MultiThreading

=> Multitasking :-

-> Performing multiple task at single time

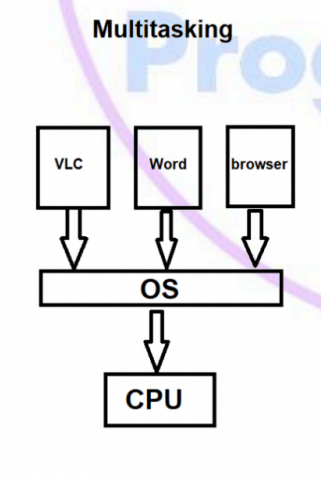
-> Examples : When we perform multiple task for example opening vlc, word, notepad, browser on single system etc

-> Multitasking use the concept of context switching internally(Switching between one task to another task)

-> Multitasking can be achieved by 2 ways :-

1. Process based multitasking (Multiprocessing)
2. Thread based multitasking (Multithreading)

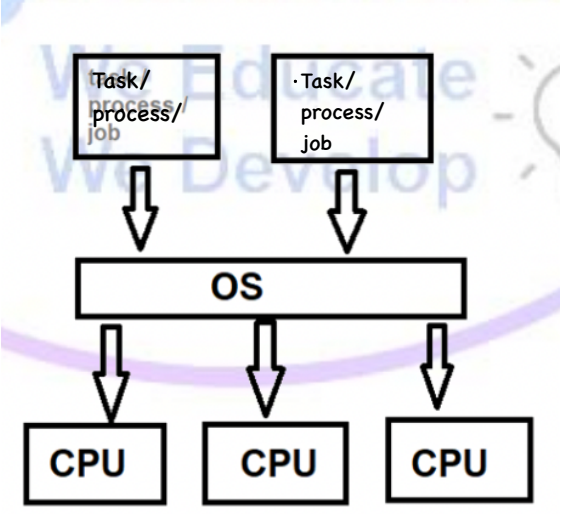
-> Multitasking is used to reduce the ideal time of CPU



=> Multiprocessing :-

-> Multiprocessing is the part when one system(OS) is connected with multiple processor (CPU)

-> Multiprocessing is best suitable at system level or OS level



* Above 2 concepts are not related to programming(Studied in OS)

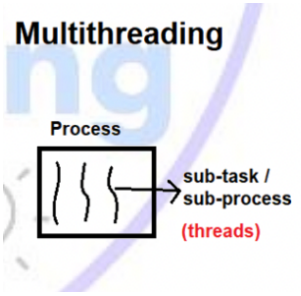
=> Multithreading :-

-> Multithreading is the part in which multiple threads (sub-process) are executed at single time

-> Examples :

1. Word
2. VLC
3. Games
4. Animations
5. Browser
6. Web-servers

etc



->Multithreading is mostly implemented by java itself, we have to implement less part of multithreading.

->Java provides API for multithreading :-

1. Thread
2. Runnable
3. ThreadGroup
4. Concureency
5. ThreadPool

=> What is difference between Process & Thread ?

1. Process is the heavy-weight task performing multiple tasks
   Thread is the light-weight sub-process performing single task
2. Process does not depends on each other
   Threads depends on each other
3. Context switching is difficult in process (takes more time)
   Context switching is easy in threads (takes less time)
4. Process shares the different address space
   Threads shares the same address space
5. Process does not require synchronization
   Threads requires synchronization

How to create threads :

-> There are 2 ways to create threads :-

1. By using "Thread" class
2. By using "Runnable" interface

=> What is "Thread" :-

-Thread is the pre-defined class which is present in java.lang package

=> Syntax :

```
class Thread implements Runnable
{
//constructors
//methods
run()
start()
etc
}
```

=> What is "Runnable" :-

Runnable is pre-defined interface present in java.lang package

=> Syntax :

```
interface Runnable
{
//method
run()
}
```

=> Which is better way to create thread, Thread or Runnable ?

-> Runnable is better way to create threads in java because if we inherit Thread class then we were not able to inherit any other class as multiple inheritance is not supported in java but if we inherit Runnable interface then we can inherit more interfaces and can inherit the class also

```
class Thread {

}

interface Runnable{

}

class Thread1 implements Runnable{

}


class Thread2 extends Thread{

}

class ATM1 extends JFrame implements Runnable, ActionListener,....{


}

class ATM2 extends JFrame, Thread... {          //cannot extends any other class


}
```

=> By using "Thread" class :-

-> Steps to create thread using "Thread" class :-

1. inherits(extends) the "Thread" class
2. override the run() method
3. create an instance of the class
4. start the thread

```
public class Test extends Thread{
    @Override
    public void run(){      //this run method is of Runnable interface since Thread class inherit Runnable
        //thread task
    }
}

public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        t.start();
    }
}
```

==========

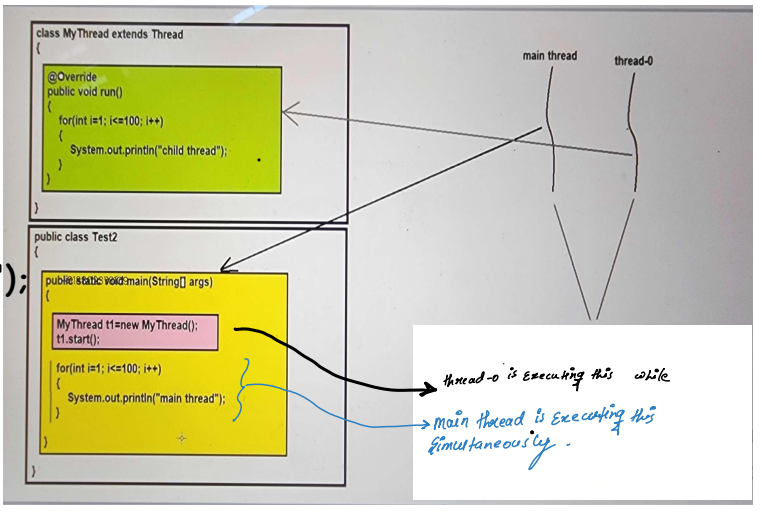
=> What is the use of start() method :-

1. It register the thread with "thread scheduler"
2. It will execute all the mentadory functionalities related to thread
3. It will invoke the run() method

etc

==========

```
public class Test extends Thread{
    @Override
    public void run(){
        for(int i=1 ; i<=100 ; i++){
            System.out.println("Child Thread");
        }
    }
}
```




```
public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        t.start();

        for(int i=1 ; i<=100 ; i++){
            System.out.println("Main Thread");
        }
    }
}
```

==========

without multithreading

```
public class VideoPlayer {
    public void playVideo(){
        for(int i=1 ; i<=100 ; i++){
            System.out.println("Playing Video");
        }
    }
}


public class SoundPlayer {
    public void playSound(){
        for(int i=1 ; i<=100 ; i++){
            System.out.println("Executing Sound");
        }
    }
}


public class Main {
    public static void main(String[] args) {
        VideoPlayer vp = new VideoPlayer() ;
        vp.playVideo();

        SoundPlayer sp = new SoundPlayer() ;
        sp.playSound();
    }
}
```

In the Above Code 1st 100 "PlayingVideo" will print and then 100 "Executing Sound" will print
for ex: if the video is of 1 hour then in that case 1st for 1 hour video will play and then after that sound will play therefor for 1 hour video it took 2 hour (1 for video and 1 for sound
)

with multithreading

```
public class VideoPlayer extends Thread {
    @Override
    public void run(){
        for(int i=1 ; i<=100 ; i++){
            System.out.println("Video playing");
        }
    }
}

public class SoundPlayer extends Thread{
    @Override
    public void run(){
        for(int i=1 ; i<=100 ; i++){
            System.out.println("Executing Sound");
        }
    }
}


public class Main {
    public static void main(String[] args) {
        VideoPlayer vp = new VideoPlayer() ;
        vp.start();

        SoundPlayer sp = new SoundPlayer() ;
        sp.start();
    }
}
```

==========

Q. What if we use 'start' as method instead of 'run'?

```
public class Test extends Thread {
    public void start() {
        System.out.println("Hello");
    }
}

public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        t.start();
    }
}
```

In this case the code will run but thread will not get created since no run method is found

Q. What if we call th.run instead of th.start() in main

```
public class Test extends Thread {
    public void run(){
        System.out.println("Hello");
    }
}

public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        t.run();
    }
}
```

the code will run but again the thread is not created

Q. Can we overload the run() method--> YES, and thread will also be created

```
public class Test extends Thread {
    public void run(){
        System.out.println("Hello");
    }
    public void run(int a){
        System.out.println(a);
    }
}

public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        t.start();          //this will call the run() method not the run(a) since start does not accept any argument
    }
}
```

Q. What if we do not provide the run method

```
public class Test extends Thread {

}
```

In this case, code will run properly but this time run method of Thread

```
public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        t.start();
    }
}
```

=> By using "Runnable" interface :-

-> Steps to create thread using "Runnable interface(Functional Interface->only 1 method) :-

1. Inherits the "Runnable" interface
2. Override the run() method
3. Create an instance of the class
4. Create an instance of Thread class and pass the above class instance in thread class constructor
5. Start the thread

Note: Runnable interface is present in java.lang package and there is no need to import java.lang package

```
public class Test implements Runnable{
    @Override
    public void run(){
        for(int i=1 ; i<=100 ; i++){
            System.out.println("Child Thread --> " + i +  "--> " + Thread.currentThread().getName());
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Test test = new Test() ;
        Thread th = new Thread(test) ;
        th.start();

        for(int i=1 ; i<=100 ; i++){
            System.out.println("Main Thread-->" + i + "--> " + Thread.currentThread().getName() );
        }
    }
}
```

```
public class NumberToPrint implements Runnable{
    int numPrint ;
    NumberToPrint(int numPrint){
        this.numPrint = numPrint ;
    }

    public void run(){
        System.out.println(numPrint+ " " + Thread.currentThread().getName());
    }


public class Main1 {
    public static void main(String[] args) {
        ExecutorService es = Executors.newFixedThreadPool(10) ;

        for(int i=1 ; i<=100 ; i++){
            NumberToPrint np = new NumberToPrint(i) ;
            es.submit(np) ;
        }
        es.shutdown();
```

```
public class NumberToPrint implements Runnable{
    int numPrint ;
    NumberToPrint(int numPrint){
        this.numPrint = numPrint ;
    }

    public void run(){
        System.out.println(numPrint+ " " + Thread.currentThread().getName());
        numPrint++ ;


public class Main1 {
    public static void main(String[] args) {
        NumberToPrint np = new NumberToPrint(1) ;
        for(int i=1 ; i<=100 ; i++){
            Thread th = new Thread(np) ;
            th.start();
        }

    }
}
```

=> Life cycle of thread :-

1. Born : New thread is created.

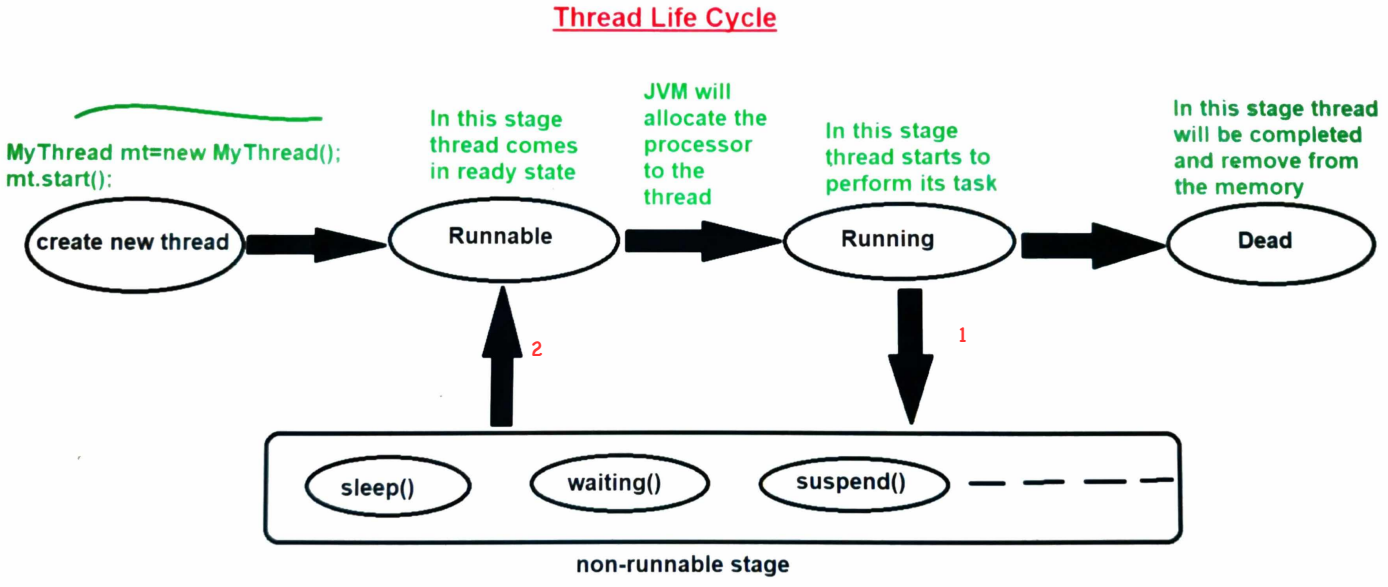
```
MyThread mt=new MyThread();
mt.start();
```

2. Runnable : In this stage thread comes in ready stage(Not in Execution stage)

3. Running : In this stage thread starts its task i.e. run() method is executing

4. Non-Runnable : In this stage thread does not perform any task. But when the thread is invoked then thread goes to the runnable stage and then it goes to the running stage

5. Dead : In this stage thread completes its task and removed from the memory



=> Note :

- We can envoke (start) the thread only once. If we try to start the thread again it will provide an exception saying java.lang.IllegalThreadStateException

```
public class Test extends Thread{
    public void run(){
        System.out.println("In the run Method");


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

        Test t = new Test() ;
        t.start();
        t.start();  //no compile time error, i.e., it is a runtime error
    }
}
```

Different cases of executing the threads :-

1. Performing Single Task from Single Thread
2. Performing Single Task from Multiple Threads
3. Performing Multiple Task from Single Thread
4. Performing Multiple Task from Multiple Threads

=> Thread class constructors :-

1. public Thread() { - }
2. public Thread(Runnable target) { - }
3. public Thread(String name) { - } --> to give name to the thread
4. public Thread(Runnable target, String name) { - }
5. public Thread(ThreadGroup group, Runnable target) { - }
6. public Thread(ThreadGroup group, String name) { - }
7. public Thread(ThreadGroup group, Runnable target, String name) { - }
8. public Thread(ThreadGroup group, Runnable target, String name, long stackSize) { - }

=> Thread class methods :-

1. Simple methods :-

- run() - this method contains the thread task
- start() - this method is used to create thread
- currentThread() - this method returns the reference of current running thread

2. Naming Methods :-

- getName() - this method is used to get the current running thread name
- setName(String name) - this method is used to set the name of current running thread

3. Daemon thread methods :-

- isDaemon() - this method is used to check wether the thread is daemon thread or not

=> Daemon Threads :-

- Daemon threads are the threads which are executed in the background of another thread
- For example : Garbage collector, finalizer(jo bhi object delete hone wala hai usse related task perform karta hai ye thread) spelling checker in word etc
- Daemon thread is used to provide the service to the other threads
- **Daemon thread methods :**
  - -> isDaemon()
  - -> setDaemon(boolean b)

Note :

1. We cannot create the running thread as daemon thread. If we try to create daemon thread after thread has started then it will throw an exception saying "java.lang.IllegalThreadStateException"
2. We cannot create main thread as daemon thread because JVM starts the main thread before we create it as daemon thread
3. Daemon thread life depends on another thread in which it is running
4. Daemon thread inherits the properties/nature from its parent thread
5. JVM can stop the daemon thread but it cannot stop the normal thread
6. It is recommended to make the lowest priority of daemon thread

⇨ Thread class methods diagram :

"Thread" class methods (Part 1)

```
public class Thread implements Runnable
{
    public void run() { - }
    public synchronized void start() { - }
    public static native Thread currentThread();
    public final native boolean isAlive();

    public final String getName() { - }
    public final synchronized void setName(String name) { - }

    public final boolean isDaemon() { - }
    public final void setDaemon(boolean on) { - }

    public final int getPriority() { - }
    public final void setPriority(int newPriority) { - }

    ----- (many more methods)
}
```

→ Basic Methods

→ Naming Methods

→ Daemon Thread Methods

→ Priority Based Methods

"Thread" class methods (Part 2)

```
public class Thread implements Runnable
{
    ----- (many more methods)

    public static native void sleep(long millis) throws InterruptedException;
    public static native void yield();
    public final void join() throws InterruptedException { - }
    public final void suspend() { - }
    public final void resume() { - }
    public final void stop() { - }
    public void destroy() { - }

    public void interrupt() { - }
    public boolean isInterrupted() { - }
    public static boolean interrupted() { - }
}
```

→ Prevent Thread Execution Methods

→ Depricated methods

→ Interrupting a thread Methods

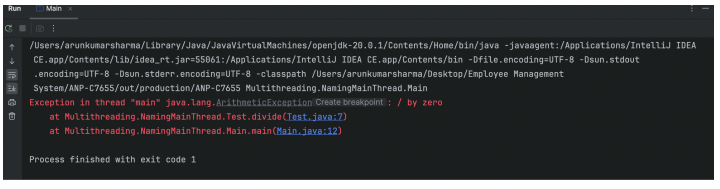
```
public class Test implements Runnable{
    @Override
    public void run(){
        System.out.println("Hi" + " from " + Thread.currentThread().getName());
```

```
public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
        Thread th = new Thread(t,"Arun") ; --> this will name the thread Arun
        th.start();
    }
}
```

==========

```
public class Test {
    public void divide(){
        int a = 100 ;
        int b = 0 ;
```

```
public class Main {
    public static void main(String[] args) {
        /*
        int a = 100 ;
        int b = 0 ;
        int c = a/b ;
         */
        Test t = new Test() ;
        t.divide();
    }
}
```




```
public class Test {
    public void divide(){
        int a = 100 ;
        int b = 0 ;
        int c = a/b ;
```

```
public class Main {
    public static void main(String[] args) {
        Thread.currentThread().setName("Arun");
        Test t = new Test() ;
        t.divide();
    }
}
```

==========

```
public class Test extends Thread{
    public void run(){
        System.out.println("hi");
        System.out.println("child thread " + Thread.currentThread().isDaemon());
    }
}

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

        Test t = new Test() ;
        t.setDaemon(true);
        t.start();
        t.setDaemon(true); --> give error, since daemon thread works in background and if we write it after t.start it wont work in background
hi
child thread false
Exception in thread "main" java.lang.IllegalThreadStateException Create breakpoint
    at java.base/java.lang.Thread.setDaemon(Thread.java:2276)
    at Multithreading.NamingMainThread.Main.main(Main.java:9)

        System.out.println("main thread " + Thread.currentThread().isDaemon());
    }
}
```

```
hi
main thread false
child thread false
```

```
hi
child thread true
main thread false
```

t.setDaemon(true);

==========

4. Priority Based Methods :

->getPriority() - This method is used to get the priority of the thread

->setPriority(int priority) - This method is used to set the priority of the thread

```java
class MyThread extends Thread 2 usages
{
    public void run()
    {
        System.out.println("hi : "+Thread.currentThread().getName());
        //System.out.println("Child thread priority : "+Thread.currentThread().getPriority());   //--> //set priority thread 0
    }
}
public class Test
{
    public static void main(String[] args)
    {
        MyThread mt=new MyThread();
        mt.start();

        mt.setPriority(8);   //set priority thread 0
        System.out.println("Child thread priority : "+mt.getPriority());

        Thread.currentThread().setPriority(8);  //--> Set priority of main Thread

        System.out.println("hello : "+Thread.currentThread().getName());
        System.out.println("Main thread priority : "+Thread.currentThread().getPriority());
    }
}
```

=> What is thread priority :-

->Thread priority is an integer value of the thread, the thread having high priority or high integer value will get the priority first to execute by JVM.

->Priority integer value ranges from 1 to 10.

->Java provides 3 pre-defined priorities :-

-> 1 - MIN_PRIORITY

-> 5 - NORM_PRIORITY

-> 10 - MAX_PRIORITY

#Below are not priorities :-

o 0, <1, >10 (these priorities will provide an exception saying java.lang.IllegalArgumentException)

o LOW_PRIORITY, MINIMUM_PRIORITY

o NORMAL_PRIORITY, MIDIUM_PRIORITY

o MAXIMUM_PRIORITY, HIGH_PRIORITY

```java
class MyThread2 extends Thread
{
    public void run() { System.out.println("hi"); }
}
public class Test2
{
    public static void main(String[] args)
    {
        MyThread2 mt2=new MyThread2();
        mt2.start();

        //mt2.setPriority(Thread.MAX_PRIORITY);
        mt2.setPriority(Thread.MIN_PRIORITY);
        System.out.println(mt2.getPriority());
    }
}
```

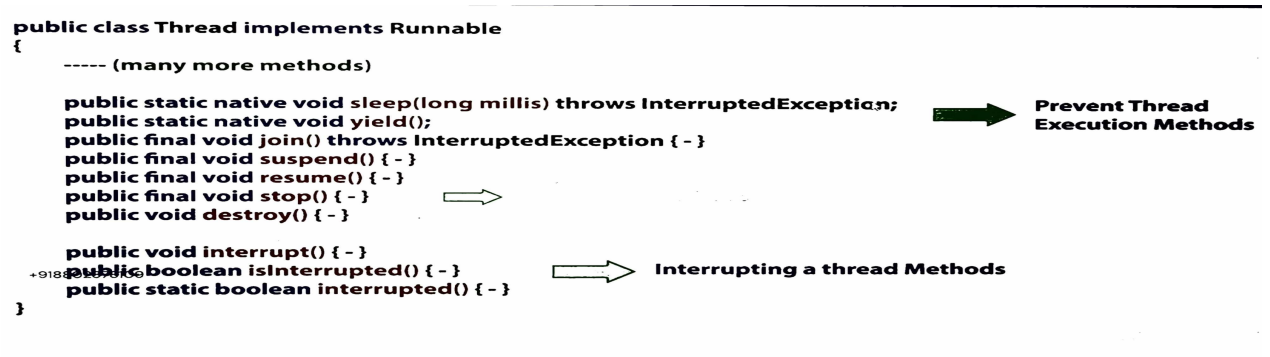
=>NOTE :

1. Priorities depends on the platform (Windows does not support thread priorities)-> we need to Mail to microsoft for this
2. By default main thread has priority 5
3. Thread default priorities are inherited by parent thread (It also depends on the platform)
4. If multiple threads have same priority then which thread will get the chance to execute first depends on the JVM (thread schedular)

5. Prevent thread execution method :-

-sleep(long mili) - This method is used to pause the current running thread for the provided time period

-yield() - This method is used to stop the current running thread execution and provide the



=> sleep() method :

-This method is used to pause the current running thread for the provided time period

-sleep() method is static method thus we have to call it by class name i.e. Thread class

-sleep() methods throws InterruptedException thus we have to use throws keyword or "trycatch block"

-We can provide time as 0 but it cannot be negative integer value (java.lang.IllegalArgumentException)

-JVM does not provide the guarantee that the sleeping thread will invoke exactly after provided time period

-When the thread goes into sleeping stage then it doesnt release the lock

```
class MyThread3 extends Thread
{
    public void run()
    {
        try
        {
            for(int i=1; i<=10; i++)
            {
                System.out.println(i+" : "+Thread.currentThread().getName());

                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
public class Test3
{
    public static void main(String[] args)
    {
        MyThread3 mt3=new MyThread3();
        mt3.start();

        try
        {
            for(int i=1; i<=10; i++)
            {
                System.out.println(i+" : "+Thread.currentThread().getName());

                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

=> yield() method :-

-This method stops the current thread execution and provide the chance to the other thread to execute

-Output is not constant in case of yield() method

-NOTE :-

o Till JDK 5 version, yield() method internally calls the sleep() method

o After JDK 5 version, yield() method working has been changed. It provides the hint to the thread schedular to yield(stop), but it totally depends on the thread-schedular that it will accept its request or not.

o If current running thread is stoped or yield, then which thread will get the chance for execution depends on the thread-schedular

```
class MyThread4 extends Thread
{
    public void run()
    {
        for(int i=1; i<=10; i++)
        {
            System.out.println(i+" : "+Thread.currentThread().getName());
        }
    }
}
public class Test4
{
    public static void main(String[] args)
    {
        MyThread4 mt4=new MyThread4();
        mt4.start();

        Thread.yield();  -> Stops the main Thread and give chance to other Thread

        for(int i=1; i<=10; i++)
        {
            System.out.println(i+" : "+Thread.currentThread().getName());
        }
    }
}
```

=> What is difference between sleep(), yield() and join()

Property	sleep()	yield()	join()
1. Purpose	This method will pause the current running thread execution for particular time period	This method will pause the current running thread and provides the chance to another thread of same or higher priority to execute	This method will pause the current running thread execution and waits for another thread to complete its task.
2. How threads are invoked ?	1. automatically after provided time period 2. if the thread is interrupted	1. automatically invoked by thread-schedular	1. automatically invokes after the another thread completes its task 2. automatically invokes after provided time period 3. if the thread is interrupted
3. Methods	1. static native void sleep(long millis) 2. static void sleep(long millis, int nanos)	1. public static native void yield();	1. final void join() 2. final synchronized void join(long millis) 3. final synchronized void join(long millis, int nanos)
4. is method overloaded	yes	no	yes
5. throws exception ?	yes (InterruptedException)	no	yes (InterruptedException)
6. Is method static ?	yes	yes	no
7. is method native ?	one is native and another is not native	yes	no
8. is method final ?	no	no	yes
9. Examples :-	timer (digital clocks), ppt, blinking blubs	billing counter	college, licence dept etc

Join Example:

```
class MyThread1 extends Thread  2 usages
{
    public void run()
    {
        try
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println("hi : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
public class Test1
{
    public static void main(String[] args)
    {
        MyThread1 mt=new MyThread1();
        mt.start();

        try
        {
            mt.join();  // this line will be exeucted by main thread, thus main thread will
                        //  wait for another thread (Thread-0 - mt) to complete its task
                       //  for which Thread it will wait will depends upon the reference,
                      //   Here 'mt' is the reference so main will wait for mt thread to complete the execution

            for(int i=1; i<=5; i++)
            {
                System.out.println("hello : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

If we want to make Thread0 i.e., mt to wait instead of main thread so for that we need main thread reference which we will pass tot he Test MyThread2 class

```
class MyThread2 extends Thread  2 usages
{
    Thread t;  no usages
    MyThread2(Thread t)  no usages
    {
        this.t=t;
    }
    public void run()
    {
        try
        {
            t.join();
            for(int i=1; i<=5; i++)
            {
                System.out.println("hi : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
public class Test2
{
    public static void main(String[] args)
    {
        Thread t=Thread.currentThread();    // main Thread reference since this line is executed by main

        MyThread2 mt=new MyThread2(t);
        mt.start();

        try
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println("hello : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

What if we have 3 Threads for example
1st admission take place then fee = deposit and then exam=rollno allocation:
Here 1st admission thread will complete and then fee deposit thread and then exam rollno allocation thread will execute

```
class TakeAdmisn extends Thread  2 usages
{
    public void run()
    {
        try
        {
            System.out.println("Admission starts");
            Thread.sleep(5000);
            System.out.println("You have taken the admisn successfully");
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

```
class DepositFee extends Thread  2 usages
{
    public void run()
    {
        try
        {
            System.out.println("depositing fee");
            Thread.sleep(3000);
            System.out.println("You have deposit the fee successfully");
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

```
class TakeRollno extends Thread  2 usages
{
    public void run()
    {
        try
        {
            System.out.println("Take roll no");
            Thread.sleep(1000);
            System.out.println("Now you can sit in exams....!!");
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

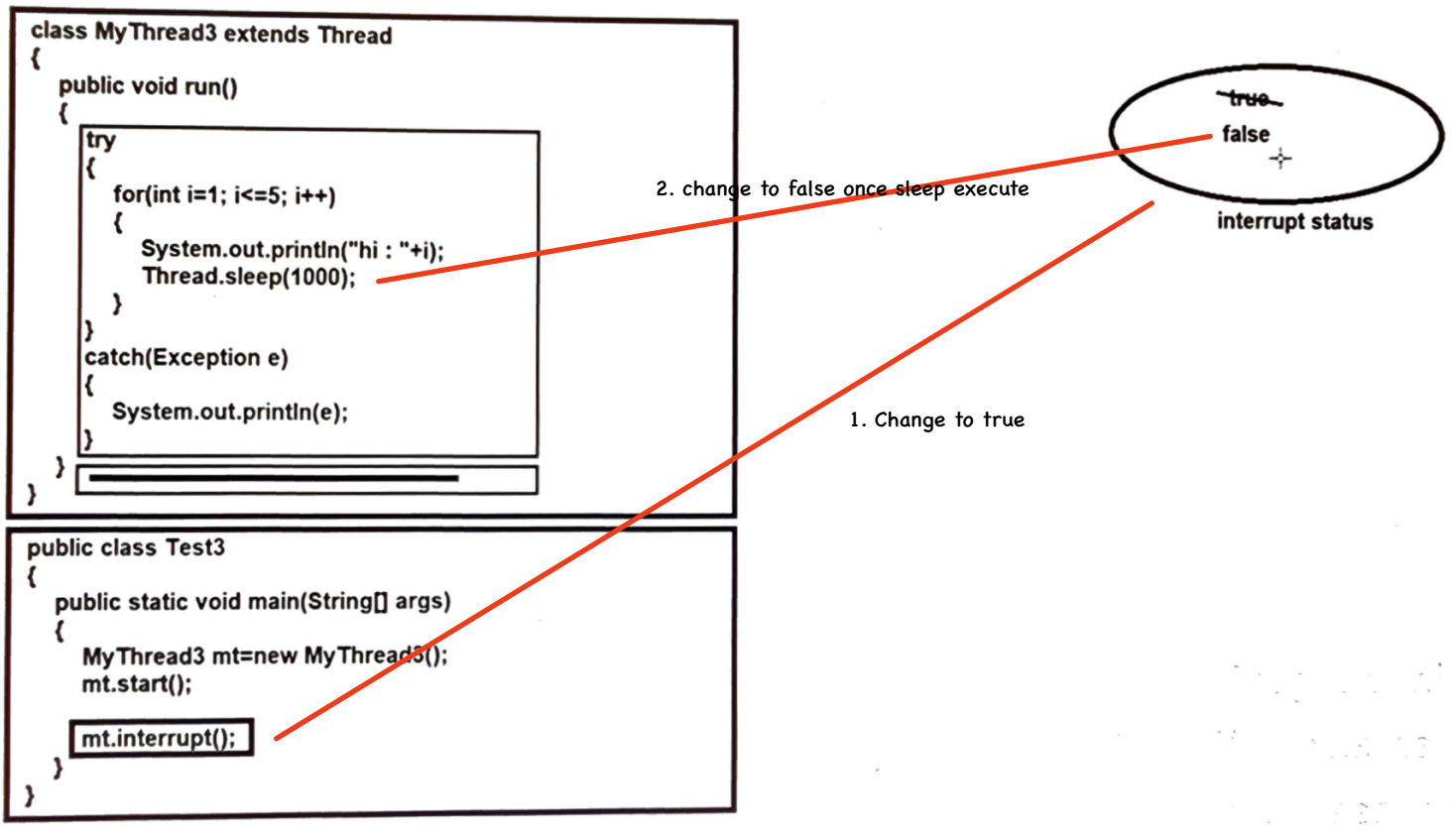
```java
public class ClgAdmin
{
    public static void main(String[] args) throws InterruptedException
    {
        TakeAdmisn t1=new TakeAdmisn();
        t1.start();

        t1.join();    //main thread will wait for t1 to complete

        DepositFee t2=new DepositFee();
        t2.start();

        t2.join();     //main thread will wait for t2 to complete

        TakeRollno t3=new TakeRollno();
        t3.start();
    }
}
```

6. Thread interrupting methods :-

interrupt() - It is used to interrupt the sleeping or waiting thread
-isInterrupted() - It is used to check the thread interrupt status. It will return true if interrupt status is true otherwise it will return false

interrupted() - It is used to check the thread interrupt status but if the interrupt status is true it will change the interrupt status into false.

=> interrupt() :-
-It is used to interrupt the thread
-It will work only when the thread is in sleeping or waiting state otherwise interrupt statement will be of no use
-It throws an exception "java.lang.InterruptedException"

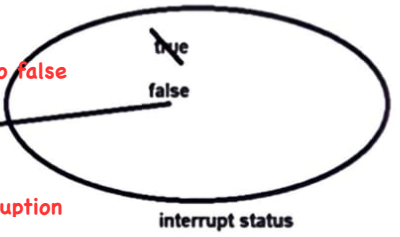


```java
class MyThread3 extends Thread
{
    public void run()
    {
        try
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println("hi : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        try
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println("hello : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
public class Test3
{
    public static void main(String[] args)
    {
        MyThread3 mt=new MyThread3();
        mt.start();

        mt.interrupt();
    }
}
```

```
class MyThread4 extends Thread
{
    public void run()
    {
        System.out.println("Is thread interrupted (1) : "+Thread.currentThread().isInterrupted());
        try
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println("hi : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        System.out.println("Is thread interrupted (2) : "+Thread.currentThread().isInterrupted());
    }
}
public class Test4
{
    public static void main(String[] args)
    {
        MyThread4 mt=new MyThread4();
        mt.start();

        mt.interrupt();
    }
```

```
class MyThread5 extends Thread
{
    public void run()
    {
        System.out.println("Is thread interrupted (1) : "+Thread.interrupted());
        try
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println("hi : "+i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        System.out.println("Is thread interrupted (2) : "+Thread.interrupted());
        System.out.println("Is thread interrupted (3) : "+Thread.interrupted());
    }
}
public class Test5
{
    public static void main(String[] args)
    {
        MyThread5 mt=new MyThread5();
        mt.start();

        mt.interrupt();
```

```
class MyThread5 extends Thread
{
   public void run()
   {
      System.out.println("Is thread interrupted (1) : "+Thread.interrupted());   (true) -> Also change the status to false
      try
      {
         for(int i=1; i<=5; i++)
         {
            System.out.println("hi : "+i);
            Thread.sleep(1000);  --> Found the status as false therefore execute normally without intruption
         }
      }
      catch(Exception e)
      {
         System.out.println(e);
      }
      System.out.println("Is thread interrupted (2) : "+Thread.interrupted());
   }
}
```




```
public class Test5
{
   public static void main(String[] args)
   {
      MyThread5 mt=new MyThread5();
      mt.start();

      mt.interrupt();
   }
}
```

==========

```
class MyThread6 extends Thread
{
    public void run()
    {
        System.out.println("Is thread interrupted (1) : "+Thread.interrupted());
        System.out.println("Is thread interrupted (2) : "+Thread.interrupted());
        System.out.println("Is thread interrupted (2) : "+Thread.interrupted());
        System.out.println("Is thread interrupted (2) : "+Thread.interrupted());
        System.out.println("Is thread interrupted (2) : "+Thread.interrupted());

        //System.out.println("Is thread interrupted (1) : "+Thread.currentThread().isInterrupted());
        //System.out.println("Is thread interrupted (2) : "+Thread.currentThread().isInterrupted());
    }
}
public class Test6
{
    public static void main(String[] args)
    {
        MyThread6 mt=new MyThread6();
        mt.start();

        mt.interrupt();
    }
}
```

=> What is difference between isInterrupted() and interrupted() method ?

1. isInterrupted() method does not change the interrupt status interrupted() method change the interrupt status

2. isInterrupted() method is non-static method interrupted() method is static method