
}
```

In this above program join() method on thread t1 ensures that t1 finishes it process before thread t2 starts.

❖ Stop Thread :-

There are two ways through which you can stop a thread in java. One is using volatile *boolean variable* flag and second one is using *interrupt()* method.

Initially we set this *flag* as *true*. Keep the task to be performed in *while* loop inside the *run()* method by passing this *flag*. This will make thread continue to run until *flag* becomes *false*. We have defined *stopRunning()* method. This method will set the *flag* as *false* and stops the thread. Whenever you want to stop the thread, just call this method.**Example: using volatile flag variable**

```
class MyThread extends Thread

{

    private volatile boolean flag = true;

     //This method will set flag as false

     public void stopRunning()
    {
        flag = false;
    }

    public void run()
    {
      while (flag)
        {
            System.out.println("I am running....");
            System.out.println("I am running....");
            System.out.println("I am running....");
            stopRunning();
        }
            System.out.println("Stopped Running....");
    }
}
 public class MainClass
{
    public static void main(String[] args)
    {
        MyThread thread = new MyThread();
       thread.start();
   }
```

}

❖ Thread Synchronization :-

Whenever multiple threads are trying to use same resource than they may be chance to of getting wrong output, to overcome this problem thread synchronization can be used.

Definition: Allowing only one thread at a time to utilized the same resource out of multiple threads is known as thread synchronization or thread safe.

In java language thread synchronization can be achieve in two different ways.

1. Synchronized block
2. Synchronized method

Synchronized block

Whenever we want to execute one or more than one statement by a single thread at a time(not allowing other thread until thread one execution is completed) than those statement should be placed in side synchronized block.

```
class  Class_Name  implement Runnable or extends Thread
{
public void run()
{
synchronized(this)
{
.......
.......
}
}
}
```

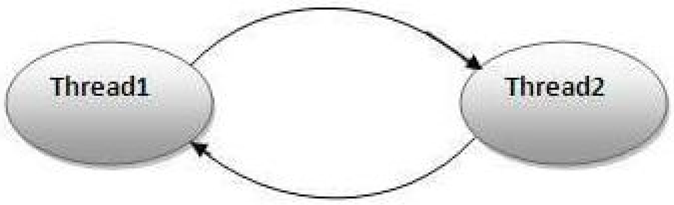
Synchronized method

Whenever we want to allow only one thread at a time among multiple thread for execution of a method than that should be declared as synchronized method.

```
class  Class_Name  implement Runnable or extends Thread
{
public void run()
{
synchronized void fun()
{
.......
.......
}
public void run()
{
fun();
....
}
}
```

❖ Deadlock in Java :-

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.


Thread1
Thread2