Bsically there are some concepts which will come when i will talk about the multithreading.

A) MultiTasking -> E.g -> chrome + Eclipse all other operation that are perform at same time.

B) MultiThreading -> its sub individual Process inside the main process like many thread can execute under Chrome application.

its main responsibility is to reduce the loading time and it gives us the fast output.

All Java class has at least one thread which is a main thread , JVM will create the main thread...

this main thread will call the main method and execute the code under the main method .

I will write the main topic here only , multi threading is simple , but u need to understand the basic concepts of it thats it...

Deamon Thread.

Deamonn thrad is less priority thread, it means that , if execution of all user thread has done then JVM will get terminated, it wont stop for Deamon thread to perform its activity .

1 ) U can create your own deamon thread

2 ) By default we have GC , GC uses deaomon thread to perform its activity.

concepts that I have practiced in IDE

1) I'll use Runnable Interface

2) I'll create two child thread , by creating two different class.

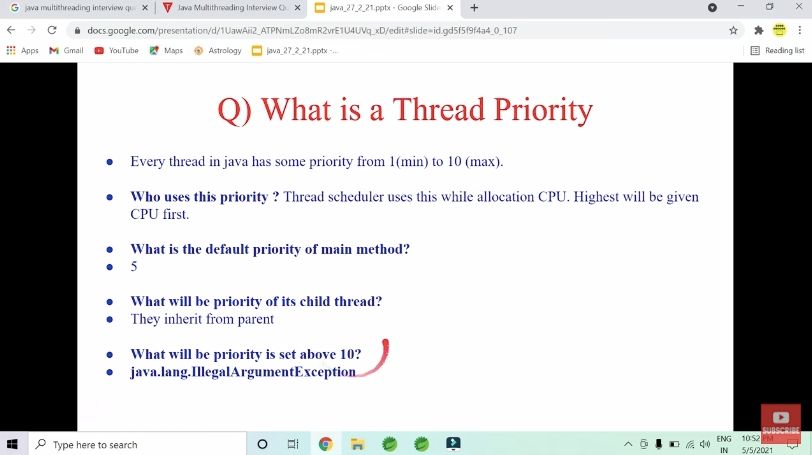
3) will set the deamon thread.

4) setName and getName of your current thread.

5) check the overriding and overrloading is possible for start() and run().

6) create the thread using inner class and with the help of java 8 also...

Important point



we can not restart the thread , means we cant use start() method after calling start() method .

## Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

* + Mutual ExclusiveSynchronized method.Synchronized block.Static synchronization.
  + Synchronized method.
  + Synchronized block.
  + Static synchronization.
  + Cooperation (Inter-thread communication in java)

Some of the important doubts and question that I have proved, I have written only highlighted points here.

Important point is here that want to make: ->

1) for example i have used t.start() method after 100 lines of my main code then child thread will only and only start after execution of that 100 lines of code ...its quit obvious BTW....

We need synchronized bcz if two thread uses the shared resources then there might be possibilities where we can get inconsistency output.

2) Write the two threads using synchronized key in methods . Compare the behaviour.

4. If we try to use a null object in the synchronized block then we may get a NullPointerException.

# THREAD LIFE CYCLE :

1) New Thread: When a new thread is created, it is in the new state. The thread has not yet started to run.

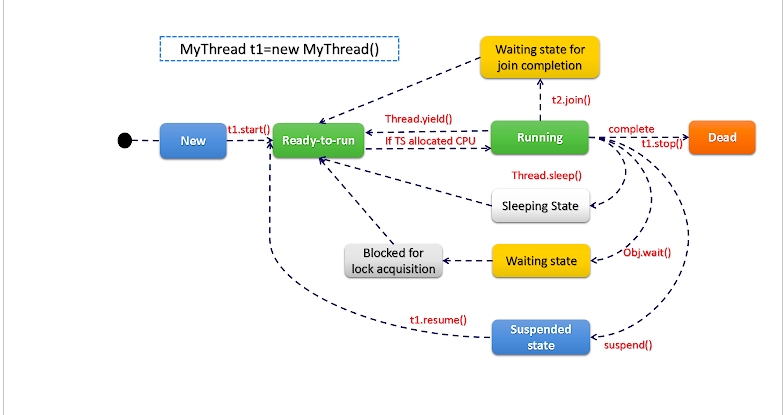
2) Runnable State: A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time

3) Blocked/Waiting state: When a thread is temporarily inactive

4) Terminated State: A thread terminates because of either of the following reasons:

* + Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
  + Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

in Java, one can get the current state of a thread using the Thread.getState() method.



wait() ->

1)wait() method is defined in Object class and hence the wait() method is declared in java.lang package.

2) In order to call wait method we require synchronized block or method as if wait() method is called outside the synchronized context it will throw IllegalMonitorStateException.

3) The wait() method releases the lock on an object and gives others chance to execute. The sleep() method does not releases the lock of an object for specified time or until interrupt.

4) The wait() is used for multi threaded synchronization, where single resource is shared among multiple thread.For example, file resources over network.

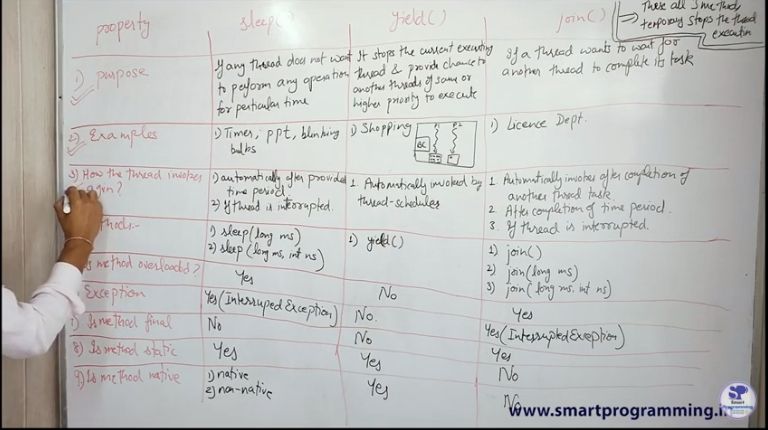
5) The wait() is used for time synchronization, where the thread actually needs a delay in background.For example, process something on specific interval.

join() ->

1) join() method, on the other hand, is also defined in java.lang package but in Thread class.

2) On the other no such condition required for calling join() method and we can call join() method with and without synchronized context in Java..

-



Important point :->

lets talk about synchronized block -->

synchronized(this){

}

this represent current object...

School b = new School();

synchronized(b){

}

b represent any object that we wnt to give lock

synchronized(School.class){

}

School.class it means we are giving class level lock

Static Synchronization :->

look we know that every object has intrinsic lock ...if we are operating/giving two thread to single object instance and we are using synchronized key word , then only one thread will execute the method RIGHT ....

BUT what happen if i have two different object and i call exact same thing with the help of this object as i did above,

then that will create a problem , two thread will go inside the same method despite the fact that the method is synchronized ....

obj1.callMethod() & obj2.callMethod() both will access the same method ....

so to overcome this problem we have concept called as class level or static synchronization...here class level lock will be given...then only one thread will execute even we have two object of that class....

Volatile keyword

DeadLock condition :->

Deadlock in Java is a condition where two or more threads are blocked forever, waiting for each other.

Nested Locks this is the main reason for a deadlock condition.

BLUE colour represnt that I have visited the question and done successfully !

Pink colour represent that I have visited but faced problem.

Questions that you can perform :->

1) write the thread using all possible method like .

a) Runnable b) Thread class c) Java-8 d) annonymous class

2) call non synchronized method from synchronized method ....

3) call synchro method and try to access other synchro method of the same object using the another thread

4) give proof why static synchro is important

5) write all the methods of thread like sleep, join ...etc

6) proove why volatile keyword is important..

7) create the demon thread

8) set the priority , set the name and get the name .

9) proove why synchronization is important

10) proove deadlock condition.

Good example of synchronization:->

public class PracticeSheetOne implements Runnable { public int balance = 5000; public int withdraw; public int remainingBalanace() { return balance - withdraw; } public PracticeSheetOne(int withdraw) { this.withdraw = withdraw; } public static void main(String[] args) throws InterruptedException { PracticeSheetOne p = new PracticeSheetOne(5000); Thread sunita = new Thread(p); Thread ram = new Thread(p); sunita.setName("Sunita"); ram.setName("ram"); sunita.start(); ram.start(); } @Override public void run() { synchronized(this) { if(this.balance > 0) { int reBalanace = remainingBalanace(); this.balance = reBalanace; System.out.println(Thread.currentThread().getName() + " Has withdrawn money"); }else { System.out.println("insufficient funds for " + Thread.currentThread().getName()); } } } }

Good example of static synchronization:->

package com.practiceSheet; public class PracticeSheetOne implements Runnable { public static int balance = 5000; public static int withdraw; public static int remainingBalanace() { return balance - withdraw; } public PracticeSheetOne(int withdraw) { PracticeSheetOne.withdraw = withdraw; } public static void main(String[] args) throws InterruptedException { PracticeSheetOne p = new PracticeSheetOne(5000); PracticeSheetOne p1 = new PracticeSheetOne(5000); Thread sunita = new Thread(p); Thread ram = new Thread(p1); sunita.setName("Sunita"); ram.setName("ram"); sunita.start(); ram.start(); } @Override public void run() { logicalPart(); } public void logicalPart() { synchronized (PracticeSheetOne.class) { if(balance > 0) { int reBalanace = remainingBalanace(); balance = reBalanace; System.out.println(Thread.currentThread().getName() + " Has withdrawn money"); }else { System.out.println("insufficient funds for " + Thread.currentThread().getName()); } } } }

Important points:->

1) one can only call the setDaemon() method before start() method otherwise it will definitely throw IllegalThreadStateException as shown below: