***Multitasking****:- Executing several tasks simultaneously is a concept of Multitasking.*

*There are 2 types of Multitaskng:-*

1. ***Process-based:-*** *Executing several tasks simultaneously where,* ***each task is a separate independent program(process)*** *is called process-based multitasking.*

***Eg:****-While typing a java program on IDE we can listen sounds on same system, at same time we can download a file from internet, all these tasks will be executed simultaneously and independent of each other. Hence it is process-based.*

***Process-****based multitasking is best suitable at* ***OS level.***

1. ***Thread-based:-*** *Executing several tasks simultaneously where,* ***each task is a separate independent part of same program*** *is called process-based multitasking. Each independent part is called* ***Thread.***

***Thread-*** is best suitable at **Programmatic level**.

* *Whether it is process based or thread-*based the main objective is to reduce response time of system and improve performance.

The main important applications ares of Multithreading are:-

1. Develop multimedia graphics.
2. Develop animations
3. Develop video games.
4. Develop web/application servers,etc

When compared with old languages , developing multithreaded applications in Java is very easy because **Java provides inbuilt support for multithreading with rich API.(Thread, Runnable,ThreadGroup…)**

**We can define Thread in 2 ways:-**

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

**-----------------------------** By extending **Thread class -------------------------------------------------------**

***Case 01: Thread Scheduler:-***

* *It is responsible to schedule threads. i.e If multiple threads are waiting to get chance of getting executed, then in which order threads will be executed is decided by Thread scheduler.*
* *We can’t except exact algorithm followed by ThreadScheduler, it varies from JVM to JVM. Hence we cannot except thread execution order and exact output.*
* *Hence whenever situation comes to multithreading there is no guarantee of exact output, but we can provide several possible outputs.*

***Case 02: Difference t.start() and t.run() :-***

* *In case of t.start() a new thread will be created which is responsible for the execution of run method.*
* *But in case of t.run() a new thread wont be created and run() will be executed just like normal method called by main thread.*

***Case 03:******Importance of thread.start() in multithreading:-***

* *.start() is responsible to register thread with thread scheduler and all other mandatory activities. Hence, without executing thread.start() there is no chance of starting a new thread in java.*
* *Due to this thread class start() is considered as heart of multithreading.*
* *Once start() method is called -> (1) Register the thread with ThreadScheduler ->(2) Perform all other mandatory operations -> (3) Invoke run() method*

***Case 04:******Overloading of run() :-***

* *It is always possible but Thread class start() will invoke run method with no parameters only.*
* *We will have to call other overloaded method explicitly as normal method.*

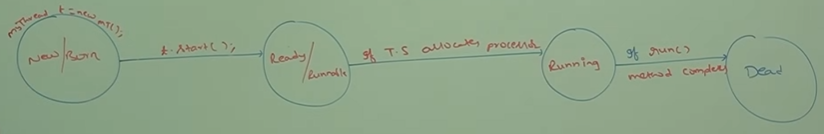
***Case 05: Overriding of run() :-***

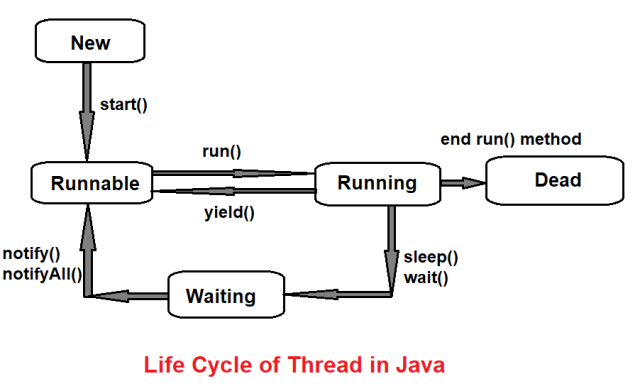
* *If we are not overriding run() method then, thread class run() method will be executed which has empty implementation. Hence, we wont get any output.*
* *It is highly recommended to override run() method otherwise don’t go for multithreading.*

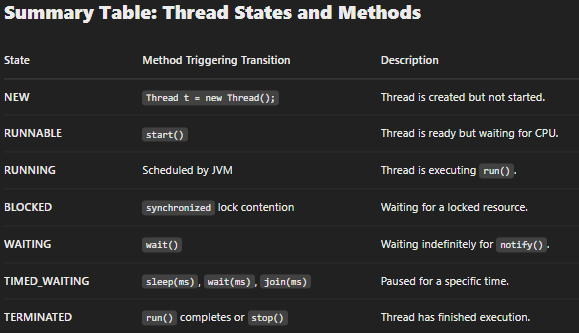
***Case 06: Overriding of start() :-***

* *If we override start() then our start() will be executed just like a normal method call and new Thread won’t be created. If we are not overriding start() then Thread class start() is invoked.*
* *It is not recommended to override start() otherwise , don’t go for Multithreading.*

***Case 07: Thread Lifecycle:-***





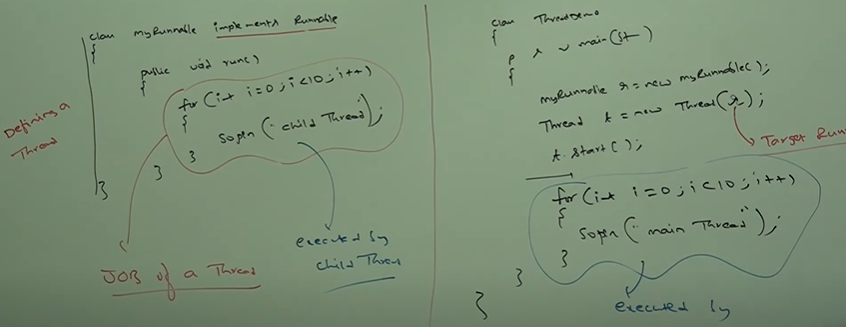


***Case 08:***

* *After starting a thread if we are trying to restart same thread then we will get* ***Runtime Exception saying IllegalThreadStateException.***

***------------------------------------------------*** By implementing **Runnable interface**.-----------------------------------

* *We can define a thread by* ***implementing Runnable Interface.***
* *Runnable interface present in java.lang package*
* *Contains only one method 🡪 run().*



*MyRunnable r=new MyRunnable();*

*Thread t1=new Thread();*

*Thread t2=new Thread(r);*

***Case 01:- t1.start()*** *🡪A new thread will be created which is responsible for execution of Thread class run() method. Which has empty implementation.*

***Case 02:- t1.run() 🡪*** *No new thread will be created and thread class run() method will be executed, just like normal method call.*

***Case 03:- t2.start()*** *🡪 A new thread will be created which is responsible for execution of MyRunnable class run() method.*

***Case 04:- t2.run()*** *🡪 No new thread will be created and MyRunnable run() method will be executed, just like normal method call.*

***Case 05:- r.start()*** *🡪 We will get Compile time error saying, MyRunnable class doesn’t have start capability.* ***CE: cannot find symbol***

***Case 06:- r.run()*** *🡪 No new thread will be created and MyRunnable run() method will be called.*

***Which approach is best to define a Thread?***

*🡪*

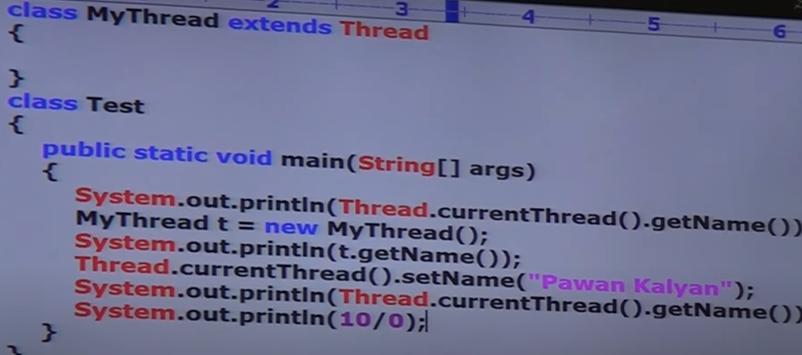
* *In Extends Thread approach there is no chance of extending any other class, hence we are missing Inheritance benefit.*
* *But in Implements Runnable Interface approach we can extend any other class hence we wont miss any inheritance benefit.*
* *Because of this reason Implementing Runnable Interface approach is best recommended*

***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 stackSize);*

***Getting and Setting name of Thread:-***

* *Every thread in java has some name, it maybe default name generated by JVM or Customised name provided by Programmer.*
* *We can set and get name of thread by using 2 methods of Thread class:-*
* ***Public final String getName();***
* ***Public final void setName(String name***



*The last line will generate exception as:* ***exception in thread Pawan Kalyan:***