## Multi Threading

**Uni Programming:-**

* The earlier days the computer’s memory is occupied only one program after completion of one program it is possible to execute another program is called uni programming.
* Whenever one program execution is completed then only second program execution will be started such type of execution is called co operative execution, this execution we are having lot of disadvantages.
  + 1. Most of the times memory will be wasted.
    2. CPU utilization will be reduced because only program allow executing at a time.
    3. The program queue is developed on the basis co operative execution

**To overcome above problem a new programming style will be introduced is called multiprogramming.**

* 1. Multiprogramming means executing the more than one program at a time.
  2. All these programs are controlled by the CPU scheduler.
  3. CPU scheduler will allocate a particular time period for each and every program.
  4. Executing several programs simultaneously is called multiprogramming.
  5. In multiprogramming a program can be entered in different states.
     + 1. New state
       2. Ready state.
       3. Running state.
       4. Waiting state.
       5. Dead state
  6. Multiprogramming mainly focuses on the number of programs.

**Advantages of multiprogramming:-**

1. The main advantage of multithreading is to provide simultaneous execution of two or more parts of a application to improve the CPU utilization.
2. CPU utilization will be increased.
3. Execution speed will be increased and response time will be decreased.
4. CPU resources are not wasted.

**Thread:-**

* Thread is nothing but separate path of sequential execution.
* The independent execution technical name is called thread.
* The thread is light weight process because whenever we are creating thread it is not occupying the separate memory it uses the same memory. Whenever the memory is shared means it is not consuming more memory.
* Executing more than one thread a time is called multithreading.

**Multithreaing is process of executing more than one thread simultaneously.**

**Process-based Multitasking (Multiprocessing):**

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

**Thread-based Multitasking (Multithreading)**  Threads share the same address space.

* Thread is lightweight.
* Cost of communication between the thread is low.  At least one process is required for each thread.

There are two types of programming models 1. Single threaded model.

2. Multi threaded model.

### Single threaded model:-

* In java whenever execution starts from main thread one thread started automatically is called main thread.
* The below example continas only one thread is available is called main thread.

class Test

{ public static void main(String[] args) **//main thread starts**

{ System.out.println("Hello World!"); **//main thread executing**

} **//main thread ended**

}

**The main important application areas of the multithreading are:-**

1. Developing video games
2. Implementing multimedia graphics.
3. Developing animations

**There are two ways to create a thread in java,** 1) By extending Thread class.

2) By implementing java.lang.Runnableinterface

#### First approach to create thread extending Thread class:-

|  |  |
| --- | --- |
| **Step 1:-** | Our normal java class will become Thread class whenever we are extending predefined Thread class.  class MyThread extends Thread  {  }; |
| **Step 2:-** | override the run() method to write the business logic of the Thread. run() method present in Thread class with empty implementations.  class MyThread extends Thread { public void run()  { //logics here  }  } |
| **Step 3:-** | Create userdefined Thread class object. |
|  | MyThread t=new MyThread(); |
| **Step 4:-** | Start the Thread by using start() method of Thread class. |
|  | t.start(); |

**Example-1:-**

class MyThread extends Thread

{ **//business logic of user defined Thread**

public void run() { for (int i=0;i<10;i++)

{ System.out.println("userdefined Thread");

}

}

};

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t=new MyThread();

t.start();

**//business logic of main Thread**  for (int i=0;i<10;i++)

{ System.out.println("Main Thread");

}

}

};

**Example-2 : Difference between t.start() and t.run():-**

* In the case of t.start(), Thread class start() is executed a new thread will be created that is responsible for the execution of run() method.
* But in the case of t.run() method, no new thread will be created and the run() is executed like a normal method call by the main thread.

#### First application Ffow of execution:-

* Whenever we are calling t.start() method then JVM will search start() method in the MyThread class since not available so JVM will execute parent class**(Thread)** start() method.

**Thread class start() method responsibilities**

* 1. It will register User defined thread is into Thread Scheduler then only decide new Thread is created.
  2. After creation of thread s start() automatically calls run() to execute logics of userdefined Thread.

**Thread Scheduler:-**

* If the application contains morethan one thread then thread execution decided by threadschedular.  Thread scheduler is a part of the JVM. It decides thread execution.
* Thread scheduler is a mental patient we are unable to predict exact behavior of Thread Scheduler it is JVM vendor dependent.
* Thread Scheduler fallows two algorithms to decide Thread execution.
  + - 1. Preemptive algorithm.
      2. Time slicing algorithm.
* We can’t expect exact behavior of the thread scheduler it is JVM vendor dependent. So we can’tsay expect output of the multithreaded examples we can say the possible outputs.

**Preemptive scheduling:-**

In this highest priority task is executed first after this task enters into waiting state or dead state then only another higher priority task come to existence.

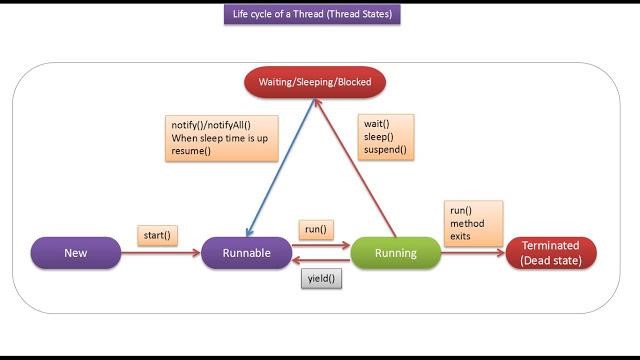
**Time Slicing Scheduling:-**

A task is executed predefined slice of time and then return pool of ready tasks. The scheduler determines which task is executed based on the priority and other factors.

#### Life cycle stages are:- 1) New

1. Ready
2. Running state
3. Blocked / waiting / non-running mode **5)** Dead state

|  |  |
| --- | --- |
| **New :-** | MyThread t=new MyThread(); |
| **Ready :-** | t.start() |
| **Running state:-** | If thread scheduler allocates CPU for particular thread then thread goes to running state. The Thread is running state means the run() is executing. |
| **Blocked State:-** | If the running thread got interrupted of goes to sleeping state at that moment it goes to the blocked state. |
| **Dead State:-** | If the business logic of the project is completed means run() over thread goes dead state. |



**Second approach to create thread by implementing Runnable interface:-**

**Step 1:**Our normal java class will become Thread class whenever we are implementing Runnable interface.class MyRunnable extends Runnable

{

};

**Step2:** override run method to write logic of Thread. class MyClass extends Runnable

{ public void run()

{ System.out.println("Rattaiah from SravyaInfotech");

System.out.println("body of the thread");

}

}

|  |  |
| --- | --- |
| **Step 3:-** | Creating a object. |
|  | MyClass obj=new MyClass(); |
| **Step 4:-** | start the thread thread by using Thread class start method. |

**Example -3 : creation of Thread implementing Runnable interface :-** class MyThread implements Runnable

{ public void run()

{ **//business logic of user defined Thread**

for (int i=0;i<10;i++)

{ System.out.println("userdefined Thread");

}

}

};

class ThreadDemo

{ public static void main(String[] args)

{ MyThread r=new MyThread();  Thread t=new Thread(r);

t.start();

**//business logic of main Thread**  for (int i=0;i<10;i++)

{ System.out.println("Main Thread");

}

}

};

There are two approaches to create a thread but the recammanded approach is implementing Runnable interface.

**First approach:-**

Iimportant point is that when extending the Thread class, the sub class cannot extend any other base classes because Java allows only single inheritance. **Second approach:-**

* Implementing the Runnable interface does not give developers any control over the thread itself, as it simply defines the unit of work that will be executed in a thread.
* By implementing the Runnable interface, the class can still extend other base classes if necessary.

**Example -4**Creating two threads by extending Thread class using anonymous inner classes class ThreadDemo

{ public static void main(String[] args)

{ **//anonymous inner class**

Thread t1 = new Thread()

{ public void run()

{ System.out.println("user Thread-1");

}

};

t1.start();

**//anonymous inner class**

new Thread()

{ public void run(){System.out.println("user Thread-1"); }

}.start();

}

};

**Example-5**Creating two threads by implementing Runnable interface using anonymous inner classes class ThreadDemo

{ public static void main(String[] args)

{ **//application with anonymous inner classes** Runnable r1 = new Runnable()

{ public void run()

{ System.out.println("user Thread-1");

}

};

Thread t1 = new Thread(r1); t1.start();

**//application with anonymous inner classes**

new Thread(new Runnable()

{ public void run()

{ System.out.println("user Thread-1");

}

}).start();

**//application with lambda**

new Thread(()->System.out.println(“user thread”)).start();

}

};

**Example-6 :-**It is not possible to start same thread twice

class MyThread extends Thread

{ public static void main(String[] args)//main thread started

{ MyThread t=new MyThread(); //MyThread is created

t.start();

t.start();

}

};

D:\DP>java MyThread

Exception in thread "main" java.lang.IllegalThreadStateException

**Example-7 :-**

class MyThread extends Thread

{ public void run()

{ System.out.println("Thread is running.......");

}

public static void main(String[] args)

{ MyThread t = new MyThread(); t.start();

}

}

**Example-8:-**

class MyThread implements Runnable

{ public void run()

{ System.out.println("Thread is running.......");

}

public static void main(String[] args)

{ MyThread r = new MyThread(); Thread t = new Thread(r); t.start();

}

}

#### Internal Implementation of multiThreading:- interface Runnable

{ public abstract void run();

}

class Thread implements Runnable

{ public void run()

{ **//empty implementation**

}

};

class MyThread extends Thread

{ public void run() **//overriding run() to write business logic** { //write the logics here

}

};

**Exampl-9** :  **sleep() method**

* sleep() method is a static method used to stop the thread particular amount of time.
* This method throws interrupted exception and it is a checked exception hence handle the checked exception by using try-catch blocks or throws keyword.

public static native void sleep(long) throws java.lang.InterruptedException; public static void sleep(long, int) throws java.lang.InterruptedException;

class MyThread extends Thread { public void run()

{ for (int i=0;i<10;i++)

{ System.out.println("Thread is running......."); try { Thread.sleep(1000); } catch(InterruptedException ie)

{ ie.printStackTrace(); }

}

}

public static void main(String[] args)

{ MyThread t = new MyThread(); t.start();

}

}

**Example-11 :-**

* It is recammanded to override run() method to write the logics of the thread.
* Here we are not overriding the run() method so thread class run method is executed which is having empty implementation so we are not getting any output.

class MyThread extends Thread

{ **//not overriding run() method**

}

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t=new MyThread(); t.start();

for (int i=0;i<5;i++ )

{ System.out.println("main thread");

}

}

}

**Example-12 :-**

* It is not recammanded to override start method.
* If we are overriding start() method then JVM is executes override start() method thread is not created.(thread is created oly when thread class start method is excuted)

class MyThread extends Thread

{ Public void start()

{ System.out.println(“override start method”);

}

}

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t=new MyThread(); t.start();

for (int i=0;i<5 ;i++ )

{ System.out.println("main thread");

}

}

}

**Example-13 :-** It is possible to overload run() but JVM always calling 0-arg run method.

class MyThread extends Thread

{ public void run()

{ System.out.println("run o-arg method");

}

public void run(int a)

{ System.out.println("run 1-arg method");

}

}

**Example-14 :-** possible to write the logics in different method then just call those methods in run() method. class MyThread extends Thread

{ public void run()

{ m1();

m2(); m3();

}

void m1(){System.out.println("m1 method");} void m2(){System.out.println("m2 method");} void m3(){System.out.println("m3 method");}

}

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t=new MyThread(); t.start();

}

}

**Example-15 :**

* Here all thread are performing same tasks.
* JVM will create separate stack memory for each and every thread.
* The below application contains four thread hence JVM will create four stack memories. class MyThread extends Thread

{ public void run()

{ System.out.println("durgasoft task");

}

}

class ThreadDemo

{ public static void main(String[] args)//main Thread is started

{ MyThread t1=new MyThread();

MyThread t2=new MyThread();

MyThread t3=new MyThread();

t1.start(); t2.start(); t3.start();

}

}

**Example-16 :-**

* Here different thread are performing different tasks.
* JVM will create separate stack memory for each and every thread.
* The below application contains four thread hence JVM will create four stack memories. class MyThread1 extends Thread

{ public void run()

{ System.out.println("balu task");

}

};

class MyThread2 extends Thread

{ public void run()

{ System.out.println("Sravya task");

}

};

class MyThread3 extends Thread

{ public void run()

{ System.out.println("anu task");

}

};

class ThreadDemo

{ public static void main(String[] args) //1- main Thread

{ MyThread1 t1 = new MyThread1();

MyThread2 t2 = new MyThread2(); MyThread3 t3 = new MyThread3();

t1.start(); //2 t2.start(); //3

t3.start();//4

}

};

**Example-17** :- There are two types of methods in java

**a.** Synchronized methods **b.**Non-synchronized methods

Only one thread is allows to access synchronized methods, these methods are thread safe methods but performance will be decreased.

More than one thread is allows to access non synchronized methods, these methods are not a thread safe methods but performance will be increased.

class A

{ public static **synchronized** void status(String msg)

{ for (int i=0;i<3;i++)

{ System.out.println("hi="+msg);

try{Thread.sleep(1000);}catch(InterruptedException ie){ie.printStackTrace();}

}

}

}

class MyThread1 extends Thread { public void run()

{ A.status("balu");

}

}

class MyThread2 extends Thread { public void run()

{ A.status("durga");

}

}

class MyThread3 extends Thread { public void run()

{ A.status("any");

}

}

class ThreadDemo

{ public static void main(String[] args)

{ new MyThread1().start(); new MyThread2().start(); new MyThread3().start();

}

}

**Status() method is non-synchronized:** G:\>java ThreadDemo hi=durga hi=balu hi=any hi=durga hi=balu hi=any hi=balu hi=durga hi=any

**status() method is synchronized:-** G:\>java ThreadDemo hi=balu hi=balu hi=balu hi=any hi=any hi=any hi=durga hi=durga hi=durga

**Example-18 :- Thread name& id & isAlive**

* Every Thread in java having name ,default name of the main thread is main & default name of user threads starts from **Thread-0**.

t1 -->Thread-0 t2 -->Thread-1 t3 -->Thread-2

* To set the name use setName() & to get the name use getName(),

Public final String getName()

Public final void setName(String name)

* To represent the current thread use currentThread() method of thread class. public static native java.lang.Thread currentThread();  To get id of a thread use getId()method. public long getId();
* To check the particular thread is running or not use isAlive() method. public final native boolean isAlive();

class MyThread extends Thread

{

}

class ThreadDemo

{ public static void main(String args[])

{ MyThread t1=new MyThread();

MyThread t2=new MyThread();

System.out.println("t1 Thread name="+t1.getName());

System.out.println("t2 Thread name="+t2.getName());

System.out.println(Thread.currentThread().getName());

t1.setName("balu");

t2.setName("anu");

Thread.currentThread().setName(“durga”);

System.out.println("t1 Thread name="+t1.getName());

System.out.println("t2 Thread name="+t2.getName());

System.out.println(Thread.currentThread().getName());

System.out.println("t1 Thread id="+t1.getId());

System.out.println("t2 Thread id="+t2.getId());

System.out.println(Thread.currentThread().getId());

System.out.println("t1 Thread alive or not="+t1.isAlive());

}

}

**Example-19 : Thread Priorities**

* In java every Thread has some property. It may be default priority provided be the JVM or customized priority provided by the programmer.
* Based on priority the thread scheddular allocates the cpu.
* The valid range of thread priorities is 1 – 10. Where 1 is lowest priority and 10 is highest priority.
* The default priority of main thread is 5 **NORM\_PRIORITY**. The priority of child thread is inherited from the parent.
* To represent the priority thread class contains three constants, **MIN\_PRIORITY = 1**

**NORM\_PRIORITY = 5**

**MAX\_PRIORITY = 10**

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

Public final int getPriority()

Public final void setPriority(int priority)

* **Thread priority decide when to switch from one running thread to another this process is called context switching.**
* If the more than one thread is having same priority then thread execution decide by thread scheduler.

class MyThread extends Thread

{ public void run()

{ System.out.println("current Thread name = "+Thread.currentThread().getName());

System.out.println("current Thread priority = "+Thread.currentThread().getPriority());

}

};

class ThreadDemo

{ public static void main(String[] args)//main thread started

{ MyThread t1 = new MyThread(); MyThread t2 = new MyThread(); t1.setPriority(Thread.MIN\_PRIORITY);

t2.setPriority(Thread.MAX\_PRIORITY);

t1.start(); t2.start();

}

};

**Case :** Priority range 1-10 if we set more than 10 JVM will generate IllegalArgumentException.

MyThread t1 = new MyThread();

t1.setPriority(15);**// java.lang.IllegalArgumentException**

**Example-20 : Java.lang.Thread.yield()**

* Yield() method causes to pause current executing Thread for giving the chance for waiting threads of same priority.
* If there are no waiting threads or all threads are having low priority then the same thread will continue its execution once again.

**Public static native void yield();**

class MyThread extends Thread { public void run()

{ for(int i=0;i<10;i++)

{ Thread.yield();

System.out.println("child thread");

}

}

}

class ThreadYieldDemo

{ public static void main(String[] args)

{ MyThread t1=new MyThread();

t1.start();

for(int i=0;i<10;i++)

{ System.out.println("main thread");

}

}

}

**Volatile:-**

* Volatile modifier is also applicable only for variables.
* If the values of a variable keep on changing such type of variables we have to declare with volatile modifier.If a variable declared as a volatile then for every Thread a separate local copy will be created.Every intermediate modification performed by that Thread will take place in local copy instead of master copy.
* Once the value got finalized just before terminating the Thread the master copy value will be updated with the local stable value. The main advantage of volatile modifier is we can resolve the data inconsistency problem.
* But the main disadvantage is creating and maintaining a separate copy for every Thread  Increases the complexity of the programming and effects performance of the system.

**Example-21 : Java.lang.Thread.join(-,-) method**

* + Join method allows one thread to wait for the completion of another thread.
  + Join() method throws interrupted exception it is a checked exception hence handle the checked exception by using try-catch blocks or throws keyword.

**public final void join()throws InterruptedExcetion**

**Public final void join(long ms)throws InterruptedException**

class MyThread extends Thread { public void run()

{ for (int i=0;i<5;i++)

{ System.out.println(“user thread”);

try{ Thread.sleep(2000); }

catch(InterruptedException e)

{ e.printStackTrace();

}

}

}

};

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t1=new MyThread();

t1.start();

try

{t1.join(); }

catch (InterruptedException ie)

{ ie.printStackTrace();

}

**//logics of main thread**  for (int i=0;i<5;i++)

{ System.out.println(“main thread”);

try{ Thread.sleep(2000); }

catch(InterruptedException e)

{ e.printStackTrace();

}

}

}

};

**Example 22:**

#### Java.lang.Thread.Interrupted():-

* A thread can interrupt another sleeping or waiting thread.
* Whenever the thread is entered in sleeping mode then only that thread is interrupted, if the thread is not entered in sleeping mode the interrupted method call will be wasted.

#### Effect of interrupt() method call:- class MyThread extends Thread

{ public void run()

{ try

{ for (int i=0;i<10;i++ )

{ System.out.println("i am sleeping ");

Thread.sleep(5000);

}

}

catch (InterruptedException ie)

{ System.out.println("i got interupted by interrupt() call");

}

}

};

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t=new MyThread(); t.start();

t.interrupt();

}

};

**No effect of interrupt() call:-** class MyThread extends Thread { public void run()

{ for (int i=0;i<10;i++ )

{ System.out.println("i am sleeping ");

}

}

};

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t=new MyThread(); t.start();

t.interrupt();

}

};

**NOTE:- The interrupt() is effected whenever our thread enters into waiting state or sleeping state and if the our thread doesn’t enters into the waiting/sleeping state interrupted call will be wasted.**

**Example-23 : Hook Thread**

* Shutdown hook used to perform cleanup activities when JVM shutdown normally or abnormally.
* Clean-up activities like o Resource release o Database closing o Sending alert message
* So if you want to execute some code before JVM shutdown use shutdown hook

**The JVM will be shutdown in fallowing cases.**

* 1. When you typed ctrl+C
  2. When we used System.exit(int)
  3. When the system is shutdown ……etc

To add the shutdown hook to JVM use addShutdownHook(obj) method of Runtime Class.

**public void addShutdownHook(java.lang.Thread);**

To remove the shutdown hook from JVM use removeShutdownHook(obj) method of Runtime Class. **public boolean removeShutdownHook(java.lang.Thread);**

To get the Runtime class object use static factory method getRuntime() & this method present in Runtime class

Runtime r = Runtime.**getRuntime();**

**Factory method:- one java class method is able to return same class object or different class object is called factory method.**

class MyThread extends Thread

{ public void run()

{System.out.println("shoutdown hook");

}

};

class ThreadDemo

{ public static void main(String[] args)throws InterruptedException

{ MyThread t = new MyThread();

**//creating Runtime class Object by using factory method**

Runtime r = Runtime.getRuntime();

r.addShutdownHook(t); **//adding Thread to JVM hook**  for (int i=0;i<10 ;i++)

{System.out.println("main thread is running");

Thread.sleep(3000);

}

}

};

D:\DP>java ThreadDemo main thread is running main thread is running main thread is running shoutdown hook while running Main thread press Ctrl+C then hook thread will be executed.

**Example-24 : Daemon threads**

* The threads wchich are executed at background to give the support to foreground thread is called daemon threads.

Example:- garbage collector,ThreadSchedular.default exceptional handler….etc

* Generally in java we have two types of threads
  1. User thread.
  2. Daemon thread.
* Generally all threads are created by user are called user threads if want make the daemon thread use the fallowing method.

public final void setDaemon(boolean); used to specify the daemon thread.

public final boolean isDaemon(); to check the thread is daemo or not.

**Properties of daemon thread:-**

* These threads are executing background to give support to fore ground threads.
* These threads are low priority threads.
* Whenever user threads are completes it’s execution all daemon threads are automatically stopped.

class MyThread extends Thread { public void run()

{ for (int i=0;i<10 ;i++)

{ System.out.println("Daemon Thread......"); try{Thread.sleep(1000);} catch(InterruptedException ie)

{ ie.printStackTrace(); }

}

}

};

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t = new MyThread();

t.setDaemon(true);//setting daemon nature to Thread

t.start();

//main thread logic for (int i=0;i<5 ;i++)

{ System.out.println("main Thread......"); try{Thread.sleep(1000);}

catch(InterruptedException ie)

{ ie.printStackTrace(); }

}

}

};

#### synchronized blocks:-

* Synchronized vlock can be used to perform synchronization on any specific resource of the code.
* if the application method contains 100 lines but if we want to synchronized only 10 lines of code use synchronized blocks.
* The synchronized block contains less scope compare to method.

#### Syntax:-

synchronized(object)

{ //code

}

**Example 25 :** class Heroin

{ public void message(String msg)

{ synchronized(this){

System.out.println("hi "+msg+" "+Thread.currentThread().getName());

try{Thread.sleep(5000);}

catch(InterruptedException e){e.printStackTrace();}

}

System.out.println("hi Sravyasoft");

}

};

class MyThread1 extends Thread

{ Heroin h;

MyThread1(Heroin h)

{ this.h=h;

}

public void run()

{ h.message("Anushka");

}

};

class MyThread2 extends Thread

{ Heroin h;

MyThread2(Heroin h)

{ this.h=h;

}

public void run()

{ h.message("Balu");

}

};

class ThreadDemo

{ public static void main(String[] args)

{ Heroin h = new Heroin();

MyThread1 t1 = new MyThread1(h);

MyThread2 t2 = new MyThread2(h);

t1.start(); t2.start();

}

};

**Inter thread communication :**

* The process of communicating more than one thread each other is called inter thread communication.
* The threads are communicating each other by using three methods o wait() o notify() o notifyAll()

the above three methods are present in Object class.

Wait vs Sleep :-

**Example-26 :-**

class MyThread extends Thread

{ int total;

public void run()

{ synchronized(this){ for (int i=0;i<10 ;i++)

{ total=total+i;

}

notify();

}

}

}

class ThreadDemo

{ public static void main(String[] args)

{ MyThread t = new MyThread(); t.start();

synchronized(t)

{ System.out.println("MyThrad total is waiting for MyThread completion...");

try{

t.wait();}

catch(InterruptedException ie){System.out.println(ie);}

}

System.out.println("MyThrad total is ="+t.total);

}

};

**Thread group in java :-**

Java provides simple way to group more than one thread in a single object. In this case we can suspend , resume , interrupt group of threads in single method call.

**Example 27 :** class MyRunnable implements Runnable

{ public void run(){ System.out.println(Thread.currentThread().getName()); }

}

class ThreadDemo

{ public static void main(String[] args)

{ ThreadGroup tg1 = new ThreadGroup("Balu"); new Thread(tg1,new MyRunnable()).start(); new Thread(tg1,new MyRunnable()).start();

ThreadGroup tg2 = new ThreadGroup("anu"); new Thread(tg2,new MyRunnable(),"one").start(); new Thread(tg2,new MyRunnable(),"two").start();

}

}

**Example-28** : **Application with lambda**  class ThreadDemo

{ public static void main(String[] args)

{ ThreadGroup tg1 = new ThreadGroup("Balu");

new Thread(tg1,()->System.out.println(Thread.currentThread().getName())).start(); new Thread(tg1,()->System.out.println(Thread.currentThread().getName())).start();

ThreadGroup tg2 = new ThreadGroup("anu");

new Thread(tg2,()->System.out.println(Thread.currentThread().getName()),"one").start(); new Thread(tg2,()->System.out.println(Thread.currentThread().getName()),"two").start();

System.out.println(tg1.getName());

tg1.list();

}

}

**Example-29 : ThreadGroup interruption** class MyRunnable implements Runnable

{ public void run()

{ System.out.println(Thread.currentThread().getName());

try{Thread.sleep(1000);}catch(InterruptedException ie){ie.printStackTrace();}

}

}

class ThreadDemo

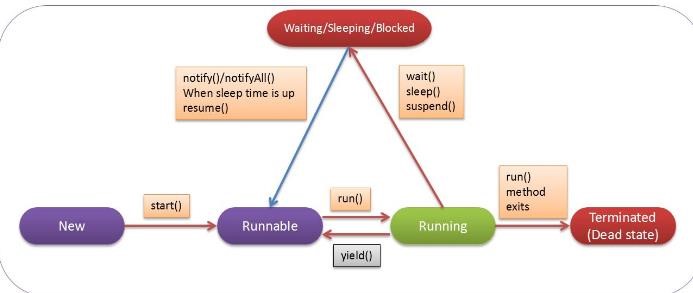
{ public static void main(String[] args)

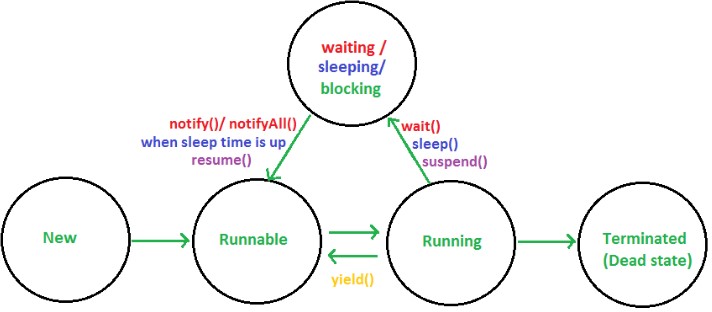
{ ThreadGroup tg1 = new ThreadGroup("Balu"); new Thread(tg1,new MyRunnable()).start(); new Thread(tg1,new MyRunnable()).start();

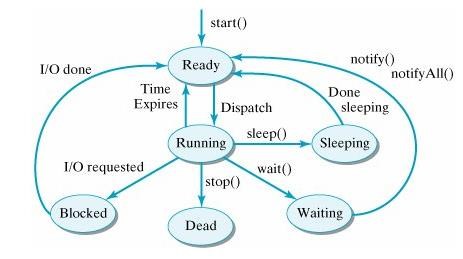
tg1.interrupt();

}

}







**Multithreading**

1. What do you mean by Thread?
2. What do you mean by single threaded model?
3. What is the difference single threaded model and multithreaded model?
4. What do you mean by main thread and what is the importance?
5. What is the difference between process and thread?
6. How many ways are there to create thread which one prefer?
7. Thread class& Runnable interface present in which package?
8. Runnable interface is marker interface or not?
9. What is the difference between t.start() & t.run() methods where t is object of Thread class?
10. How to start the thread?
11. What are the life cycle methods of thread?
12. Run() method present in class/interface ?Is it possible to override run() method or not?
13. Is it possible to override start method or not?
14. What is the purpose of thread scheduler?
15. Thread Scheduler fallows which algorithm?
16. What is purpose of thread priority?
17. What is purpose of sleep() & isAlive() & isDemon() & join() & getId() & activeCount() methods?
18. Jvm creates stack memory one per Thread or all threads only one stack?
19. What is the thread priority range & how to set priority and how to get priority?
20. What is the default name of user defined thread and main thread? And how to set the name and how to get the name?
21. What is the default priority of main thread?
22. Which approach is best approach to create a thread?
23. What is the difference between synchronized method and non-synchronized method?
24. What is the purpose of synchronized modifier?
25. What is the difference between synchronized method and non synchronized method?
26. What do you mean by demon thread tell me some examples?
27. what is the purpose of volatile modifier?
28. What is the difference between synchronized method and synchronized block?
29. Wait() notify() notifyAll() methods are present in which class?
30. When we will get Exception like “IllegalThreadStateException” ?
31. When we will get Exception like “IlleglArgumentException” ?
32. If two threads are having same priority then who decides thread execution?
33. How two threads are communicate each other?
34. What is race condition?
35. How to check whether the thread is demon or not? Main thread is demon or not?
36. How a thread can interrupt another thread?
37. Explain about wait() motify() notifyAll()?
38. Once we create thread what is the default priority?
39. What is the max priority & min priority & norm priority?
40. What is the difference between preemptive scheduling vs time slicing?