**Multi Threading**

**Link:**

**<https://www.youtube.com/watch?v=Hysb7hXp8B0&list=PLd3UqWTnYXOkWZUcbW68CbN9fyPFQ0LDk&ab_channel=DurgaSoftwareSolutions>**

1. introduction
2. the ways to define a thread
   1. by extending thread class
   2. by implementing runnable interface
3. getting and setting name of thread
4. thread priorities
5. the methods to prevent thread execution
   1. yield();
   2. join();
   3. sleep();
6. Synchronization
7. inter thread communication
8. deadlock
9. daemon thread
10. multi threading enhancement

**Video 1 : Multi threading introduction**

**Multitasking :** executing multiple task simultaneously is called as multitasking.

there are two type of multitasking

1. **Process based** :
   1. executing several task simultaneously where each task is separate independent program(process) is called process based multi tasking.
   2. **eg**. while running java program in editor we can listen audio song from same system at same time we can download file from internet all these task will be executed simultaneously and independent of each other hence it is process based multitasking.
   3. process based multitasking is best suitable at OS level.
2. **Thread based** :
   1. executing several task simultaneously where each task is a separate independent part of same program this type of multitasking called thread based multitasking. and each independent part is called thread.
   2. thread based multitasking is best suitable at programmatic level.

**Note :** whether it is process based or thread based the main objective of multitasking is to reduce response time of the system and to improve preforms.

**The main important application areas of multi threading are :**

1. to develop multi media graphic.
2. to develop animations.
3. to develop video games.
4. to develop web severs and application servers etc.

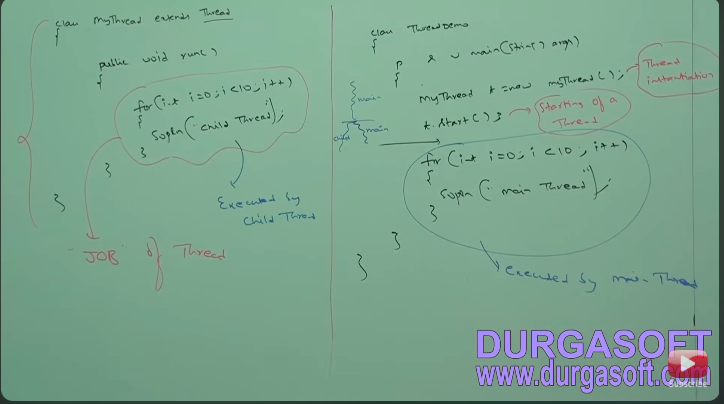
Tomcat bydefault handle 60 request at time that means it uses multi threading concept. each thread is handling separate request at time.

when compare with old languages developing multi threaded application in java is very easy because java provides inbuilt support for multi threading with reach API (thread, Runnable, Thread Group....).

**Video 2 : the ways of defining a thread**

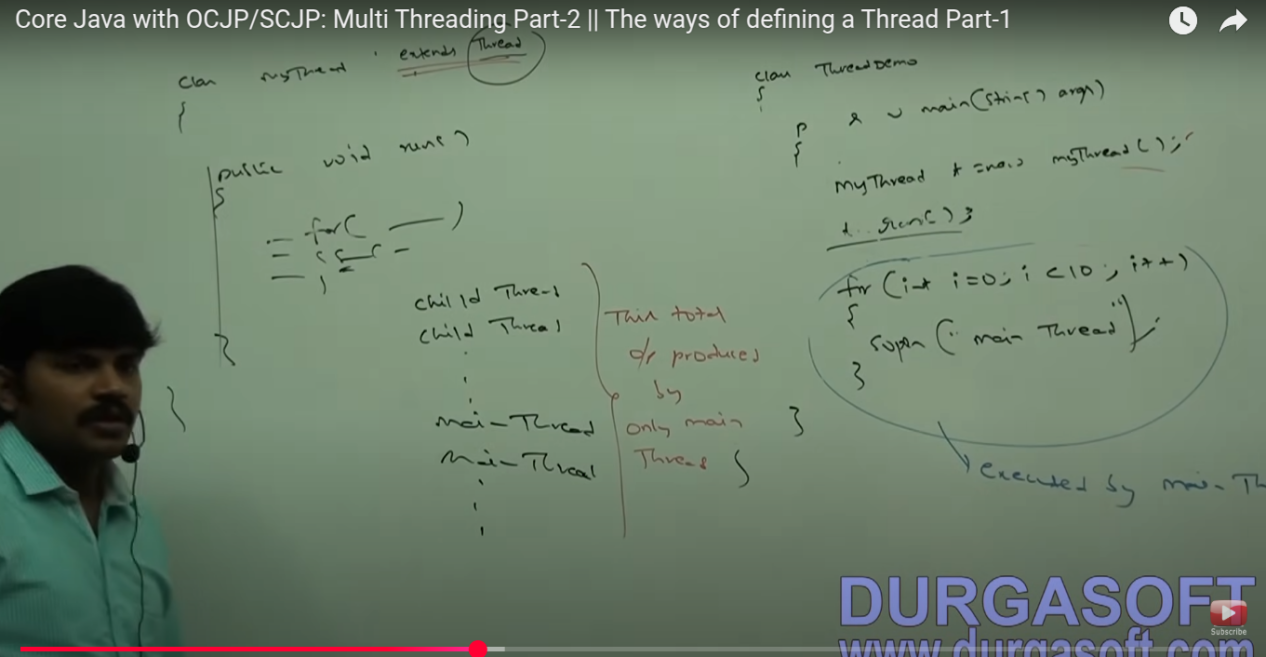
1. we can defining a thread in the following two ways:
   1. by extending **thread** class
   2. by implementing **runnable** interface

**example:**



**Case 1. thread scheduler**

1. it is the **part** of **jvm**
2. it is responsible to **schedule threads** that if multiple threads are **waiting** to get **chance of execution** then in which order thread will be **executed** is **decided** by **thread scheduler**.
3. we can except exact algorithm followed by thread scheduler it is varied from jvm to jvm hence we can’t except thread execution order and exact output hence whenever situation comes to multi threading there is no guaranty to exact output but we provide several possible outputs.
4. the following are various possible output for above programs



**Case 2 : difference between t.start() and t.run()**

1. in the case of t.start() a **new thread** will be **created** which is responsible for **execution of run method**
2. but in the case t.run() a **new thread won’t be created** and run method will be executed just like a normal method call by main thread.
3. hence in the above program we can replace t.start() with t.run() then the output is child thread child thread 10 time and main thread main thread 10 times.
4. this total output produced by main thread.

**Case 3 : Importance of thread class start method.**

1. thread class **start method** is responsible to **register the thread** with thread scheduler and all other **mandatory activities** hence without **executing thread class start method** there is no chance of **starting new thread** in java due to this thread class **start method** is consider as **heart to multi threading.**
2. eg.

**start(){  
 1. Register this thread with thread scheduler.**

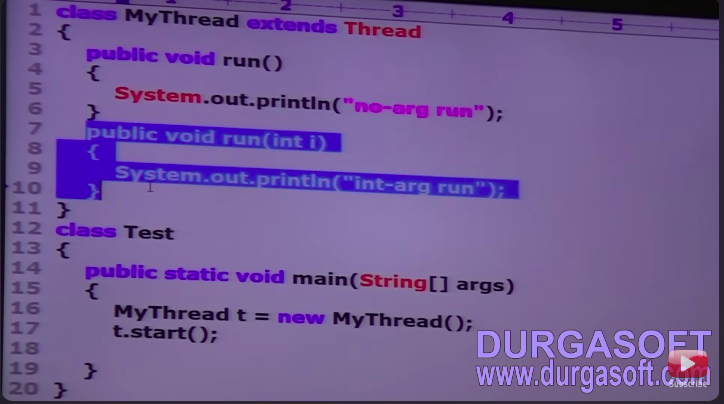
**2. Perform all other mandatory activities.**

**3. Invoke run();**

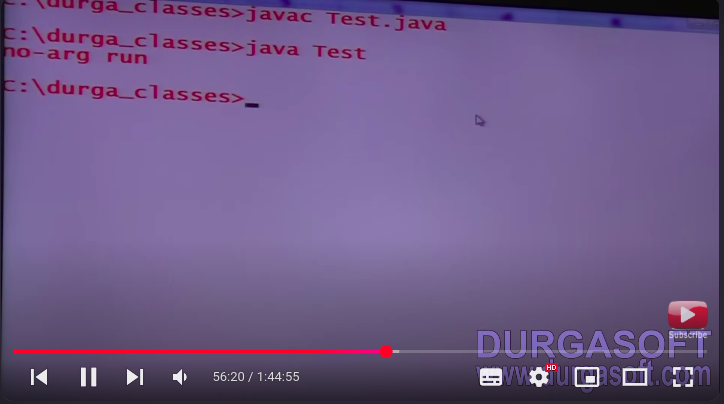
**}**

**Case 4 : overloading run method**

1. overloading of run method is always possible but thread class start method can invoke no-agrument run method.
2. the other overloaded method we have called explicitly like a normal method call.

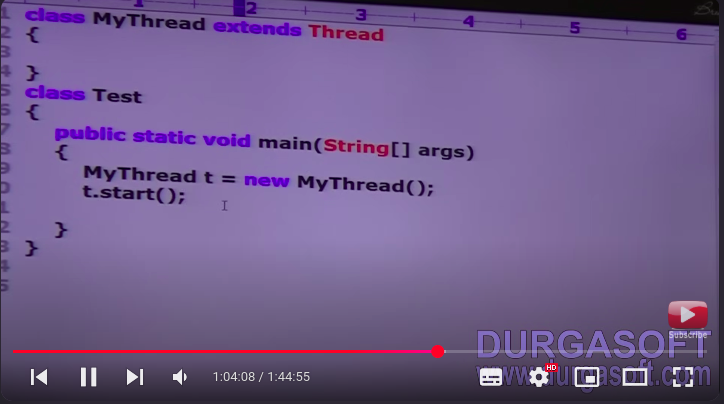


output :

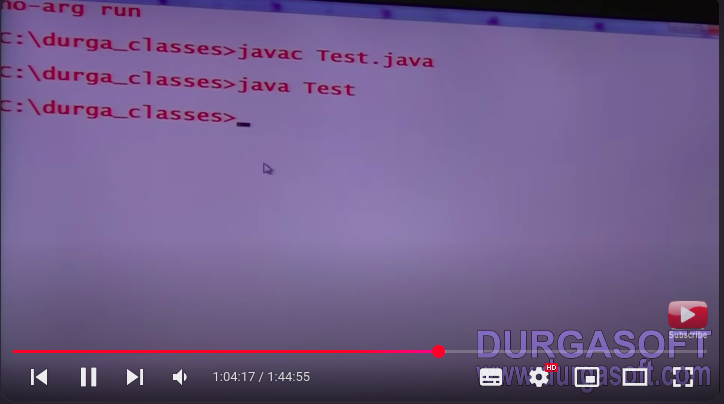


**Case 5 : If we are not overriding run method**

1. if we are not overriding run method then thread run method will be executed which has empty implementation hence won’t get any output.
2. it is highly recommended override run method other wise don’t go for multi threading concept.

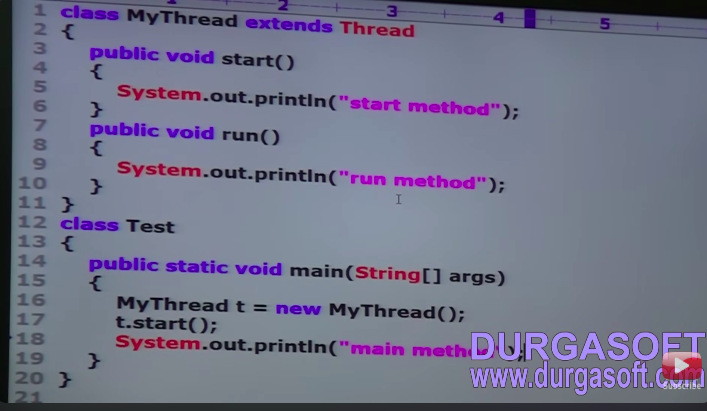


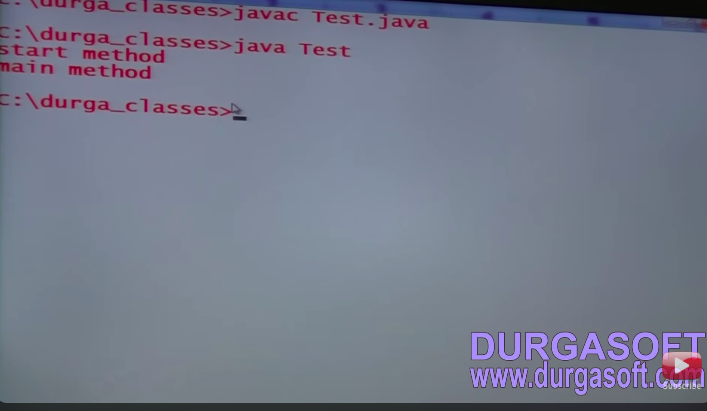
output:



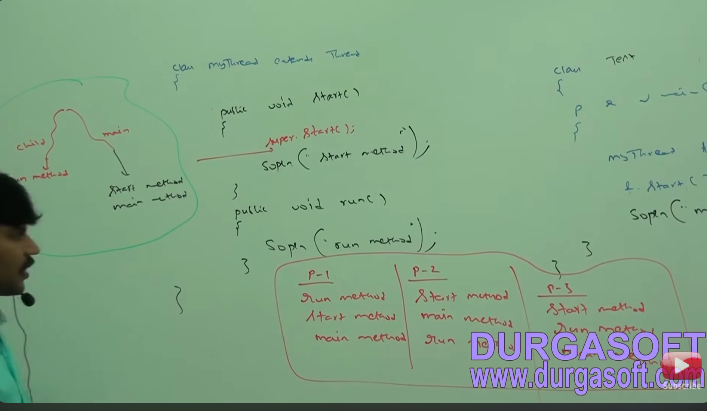
**Case 6 : overriding of start method.**

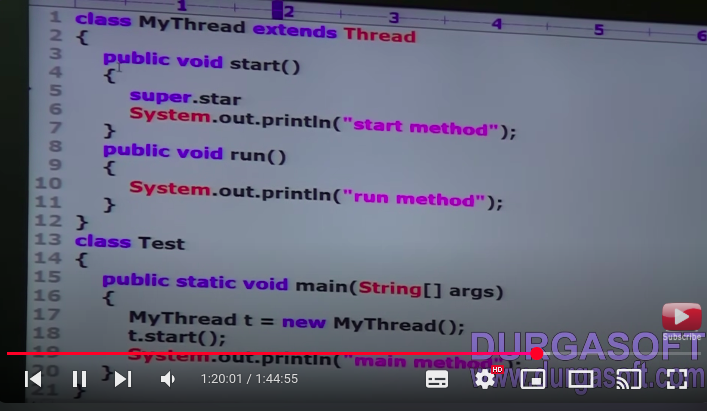
1. if we override start method then our start method will be executed just like normal method call and new thread won’t be created.
2. it is not recommended to over ride start method otherwise don’t go for multi threading concept.



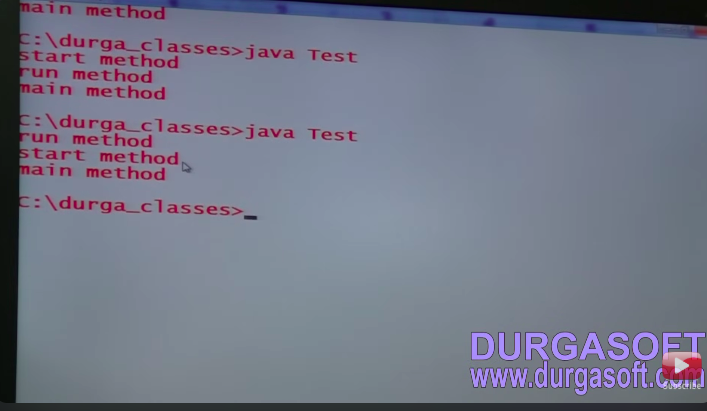


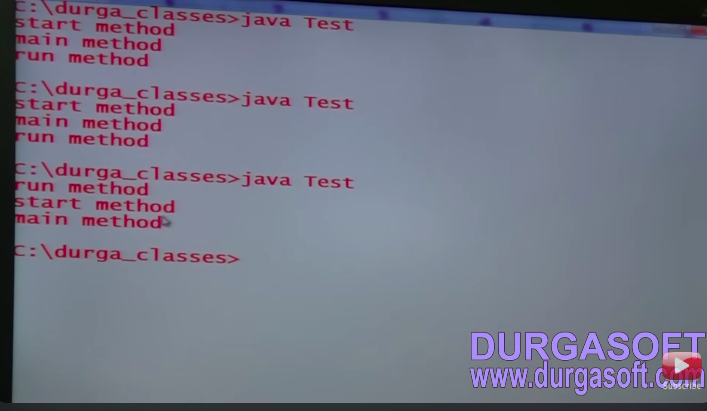
eg.2



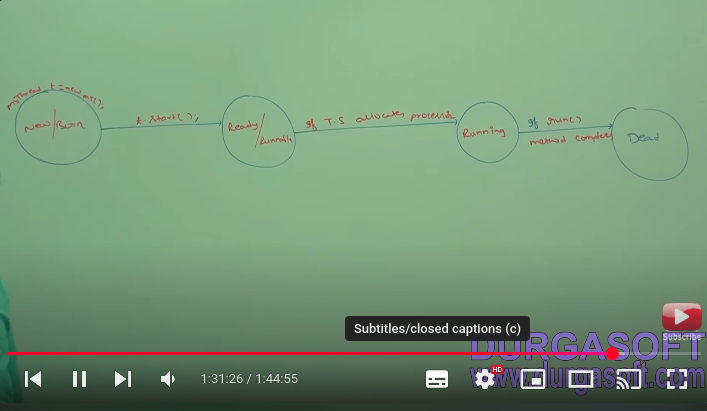






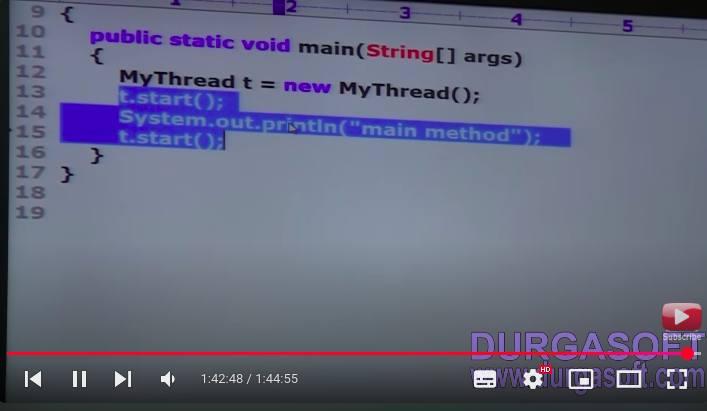


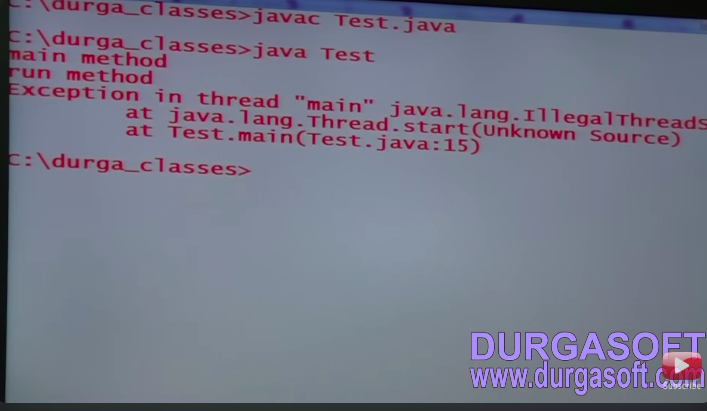
**Thread life cycle**



Case 7 :

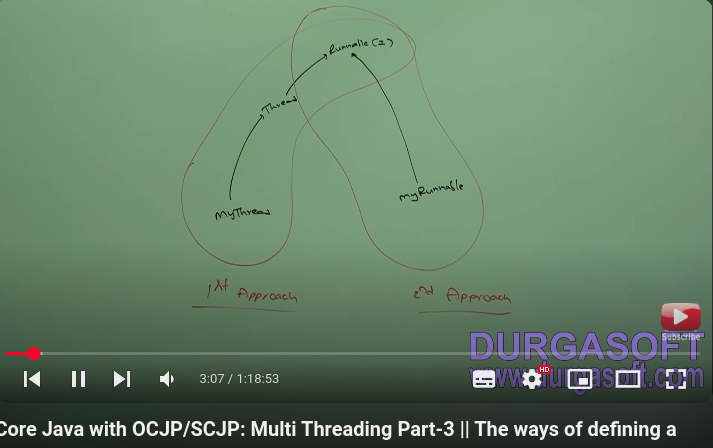
1. after starting a thread if we trying to restart the same thread then we will get run time exception saying illegal thread state exception.



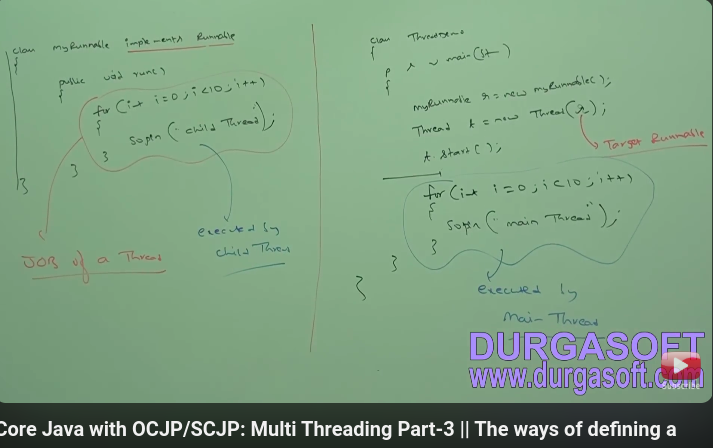


**Video 3 : The ways to defining a Thread**

1. **Define a thread by implementing Runnable interface.**



1. runnable interface present in java.lang package it only contain one method run method.



**we will get mixed output and we can’t tell exact output.**

**Case study**

**myRunnable r = new myRunnable();**

**Thread t1= new Thread();**

**Thread t2 = new Thread(r);**

**Case 1: t1.start();**

a new thread will be created and which is responsible for execution of thread class run method which has empty implementation.

**Case 2 : t1.run();**

no new thread will be created and thread class run will be executed just like normal method call.

**Case 3 : t2.start();**

a new thread will be created which is responsible for execution of myRunnable class run method

**Case 4 : t2.run();**

a new won’t be created and myRunnable run method will be executed just like normal method call.

**Case 5 : r.start();**

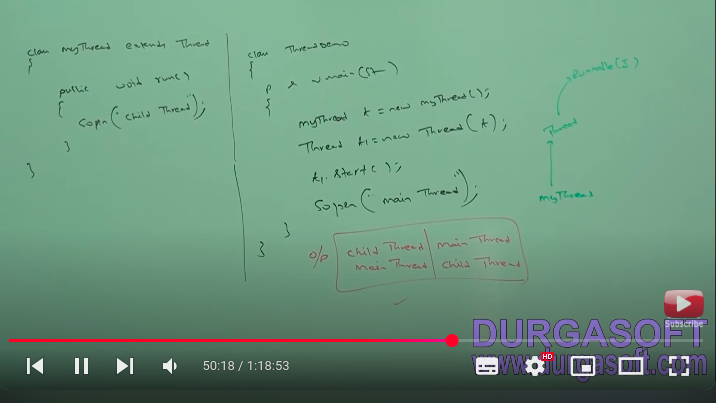
we will get compile time error saying myRunnable class doesn’t have start capability.

**compile time error - cannot find symbol method start() location class myRunnable**

**Case 6 : r.run();**

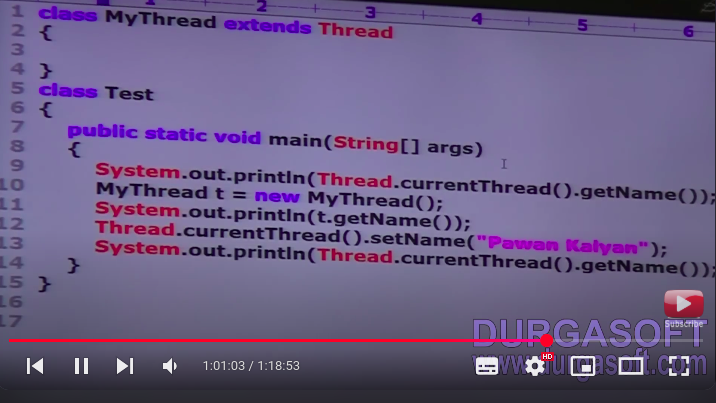
no new thread will be create and myRunnable run method will be executed like normal method call.

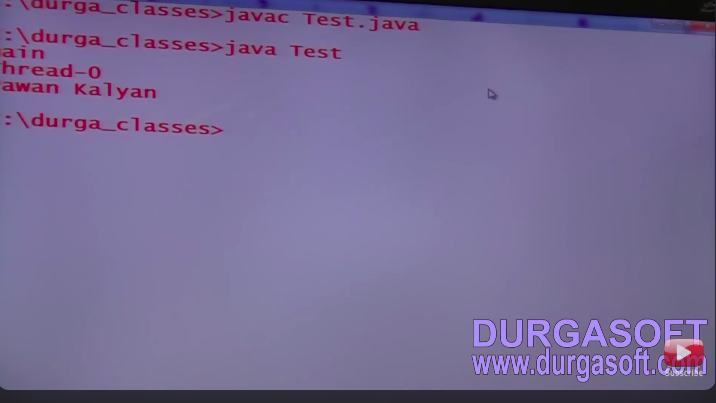
1. **which approach best to define a thread?**
   1. among two way to defining thread implement runnable approach is recommended.
   2. in the first approach our class our always extends thread class , there is not chance of extending any other class. Hence we are missing inheritance benefit.
   3. but in the second approach while implementing runnable interface we can extends any other class hence we won’t miss any inheritance benefit.
   4. because of above reason implementing runnable interface approach is recommended than extending thread class.
2. **Thread class constructors**
   1. Thread t = new Thread();
   2. Thread t = new Thread(Runnable r);
   3. Thread t = new Thread(String name);
   4. Thread t = new Thread(Runnable r,String name);
   5. Thread t = new Thread(ThreadGroup g , String name);
   6. Thread t = new Thread(ThreadGroup g, Runnable r);
   7. Thread t = new Thread(ThreadGroup g, Runnable r,String name);
   8. Thread t = new Thread(ThreadGroup g, Runnable r,String name,long statckSize);
3. **Durga’s approach to define a thread (not recommended to use)**



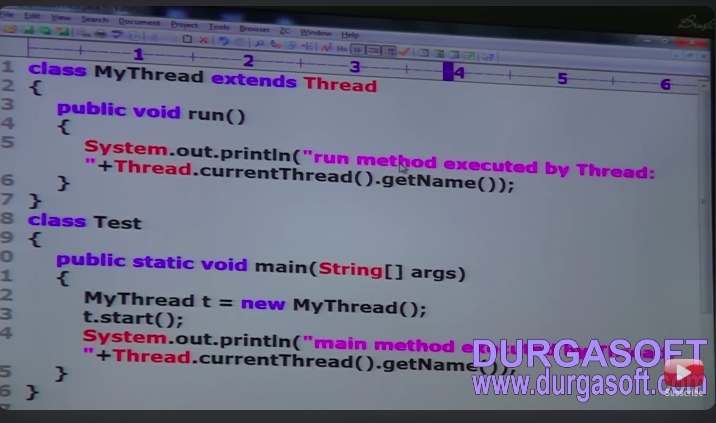
**Note: create Thread class extra object and pass the myThread class object in it.**

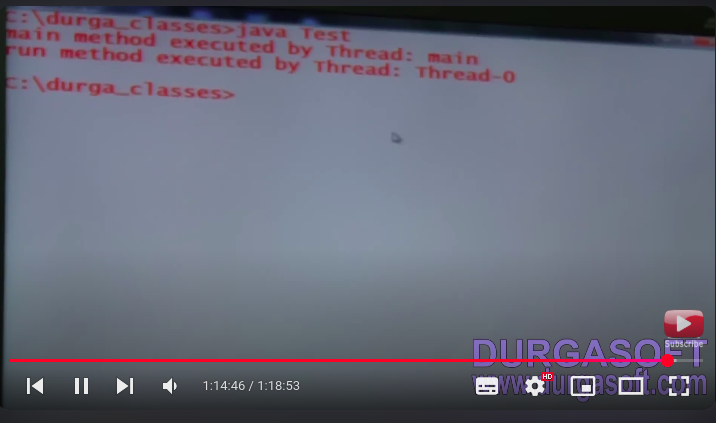
1. **Getting and setting name of the thread**
   1. every thread in java has some name it may be default name generated by JVM or customized name provided by programmer
   2. we can get and set name of the thread by using following the two method of thread class
      1. public final String getName();
      2. public final void setName(String name);





eg.2

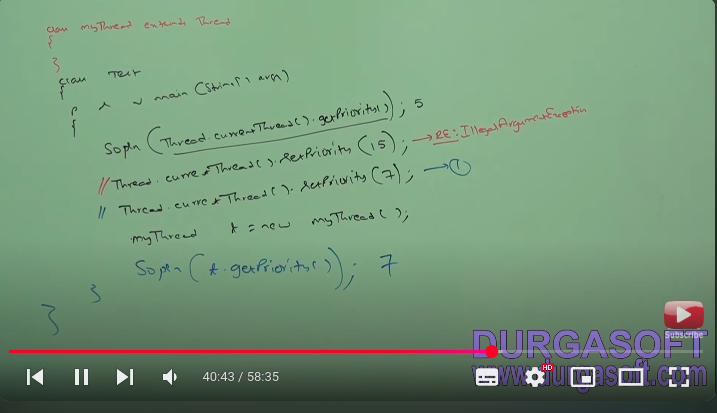


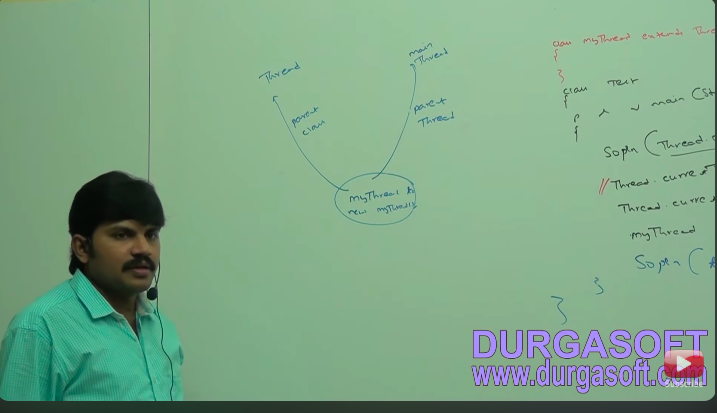


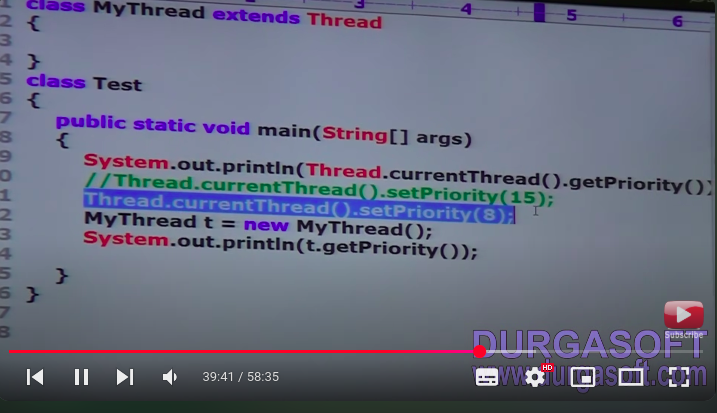
**Note : we can get current executing thread object by using thread.currentThread() method.**

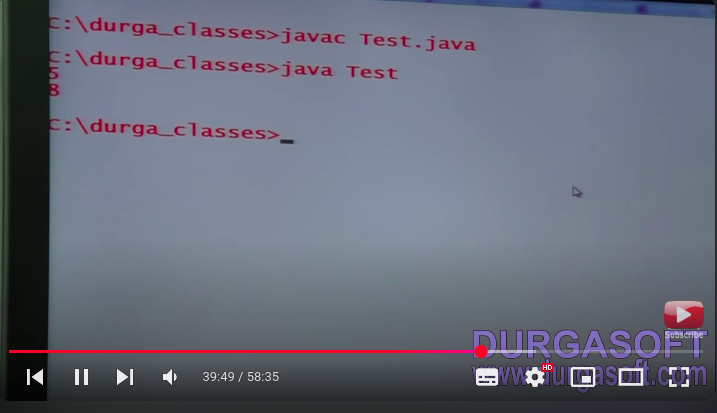
**Video 4 : Thread Priorities**

1. every thread in java has some priority it may be default priority generated by JVM or customized provided by programmer.
2. the valid range of thread priority is 1 to 10 where 1 is min priority and 10 is max priority.
3. thread class defines the following constant to represent some standard priorities:
   1. Thread.MIN\_PRIORITY ---------> 1
   2. Thread.NORM\_PRIORITY ------> 5
   3. Thread.MAX\_PRIORITY --------> 10
4. thread scheduler will use priorities while allocating processor.
5. the thread which is having highest priority will get change first.
6. if two thread having same priorities then we can’t expect exact execution order it depends on thread scheduler.
7. thread class define the following method to get and set priorities of thread:
   1. public final int getPriority();
   2. public final void setPriority(int p);
8. allowed value range 1 to 10 other wise RunTimeException - illegal argument exception.
   1. example
      1. t.setPriority(7); --- valid
      2. t.setPriority(17); --- illegal argument exception.
9. Default Priority : the default priority only for main thread is 5 but all remaining threads default priority inherited parent to child that is whatever parent thread has the same priority will be there for child thread.
10. **program 1:**

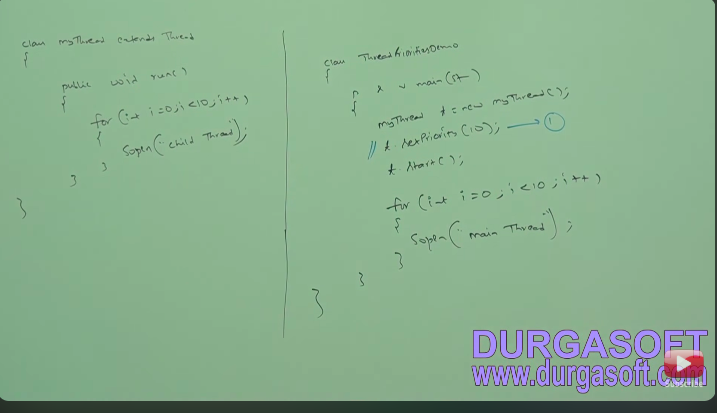








1. **program 2:**



**Note : if we commenting line 1 then both main and child threads have same priority 5 hence we can’t expect execution order and exact output.**

**if we are not commenting line 1 then main thread has priority 5 and child thread has the priority 10 hence child thread will get change first followed by main thread in this case output is child thread child thread 10 times followed by main thread main thread 10 times.**

**Note : some platforms won’t provide support for thread priorities.**

