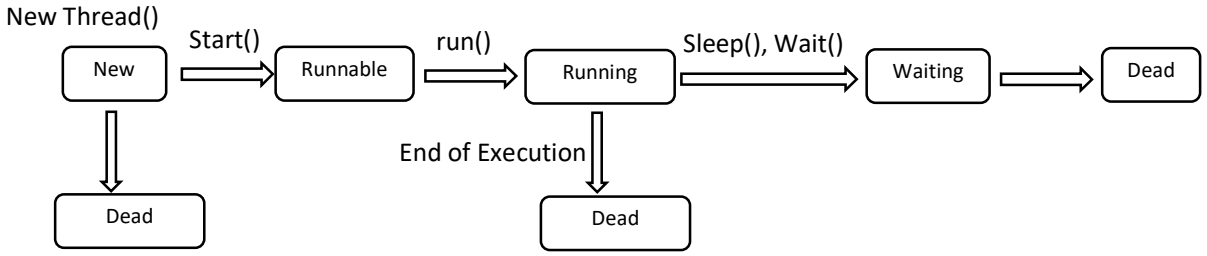
Multithreading: Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs. By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application. Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

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. The following diagram shows the complete life cycle of a thread.



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 (Dead): A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Creating a Thread: There are two ways to create a thread in java .

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

By extending Thread class: We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Start(): A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object**.**

For Example:

```
class duper extends Thread
{
public void run()
    {
inti=0;
while(i<5)
    {
System.out.println("java program -------");
i++;
}

    }
}
class duper2 extends Thread
{
```

```
public void run()
{
inti=0;
while(i<5)
    {
System.out.println("-------core java");
i++;
}
}
}
public class super1
{
public static void main(String[] args)
    {
duper d=new duper();
        duper2 d2=new duper2();
d.start();
d2.start();
System.out.println("progam is running mode");
    }
}
```

By implementing the Runnable Interface: We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

For Example:

```
Import java.io.IOException;
class duper implements Runnable
{
public void run()
    {
try{
inti=0;
while(i<5)
    {
System.out.println("java program -------");
```

```
i++;
}
        }
catch(Exception e)
        {
System.out.println("program is dead");
        }
    }
}
class duper2 implements Runnable
{
public void run()
{
try{
inti=0;
while(i<5)
    {
System.out.println("-------core java");
i++;
}
    }
catch(Exception e)
    {
System.out.println(e);
    }
}
}
public class super2
{
public static void main(String[] args)
    {
        // duper d=new duper();
        // duper2 d2=new duper2();
        Thread d=new Thread(new duper());
        Thread d2=new Thread(new duper2());
d.start();
```

```
d2.start();
System.out.println("progam is running mode");
    }
}
```

Create a Thread by Extending a Thread Class: The second way to create a thread is to create a new class that extends Thread class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in **Current Thread :**Note(Print the current thread)

For Example:

```
classth
{
public static void main(String[] args)
    {
        Thread t=Thread.currentThread();
System.out.println("Current thread = " +t);
t.setName("My thread");
System.out.println("After change my thread name = " +t);
    }
}
```

Thread class: Note(Multiple Thread create in different ways)

Step 1: You will need to override run( ) method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method.

Syntax : run() method

public void run( )

Step 2: Once Thread object is created, you can start it by calling start() method, which executes a call to run( ) method.

Syntax:start() method

void start( );

For Example:

```
class ThreadDemo extends Thread {
public Thread t;
```

```
public StringthreadName;

ThreadDemo( String name) {
threadName = name;
System.out.println("Creating " +  threadName + "come here" );
   }

public void run() {
System.out.println("Running " +  threadName + " ------------- " );
try {
for(inti = 4; i> 0; i--) {
System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
Thread.sleep(50);
         }
      } catch (InterruptedException e) {
System.out.println("Thread " +  threadName + " interrupted.");
      }
System.out.println("Thread " +  threadName + " exiting.");
   }

public void start () {
System.out.println("Starting " +  threadName+ " go back " );
if (t == null) {
         t = new Thread (this, threadName);
t.start ();
      }
   }
}

public class TestThread {

public static void main(String args[]) {
ThreadDemo T1 = new ThreadDemo( "Thread-1");
T1.start();
```

```
ThreadDemo T2 = new ThreadDemo( "Thread-2");
T2.start();
   }
}
```

Using the Thread Class: Note(In which program using Runnable interface)

Syntax:

Thread(Runnable r, String name)

For Example:

```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}

// main method
public static void main(String argvs[])
{
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r,
String name)
Thread th1 = new Thread(r1, "My new thread");

// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```

Sleep(): The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is known as the sleeping time of the thread. After the sleeping time is over, the thread starts its execution from where it has left.

For Example:

```
class Cd {
public static void main(String args[]) {
Thread t = Thread.currentThread();
System.out.println("Current thread: " + t);
// change the name of the thread
t.setName("My Thread");
System.out.println("After name change: " + t);
try {
for(int n = 15; n > 5; n--) {
System.out.println(n);
Thread.sleep(1000);
}
}
catch (InterruptedException e) {
System.out.println("Main thread interrupted");
}

finally{
System.out.println("End");
}
}
}
```

Sleep(): On the main Thread

For Example:

```
importjava.lang.Thread;
import java.io.*;
classDt
{
public static void main(String[] args)
```

```
    {
try{
for(inti=1;i<5;i++)
  {
Thread.sleep(900);
System.out.println("value of i= "+i);
  }
}
catch(Exception e)
  {
System.out.println(e);
  }
 }
}
```

Note: In this program run as the value of amount in not same.

For Example:

```
public class Main extends Thread
 {
public static int amount = 0;

public static void main(String[] args)
  {
    Main thread = new Main();
thread.start();
System.out.println(amount);
amount++;
System.out.println(amount);
  }
public void run() {
amount++;
  }
}
```

Alive() : The isAlive() method of thread class tests if the thread is alive. A thread is considered alive when the start() method of thread class has been called and the thread is not yet dead. This method returns true if the thread is still running and not finished.

For Example:

```
class boot extends Thread
{
public void run()
    {
try {
            {
System.out.println("hello   this   is   first   task   :      "+
Thread.currentThread().isAlive());
            }
        } catch (Exception e) {
System.out.println("this is not work");
        }
    }
public static void main(String[] args)
{
boot b=new boot();
System.out.println("Before Ending :  "+ b.isAlive());

b.start();

System.out.println("END task :  "+ b.isAlive());
}

}
```

join() : Join method in Java allows one thread to wait until another thread completes its execution. In simpler words, it means it waits for the other thread to die. It has a void type and throws InterruptedException.

For Example:

```
class boot1 extends Thread
{
```

```
public void run()
    {
System.out.println("hello 1");
System.out.println("hello2");
System.out.println("hello this is first task :  ");
System.out.println("end------------");

        }
    }
class boot2 extends Thread{
public void run()
    {
System.out.println("hello this is 2 task :  ");
System.out.println("second");
        }

    }

class boot3 extends Thread{
public void run()
    {
System.out.println("hello this is 30 task :  ");
System.out.println("hello this is 325 task :  ");
System.out.println("hello this is 35 task :  ");
System.out.println("hello th is is 253 task :  ");
        }
    }
class soot{


public static void main(String[] args)
{
boot1 b=new boot1();

b.start();
try{
```

```
b.join();
    }
catch(Exception e)
    {
System.out.println(e);
    }
boot2 r=new boot2();
r.start();
boot3 g=new boot3();
g.start();
}
}
```

For Example: Note(In program use the join() with sleep() in thread).

```
class booty extends Thread
{
public void run()
    {
for(inti=1;i<=6;i++)
        {
try{
Thread.sleep(500);
        }
catch(Exception e)
        {
System.out.println(e);
        }
System.out.println(i);
    }
        }
    }
class moot{
public static void main(String[] args)
{
booty b=new booty();
b.start();
```

```
try{
b.join();
    }
catch(Exception e)
    {
System.out.println(e);
    }
booty r=new booty();
r.start();
booty g=new booty();
g.start();
}
}
```

For Example: Note(In program use the join() with sleep() in thread and use join(time) in block).

```
class booty extends Thread
{
public void run()
    {
for(inti=1;i<=6;i++)
        {
try{
Thread.sleep(500);
        }
catch(Exception e)
        {
System.out.println(e);
        }
System.out.println(i);
    }
        }
    }

class moot{
public static void main(String[] args)
```

```
{
booty b=new booty();
b.start();
try{
b.join(1200);
    }
catch(Exception e)
    {
System.out.println(e);
    }
booty r=new booty();
r.start();
booty g=new booty();
g.start();
}
}
```

Thread getPriority(): The getPriority() method of thread class is used to check the priority of the thread. When we create a thread, it has some priority assigned to it. Priority of thread can either be assigned by the JVM or by the programmer explicitly while creating the thread.The thread's priority is in the range of 1 to 10. The default priority of a thread is 5.

For Example:

```
classpr extends Thread
{
public void run()
    {
System.out.println("running     thread     anme:                "
+Thread.currentThread().getName());
}
public static void main(String[]args)
{
pr p=new pr();
pr p1=new pr();
System.out.println("first priority is  :  "+p.getPriority());
```

```
System.out.println("second  priority is  :  "+p1.getPriority());

p.start();
p1.start();
}
}
```

Thread setPriority() method: The setPriority() method of thread class is used to change the thread's priority. Every thread has a priority which is represented by the integer number between 1 to 10.

Thread class provides 3 constant properties:

MAX_PRIORITY: It is the minimum priority of a thread. The value of it is 10.

We can also set the priority of thread between 1 to 10. This priority is known as custom priority or user defined priority.

Syntax

public final void setPriority(int a)

For Example:

```
classpr extends Thread
{
public void run()
    {
System.out.println("running     thread     anme:                "
+Thread.currentThread().getPriority());
}
public static void main(String[]args)
{
pr p=new pr();

System.out.println("first priority is  :  "+p.getPriority());

p.setPriority(Thread.MAX_PRIORITY);
p.start();
}
}
```

MIN_PRIORITY: It is the maximum priority of a thread. The value of it is 1.

```
classpr extends Thread
{
public void run()
    {
System.out.println("running     thread     anme:                "
+Thread.currentThread().getPriority());
}
public static void main(String[]args)
{
pr p=new pr();

System.out.println("first priority is  :  "+p.getPriority());

p.setPriority(Thread.MIN_PRIORITY);
p.start();

}
}
```

NORM_PRIORITY: It is the normal priority of a thread. The value of it is 5.

```
classpr extends Thread
{
public void run()
    {
System.out.println("running     thread     anme:                "
+Thread.currentThread().getPriority());
}
public static void main(String[]args)
{
pr p=new pr();

System.out.println("first priority is  :  "+p.getPriority());

p.setPriority(Thread.NORM_PRIORITY);
p.start();
```

```
}
}

class pr1 extends Thread
{
public void run()
    {
System.out.println(Thread.currentThread().getName()   +   "   ---
running thread name:   " +Thread.currentThread().getPriority());
System.out.println("---------------");
}
}
```

Custom Priority: The thread scheduler chooses that thread for execution that has the highest priority.

For Example: Note(In this program use the single task with multiple threads).

```
classpr{
public static void main(String[]args)
{
pr1 p=new pr1();
Thread h=new Thread(p,"java");
Thread g=new Thread(p,"html");
Thread o=new Thread(p, "css");

h.setPriority(9);
g.setPriority(5);
o.setPriority(6);
System.out.println( "first priority is  :  "+h.getPriority());
System.out.println("Second piority is : " +g.getPriority());
System.out.println("Third priority is :  "+o.getPriority());

h.start();
g.start();
o.start();
}
```

```
}
```

For Example: Note(In this program use the single task with multiple threads).

```
class top extends Thread
{
public void run()
    {
System.out.println("running   thread   1   name:----------        "
+Thread.currentThread().getPriority());

}
}
class mid extends Thread
{
public void run()
    {
System.out.println("running   thread   2   name:   ------------     "
+Thread.currentThread().getPriority());
}
}
classbotom extends Thread
{
public void run()
    {
System.out.println("running   thread   3   name:   ------------     "
+Thread.currentThread().getPriority());
}
}

classmt{
public static void main(String[]args)
{
topht=new top();
mid g=new mid();
botom o=new botom();
```

```
ht.setPriority(4);
g.setPriority(9);
o.setPriority(2);
System.out.println("first priority is  :  "+ht.getPriority());
System.out.println("Second piority is : " +g.getPriority());
System.out.println("Third priority is :  "+o.getPriority());

ht.start();
g.start();
o.start();
}
}
```

Stop():A thread can be stopped using a boolean value in Java. The thread runs while the boolean value stop is false and it stops running when the boolean value stop becomes true

For Example:

```
classThreadDemo extends Thread {
publicboolean stop = false;
inti = 1;
public void run() {
while (!stop) {
try {
sleep(1000);
         } catch (InterruptedException e) {
         }
System.out.println(i);
i++;
      }
   }
}
public class Demo12 {
public static void main(String[] args) {
ThreadDemo t = new ThreadDemo();
t.start();
try {
```

```
Thread.sleep(10000);
       } catch (InterruptedException e) {
       }

t.stop = true;

System.out.println("The thread is stopped" );
   }
}
```

Wait(): Simply put, calling wait() forces the current thread to wait until some other thread invokes notify() or notifyAll() on the same object. For this, the current thread must own the object's monitor.

Program: Note(In which use without wait() in thread).

For Example:

```
Class mov extends Thread
{
int total=0;
public void run()
    {
for(inti=1;i<=10;i++)
        {
total=total+100;
    }
}
}
class move
{
public static void main(String[] args)
    {
mov j=new mov();

System.out.println("Total amount " +j.total);
j.start();
    }
}
```

Notify(): notify() wakes up a single thread that is waiting on this object's monitor. If many threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object's monitor by calling one of the wait methods.

Program: Note(In which use wait() and notify() in thread).

For Example:

```
classmov extends Thread
{
int total=0;
public void run()
    {
synchronized(this)
        {
for(inti=1;i<=10;i++)
        {
total=total+100;
    }
this.notify();
}
}
}
class move
{
public    static    void    main(String[]    args)         throws
InterruptedException
    {
mov j=new mov();


j.start();
synchronized(j)
{
j.wait();
System.out.println("Total amount " +j.total);
```

```
}
    }
}
```

Synchronization: Synchronization in Java is the capability to control the access of multiple threads to any shared resource. Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Program : Note (In this example, there is no synchronization, so output is inconsistent.)

For Example:

```
class kit
{
void show(int n)
    {
for(int i=1;i<=5;i++)
{
try
{
Thread.sleep(400);
    }
catch(Exception e)
    {
System.out.println(e);
    }
System.out.println(n*i);
}
    }
}
class kit1 extends Thread
{
kit t;
kit1(kit t)
    {
    this.t=t;
    }
public void run()
    {
```

```
t.show(4);
    }
}
class kit2 extends Thread
    {
kit t;
kit2(kit t)
    {
        this.t=t;
    }
public void run()
    {
t.show(10);
    }
}
class hit
{
public static void main(String[] args)
    {
kitobj=new kit();
kit1 k=new kit1(obj);
      kit2 k2=new kit2(obj);
k.start();
k2.start();

    }
}
```

Program : Note (In this example, there is no synchronization, so output is inconsistent.)

For Example:

```
class kit
{
synchronized void show(int n)
    {
for(inti=1;i<=5;i++)
{
```

```
try
{
Thread.sleep(400);
    }
catch(Exception e)
    {
System.out.println(e);
    }
System.out.println(n*i);
}
    }
}
class kit1 extends Thread
{
kit t;
kit1(kit t)
    {
    this.t=t;
    }
public void run()
    {
t.show(3);
    }
}
class kit2 extends Thread
    {
kit t;
kit2(kit t)
    {
        this.t=t;
    }
public void run()
    {
t.show(10);
    }
}
```

```
class hit
{
public static void main(String[] args)
    {
Kit obj=new kit();
kit1 k=new kit1(obj);
      kit2 k2=new kit2(obj);
k.start();
k2.start();
    }
}
```

Yield(): The yield() method of thread class causes the currently executing thread object to temporarily pause and allow other threads to execute.

For Example:

```
Class jin extends Thread
{
public void run()
    {
        //Thread.yield();
for(inti=1;i<=5;i++)
       {
System.out.println(Thread.currentThread().getName()+i);
}}
}
class yes
{
public static void main(String[] args)
 {
jin j=new jin ();
j.start();
Thread.yield();

for(inti=1;i<=5;i++)
{
```

```
System.out.println("java   " + i);
}
 }
 }
```