1. Two ways of creating Thread
2. By Extending Thread class
3. By Implementing Runnable interface
4. Getting and setting name of Thread
5. Thread Priorities
6. Methods to prevent Thread execution
7. Yield()
8. Join() method
9. Sleep() method

Synchronization

Inter Thread communication

Dead Lock

Deamon Threads.

MultiThreading Enhancemenets

Introduction

MultiTasking – executing multiple tasks simultaneously. - two types.

1. Process based multitasking
2. Thread based multitasking

**Process based multitasking : -**

Executing several tasks simultaneously, where each task is independent program(process) is called Process Based multitasking

Ex : - same time below all tasks happening simultaneously and independent of each other.

1. Typing Java Program in same system
2. Downloading a file from internet
3. Listening music from a same system.

Process based multitasking is best suitable at OS level(Operating system)

No dependency in all the above tasks, these are happening at Operating system level.

**Thread based multitasking: -**

Executing several tasks simultaneously, where each task separate independent part of the same program is called Thread based multitasking and each independent part is called a Thread.

Best suitable at programmatic level. – Thread based multitasking.

AdvanTage : - To increase the performance of the application and reduces idle time of process

Performance will be improved of the system .

Whether it is process based or thread based, the main objective multitasking is to reduce the response time of the system and to improve performance

Main important application areas Multi-Threading can be used in

1. To develop Multi Media Graphics.
2. To develop animations
3. To develop video games.
4. To develop and webservers and applications servers etc.
5. Ever Web/application server maintains the multiple Threads to handle multiple requests and send back responses simultaneously.

Tomcat – supports – 60 threads internally

A Flow of execution is called Thread.

Multiple flows -Multiple Threads.

Every Thread is separate flow of execution. – There is some separate Job

Thread is flow of execution.

Defining a Thread in two ways.

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

Main Thread starts child Thread .

Thread t=new Thread(); -> Thread Instantiation

t.start() -> starting of new Thread.

Thread scheduler:- scheduling threads.

Order of threads executing used by Thread scheduler vary from JVM to JVM.

If multiple threads are waiting to get the chance of execution, then which order threads will be executed is decided by Thread scheduler.

We cannot expect exact algorithm followed by Thread Scheduler it is varied from JVM to JVM, hence we cant expect threads execution order and exact output, hence whenever situation comes to Multithreading , there is no guarantee for exact output , but we can provide several possible outputs.

Difference between t.start() and t.run()

t.start() -> new thread will be created to run method

t.run() will call as normal method , it wnt create any new thread.

If we use t.run() method – total out will be produced by main thread().

Importance of Thread.start()-> heart() of multithreading

1. Register this thread with the Thread scheduler.
2. Perform all other mandatory activities
3. Invoke run() method

Overloading of run() method possible and always Thread.start() always will call the no arguments run method and will not call any other arguments run method.

If you are not overidding run method then , Thread class run method will be executed , which has empty implementation , hence we wont get any output.

It is highly recommended override run() method , other wise do not go for multithreading concept.

If Overriding of start() method, then our start method will be executed , just like a normal method call and new thread wont be created .

It is not recommended to override start() method, other wise do not go or multithreading concept

1. <https://www.youtube.com/watch?v=fd3TAYCHRfw-> Part2 MultiThreading
2. <https://www.youtube.com/watch?v=8xnNgR0zeS8> – Part 3 MultiThreading

Thread Lifecycle

1. Base life cycle (Top level life cycle)

Thread th=new Thread() 🡪 new state or born state

Th.start()🡪 Thread will enter to ready state/Runnable state.

If Thread scheduler allocates processers then thread will be running state

Once run() method completed thread will go in dead state.

If you are stating thread which is already started, then we will get illegalthreadstate exception

(IllegalThreadStateException)- RuntimeException

(RuntimeException)

3rd Video

* Defining a Thread by implementing Runnable interface.
* Defining a Thread by extending Thread class.
* Runnable interface present in java.lang package and it contains only method run().

Public class MyRunnable implements Runnable

{

}

MyRunnable r=new MyRunnable();

Thread t1=new Thread();

Thread t2=Thread(r);

Case 1 : t1.start(); 🡪 new Thread will be created which class Thread run method will be called, which has empty implementation

Case 2 : t1.run();🡪 no new Thread will be created and run will be executed like a normal method

Case 3 : t2.start(); 🡪 yes , new Thread will be started , MyRunnable object will be created and run () method will be called

Case 4 : t2.run(); 🡪 directly calls the run method , without creating any new Thread

Case 5 : r.start(); 🡪 compile time error , can’ find symbol start() location class MyRunnable.

Case 6 : r.run() method will be executed -My Runnable class run method will be executed.

Which approach is best to define a Thread?

Among two ways of defining a Thread implements Runnable approach is recommended

In the first approach our class always extends Thread class, there is no chance of extending any other class. Hence we are missing inheritance benefit.

But in the second approach while implementing Runnable interface, we can extend any other class, hence we won’t miss any inheritance benefit.

Bcz of above reason implementing Runnable interface approach is recommended than extending Thread class.

Thread Class constructors

Thread t=new Thread();

Thread t=new Thread(Runnable r);

Thread t=new Thread(String name);

Thread t=new Thread(Runnable r , String name);

Thread t=new Thread(ThreadGroup tg, String name);

Thread t=new Thread(ThreadGroup tg, Runnable r);

Thread t=new Thread(ThreadGroup tg,Runnable r,String name);

Thread t=new Thread(ThreadGroup tg,Runnable r,String name,long stackSize);

Durgas apparoch to define a Thread – not recommended to use

We can set and get name by using following methods

Public final String getName();

Public final void setName(String name);

We can get current executing Thread object by using Thread.currentThread().getName();

<https://www.youtube.com/watch?v=DpM_wER1gAY> – seesion - 4

Every thread has its own priority

Thread valid - range priority 1-10 ( where 1 is Min priority and 10 is Max Priority)

We can set the Thread Priority.

Thread Class defines the following constants – to represent the some standard priorities.

Every Thread in Java has some priority, it may be default priority generated by JVM or customized priority provided by programmer.

The valid range of Thread priorities is (1-10 ) where 1 is MIN priority and 10 is MAX priority .

Thread class defines the following constants to represent some standard priorities.

Thread.MIN\_PRIRORITY =1

Thread.NORM\_PRIROIRTY =5

Thread.MAX\_PRIRORITY =10

Thread has highest priority will get first chance to run.

If two threads having same priority then we can’t expect exact execution order, it depends on Thread scheduler.

The below two methods get and set the priority of a Thread.

Public final int getPriority()

Public final void setPriority(int p) – allowed range 1-10 .

If the range is not in 1-10 , we will get IllegalArgumentException will occur.

Default Priority of main thread is 5 and other threads priority will be inherited from parent thread. Child threads has same priority as Parent Thread.