Multithreading

Thread :-

(i) Thread represents a separate path of execution of a group of statements.
(ii) In a java program if we write a group of statement, then the statements are executed by JVM one by one.
(iii) This execution is called a thread, because JVM uses a thread to execute these statements.

WAP to find a thread used by JVM to execute the statement.

```
class ThreadDemo
{
public static void main(String args[])
{
System.out.println("Current thread");
Thread t = Thread.currentThread();
System.out.println("Current thread is = " + t);
System.out.println("Its name is = " + t.getName());
}
}
```

(Static method; 1 5 10)

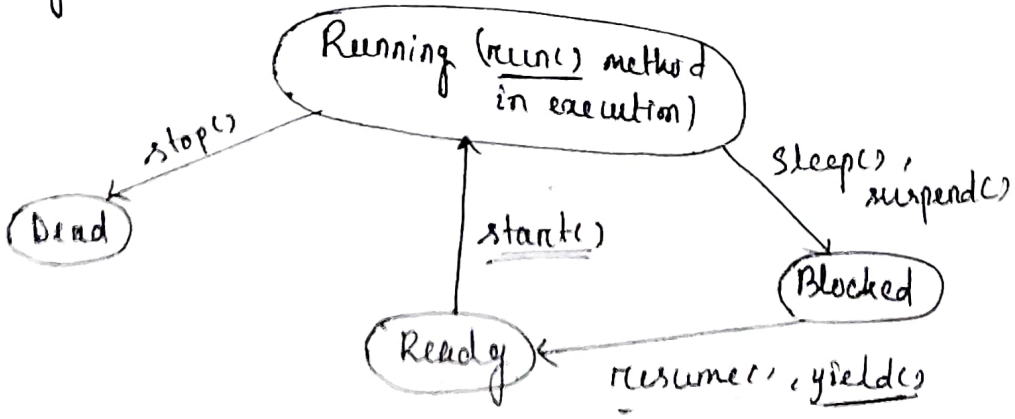
Output :- Current thread.
Current thread is = Thread [main, 5, main]
Its name is = main

(main thread name; main group; Priority)

Life cycle of a thread :-

Life cycle of a thread contains the following states:-

a) ready
b) running
c) blocked
d) dead



→ When the start() method is called the thread doesn't start working immediately. It goes into ready state and stay there until the task scheduler or thread scheduler selects it for execution.

→ When the thread starts execution it goes into the running state by invoking resume() method.

→ During its execution state the thread may temporarily give up the CPU. The various reasons are:

(i) It may go to a sleep or

(ii) It might be waiting for some event to happen.

→ This is known as blocked state.

→ Finally, when the thread completes its execution it goes to the dead state by calling stop().

→ From the blocked state the thread goes to the ready state when it wakes up by the invokation of the resume() method.

Various methods in Thread class :-

The Thread class is situated in java.lang package

java.lang.Thread

To create a thread we can use the following forms :-

```
Thread t1 = new Thread();
Thread t2 = new Thread(obj);
Thread t3 = new Thread(obj, "thread-name");
```

constructs

→ To know the current running thread :

```
Thread t = Thread.CurrentThread();
```

Synat: static Thread currentThread()

→ To start a thread :

```
t.start();
```

→ To stop execution of a thread for a specified time :

```
Thread.sleep(milliseconds);
```

→ To get the name of a thread :

```
String name = t.getName();
```

Syntax: String getName()

→ To set name to a thread :

t.setName("new Name");

→ To get the priority of a thread : 1 — 10

int priority_no = t.getPriority(); int

→ To set the priority of a thread :

t.setPriority(int Priority_no); 1 to 10

→ To test if a thread is still alive :

t.isAlive() returns true / false; boolean isAlive()

→ To wait till a thread dies :

t.join();

How to create a thread ? Give the steps :- implicitly

Step-1 :- Thread implements Runnable

Create a class that extends a Thread class or implements Runnable interface. Both the thread classes and Runnable interface situated in java.lang.package.

Note :-

→ In order to run a thread we need run() method. This run() method is in Runnable interface.

→ The Thread class implicitly implements Runnable interface. If a user wants to extend the thread class then by default run() method is available to this class.

IQ Difference between 'extends Thread' and 'implements Runnable' :- Which one has more advantages?

'extends Thread' and 'implements Runnable' both are functionally same, but when we write 'extends Thread' there is no scope to extend another class, as multiple inheritance is not supported in java.

If we write 'implements Runnable' then still there is a scope to extend another class. This is definitely advantageous when the programmer

wants to use Threads and also wants to access the features of another class.

```
class MyClass extends Thread
{
    public void run()
    {
        statements;
    }
```

OR

```
class MyClass implements Runnable
{

}
```

Step-2 :-

Now in this class write a run() method as public void run() { }. By default, this run() method is recognized and executed by a thread.

Step-3 :-

Create an object to MyClass, so that the run() method is available for execution.

```
MyClass obj = new MyClass();
```

Step-4 :-

Now, create a thread and attach the thread to the object obj.

```
Thread t = new Thread(obj);
Thread t = new Thread(obj, "ThreadName");
```

Step-5 :-

Now to run the thread we should start() of the Thread class

```
t.start();
```

|

run()

WAP to create a thread and display 0 to 10 using run()

```
class MyThreadClass extends Thread
  {
   public void run()
     {
      for(int i=0; i<11; i++)
        System.out.println(i);
     }
  }
class Demo
  {
   public static void main(String args[])
     {
      MyThreadClass obj = new MyThreadClass();
      Thread t = new Thread(obj);
      t.start();
     } //S.o.p("Hello");
  }
```

Output: Hello 0 1 2 3 4 5 6 7 8 9 10

Main Group → 1st Thread - main thread → t

WAP to create 2 threads and find priority of it..

```
class MyThreadClass extends Thread
  {
   public void run()
     {
      for int S.o.p(Thread.CurrentThread() + "Hello");
     }
  }
class Demo
  {
   public void static void main(String args[])
     {
      MyThreadClass obj = new MyThreadClass();
      Thread t = new Thread(obj, "First");
      S.o.p(t.getPriority());
      t.start();
     }
  }
```

1 (High) 5 (Avg) 10 (Least)

main Thd → main Group → JVM → main thread → First

Output:-

5

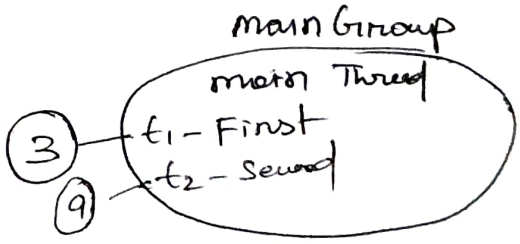
Thread [First, 5, main] Hello

- WAP to set priority of 2 threads. First thread as 3 and 2nd thread as 9.

```
class MyThreadClass extends Thread
  {
  public void run
  main()
      {
      S.o.p.(Thread.currentThread());
      }
  }
class Demo
  {
  public static void main(String args[])
      {
      MyThreadClass obj1= new MyThreadClass();
      Thread t1 = new Thread(obj1,"First");
      Thread t2 = new Thread obj
      S.o.p(t1.setPriority(3));
      t1.start();
      MyThreadClass obj2 = new MyThread(class);
      Thread t2 = new thread (obj2, "Second");
      S.o.p(t2.setPriority(9));
      t2.start();
      S.o.p("Hello");
      }
  }
```

1 → highest Priority

10 → lowest priority



Output:-

3

Thread [First, 9, main] → Hello

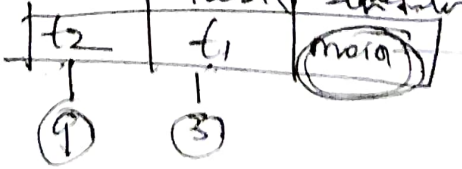
Second

9 Hello

Thread [First, 3, main]

Thread [Second, 9, main]

OS - Task Scheduler



Yield() method :-

Yielding is the process through which a currently executed thread goes to a ready state from running. If any other thread is waiting for execution then it might get chance for execution. Same priority

Ex:-

```
class X implements Runnable
  {
  public void ~~main~~ run()
      {
      System.out.println(Thread.currentThread());
      System.out.println("First");
      Thread.yield();        // public static void yield()
      System.out.println("Second");
      System.out.println(Thread.currentThread());
      System.out.println("Third");
      }
  }
class Demo
  {
  public static void main(String args[])
      {
      ~~publ~~ X o = new X();
      X p = new X();
      Thread t1 = new Thread(o,"1st");
      Thread t2 = new Thread(p,"2nd");
      t1.start();
      t2.start();
      }
  }
```

Output:

```
Thread [1st,5,main]
First
Thread [2nd,5,main]
First
Second [1st,5,main]
Third [1st,5,main]
~~Second~~ Third
Second
```

```
Thread [2nd,5,main]
Third
```

t2 - run()
↓ yield()
t2 - run()
↓ yield()
t1 - complet
then t2 complet.

sleep() method :-

(i) When a thread calls a sleep() method it causes the thread for sleeping for a particular time period as given by the programmer as parameter of the sleep() method

(ii) During a thread's sleep position if an interrupt occurs then it throws an InterruptedException

(iii) The syntax is :-

→ public static void sleep (long millisec, int nanosec) throws InterruptedException

```
{

}
```

→ public static void sleep(int millisec) throws InterruptedException

```
{

}
```

Ex-8 :-

```
class X implements Runnable
  {
  public void run()
      {
      try {
          Thread.sleep(2000);
          }
      catch (InterruptedException e)
          {
          System.out.println("I am handling it");
          }
      System.out.println (Thread.currentThread());
      }
  }
class Demo
  {
  public static void main(String args[])
      {
      X o = new X();
```

```
        Thread t1 = new Thread(0, "1st");
        t1.start();
        t1.interrupt();                    public void interrupt()
        }
    }
```

Output:- I am handling it
Thread [1st, 5, main]

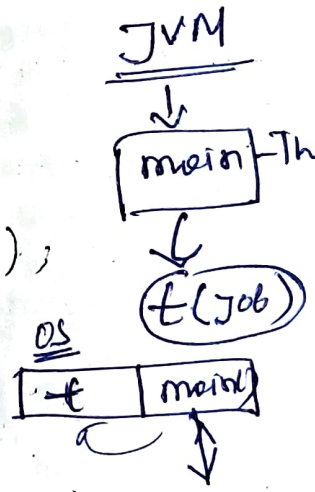
join() method :-

```
final void join() throws InterruptedException
```

This method waits until the thread upon which is called is terminated. That means the calling thread waits until the specified thread joins it.

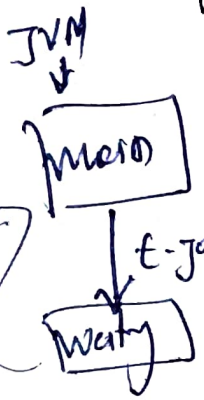
Eg:-

```
class X implements Runnable
    {
    public void run()
        {
        System.out.println("Inside X");
        }
    }
class ThreadDemo
    {
    public static void main(String args[])
        {
        X job = new X();
        Thread t = new Thread(job, "job");
        t.start();
        System.out.println("Inside main");
        }
    }
```



Output:- Inside main
Inside X

Ex:-
```
package demo;
class X implements Runnable
  {
  public void run()
    {
    System.out.println("Inside X");
    }
  }
class Thread1
  {
  public static void main(String args[])
    {
    X job = new X();
    Thread t = new Thread(job, "job");
    t.start();
    try {
      t.join();
      }
    catch (InterruptedException e)
      {
      }
    System.out.println("Inside main");
    }
  }
```



Output:-
Inside X
Inside main

Interthread communication :-

(i) If 2 or more threads communicate with each other by using some methods. are called inter thread communication

(ii) To improve communication between threads java.lang. Object class provides 3 methods.

→ public final void notify() :-

This method releases an object and sends a notification

to a waiting thread that the object is available.

→ public final void notifyAll() :- public final void notifyAll()

This method is useful to send notification to all the waiting threads that once you get the notification that object is available for running.

Syntax of wait()

public final void wait() throws InterruptedException

OR) public final void wait(long timeout) throws InterruptedException

OR) public final void wait(int nanosec) throws InterruptedException

↳ It is used only synchronized block.

Ex:-

```
public class WaitThreadDemo
{
public static void main(String args[])
{
ThreadB b = new ThreadB();
b.start();
System.out.println("Inside Main Thread");
synchronized(b)
{
try {
System.out.println("Waiting for b to complete:");
b.wait();
System.out.println("Thread is alive : " + b.isAlive());
}
catch(InterruptedException e)
{
e.printStackTrace();
}
System.out.println("Total is = " + b.total);
}
}
class ThreadB extends Thread
{
int Total; // Instance variable
```

```
public void run()          // override run()
  {
  System.out.println("Inside Thread B.");
  synchronized(this)
    {
    System.out.println("Inside synchronized thread B");
    for (int i=0; i<10; i++)
      {
      System.out.println(i);
      total += i;          (1+2+3+ ... +9)
      }
    notify();                          | total = total + i |
    }
  }
}
```

Output :- Inside main thread ✓
~~Inside~~ Waiting for b to complete ✓
Inside thread B
Inside synchronized thread B ✓
~~Total is: 45~~ 0 1 2 3 4 5 6 7 8 9
Thread is alive : true
Total is : 45

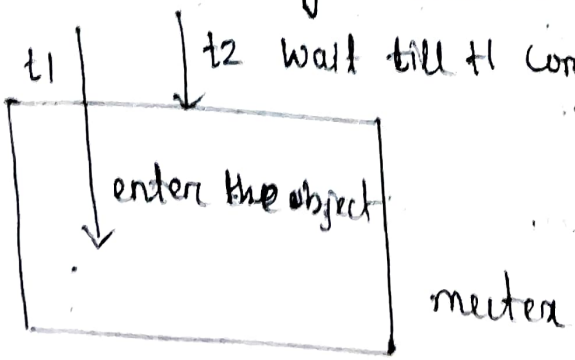
Difference between yield and wait :-

wait()	yield()
(i) wait() is declared in java.lang.Object	(i) yield() is declared on java.lang.Thread
(ii) Wait() is an overloaded method and has 2 versions of wait normal and timed wait.	(ii) yield() is not overloaded
(iii) wait() is an instance method.	(iii) yield() is a static method and work on current thread
(iv) wait() method must be called from either synchronized block or synchronized method.	(iv) There is no such method here.

Thread Synchronization :-

When a thread is already acting on an object, preventing any other thread from acting on the same object is called 'Thread synchronization' or 'Thread safe'. The object on which the threads are synchronized is called synchronization object. Thread synchronization is recommended when multiple threads are used on the same object (in multithreading).

Synchronized object is like a locked object, locked on a thread. It is like a room with only one door. A person has entered the room and should locked it from behind. The second person who wants to enter the room should wait till the first person comes out. In this way, a thread also locks the object after entering it. Then the next thread can't enter it till the first thread comes out. It means the object is locked mutually on threads. So, this object is called 'mutex' (mutually exclusive lock)

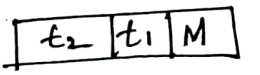


[Fig :- Thread synchronization]

Ex: WAP to synchronise the thread acting on the same object. The synchronized locks in the program can be executed by only one thread at a time.

```
class Reserve implements Runnable
{
int available = 1;
int wanted;
Reserve (int i)
        {
        wanted = i;
        }
public void run()
    {
    synchronized(this)
        {
        System.out.println("Available berths = "
                                    + available);
        if(available >= wanted)
        {
        String name = Thread.currentThread().
                                    getName();
        System.out.println(wanted + " Berths
                        reserved for " + name);
        try{
            Thread.sleep(1500);
            available = available - wanted;
            }
        catch(InterruptedException ie)
            {
            }
        }
    else
        S.o.p.("Sorry no berths");
    }
}
}
```

```
class Safe
  {
  public static void main(String args[])
      {
      Reserve obj = new Reserve(1);
      Thread t1 = new Thread(obj);
      Thread t2 = new Thread(obj);
      t1.setName("First person");
      t2.setName("Second person");
      t1.start();
      t2.start();
      }
  }
```



Output :- Available berths = 1
1 berth reserved for first person
Available berth = 0
Sorry, no berths

Note :-

If synchronized block is not inside reen method

Available berth = 1
1 berth reserved for 1st person
Available berth = 1
1 berth reserved for 2nd person.
Available berth = 0
Available berth = 0