Multi Threading

Q1: What is multi tasking? Howmany types of multi tasking are there?

Executing several tasks simultaneously is the concept of multi tasking.

There are two types of multi tasking.

1. Process based multi tasking
2. Thread based multi tasking

Q2: What is Process based multi tasking?

Executing several tasks simultaneously, where each task is a separate independent process, such type of multi tasking is called process based multi tasking.

Ex: while typing a java program in the editor, at the same time we can able to listen to songs, we can download a file from internet…. All these tasks are executing simultaneously & independent of each other. This kind of process is nothing but process based multi tasking. And this kind of multi tasking is best suitable for operating system level.

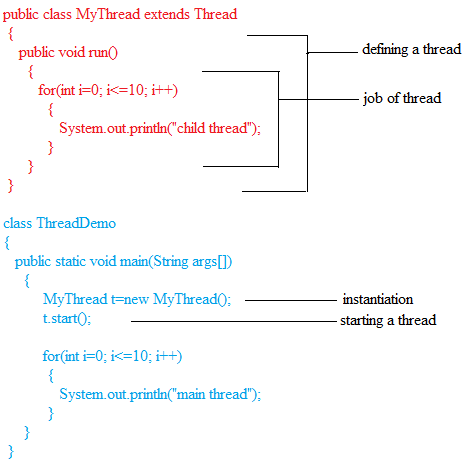
Q3: What is Thread based multi tasking?

1. Executing several tasks simultaneously, where each task is separate independent part of the same program. Such type of multi tasking is called thread based multi tasking.
2. This type of multi tasking is best suitable for programmatic level & each independent part is called a thread.
3. Java provides inbuilt support for multi threading by introducing a rich library. Ex: Thread, Runnable, Thread group etc..
4. Whether it is process based or thread based, the main objective(means duty) of multi tasking is to improve performance by reducing response time.
5. The main application areas of multi threading are developing video games & multi media graphics etc…

Q4: how many ways are there to define, instantiate and starting a thread?

There are two ways.

1. By extends Thread class.
2. By implements Runnable interface.

Q5:how to define, instantiate and starting a thread by extends Thread class

Q6:What is a thread scheduler?

If multiple theads are waiting for getting the chance of execution, then which tread will get chance decided by thread scheduler, which is the component of jvm, its behavior is vendor dependent. We can’t expect exactly the same output every time we run. The guarantee behavior is very low. To see the effect of thread scheduler, run the above program for multiple times and observe the output.

Q7:What is the difference between t.start() and t.run()?

1. In case of t.start(), a new thread will be created, which is responsible for the execution of run() method.
2. In case of t.run(), no new thread will be created, and run() method will be executed just like a normal method.

Q8:What is the importance of start() method?

1. The start() method present in thread class is responsible to perform all low level mandatory activities like registering the thread with thread scheduler.
2. After performing these activities, it will invoke run() method. So programmer can concentrate only on the job, instead of low level activities.
3. So we can say,

Start()

{

1.register this thread with thread scheduler, 2.invoke run() method.

}

Note: with out executing Thread class start() method, there is no chance of starting the new thread in java.

Note: if we are not overriding run() method, then Thread class’s run() method will be executed, which is empty implementation. hence we won’t get any output.

Note: if we override Thread class’s start() method then it will be executed just like a start() method i.e it won’t create any new thread. So it is not recommended to override Thread class’s start() method.

Ex:

Class MyThread extends Thread

{

}

Class Test

{

Public static void main(String args[])

{

MyThread t=new MyThread();

t.start();

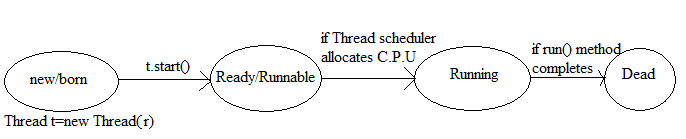
}

}

o/p is nothing

Note: it is highly recommended to override Thread classe’s run() method to define our job.

Q9:what is the life cycle of a Thread?



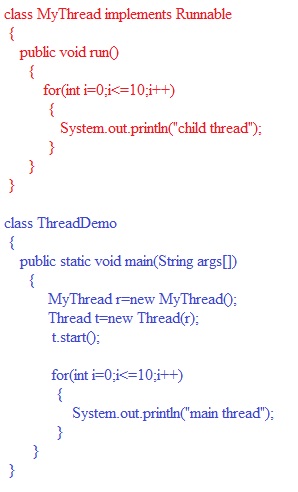
Note: after starting thread, we are not allowed to start the same thread once again, violation leads to IllegalThreadStateException.

Ex:

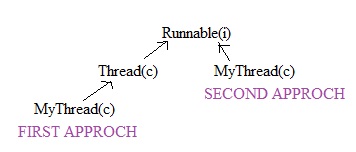
MyThread t=new MyThread();

t.start();

t.start(); -------> it raises IllegalThreadStateException

Q10:how to define, instantiate and starting a new thread by implements Runnable interface?

Note: observe the following diagram to get clarity



Q11: observe the following cases

Class MyThread implements Runnable

{

Public void run()

{

System.out.println(“child thread”);

}

}

MyThread r=new MyThread();

Thread t1=new Thread();

Thread t2=new Thread(r);

Case(i): t1.start();

A new thread will be created which is responsible for the execution of Thread class’s run() method.

Case(ii): t2.start();

A new thread will be created which is responsible for the execution of MyThread class’s run() method.

Case(iii): t1.run();

No new thread will be created and Thread class’s run() method will be executed as a normal method.

Case(iv): t2.run();

No new thread will be created and MyThread class’s run() method will be executed as a normal method.

Case(v): r.run();

No new thread will be created and MyThread class’s run() method will be executed as a normal method.

Case(vi): r.start();

Can’t find symbol.

Q12:for thread creation, which approach is the best and why?

Creating a thread by implementing Runnable interface is the best approach than by extending Thread class. Because in the first approach, our class already extending Thread class, so there is no chance of extending any other. Thus we are missing inheritance concept. That’s why it is not recommended to use.

Q13: there is a third approach but not recommended to use.

Class MyThread extends Thread

{

Public void run()

{

System.out.println(“child thread”);

}

}

Class ThreadDemo

{

Public static void main(String args[])

{

MyThread t=new MyThread();

Thread t1=new Thread(t);

. t1.start();

}

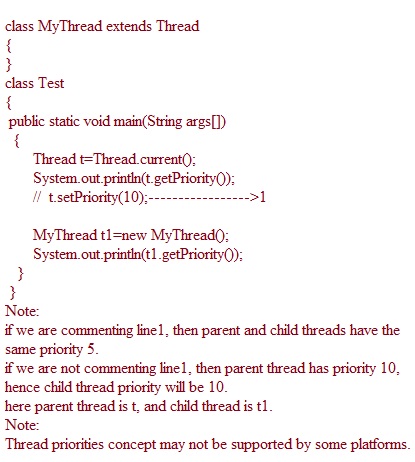
}

Q14:tell me about Thread priorities?

1. every thread in java has some priority. The range of thread priority is 1 to 10. Max priority is 10 and min priority is 1, normal priority is 5.
2. Generally thread priority is used by thread scheduler while allocating C.P.U, the thread which has highest priority will get first chance for execution.

Q15:tell me about default priority?

1. The default priority for only main method is five.
2. For all the remaining thread, it will be inherited from parent i.e whatever the parent thread’s priority will be the priority for their child thread.
3. Priority should be 1 to 10. Otherwise we will get IllegalArgumentException.
4. See the following example for clariry

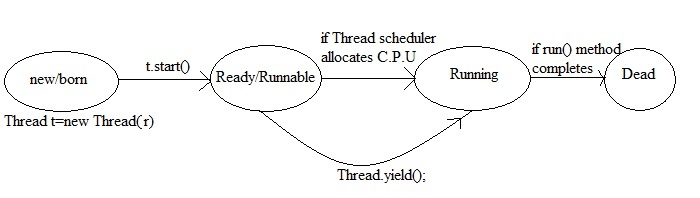


Q16:What are the methods to prevent a thread from execution?

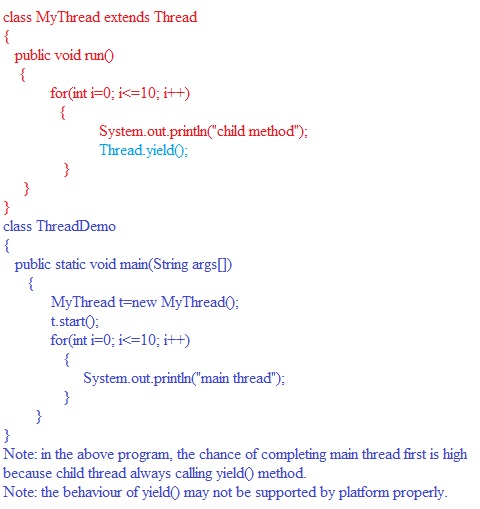
We can prevent a thread from execution by using three methods. They are yield(), join(), sleep().

Q17:tell me about the yield() method?

1. Yield() method pauses the currently executing thread temporarily and gives the chance for remaining waiting threads of same priority.
2. If there is no waiting thread (or) all the remaining threads have low priority then the same thread will get chance once again for execution.
3. Public static native void yield();
4. The temporary time depends upon mercy of thread scheduler.
5. Yield() method life cycle is the following

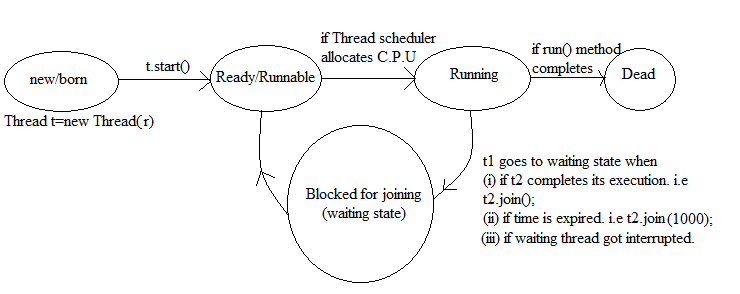


1. Example on yield() method

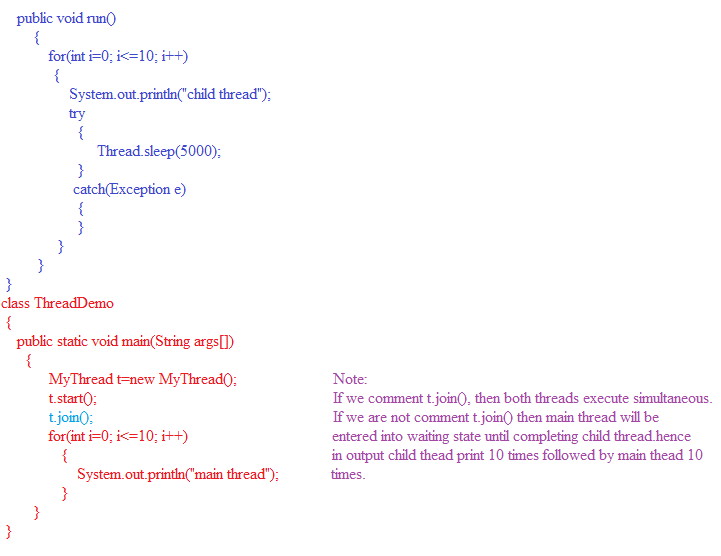


Q18:tell me about join() method?

1. If a thread wants to wait until completing some other threads, then we should go for join() method.
2. If a thread t1 executes t2.join(), then t1 will go waiting state until thread t2 completes.
3. Public final void join() throws InterruptedException
4. Public final void join(long ms) throws InterruptedException
5. Public final void join(long ms, int nanosec) throws InterruptedException
6. Whenever we call join() method, compulsory we should handle InterruptedException either by try-catch or throws keyword, otherwise we will get compile time error.
7. See this life cycle diagram



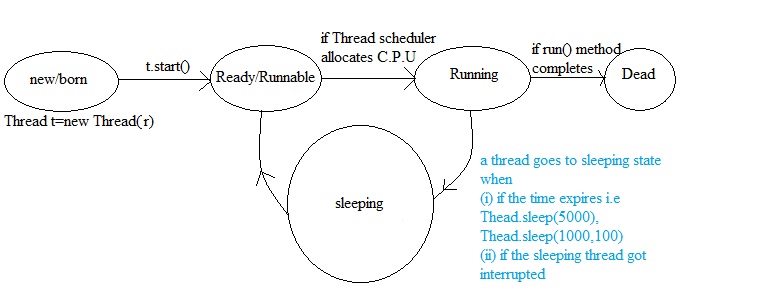
1. Example on join() method



Note: with in run(), if it is required to handle any checked exception then compulsory we should go for try-catch. (i.e we shouldn’t use throws).

Q19: tell me about sleep() method?

1. If a thread doesn’t want to perform any activity for a particular amount of time, then we should go for sleep() method.
2. Public static native void sleep(long ms)
3. Public static void sleep(long ms, int ns) throws InterruptedException
4. for example, we used sleep() method in the previous example. Check it out.
5. see the life cycle of sleep() method



Q20:interrupting a thread

A thread can interrupt another waiting or sleeping thread by using interrupt() method of Thread class. Public void interrupt();

ex:

class MyThead extends Thread

{

Public void run()

{

Try

{

For(int i=0; i<50; i++)

{

System.out.println(“I am lazy thread”);

}

Tread.sleep(1000);

}

Catch(Exception e)

{

System.out.println(“I am interrupted”);

}

}

}

Class ThreadDemo

{

Public static void main(String args[])

{

MyThread t1=new MyThread();

. t1.start();

. t1.interrupt();

System.out.println(“end of main”);

}

}

Note:

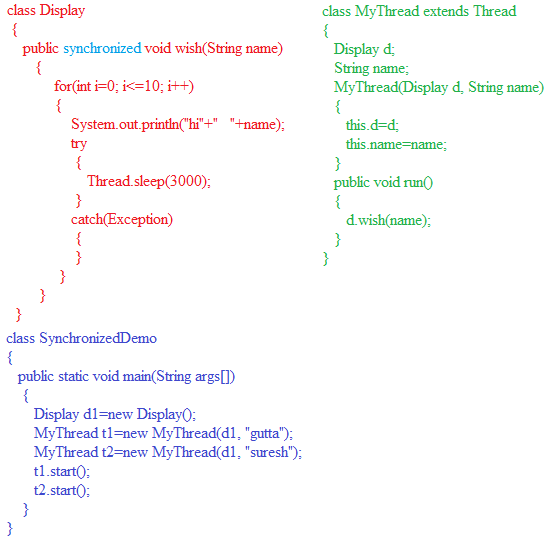
Whenever a thread calls interrupt() method, it may not effect immediately, if the target thread is not in sleeping or waiting state. Interrupt() will wait until target thread comes into sleeping or waiting state and then interrupt call will effect on that target thread.

Q21:comparision of yield(), join() and sleep()

|  |  |  |  |
| --- | --- | --- | --- |
| property | Yield() | Join() | Sleep() |
| Is it static? | yes | no | Yes |
| Is it final? | No | Yes | No |
| Is it overloaded? | No | Yes | Yes |
| Is it native? | Yes | No | Sleep(long ms), sleep(long ms, int ns) are native. |
| Is it throws InterruptException? | no | yes | Yes |

Q22:tell me about synchronization?

1. Synchronization is the keyword applicable for methods and blocks, but not applicable for variables and classes.
2. If a method declared as synchronized then at a time only one thread is allowed to execute that method on the given object.
3. The main advantage of the synchronized keyword is to prevent data inconsistency problem. But the main limitation is it increases the waiting time of thread and effects on performance of the system.hence until and unless there is no specific requirement, it is not recommended to use. It is dangerous.
4. Every object in java has the unique lock concept. Whenever we are using synchronized keyword, then this lock concept will come into picture.
5. If a thread wants to execute any synchronized method on the given object, first it should get the lock of the object.
6. Once a thread got the lock of object, then it is allowed to execute any synchronized method on that object.
7. While a thread executing any synchronized method on a object x, then remaining threads are not allowed to execute any synchronized method of that object simultaneously.
8. But remaining threads are allowed to execute any non synchronized methods simultaneously.
9. For example see the following



In the above example, if we are not declaring wish() method as synchronized, then both threads will be executed simultaneously and we will get irregular output.

If we are declaring wish() method as synchronized, then the threads will be executed one by one and there is no chance of executing simultaneously, hence we will get regular output.

Note:

Display d1=new Display();

Display d2=new Display();

MyThread t1=new MyThread(d1, “gutta”);

MyThread t2=new MyThread(d2, “suresh”);

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

o/p: we will get irregular output even though wish() method is synchronized. Because synchronization concept is applicable whenever we apply multiple threads simultaneously on the same object. But not on the separate objects.

Q23: What is class level lock?

1. Every class in java has a unique lock. If we want to execute any static synchronized method, thread requires class level lock.
2. Whenever a thread executing any static synchronized method, then no other threads are allowed to execute any static synchronized method of the same class simultaneously.
3. But remaining threads are allowed to execute the following methods simultaneously.
4. Non synchronized static method
5. Synchronized instance method
6. Non synchronized instance method

Q24:what is synchronized block?

1. If very few lines of code requires synchronization, then it is not recommended to declare entire method as synchronized. We have to declare those few lines of code inside synchronized block.
2. The main objective(means duty) of synchronized block is to improve the performance when compared with synchronized method.
3. We can declare synchronized block as follows

. synchronized( d )

{

-------

-------

}

Here d is object reference. This synchronized block code will be executed if and only if thread got the lock of d.

To get lock of current object, we have to declare synchronized block as follows.

. synchronized(this)

{

-------

-------

}

Here if thread get the lock of current obj, then only it will be executed.

To get class level lock, we have to declare synchronized block as follows.

synchronized(Display.class)

{

-------

-------

}

Here if thread get the class level lock of Display class, then only it will be executed.

Note:

Synchronized block concept is applicable for classes and objects, but not for primitives. If we use we will get compile time error.

. int i=10;

synchronized(i)

{

-------

-------

} // this block raises compile time error.

Q25:what is inter thread communication?

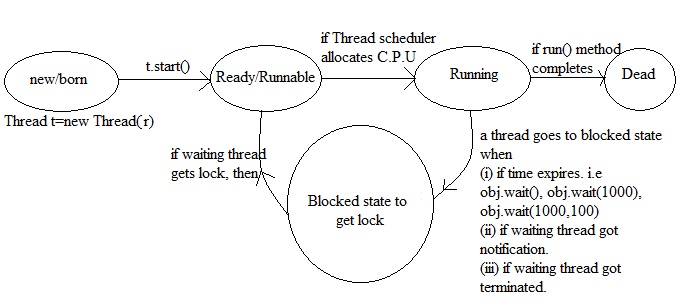
1. Two threads can communicate with each other by using wait(), notify(), notifyall() methods.
2. Wait(), notify(), notifyall() methods are defined in Object class, but not in Thread class. Because thread has to call these methods on any shared object.
3. To call wait(), notify(), notifyall() methods, compulsory the current thread should be owner of the object i.e the current thread should have the lock of object. i.e from synchronized area only we can call wait(), notify(), notifyall() methods. Otherwise we will get IllegalMoniterStateException.
4. Whenever a thread calls wait(), it releases the lock of that object(but not all locks) and entered into waiting state.
5. After giving the notification, thread releases lock but may not be immediately.
6. Frankly saying these three methods are there to release lock of their object.
7. Public final void wait() throws InterruptedException

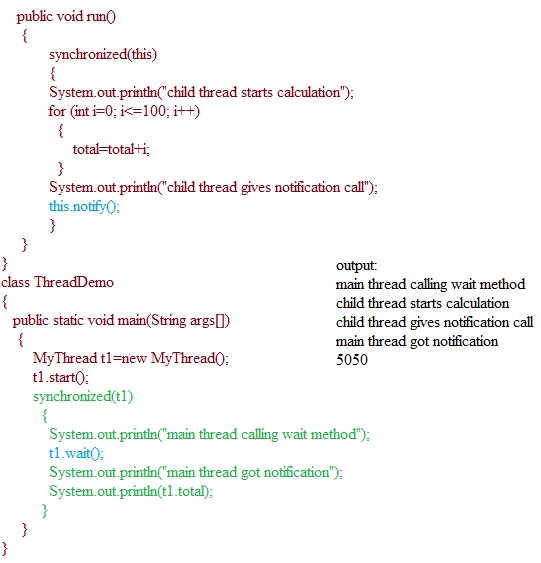
Public final native void wait(long ms) throws InterruptedException

Public final void wait(long ms, int ns) throws InterruptedException

Public final native void notify()

Public final native void notifyall()

1. See this thread life cycle diagram
2. See the following example



Q26: if waiting thread got notification, then it will enter into which of the following state?

1. Ready
2. Runnable
3. Dead state
4. Another waiting state

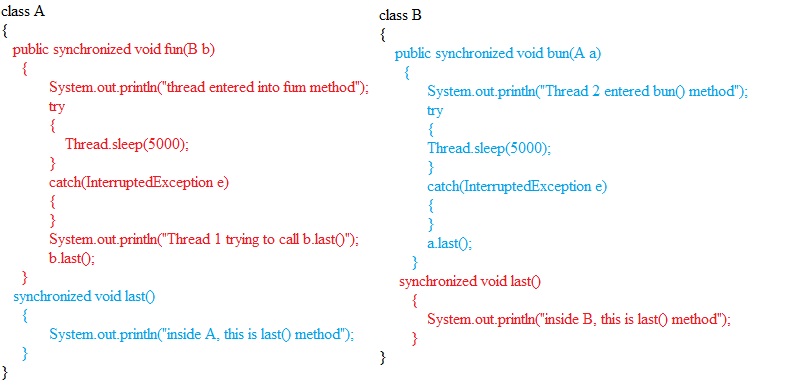
Answer: d

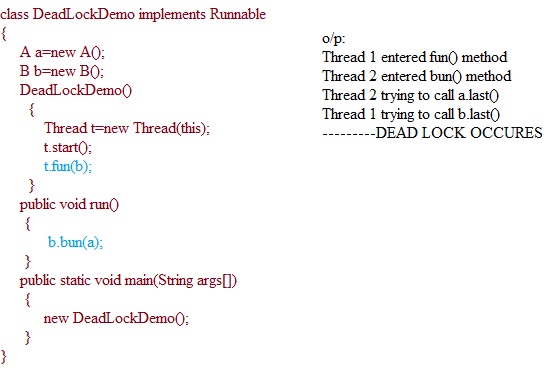
Q27:What is Dead lock?

If two threads are waiting for each other forever, then the threads are said to be in dead lock state.

for the dead lock, there is no solution technique, but several prevention techniques are possible.

See the following program





Note: if we are not using synchronized keyword properly, there may be a chance of entering the program into deadlock.

Q28: What is a Daemon Thread?

1. the threads which are running in the background are called Daemon threads.
2. The main purpose of Daemon threads is to provide support for non daemon thread.
3. Ex: Garbage Collector, Thread Scheduler
4. We can check whether the thread is daemon or not by using isDaemon() method of Thread class.
5. Main() is always non-daemon thread and for the remaining threads, Daemon nature will be inherited from parent to child.
6. If the parent is daemon, then child also will be daemon.
7. If the parent is non daemon, then child won’t be daemon either.
8. We can change daemon nature of a thread by

using public final void setDaemon(Boolean b);

1. We can change the daemon nature at before starting a thread, after starting a thread only but not at running stage thread. We can not change daemon nature of a main thread because it was started at the beginning. Usually daemon threads are running with low priority. But based on our requirement, we can provide high priority.
2. Whenever last non daemon thread terminates automatically all the daemon threads will be terminated.

See the following example

Class MyThread extends Thread

{

Public void run()

{

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

{

System.out.println(“I am a lazy thread”);

Try

{

Thread.sleep(2000);

}

Catch(InterruptedException e)

{

}

}

}

}

Class DaemonDemo

{

Public static void main(String args[])

{

MyThread t=new MyThread();

t.setDaemon(true);---------------->1

t.start();

}

System.out.println(“end of main thread”);

}

o/p:

if we are commenting line 1, then both main and child threads are non daemon and both will be executed until completion.

If we are not commenting line 1, then main thread is non daemon and child thread is daemon. Hence whenever main() thread completes automatically child thread will be terminated.

T H E E N D