**Multitasking: Executing several tasks simultaneously is the concept of multitasking.**

**There are two types of multitasking's.**

1. **Process based multitasking.**
2. **Thread based multitasking.**

***Process based multitasking:***

**Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called process based multitasking.**

**Example:**

1. **While typing a java program in the editor we can able to listen mp3**

**audio songs at the same time we can download a file from the net all these tasks are independent of each other and executing simultaneously and hence it is Process based multitasking.**

1. **This type of multitasking is best suitable at "os level".**

***Thread based multitasking:***

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

**And each independent part is called a "Thread".**

1. **This type of multitasking is best suitable for "programatic level".**
2. **When compared with "C++", developing multithreading examples is**

**very easy in java because java provides in built support for**

**multithreading through a rich API (Thread, Runnable,**

**ThreadGroup, ThreadLocal...etc).**

1. **In multithreading on 10% of the work the programmer is required to do and 90% of the work will be down by java API.**
2. ***The main important application areas of multithreading are:***
3. **To implement multimedia graphics.**
4. **To develop animations, video games, web and application servers.**
5. **Whether it is process based or Thread based the main objective of**

**Multitasking is to improve performance of the system by reducing response time.**

**The ways to define instantiate and start a new Thread:**

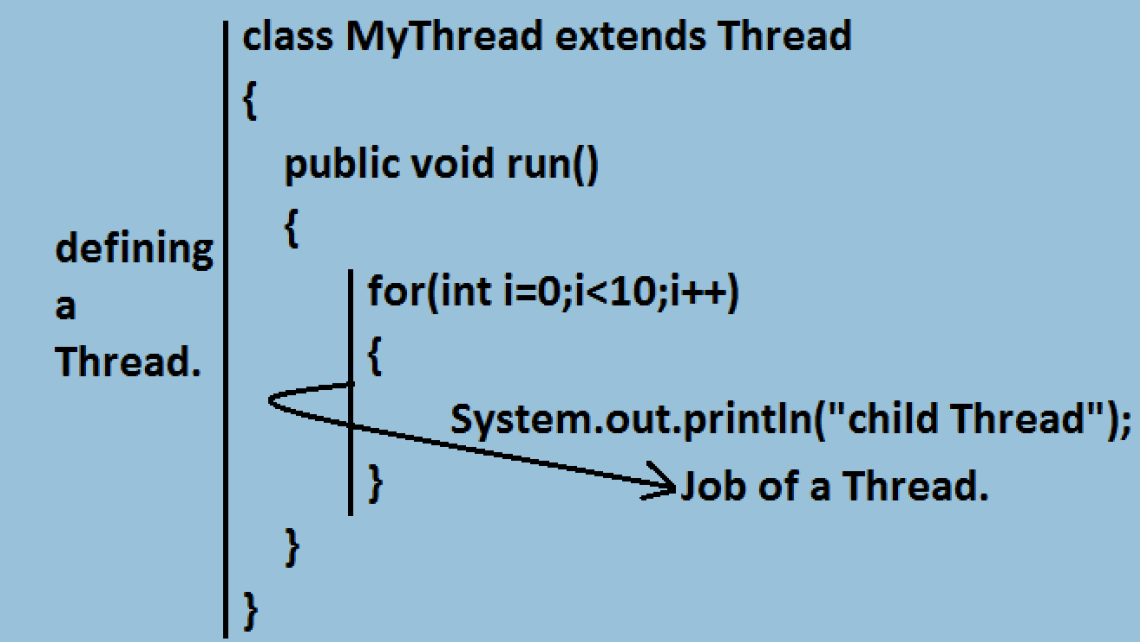
**What is singleton? Give example?**

**We can define a Thread in the following 2 ways.**

**1. By extending Thread class.**

**2. By implementing Runnable interface.**

**Defining a Thread by extending "Thread class":**



***Case 1: Thread Scheduler:***

1. **If multiple Threads are waiting to execute then which Thread will execute 1st is decided by "Thread Scheduler" which is part of JVM.**

2. **Which algorithm or behavior followed by Thread Scheduler we can't expect exactly it is the JVM vendor dependent hence in multithreading examples we can't expect exact execution order and exact output.**

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

1. **In the case of t.start() a new Thread will be created which is**

**responsible for the execution of run() method.**

1. **But in the case of t.run() no new Thread will be created and run()**

**method will be executed just like a normal method by the main**

**Thread.**

1. **In the above program if we are replacing t.start() with t.run() the following is the output.**

***Case 3: importance of Thread class start() method.***

**For every Thread the required mandatory activities like registering the Thread with Thread Scheduler will takes care by Thread class start() method and programmer is responsible just to define the job of the Thread inside run() method.**

**That is start() method acts as best assistant to the programmer.**

**Example:**

**start()**

**{1. Register Thread with Thread Scheduler**

**2. All other mandatory low level activities.**

**3. Invoke or calling run() method.**

**}**

**We can conclude that without executing Thread class start() method there is no chance of starting a new Thread in java. Due to this start() is considered as heart of multithreading.**

can we override start method in thread ?

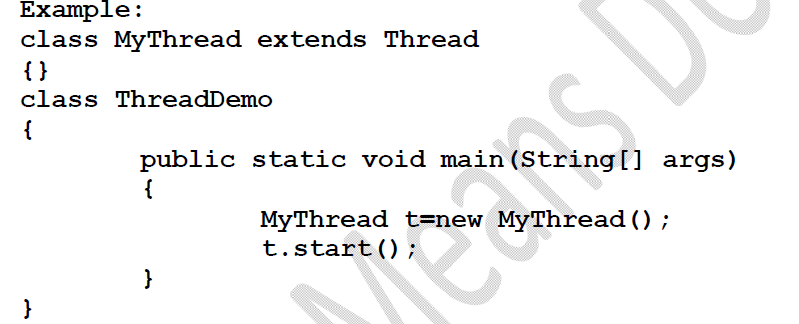
Whenever **we override start**() **method** then our **start**() **method will** be executed just like a normal **method** call and new **thread** wont be created. **We can override start**/**run method** of **Thread** class because it is not final. But it is not recommended to **override start**() **method**, otherwise it ruins multi-threading concept.

***Case 4: If we are not overriding run() method:***

**If we are not overriding run() method then Thread class run() method will be executed which has empty implementation and hence we won't get any output.**

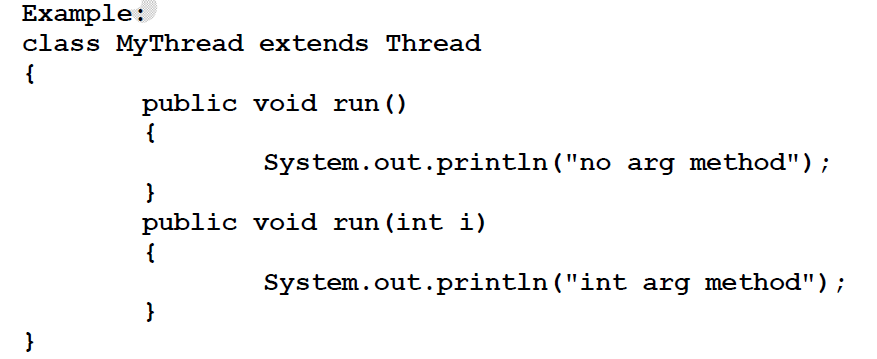
**It is highly recommended to override run() method. Otherwise don't go for**

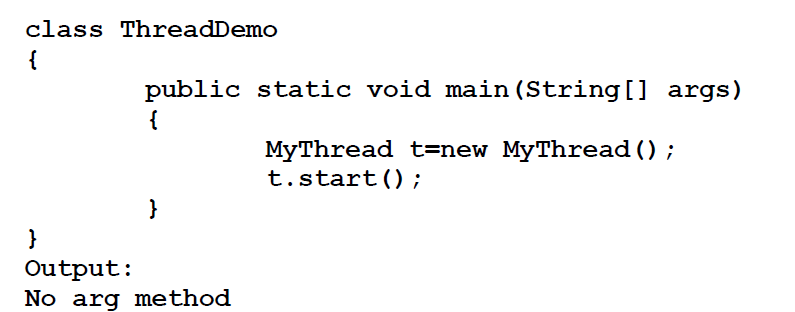
**multithreading concept.**



***Case 5: Overloding of run() method.***

**We can overload run() method but Thread class start() method always invokes no argument run() method the other overload run() methods we have to call explicitly then only it will be executed just like normal method.**



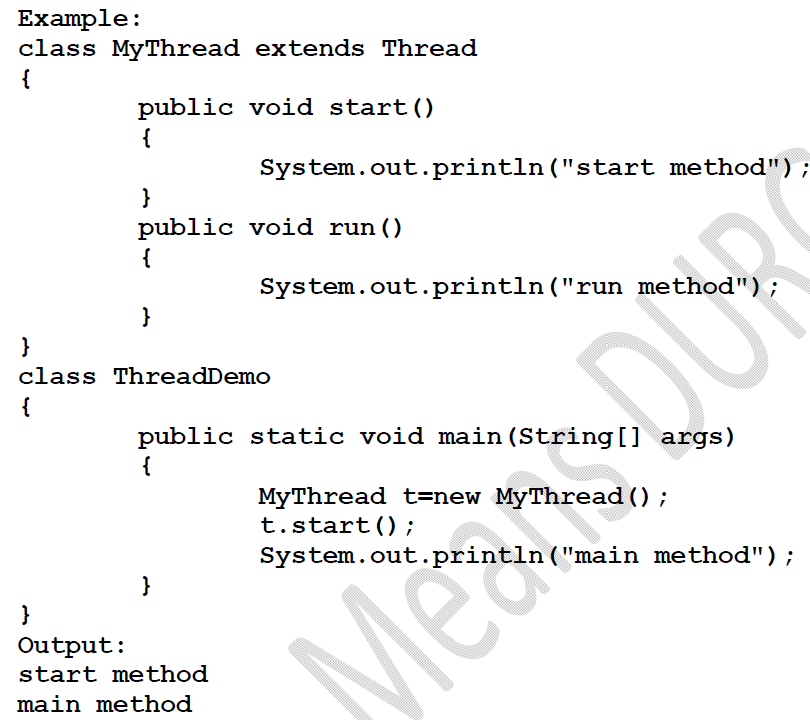


***Case 6: overriding of start() method:***

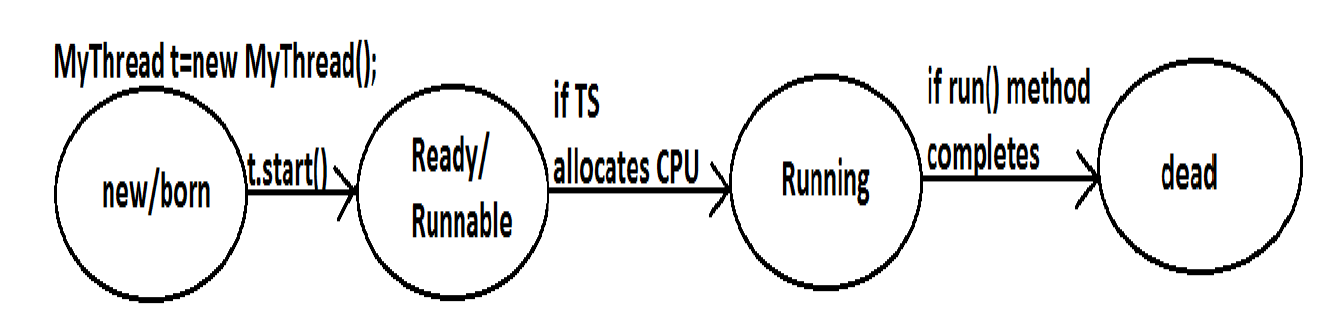
**If we override start() method then our start() method will be executed just like a normal method call and no new Thread will be started.**

**Entire output produced by only main Thread.**

**Note : It is never recommended to override start() method.**



***Case 8: life cycle of the Thread:***



1. **Once we created a Thread object then the Thread is said to be in new**

**state or born state.**

1. **Once we call start() method then the Thread will be entered into**

**Ready or Runnable state.**

**3.If Thread Scheduler allocates CPU then the Thread will be entered into running state.**

4. **Once run() method completes then the Thread will entered into dead state.**

***Case 9:***

**After starting a Thread we are not allowed to restart the same Thread once again otherwise we will get runtime exception saying "IllegalThreadStateException".**

**Example:**

**MyThread t=new MyThread();**

**t.start();//valid**

**;;;;;;;;**

**t.start();//we will get R.E saying: IllegalThreadStateException**

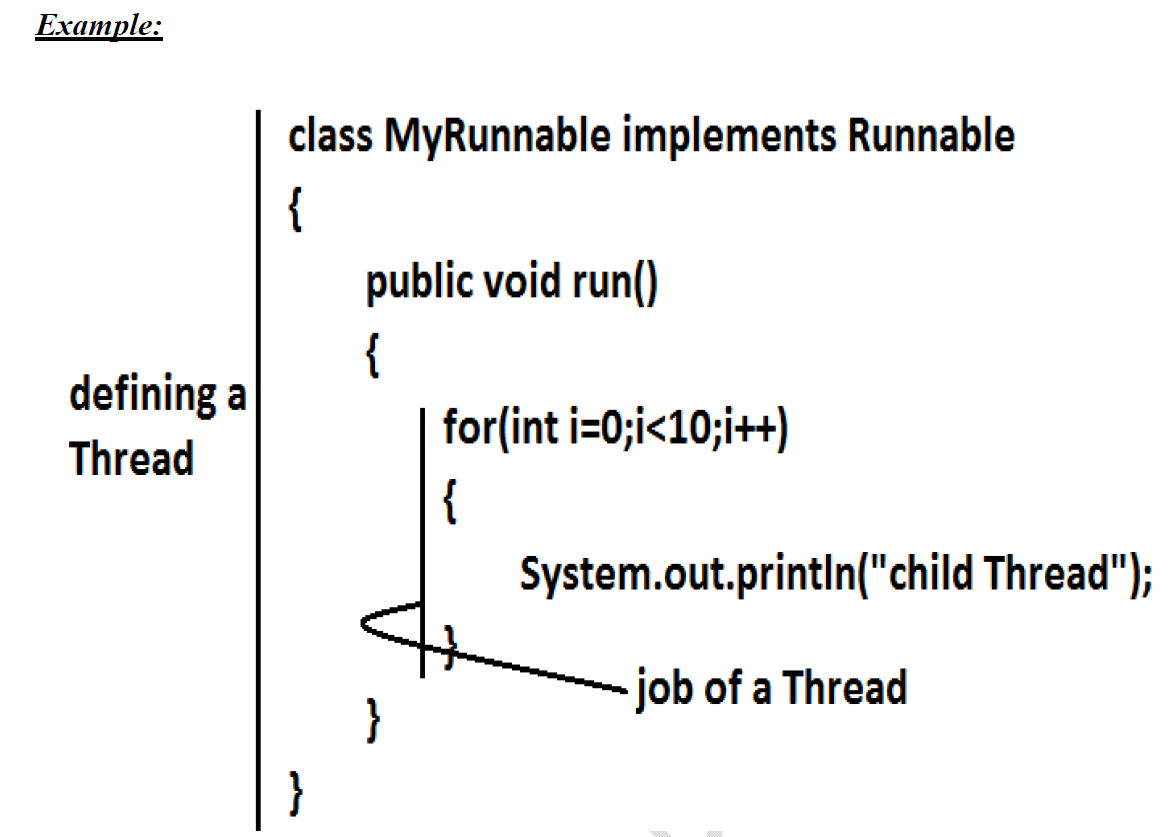
**Defining a Thread by implementing Runnable interface:**

**We can define a Thread even by implementing Runnable interface also.**

**Runnable interface present in java.lang.pkg and contains only one method run().**

***Diagram:***

******



**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 which is responsible for the execution of Thread class run()method.**

**Output:**

**main thread**

**main thread**

**main thread**

**main thread**

**main thread**

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

**No new Thread will be created but Thread class run() method will be executed just like a normal method call.**

**Output:**

**main thread**

**main thread**

**main thread**

**main thread**

**main thread**

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

**New Thread will be created which is responsible for the execution of MyRunnable run() method.**

**Output:**

**main thread**

**main thread**

**main thread**

**main thread**

**main thread**

**child Thread**

**child Thread**

**child Thread**

**child Thread**

**child Thread**

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

**No new Thread will be created and MyRunnable run() method will be executed just like a normal method call.**

**Output:**

**child Thread**

**child Thread**

**child Thread**

**child Thread**

**child Thread**

**main thread**

**main thread**

**main thread**

**main thread**

**main thread**

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

**We will get compile time error saying start()method is not available in MyRunnable class.**

**Output:**

**Compile time error**

**E:\SCJP>javac ThreadDemo.java**

**ThreadDemo.java:18: cannot find symbol**

**Symbol: method start()**

**Location: class MyRunnable**

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

**No new Thread will be created and MyRunnable class run() method will be executed just like a normal method call.**

**Output:**

**child Thread**

**child Thread**

**child Thread**

**child Thread**

**child Thread**

**main thread**

**main thread**

**main thread**

**main thread**

**main thread**

***Best approach to define a Thread:***

 **Among the 2 ways of defining a Thread, implements Runnable approach is always recommended.**

 **In the 1st approach our class should always extends Thread class there is no chance of extending any other class hence we are missing the benefits of inheritance.**

 **But in the 2nd approach while implementing Runnable interface we can extend some other class also. Hence implements Runnable mechanism is recommended to define a Thread.**

**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);**

****

**Output:**

**main method**

**run method**

**Getting and setting name of a Thread:**

 **Every Thread in java has some name it may be provided explicitly by the**

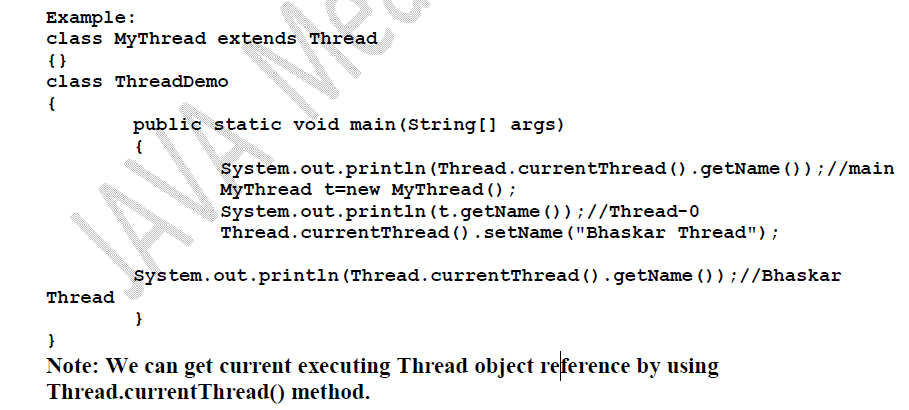
**programmer or automatically generated by JVM.**

 **Thread class defines the following methods to get and set name of a Thread.**

***Methods:***

**1. public final String getName()**

**2. public final void setName(String name)**



**Thread Priorities**

**Every Thread in java has some priority it may be default priority generated by**

**JVM (or) explicitly provided by the programmer.**

**The valid range of Thread priorities is 1 to 10[but not 0 to 10] where 1 is the least**

**priority and 10 is highest priority.**

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

**1. Thread. MIN\_PRIORITY----------1**

**2. Thread. MAX\_PRIORITY----------10**

**3. Thread. NORM\_PRIORITY--------5**

**There are no constants like Thread.LOW\_PRIORITY, Thread.HIGH\_PRIORITY**

**Thread scheduler uses these priorities while allocating CPU.**

**The Thread which is having highest priority will get chance for 1st execution.**

**If 2 Threads having the same priority then we can't expect exact execution order**

**it depends on Thread scheduler whose behavior is vendor dependent.**

**We can get and set the priority of a Thread by using the following methods.**

**1. public final int getPriority()**

**2. public final void setPriority(int newPriority);//the allowed values are 1 to 10**

**The allowed values are 1 to 10 otherwise we will get runtime exception saying**

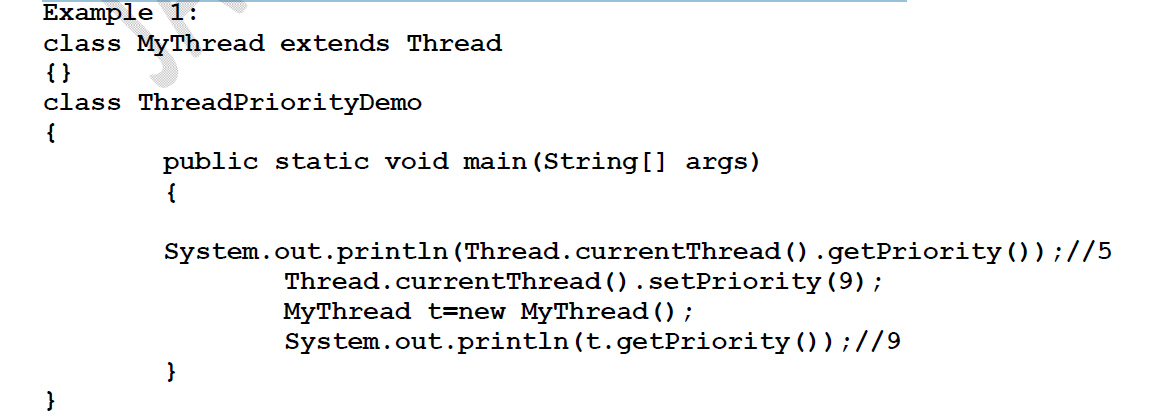
**"IllegalArgumentException".**

***Default priority:***

**The default priority only for the main Thread is 5. But for all the remaining Threads**

**the default priority will be inheriting from parent to child. That is whatever the priority**

**parent has by default the same priority will be for the child also.**



**The Methods to Prevent a Thread from Execution:**

**We can prevent(stop) a Thread execution by using the following methods.**

**1. yield();**

**2. join();**

**3. sleep();**