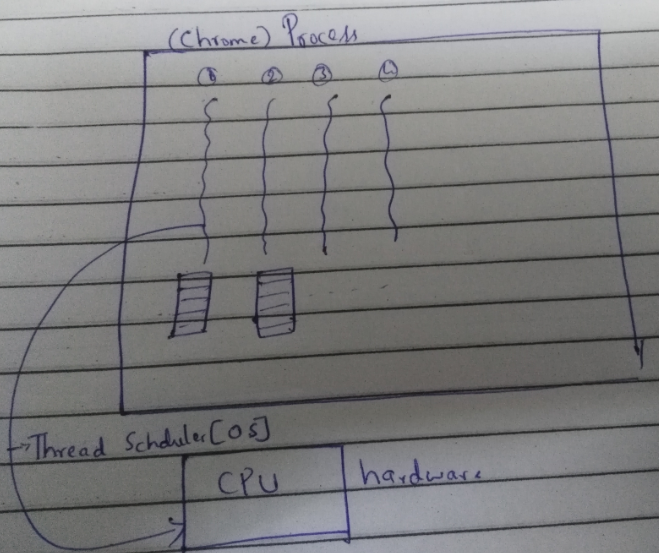
* Multi-Threading



1. **What is the difference between Process and Thread?**

**Process:**  It is an execution environment. Process is nothing but program or application.

**Thread:** Thread is single executing task within the process.

A thread is a lightweight.

Thread runs in a separate stack frame.

1. **What is multithreading and what are its benefits?**

Executing multiple threads simultaneously.

**Advantages:**

**Saves time**: As multiple threads are running together it saves time.

In case some thread is waiting to get some resources. CPU is not idle

**Threads are independent so it doesn't affect other** threads if exception occurs in a single thread.

Thread is lightweight.

Threads share the same address space.

1. **What about the daemon threads?**

Daemon thread provides services to the user thread.

Life cycle of Daemon threads is purely depending on of user threads i.e. when all user threads die, JVM terminates this thread automatically.

Ex: gc, finalizer etc

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc

1. **How to create daemon thread in Java?**

In Thread class there is a method setDaemon(true); if we invoke that method with true then invoking thread will become daemon thread.

We need to call this method before calling start() method else it will throw IllegalThreadStateException.

1. **What are user threads?**

Any thread we create in java is user thread.

1. **How can we create a Thread in Java?**

There are 2 ways,

First one: implement Runnable interface

Second one: extend the Thread Class

1. **Which one you prefer Implementing Runnable or extending Thread? Why?**

I prefer implementing Runnable over extending thread.

Why:

**Implementing Runnable is good object oriented design practice**. In object oriented programming, extending a class means modifying or improving the existing class. But we are not doing that in case of thred. So it’s not a good practice.

It promotes **loose coupling.**

One Class can only extend one class in Java. So if you extend the Thread class then your class lose that option and it cannot extend another class, but if you implement Runnable then your Thread class can still extend another class.

Another difference between Thread and Runnable comes from the fact that you are extending Thread class just for run() method but you will get overhead of all other methods which come from Thread class. So, if your goal is to just write some code in run () method for parallel execution then use Runnable instead of extending Thread class.

1. **What are different states in lifecycle of Thread? IMP**

A thread state. A thread can be in one of the following states:

**NEW**

A thread that has not yet started is in this state.

**RUNNABLE**

A thread executing in the Java virtual machine is in this state.

**BLOCKED**

A thread that is blocked, waiting for a monitor lock is in this state.

**WAITING**

A thread that is waiting for another thread to perform a particular action indefinitely , is in this state.

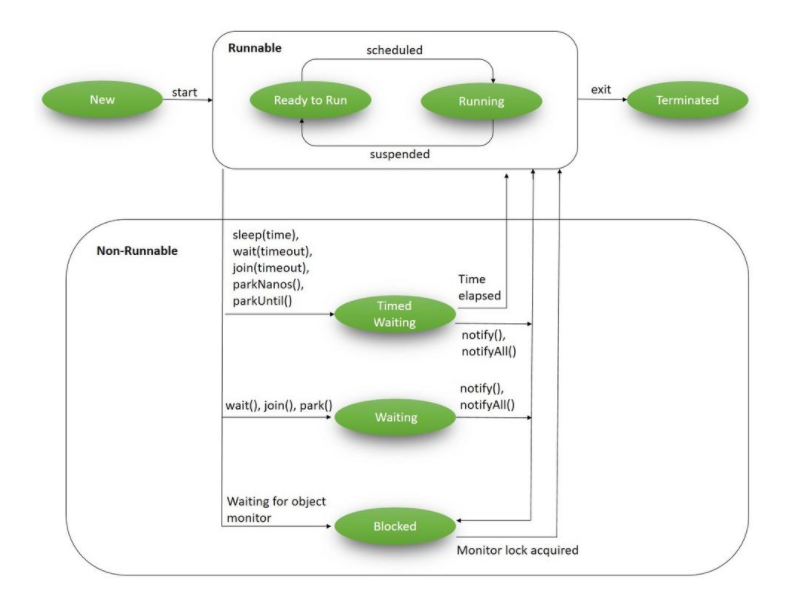
**TIMED\_WAITING**

A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.

**TERMINATED**

A thread that has exited is in this state.

When we create a Thread in java program, its state is New. Then we start the thread that changes its state to Runnable. Thread Scheduler is responsible to allocate CPU to threads in Runnable thread pool and change their state to Running. Other Thread states are Waiting, Blocked and Terminated.



https://www.baeldung.com/java-thread-lifecycle

<http://www.journaldev.com/1044/thread-life-cycle-in-java-thread-states-in-java>

1. **In which all states thread throws interrupted exception?**

Blocked, Waiting and timed waiting.

**Sleep (); yield(); yield meaning demand/pressurise**

1. **How can we pause the execution of a Thread for specific time? Sleep()**

Using sleep() method we can pause the execution of Thread for certain time.

Once the thread awake from sleep, it’s state gets changed to runnable and based on thread scheduling it comes back to running state.

1. **Explain yield() method?**

Yield method pauses the currently executing thread temporarily so that it can give chance to the other waiting threads with same priority.

If there are no threads in the waiting state with the same priority then current thread will continue its execution.

If there are threads with same priority in waiting state then current thread will go to runnable state and moving the thread back to running state is purely depending on thread scheduler

1. **Can I assign same priority to multiple threads? IMP**
2. **Why Thread sleep () and yield () methods are static?**

Thread sleep () and yield () methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

**Join ();**

1. **What does join() method?**

It causes the currently running threads to stop executing until the thread it joins with completes its tasks.

JoinExample.java

1. **How can we make sure main() is the last thread to finish in Java Program?**

We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function.

**Thread Priority and Scheduling**

1. **What do you understand about Thread Priority?**

Every thread has a priority, usually higher priority thread will be executed first.

But it depends on Thread Scheduler implementation that is OS dependent. We can specify the priority of thread but it doesn’t guarantee that higher priority thread will get executed before lower priority thread.

Thread priority is an int whose value varies from 1 to 10 where 1 is the lowest priority thread and 10 is the highest priority thread.

1. **What is Thread Scheduler?**

Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads.

Once we create and start a thread, it’s execution depends on the implementation of Thread Scheduler.

1. **What is the difference between preemptive scheduling and time slicing?**

**Pre-emptive scheduling**: highest priority thread executes until it enters the waiting or dead states or a higher priority thread comes into existence

**Time slicing**: thread executes for a predefined slice of time and then reenters the pool of ready threads. The scheduler then determines which task should execute next, based on priority and other factors.

1. **What is context-switching in multi-threading?**

Context Switching is the process of storing and restoring of CPU state so that Thread execution can be resumed from the same point at a later point of time. Context Switching is the essential feature for multitasking operating system and support for multi-threaded environment**.**

**wait (); notify(); notifyAll();**

1. **How does thread communicate with each other?**

Object class wait (), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource.

When threads share resources, communication between Threads is important to coordinate their efforts.

1. **Explain wait(); notify(); notifyAll(); ?**

Allows threads to communicate about the lock status of a resource

**wait() :**

*synchronized( lockObj )*

*{*

*while( ! condition )*

*{*

*lockObj.wait();*

*}*

*//take the action here.*

*}*

When wait method is invoked on the object that is locked it tells the executing thread to release the lock and go to sleep until some other thread enters the same monitor and calls notify() on the same object. When the thread returning from the wait state it reacquires the lock.

The wait() method releases the lock prior to waiting and reacquires the lock prior to returning from the wait() method.

**notify() :**

General syntax for calling notify () method is like this:

*synchronized(lockObject)*

*{*

*//establish\_the\_condition;*

*lockObject.notify();*

*//any additional code if needed*

*}*

When notify method is invoked on the lock object It wakes up thread that called wait () on the same object. But calling notify will not release the lock until it completed execution of synchronized block

**notifyAll() : It wakes up all the threads that called wait() on the same object.**

The highest priority thread will run first in most of the situation, though not guaranteed. Other things are same as notify () method above.

General syntax for calling notify() method is like this:

synchronized(lockObject)

{

establish\_the\_condition;

lockObject.notifyAll();

}

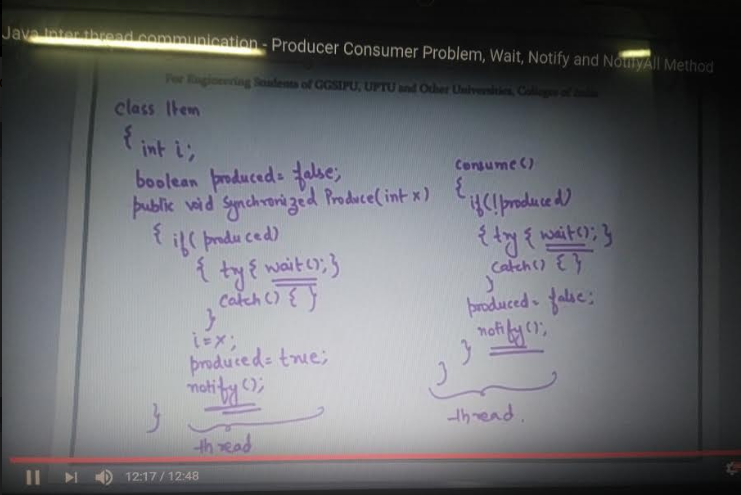
<http://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example>

<http://howtodoinjava.com/core-java/multi-threading/how-to-work-with-wait-notify-and-notifyall-in-java>

1. **Producer consumer problem?**

[**https://www.youtube.com/watch?v=J7-JuD5oDK4**](https://www.youtube.com/watch?v=J7-JuD5oDK4)

This problem can be solved by using inter thread communication.



Code: multithreading. Producerconsumer **need to learn to write the code based on queue. Currently you know simple one**

1. **Why wait (), notify () and notifyAll () methods have to be called from synchronized method or block?**

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify () on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

1. **Why thread communication methods wait(), notify() and notifyAll() are in Object class?**

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication.

1. **Is it possible to start a thread twice?**

No, there is no possibility to start a thread twice. If we does, it throws an exception.

1. **Can we call the run () method instead of start ()?**

yes, but it will not work as a thread rather it will work as a normal object so there will not be context-switching between the threads.

1. **What will happen when notify is called first? IMP**

.

1. **Can we make the user thread as daemon thread if thread is started?**

No, if you do so, it will throw IllegalThreadStateException

1. **What is shutdown hook?**

The shutdown hook is basically a thread i.e. invoked implicitely before JVM shuts down. So we can use it perform clean up resource.

[**http://www.javatpoint.com/ShutdownHook-thread**](http://www.javatpoint.com/ShutdownHook-thread)

1. **When should we interrupt a thread? IMP**

We should interrupt a thread if we want to break out the sleep or wait state of a thread.

1. **Why threads are interrupted and what are its real time uses? IMP**
2. **Interrupting a Thread? IMP**

If any thread is in sleeping or waiting state calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException.

If the thread is not in the sleeping or waiting state, calling the interrupt() method will have no effect but sets the interrupt flag to true.

<https://www.javatpoint.com/exception-handling-with-method-overriding>

1. **How to use callable?** **IMP**
2. **Difference between runnable and callable? IMP**
3. **Why threads are interrupted and what are its real time uses?**
4. **Say you have 3 threads how will you run one by one? IMP**

**https://stackoverflow.com/questions/12989397/running-3-threads-in-sequence-java**

1. **What is difference between wait () and sleep () method?**

* **Executor Framework (Thread framework)**

1. **What do you understand by Executor Framework in Java? What is executor framework?**

Executor Framework has been introduced in JDK 5.

Executor Framework helps in

Creation of thread,

Creating the thread pool

Checking health while running and also terminates if needed.

1. **What is the role of Executor Service in Java?**

Executor Service provides different methods to start and terminate thread

An Executor Service is thus very similar to a thread pool.

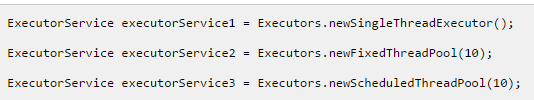
1. **Which are implementation classes of ExecutorService interface?**

ThreadPoolExecutor

ScheduledThreadPoolExecutor

1. **What is Executors in java Executor Framework? How you create ExecutorService?**

Executors is a factory that provides the methods to return ExecutorService, ScheduledExecutorService, ThreadFactory.



Some of the factory methods are:

**ExecutorService newFixedThreadPool(int nThreads)**: We need to pass the number of threads to this method .It returns the pool of that size.If concurrently task are submitted more than the pool size, then rest of task need to wait in queue.

It returns ExecutorService.

**ScheduledExecutorService newScheduledThreadPool(int corePoolSize):** This also creates a fixed size pool but it can schedule the thread to run after some defined delay. It is useful to schedule the task.

It returns ScheduledExecutorService.

**ExecutorService newCachedThreadPool()**:

There is no fixed size of this pool. Thread will be created at run time and if there is no task it will alive for 60 second and then die. For short lived threads this pool works well.

It returns ExecutorService.

**ExecutorService newSingleThreadExecutor():**

1. **ExecutorService Usage?**
   1. **execute(Runnable)**

The execute(Runnable) method takes a java.lang.Runnable object, and executes it asynchronously. Here is an example of executing a Runnable with an ExecutorService:

ExecutorService executorService = Executors.newSingleThreadExecutor();

executorService.execute(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

executorService.shutdown();

There is no way of obtaining the result of the executed Runnable, if necessary. You will have to use a Callable for that (explained in the following sections).

* 1. **submit(Runnable)**

The submit(Runnable) method also takes a Runnable implementation, but returns a Future object. This Future object can be used to check if the Runnable as finished executing.

Here is a ExecutorService submit() example:

Future future = executorService.submit(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

future.get(); //returns null if the task has finished correctly.

* 1. **submit(Callable)**

The submit (Callable) method is similar to the submit(Runnable) method except for the type of parameter it takes. The Callable instance is very similar to a Runnable except that its call() method can return a result. The Runnable.run() method cannot return a result.

The Callable's result can be obtained via the Future object returned by the submit(Callable) method. Here is an ExecutorService Callable example:

Future future = executorService.submit(new Callable(){

public Object call() throws Exception {

System.out.println("Asynchronous Callable");

return "Callable Result";

}

});

System.out.println("future.get() = " + future.get());

The above code example will output this:

Asynchronous Callable

future.get() = Callable Result

* 1. **invokeAny(**Collection callable**)**

The invokeAny() method takes a collection of Callable objects, or subinterfaces of Callable. Invoking this method does not return a Future, but returns the result of one of the Callable objects. You have no guarantee about which of the Callable's results you get. Just one of the ones that finish

If one of the tasks complete (or throws an exception), the rest of the Callable's are cancelled.

Here is a code example:

ExecutorService executorService = Executors.newSingleThreadExecutor();

Set<Callable<String>> callables = new HashSet<Callable<String>>();

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 1";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 2";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 3";

}

});

String result = executorService.invokeAny(callables);

System.out.println("result = " + result);

executorService.shutdown();

* 1. **invokeAll()**

The invokeAll() method invokes all of the Callable objects you pass to it in the collection passed as parameter. The invokeAll() returns a list of Future objects via which you can obtain the results of the executions of each Callable.

Keep in mind that a task might finish due to an exception, so it may not have "succeeded". There is no way on a Future to tell the difference.

Here is a code example:

ExecutorService executorService = Executors.newSingleThreadExecutor();

Set<Callable<String>> callables = new HashSet<Callable<String>>();

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 1";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 2";

}

});

callables.add(new Callable<String>() {

public String call() throws Exception {

return "Task 3";

}

});

List<Future<String>> futures = executorService.invokeAll(callables);

for(Future<String> future : futures){

System.out.println("future.get = " + future.get());

}

executorService.shutdown();

1. **ExecutorService Shutdown?**

**shutdown():** To terminate the threads inside the ExecutorService you call its method. The ExecutorService will not shut down immediately, but it will no longer accept new tasks, and once all threads have finished current tasks, the ExecutorService shuts down. All tasks submitted to the ExecutorService before shutdown () is called, are executed.

**shutdownNow()**: If you want to shut down the ExecutorService immediately, you can call the smethod. This will attempt to stop all executing tasks right away, and skips all submitted but non-processed tasks. There are no guarantees given about the executing tasks. Perhaps they stop, perhaps the execute until the end. It is a best effort attempt.

1. **What is the role of Future Task and Future in java?**

**Future:** is result of asynchronous computation. Future checks if task is complete and if completed it gets the output.

**FutureTask:** FutureTask is a cancellable asynchronous computation in java. It can cancel the task which is running. Once the FutureTask will be cancelled, it cannot be restarted.

1. **How to get return value of a callable thread in java Executor Framework?**

Using Future, we can get the return value of callable thread.

ExecutorService exService = Executors.newCachedThreadPool();

Future<Integer> future=exService.submit(new CallableThread());

int val=future.get();

1. **How to terminate a thread in Executor Framework in java?**

ExecutorService provides a method awaitTermination(long timeout, TimeUnit unit) that takes time and unit of time as an arguments. After that time thread pool is terminated. Suppose we need to terminate a task just now, then we can do as.

1. **What is the role of Executors.unconfigurableExecutorService in Executor Framework?**

unconfigurableExecutorService returns an object that delegates all methods of ExecutorService to the given executor so that any other method cannot accessed by cast.

1. **What are the different policy in Executor Framework?**

There are different policy within ThreadPoolExecutor in java.

a. ThreadPoolExecutor.AbortPolicy : AbortPolicy is a handler for rejected task. It handles those task which has been rejected.

b. ThreadPoolExecutor.CallerRunsPolicy : This also handles the rejected task and runs the rejected task directly.

c. ThreadPoolExecutor.DiscardOldestPolicy : This handles those rejected task that is oldest and unhandled. It discards those that oldest task.

d. ThreadPoolExecutor.DiscardPolicy : This is the handler for those rejected task that are rejected silently.

1. **What happens to exception when pooled executor is processing threads? IMP**
2. **ThreadPoolExecutor?**

<http://tutorials.jenkov.com/java-util-concurrent/scheduledexecutorservice.html>

1. **ScheduledExecutorService?**

[**http://tutorials.jenkov.com/java-util-concurrent/scheduledexecutorservice.html**](http://tutorials.jenkov.com/java-util-concurrent/scheduledexecutorservice.html)

1. **How to create 10 concurrent thread in java?**

**Say i have servers in Switzerland London etc so i want to hit every server at same time and get the data how do i do that?**

* **Synchronization**

1. **What is Race condition? IMP**

Race condition occurs when two or more threads access shared data and they try to change it at the same time without taking lock.

Here we do not know which thread access the data first because it all completely depends on thread scheduler.

1. **How will you avoid race condition?**

Race condition can be avoided by taking lock on shared resources.

Understand with example:

Problems often occur when one thread does a "**check-then-act**" (e.g. "check" if the value is X, then "act" to do something that depends on the value being X) and another thread does something to the value in between the "check" and the "act". E.g:

If (x == 5) // The "Check"

{

y = x \* 2; // The "Act"

// If another thread changed x in between "if (x == 5)" and "y = x \* 2" above,

// y will not be equal to 10.

}

The point being, y could be 10, or it could be anything, depending on whether another thread changed x in between the check and act. You have no real way of knowing.

In order to prevent race conditions from occurring, you would typically put a lock around the shared data to ensure only one thread can access the data at a time. This would mean something like this:

// obtain lock for x

if (x == 5)

{

y = x \* 2; // Now, nothing can change x until the lock is released.

// Therefore y = 10

}

// release lock for x

1. **How can we achieve thread safety in Java? Mean how many ways?**

**S**ynchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes.

1. **What is synchronization?**

Control the access of multiple threads to any shared resource.

It is used:

To prevent thread interference.

To prevent consistency problem.

1. **What is synchronized method?**

Method declared with synchronization key word is called as synchronized method.

*When a thread invokes a synchronized method, it automatically acquires the lock for that object (this) and releases it when the thread completes its task. (instance method synchronization)*

1. **What is Synchronized block?**

Using synchronized block we can perform synchronization on any specific resource of the method.

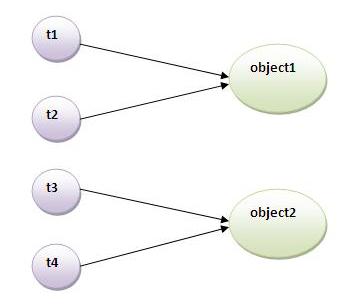
Points to remember for Synchronized block

Synchronized block is used to lock an object for any shared resource.

Scope of synchronized block is smaller than the method.

1. **What is Static synchronization?**

If you make any static method as synchronized, the lock will be on the class not on object. (A.class)



Problem without static synchronization

Say we have a class as below

**Class A**

**{**

**synchronized test()**

**{**

**//Logic**

**}**

**synchronized test1()**

**{**

**//Logic**

**}**

**}**

In case of synchronized instance method when any thread executes this method it takes lock on **this** object. So if there are multiple synchronized methods one method will be executed at one time.

If we create 2 object for above class as below.

A a = new A ();

A a1 = new A();

In this say we have 2 threads first one is calling a.test(); and second one can also can call a1.test(); as both takes lock on their own this object so there is chance of interference.

Suppose if we have class as below

**Class A**

**{**

**synchronized static test()**

**{**

**//Logic**

**}**

**Synchronized static test1()**

**{**

**//Logic**

**}**

**}**

In case of synchronized static method when any thread executes test method it takes lock on A.class so executing test() and test1() at the same time by different thread is not possible like instance method synchronization.

If we create 2 object for above class as below.

A a = new A ();

A a1 = new A();

In this say we have 2 threads first one is calling a.test(); so lock obtained on A.class Here the second thread can not call a1.test(); though it is different object.

1. **Can Java object be locked down for exclusive use by a given thread?**

Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it.

1. **Can synchronized keyword used for Constructors?**

No

1. **Can synchronized keyword used for variables?**

No

1. **Can we use static object in synchronized block as monitor? IMP**

Yes

1. **Which is more preferred – Synchronized method or synchronized block?**

Synchronized block is more preferred way because it doesn’t lock the Object, synchronized methods lock the Object and if there are multiple synchronization blocks in the class, even though they are not related, it will stop them from execution and put them in wait state to get the lock on Object.

**Volatile Keyword**

1. **What is volatile keyword in Java?**

When we use volatile keyword with a variable, all the threads read its value directly from the memory and not from cache.

This makes sure that the value read is the same as in the memory.

1. **When to use Volatile variable in Java?**

1) Any variable which is shared between multiple threads should be made variable, in order to ensure that all thread must see the latest value of the volatile variable.

2) A signal to compiler and JIT to ensure that compiler does not change ordering or volatile variable and moves them out of synchronized context.

3) You want to save the cost of synchronization as volatile variables are less expensive than synchronization.

1. **Can we use volatile when 1 thread is modifying it and other two threads are accessing it? IMP**
2. **Can we use volatile when 2 threads modifying and one thread accessing? IMP**
3. **What is atomicity?**
4. **What is Deadlock? How to analyze and avoid deadlock situation? IMP**

Two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

Example program is there in eclipse.

http://www.journaldev.com/1058/deadlock-in-java-example

**How to detect deadlock in java?**

Need to look into thread dump of the application.

We need to look out for the threads with state as BLOCKED and then the resources it’s waiting to lock; every resource has a unique ID using which we can find which thread is already holding the lock on the object.

**How to avoid deadlock? Check for better answer if there are some**

**Avoid Nested Locks**: This is the most common reason for deadlocks, avoid locking another resource if you already hold one. It’s almost impossible to get deadlock situation if you are working with only one object lock.

**Lock Only What is Required**: You should acquire lock only on the resources you have to work on, for example in above program I am locking the complete Object resource but if we are only interested in one of it’s fields, then we should lock only that specific field not complete object.

**Avoid waiting for long time**: You can get deadlock if two threads are waiting for each other to finish indefinitely using thread join. If your thread has to wait for another thread to finish, it’s always best to use join with maximum time you want to wait for thread to finish.

# Fork join

## What is fork join?

The fork/join framework is an implementation of the Executor Service interface that helps you take advantage of multiple processors

It is designed for work that can be broken into smaller pieces recursively.

The goal is to use all the available processing power to enhance the performance of your application

As with any ExecutorService implementation, the fork/join framework distributes tasks to worker threads in a thread pool. The fork/join framework is distinct because it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy.

References:

<http://www.javatpoint.com/java-multithreading-interview-questions>

Pending Question in journal dev interview questions?

20-26