Java Keywords
Key Concepts
Key Concepts #2
OOPS in java
Java Collections#1
Java Collections #2
Exceptions #1
Exceptions #2
Threads#1
Threads#2
InnerClass
More InnerClass
Serialization
Immutable Class
Cloning in Java
Garbage Collection
JSP #1
JSP #2
Programming Q#1
Programming Q#2
Programming Q#3

Java Multithreading Interview Questions

Multithreading or Concurrency is one of the popular topic for **java interview questions**. Large scale applications such as **Banking**, **Big data processing** or **services built to scale for millions of users** often rely on multithreading and async functionality.

Q1) What is a Thread?

Ans) In Java, "thread" means two different things:

- An instance of class java.lang.Thread.
- A thread of execution.

An instance of Thread is just.. an object. Like any other object in Java, it has variables and methods, and lives and dies on the heap. But a thread of execution is an individual process (a "lightweight" process) that has its own call stack. In Java, there is one thread per call stack—or, to think of it in reverse, one call stack per thread. Even if you don't create any new threads in your program, threads are back there running.

The main() method, that starts the whole ball rolling, runs in one thread, called (surprisingly) the main thread. If you looked at the main.call stack (and you can, any time you get a stack trace from something that happens after main begins, but not within another thread), you'd see that main() is the first method on the stack— the method at the bottom. But as soon as you create a new thread, a new stack materializes and methods called from that thread run in a call stack that's separate from the main() call stack.

Q2) What is difference between a thread and a process?

Ans)

- Threads share the address space of the process that created it; process has it's own address space.
- Threads have direct access to the data segment of its process; processes have their own copy of the data segment of the parent process.
- Threads can directly communicate with other threads of its process; processes must use interprocess communication to communicate with sibling processes.
- Threads have almost no overhead; processes have considerable overhead.
- New threads are easily created; new processes require duplication of the parent process.
- Threads can exercise considerable control over threads of the same process; processes can only exercise control over child processes.
- Changes to the main thread (cancellation, priority change etc.) may affect the behavior of the other threads of the process; changes to the parent process do not affect child processes.

Q3) What are the advantages or usage of threads?

Ans)**Threads support concurrent operations. For example,**
• Multiple requests by a client on a server can be handled as an individual client thread.
• Long computations or high-latency disk and network operations can be handled in the background without disturbing foreground computations or screen updates.

Threads often result in simpler programs.
• In sequential programming, updating multiple displays normally requires a big while-loop that performs small parts of each display update. Unfortunately, this loop basically simulates an operating system scheduler. In Java, each view can be assigned a thread to provide continuous updates.
• Programs that need to respond to user-initiated events can set up service routines to handle the events without having to insert code in the main routine to look for these events.

Threads provide a high degree of control.
• Imagine launching a complex computation that occasionally takes longer than is satisfactory. A "watchdog" thread can be activated that will "kill" the computation if it becomes costly, perhaps in favor of an alternate, approximate solution. Note that sequential programs must muddy the computation with termination code, whereas, a Java program can use thread control to non-intrusively supervise any operation.

Threaded applications exploit parallelism.
• A computer with multiple CPUs can literally execute multiple threads on different functional units without having to simulating multi-tasking ("time sharing").
• On some computers, one CPU handles the display while another handles computations or database accesses, thus, providing extremely fast user interface response times.

Q4)What are the two ways of creating thread?

Ans) There are two ways to create a new thread.

- **Extend the Thread class** and override the run() method in your class. Create an instance of the subclass and invoke the start() method on it, which will create a new thread of execution.

```
public class NewThread extends Thread{
  public void run(){
    // the code that has to be executed in a separate new thread goes here
    ...
    public static void main(String [] args){
    NewThread c = new NewThread();
    c.start();
  }
}
```

- **Implements the Runnable interface.**The class will have to implement the run() method in the Runnable interface. Create an instance of this class. Pass the reference of this instance to the Thread constructor a new thread of execution will be created.

```
public class NewThread Implements Runnable{
    public void run(){
      // the code that has to be executed in a separate new thread goes here
    public static void main(String [] args){
      NewThread c = new NewThread();
      Thread t = new Thread(c);
      t.start();
    }
  }
```

Q5) What are the different states of a thread's lifecycle?

Ans) The different states of threads are as follows:

New – When a thread is instantiated it is in New state until the start() method is called on the thread instance. In this state the thread is not considered to be alive.
Runnable – The thread enters into this state after the start method is called in the thread instance. The thread may enter into the Runnable state from Running state. In this state the thread is considered to be alive.
Running – When the thread scheduler picks up the thread from the Runnable thread's pool, the thread starts running and the thread is said to be in Running state.
Waiting/Blocked/Sleeping – In these states the thread is said to be alive but not runnable. The thread switches to this state because of reasons like wait method called or sleep method has been called on the running thread or thread might be waiting for some i/o resource so blocked. 5) Dead – When the thread finishes its execution i.e. the run() method execution completes, it is said to be in dead state. A dead state can not be started again. If a start() method is invoked on a dead thread a runtime exception will occur.

Q6) What is use of synchronized keyword?

Ans) synchronized keyword can be applied to static/non-static methods or a block of code. Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method by one thread. Synchronized keyword provides a lock on the object and thus prevents race condition. E.g.

```
public void synchronized method(){}
  public void synchronized staticmethod(){}
  public void mymethod(){
    synchronized (this){
      //synchronized keyword on block of code
    }
  }
```

Q7) What is the difference when the synchronized keyword is applied to a static method or to a non static method?

Ans) When a synch non static method is called a lock is obtained on the object. When a synch static method is called a lock is obtained on the class and not on the object. The lock on the object and the lock on the class don’t interfere with each other. It means, a thread accessing a synch non static method, then the other thread can access the synch static method at the same time but can’t access the synch non static method.

Q8) What is a volatile keyword?

Ans) In general each thread has its own copy of variable, such that one thread is not concerned with the value of same variable in the other thread. But sometime this may not be the case. Consider a scenario in which the count variable is holding the number of times a method is called for a given class irrespective of any thread calling, in this case irrespective of thread access the count has to be increased so the count variable is declared as volatile.

The copy of volatile variable is stored in the main memory, so every time a thread access the variable even for reading purpose the local copy is updated each time from the main memory. The volatile variable also have performance issues.

Q9) What is the difference between yield() and sleep()?

Ans) yield() method pauses the currently executing thread temporarily for giving a chance to the remaining waiting threads of the same priority to execute. If there is no waiting thread or all the waiting threads have a lower priority then the same thread will continue its execution. The yielded thread when it will get the chance for execution is decided by the thread scheduler whose behavior is vendor dependent. It doesn't release the lock on the objects acquired.
sleep() allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn't releases the lock.

Q10) What is the difference between wait() and sleep()?

Ans)

- wait() is a method of Object class. sleep() is a method of Thread class.
- sleep() allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn't release the lock. wait() allows thread to release the lock and goes to suspended state. The thread is only active when a notify() or notifAll() method is called for the same object.

Q11) What is difference between notify() and notflyAll()?

Ans) notify() wakes up the first thread that called wait() on the same object. notifyAll() wakes up all the threads that called wait() on the same object. The highest priority thread will run first.

Q12) What happens if a start method is not invoked and the run method is directly invoked?

Ans)If a thread has been instantiated but not started its is said to be in new state. Unless until a start() method is invoked on the instance of the thread, it will not said to be alive. If you do not call a start() method on the newly created thread instance thread is not considered to be alive. If the start() method is not invoked and the run() method is directly called on the Thread instance, the code inside the run() method will not run in a separate new thread but it will start running in the existing thread.

Q13) What happens when start() is called?

Ans) A new thread of execution with a new call stack starts. The state of thread changes from new to runnable. When the thread gets chance to execute its target run() method starts to run.

Q14) If code running is a thread creates a new thread what will be the initial priority of the newly created thread?

Ans) When a code running in a thread creates a new thread object, the priority of the new thread is set equal to the priority of the thread which has created it.

Q15) When jvm starts up, which thread will be started up first?

Ans) When jvm starts up the thread executing main method is started.

Q16) What are the daemon threads?

Ans) Daemon thread are service provider threads run in the background,these not used to run the application code generally.When all user threads(non-daemon threads) complete their execution the jvm exit the application whatever may be the state of the daemon threads. Jvm does not wait for the daemon threads to complete their execution if all user threads have completed their execution.

To create Daemon thread set the daemon value of Thread using setDaemon(boolean value) method. By default all the threads created by user are user thread. To check whether a thread is a Daemon thread or a user thread use isDaemon() method.

Example of the Daemon thread is the Garbage Collector run by jvm to reclaim the unused memory by the application. The Garbage collector code runs in a Daemon thread which terminates as all the user threads are done with their execution.

Q18) Can the variables or classes be Synchronized?

Ans) No. Only methods and blocks can be synchronized.

Q19) How many locks does an object have?

Ans) Each object has only one lock.

Q20) Can a class have both Synchronized and non-synchronized methods?

Ans) Yes a class can have both synchronized and non-synchronized methods.

If you reach so far, check More Questions on Thread

Recommend Reading

How java manages Memory?
Why is it advised to use hashcode and equals together?
Comparable and Comparator in java
How to create Singleton class in Java?
Difference between equals and ==?
When to use abstract class over interface?
Cap Theorem in Details
Why is it recommended to use Immutable objects in Java

Recommended Reading
Java 8 Lambda Explained
How java manages Memory?
When to use hashcode and equals?
Comparable and Comparator in java
How to create a Singleton class in Java?
When to use abstract class over interface?