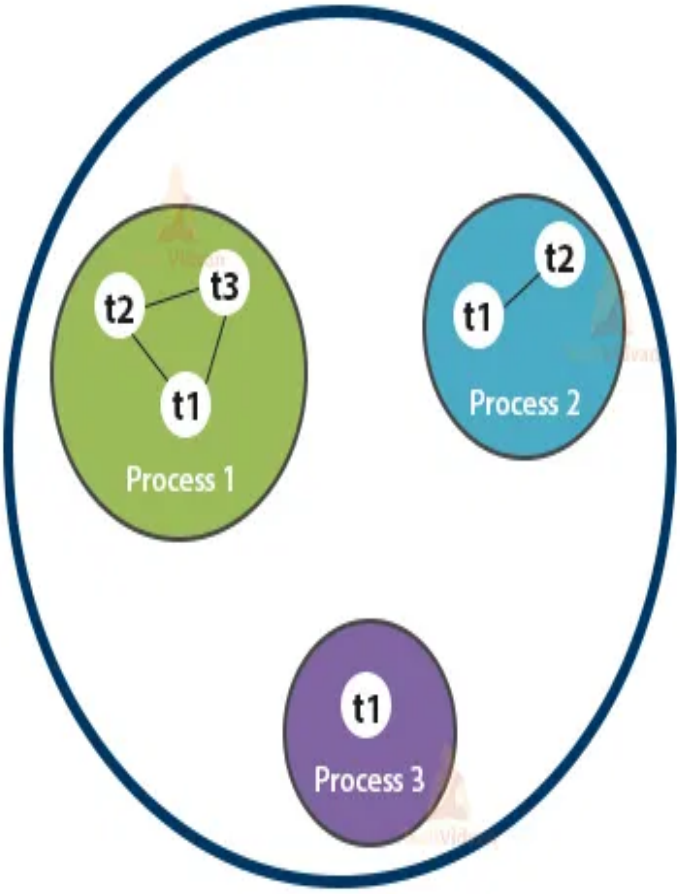
Multithreading in java

- It is a process of executing multiple threads simultaneously.
- Thread is basically a lightweight sub-process, a smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- Threads share a common memory area. They don't allocate separate memory area so saves memory
- context-switching between the threads takes less time than process.
- Java Multithreading is mostly used in games, animation etc.

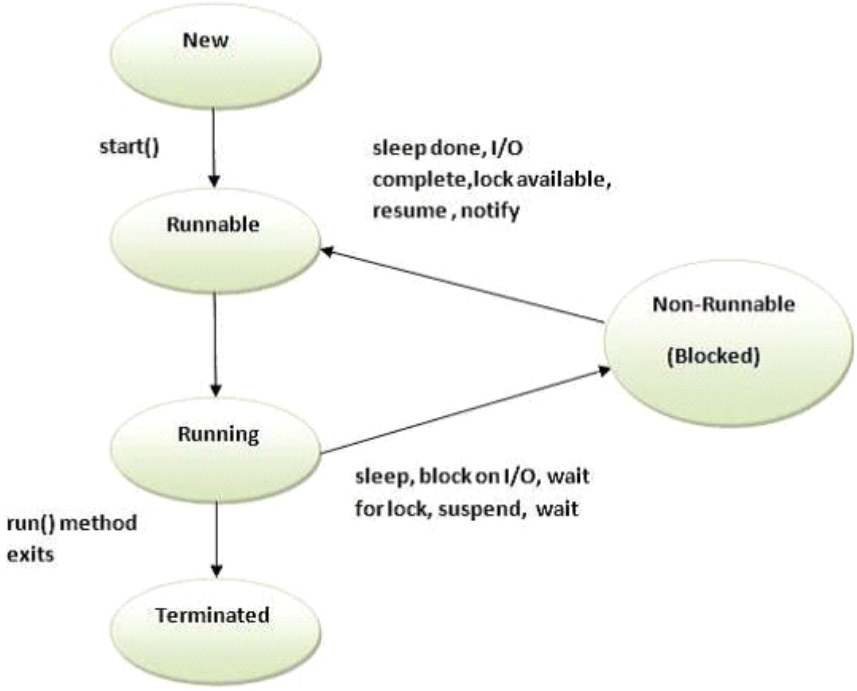


Life cycle of a Thread (Thread States)

- Every thread, while executing goes through its own life cycle.
- A thread can be in one of the five states during its life cycle. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

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

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


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

Life cycle of Thread.

A Thread can be in any of the following states.

1. Newborn state

When a thread object is created a new thread is born an

2. Runnable State

When a thread is ready for execution and waiting for the availability of processor. If all threads in the queue are of same priority, then they are given time slots for execution in round robin fashion

3. Running State

The processor has given its time to the thread for execution. A Thread keeps running until the following condition occures.

a) Thread give up its control on its own
   (i) A thread is made to sleep for a specified time period using

sleep (time) method

(ii) A thread is made to wait for some event to occure using wait() method On this case a thread can be scheduled to run again using notify() method

(iii) A thread gets suspended using suspend() method which can again be resume().

b) A thread is pre-empted by higher — priority thread.

- Blocked state

  If a thread is prevented from entering into runnable and subsequently running state then thread is in blocked state.

- Dead state

  When the thread completes its task.

Thread class
Runnable interface

Present in java.lang provides methods to create multi threaded program

Class Thread extends Object implements Runnable
{
}

How to create thread

There are two ways to create a thread:

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

Thread class:

- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:
Thread()
Thread(String name)
Thread(Runnable r)
Thread(Runnable r,String name)

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().

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

Thread Example by extending Thread class

```
class demothread extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
demothread t1=new demothread();
t1.start();
```

}}

Thread Example by implementing Runnable interface

```
class demothread1 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
demothread1 m1=new demothread1();
Thread t1 =new Thread(m1);
t1.start(); } }
```

Commonly used 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 static 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 void 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.StategetState():** 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

Starting a thread:

- **start() method** of Thread class is used to start a newly created thread.

It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- The thread is executed by run() method, when the state of the thread is changed from Runnable to Running state.

Priority of a Thread (Thread Priority):

- Each thread has a priority.
- Priorities are represented by a number between 1 and 10.
- In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

- 1. public static int MIN_PRIORITY
- 2. public static int NORM_PRIORITY
- 3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example

```
class TestPriority1 extends Thread{
 public void run(){
System.out.println("running thread name is:"+Thread.currentThread().getName());
System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
}
public static void main(String args[]){
TestPriority1 m1=new TestPriority1();
TestPriority1 m2=new TestPriority1();
m1.setPriority(Thread.MIN_PRIORITY);
m2.setPriority(Thread.MAX_PRIORITY);
m1.start();
m2.start();
} }
```

Join():

It is method of thread class, which allows a executing thread to die, In the queue , other thread has to wait till the completion of the thread , which has called the jain() method.

public void join() throws InterruptedException

```
class thread1 extends Thread
{
 public void run()
{
 for(int i=1;i<=10;i++)
 System.out.println("i is"+i);
}
}
class thread2 extends Thread
{
 public void run()
{
for(int i=1;i<=10;i++)
 System.out.println("i*i is"+(i*i));
}
}

class demo
{
 public static void main(String[] s)
 {
  thread1 t1=new thread1();
  thread2 t2=new thread2();
   t1.setPriority(1);
   t2.setPriority(10);
  t1.start();
 try
{
 t2.join();
}
catch(InterruptedException e)
{
}
  t2.start();
 System.out.println(t1.getPriority());
System.out.println(t2.getPriority());


}
```

```
}
```

Thread synchronization

It is process of locking the object of a sharable resource to be accessed by multiple thread simultaneously.

synchronization can be achieved in two ways.

- **By synchronized method :**
  If you declare any method as synchronized, it is known as synchronized method. Synchronized method is used to lock an object for any shared resource. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
synchronized returntype methodname(parameter)
{ }
```

- **By synchronized block :**

Syntax

```
synchronized {
        // codes for accessing sharable resource

        }
```

Example:

```
class customer{
int amount=10000;
synchronized void withdraw(int amount)
{ System.out.println("going to withdraw...");
if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}
catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
synchronized void deposit(int amount){ System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class th1 extends Thread
{
```

```
 customer c;
 th1(customer c)
 {
  this.c=c;
}
public void run()
{c.withdraw(15000);}
}
class th2 extends Thread
{
 customer c;
 th2(customer c)
 {
  this.c=c;
}
public void run(){
c.deposit(10000);
}
}
class Test{
public static void main(String args[])
{
final customer c=new customer();
th1 t1=new th1(c);
th2 t2=new th2(c);
t1.start();
t2.start();
}}
```

Another way of writing same program

```
class customer
{
int amount=10000;
synchronized void withdraw(int amount)
{ System.out.println("going to withdraw...");
if(this.amount<amount)
{
System.out.println("Less balance; waiting for deposit...");
try{wait();}
catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
```

```
synchronized void deposit(int amount)
{ System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class Test{
public static void main(String args[])
{
final customer c=new customer();
new Thread(){
public void run()
{c.withdraw(15000);}
}.start();
new Thread(){
public void run(){
c.deposit(10000);
}}.start();
}}
```