Multithreading in Java

- Multithreading in Java is a process of executing multiple threads simultaneously.

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

→ Threads provide a way to improve application performance through parallelism.

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.
Each part of such Program is called thread.

* $\boxed{\text{Advantages}}$

you can perform many operations together, so it saves time.

Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

__/__/____

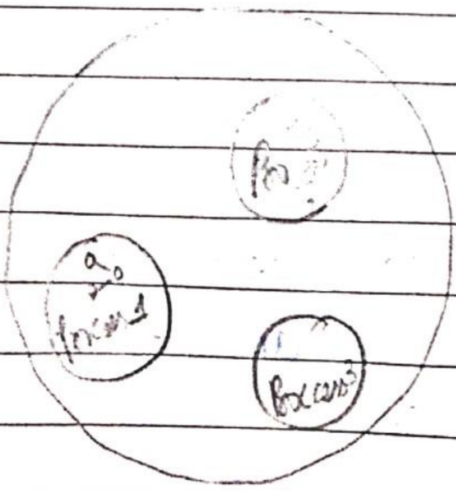
Multitasking

- Multitasking, it is an operating system concept in which multiple tasks are performed simultaneously.
- It is supports execution of multiple programs simultaneously.

* **Thread**

→ A thread is a lightweight subprocess, the smallest unit of processing.

→ Threads are independent. If Exception occurs in one thread, It doesn't affect other threads.



__/__/__

* CREATING THREADS:-

Threads are implemented in the form of objects that contain a method called run().

✓ The run() method is the heart and soul of the thread.

```
✓ public void run ()
   {
     = = =
   }
```

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.

Constructors of Thread class:-

Thread ()
Thread (String name)

__/__/____

Thread (Runnable target, String Name)

* 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.

3. public void join():- waits for a thread to die.

4. public void sleep(miliseconds) - It causes the currently executing thread to sleep for the specified number of milliseconds.

5. public void yield():- It causes the currently executing thread object to temporarily pause and allow other threads to execute.

6. isAlive(), suspend(), resume(), stop(), getName(), setName(String name), setPriority, getPriority(), currentThread().

→ Runnable interface have only one method named run().

_/_/__

1. EXTENDING THE THREAD CLASS :-

We can make our class runnable as thread by extending the class java.lang.Thread.

It include the following steps:-

(i) Declare the class as extending the Thread class.

(ii) implement the run() method

(iii) Create a thread object and call the start().

declaring the class :-

```
class MyThread extends Thread
{
    ----
    ----
}
```

Example :-

```
        Class thr extends Thread
        {
            Public void run()
            {
                System.Out.Println ("thread is Running");
            }
```

_/_/__

```
public static void main (String args[])
      thr obj = new thr();
        obj.start();
    }
  }
```

* IMPLEMENTING THE RUN() METHOD :-

The run() method has been inherited by the class MyThread.

• Starting New Thread

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

Example # Creating threads using thread class.

```
   class A extends Thread
     {
        public void run()
         {
             for (int i=1; i<=5; i++)
              {
                 S.O.P (" Thread A" + i);
              }
         }
   }
   class B extends Thread
```

```
        {
          public void run()
            {
              for(int j=1; j<=5; j++)
              {
                S.O.P.("Thread B"+j);
              }
        }
  }

  class Main
    {
      Public static void main (String args[])
        {
          A obj1= new A();
           Obj1.start();
          B obj2 = new B();
           Obj2.start();
        }
     }
```

start thread A

start thread B.

__/__/____

(2) Implementing the 'Runnable' Interface

We can create threads also in implementing the Runnable Interface.

Steps :-

(i) Declare the class as implementing the Runnable interface.
(ii) implement the run() method.
(iii) Create a thread by defining an Object from this 'Runnable' class as the target of the thread.
(iv) call the start() method to run the thread.

Program using Runnable interface

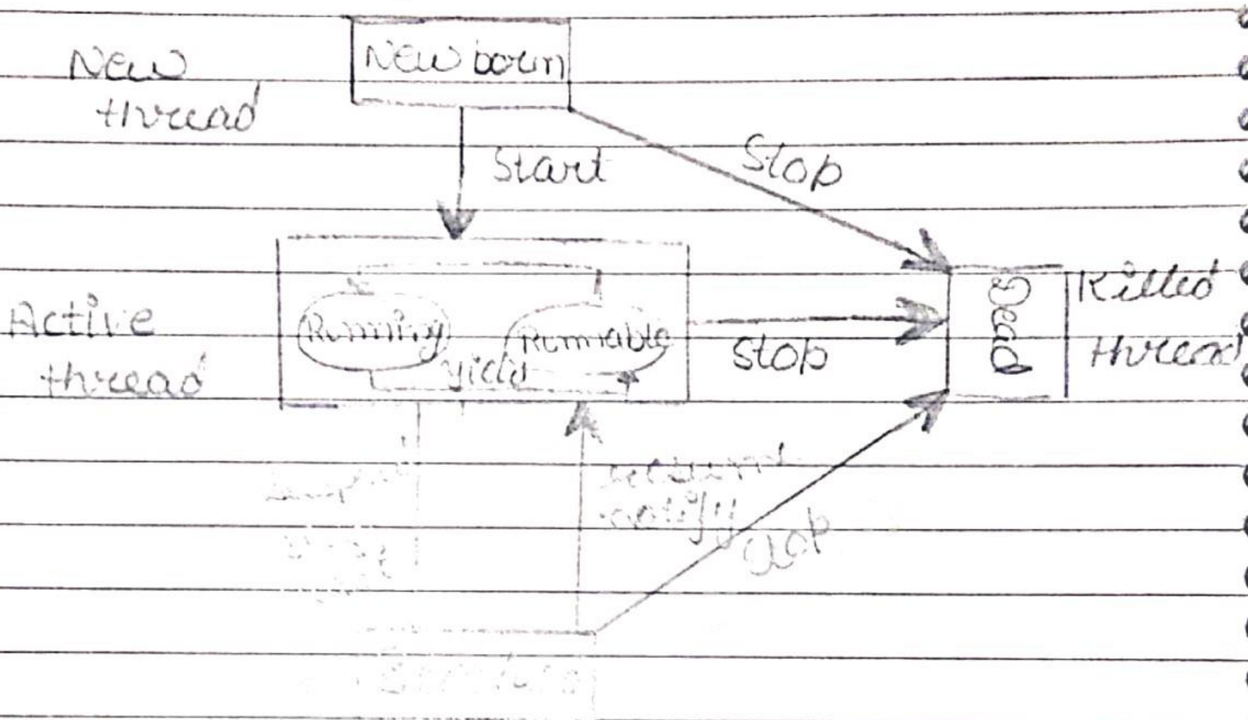
```
class abc implements Runnable
{
    Public void run()
    {
        for (int i=1 ; i<=10; i++)
        {
            System.out.println("Threadabc"+i);
        }
    }
}
class main
{
    Public static void main (String args[])
    {
        abc runnable = new abc();
        Thread threadabc = new Thread(runnable);
        threadabc.start();
    }
}
```

Output

```
Threadabc : 1
   "      : 2
   "      : 3
   "      : 4
   "      : 5
   "      : 6
   "      : 7
   "      : 8
   "      : 9
   "      : 10
```

_/_/_

* Java thread Model : (Life cycle of Thread)



1. **New :-** A new thread begins its life cycle in the new state. It is also referred to as a born thread.

2. **Runnable :-** After a newly born thread is started, the thread becomes runnable.

   * When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it :- One is runnable and other is running.

3. **Running :-** When the thread gets the CPU, it moves from th runnable to the

running state.

4. Blocked / Waiting :- When a thread is temporarily inactive, then it's in one of the following states :-
   - Blocked
   - Waiting

A thread is blocked state when it tries to access a protected section of code that is currently locked by some other thread.

On Becoming blocked :-

A thread can become blocked for five reasons :-

→ sleeep() :- It sleeps a thread for the specified amount of time.
→ suspend () :- It is used to suspend the thread.
→ resume () :- It is used to resume the suspended thread.
→ notify () :- It is used to give the notification for only one thread which is waiting for a particular object.
→ notifyAll () :- It is used to give the notification to all waiting threads of a particular object.

→ The thread is waiting for some I/O to Complete.

__/__/____

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

* Stopping and Blocked a Thread

(i) Stopping a Thread :- stop() method is used to stop a thread from running.

```
obj. stop();
```

(ii) Blocking a Thread :- A thread can also be a temporarily or blocked from entering into the runnable or running state by using following thread methods-

```
sleep()     // blocked for specified time
suspend()   // blocked until orders.
wait()      // blocked for certain condition.
```

_ / _ / _

Program using Thread Methods

```
class A extends Thread
  {
    Public void run()
   {
      for (int i = 1 ; i<=5 ; i++)
       {
         if (i=1)
         yield();
      System.out.println(" A"+ i);
     }
   System.out.println(" Exit from A");
}
}
 class B extends ~~A~~ Thread
    {
      Public void run()
       {
          for (int j=1; j<=5; j++) {
            System.out.println ("B"+j);
  if(j==3) stop();
     }
     System.out.println ("Exit B");
   }
}

 class C extends Thread
```

```
{
   public void run()
    {
      for (int k=1; k<=5; k++)
      {
        System.out.println ("           C"+k);
          if (k==1)
          try
             {
                sleep (1000);
             }
          catch (Exception e)
          {
          }
      }
      System.out.println (" Exit from C");
   }
}
  class Main
   {
   public static void main (String args[])
    {
         A obja = new A();
         B objb = new B();
         C objc = new C();
           obja.start();
          objb.start();
         objc.start();
     }
 }
```

__/__/____

* THREAD PRIORITY:-

In Java, each thread is assigned a Priority, which affects the order in which it is scheduled for running

The threads of the same priority are given equal treatment by the Java schedular, they share the processor on a FCFS- (first come first serve).

set the priority of thread :- setPriority()

Syntax :- ThreadName.setPriority(int Number);

$\downarrow$ integer value

Thread class defines sevral priority Constants:-

MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10

* if you don't set the priority, Java set the default priority with the help of getPriority() method.

$\uparrow$ 5

_/_/__

Program

```
Class th extends Thread
{
  Public void run()
  {
   System.out.println("Current Thread" +
                         Thread.currentThread.getName());
  }
}

Class Main
{
public static void main (String[] args)
{
  th obj1 = new th();
  th Obj2 = new th();

   Obj1.setName("First");
   obj2.setName("Second");

  Obj1.setPriority(4);
  obj2.setPriority(Thread.MAX_PRIORITY);

  Obj1.start();
  Obj2.start();
 }
}
```

```
Current Thread: second
Current Thread: first
```

_/_/_

* SYNCHRONIZATION :-

→ Synchronization is a technique through which we can control multiple threads. ~~or~~

→ Only one thread will enter inside the Synchroized area.

Note :- The main purpose of Synchronization is to overcome the problem of multithreading when multiple threads are trying to access some resource at same time on that situation it may provides some wrong result.

For example The method that will read information from a file

~~and method~~ that will update the same file may be declared as synchronized.

```
synchronized void update()
{
    - - -
    - - -
}
```

__/__/__

Types of Synchronization

(i) Method Level Synchronization

Method Level Synchronization is a simple method to prevent threads from interfering with each other while accessing same resources.

2) Block level Synchronization

Block level Synchronization just locks the block that requries updation of shared resources.

Syntax

```
Synchronized(<object reference>
{
    ---
    --
}.
```

__/__/____

* INTER- THREAD COMMUNICATION :-

- It can be defined as the exchange of messages between two or more threads
- Java implements inter- thread communication with the help of three methods :-

(i) notify() :- Resume the first method that went into the sleep mode.

final void notify()

(ii) notifyall() :- Resume all the threads that are in sleep mode.

final void notifyall()

(iii) wait() :- The desired waiting time period is specified as an argument to the wait() method.

final void wait()