1. What is difference between Path and Classpath? 🡺Path and Classpath are operating system level environment variales. Path is used define where the system can find the executables(.exe) files and classpath is used to specify the location .class files.

2. What are instance variables? 🡺Instance variables are those which are defined at the class level. Instance variables need not be initialized before using them as they are automatically initialized to their default values.

3. What is a pointer and does Java support pointers? 🡺Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and reliability issues hence Java doesn't support the usage of pointers.

4. What are local variables? 🡺Local variables are those which are declared within a block of code like methods. Local variables should be initialized before accessing them.

5. Is Java a pure object oriented language? 🡺Java uses primitive data types and hence is not a pure object oriented language.

6. Is JVM platform independent? 🡺JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.

7. Can a main() method be overloaded? 🡺Yes. You can have any number of main() methods with different method signature and implementation in the class.

8. What do you mean by platform independence? 🡺Platform independence means that we can write and compile the java code in one platform (eg Windows) and can execute the class in any other supported platform eg (Linux,Solaris,etc).

9. Are arrays primitive data types? 🡺 In Java, Arrays are objects.

10. What is the difference between a JDK and a JVM? 🡺JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

11. Does the order of public and static declaration matter in main() method? 🡺No. It doesn't matter but void should always come before main().

12. What is the impact of declaring a method as final? 🡺A method declared as final can't be overridden. A sub-class can't have the same method signature with a different implementation.

13. Can a class be declared as protected? 🡺A class can't be declared as protected. only methods can be declared as protected.

14. How to define a constant variable in Java? 🡺The variable should be declared as static and final. So only one copy of the variable exists for all instances of the class and the value can't be changed also. static final int PI = 2.14; is an example for constant.

15. Which package is imported by default? 🡺java.lang package is imported by default even without a package declaration.

16. What is the argument of main() method? 🡺main() method accepts an array of String object as argument.

17. Can a source file contain more than one class declaration 🡺 Yes a single source file can contain any number of Class declarations but only one of the class can be declared as public.

18. What is the access scope of a protected method 🡺A protected method can be accessed by the classes within the same package or by the sub classes of the class in any package.

19. Can a class declared as private be accessed outside it's package? 🡺Not possible.

20. Why is the main() method declared static 🡺main() method is called by the JVM even before the instantiation of the class hence it is declared as static.

21. What is the purpose of declaring a variable as final 🡺A final variable's value can't be changed. final variables should be initialized before using them.

22. What is a package 🡺 Package is a collection of related classes and interfaces. package declaration should be first statement in a java class.

23. Can a main() method be declared final 🡺Yes. Any inheriting class will not be able to have it's own default main() method.

24. I don't want my class to be inherited by any other class. What should i do 🡺You should declared your class as final. But you can't define your class as final, if it is an abstract class. A class declared as final can't be extended by any other class.

26. When will you define a method as static 🡺 When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

27. What is the importance of static variable? 🡺 static variables are class level variables where all objects of the class refer to the same variable. If one object changes the value then the change gets reflected in all the objects.

28. How is final different from finally and finalize()?

final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited,final method can't be overridden and final variable can't be changed.

finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.   
finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

29. I want to print "Hello" even before main() is executed. How will you achieve that?

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the main() method. And it will be executed only once.

30. Can we declare a static variable inside a method?  
Static variables are class level variables and they can't be declared inside a method. If declared, the class will not compile.

31. Can a abstract class be defined without any abstract methods 🡺Yes it's possible. This is basically to avoid instance creation of the class.

32. Can you give few examples of final classes defined in Java API 🡺java.lang.String, java.lang.Math are final classes.

33. Can a abstract class be declared final 🡺 Not possible. An abstract class without being inherited is of no use and hence will result in compile time error.

34. Can you create an object of an abstract class 🡺Not possible. Abstract classes can't be instantiated.

35. What are the restriction imposed on a static method or a static block of code 🡺A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

**36 . Class C implements Interface I containing method m1 and m2 declarations. Class C has provided implementation for method m2. Can i create an object of Class C?**

No not possible. Class C should provide implementation for all the methods in the Interface I. Since Class C didn't provide implementation for m1 method, it has to be declared as abstract. Abstract classes can't be instantiated.

37. Can a class be defined inside an Interface? 🡺Yes it's possible.

38. Can an Interface extend another Interface? 🡺 Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface.

39. What is use of a abstract variable 🡺Variables can't be declared as abstract. only classes and methods can be declared as abstract.

40. Can an Interface be defined inside a class 🡺Yes it's possible.

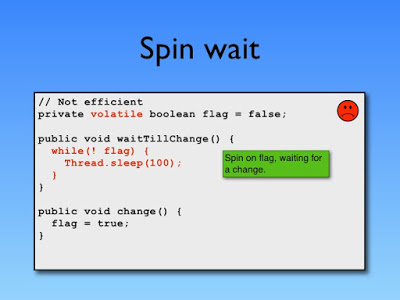
41. What is an Abstract Class and what is it's purpose 🡺A Class which doesn't provide complete implementation is defined as an abstract class. Abstract classes enforce abstraction.

42. Can an Interface be final 🡺 Not possible. Doing so so will result in compilation error.

43. Can an Interface implement another Interface 🡺Interfaces doesn't provide implementation hence a interface cannot implement another interface.

45. Why does Java not support operator overloading?

Operator overloading makes the code very difficult to read and maintain. To maintain code simplicity, Java doesn't support operator overloading.

**1. What is Busy Spinning? Why Should You Use It in Java?**  
busy spinning is a *waiting strategy*, in which a thread just wait in a loop, without releasing the CPU for going to [sleep](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html). This is a very advanced and specialized waiting strategy used in the high-frequency trading application when wait time between two messages is very minimal.  
By not releasing the CPU or [suspending the thread](http://java67.blogspot.com/2015/06/how-to-pause-thread-in-java-using-sleep.html), your thread retains all the cached data and instruction, which may be lost if the thread was suspended and resumed back in a different core of CPU.   
  
  
  
**2. What is Read-Write Lock? Does ConcurrentHashMap in Java Uses The ReadWrite Lock?**  
ReadWrite Lock is an implementation of *lock stripping* technique, where two separate locks are used for read and write operation. Since read operation doesn't modify the state of the object, it's safe to allow multiple thread access to a shared object for reading without locking, and by splitting one lock into [read and write lock](http://javarevisited.blogspot.com/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html), you can easily do that.   
Java provides an implementation of read-write lock in the form of ReentrantReadWritLock class in the java.util.concurrent.lock package. This is worth looking before you decide to write your own read-write locking implementation.   
Also, the current implementation of java.util.ConcurrentHashMap doesn't use the ReadWriteLock, instead, it divides the Map into several segments and locks them separately using different locks. This means any given time, *only a portion of the ConcurrentHashMap is locked*, instead of the whole Map. See [how ConcurrentHashMap internally works in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) for more detail.   
  
**3. How to Make an Object Immutable in Java? Why Should You Make an Object Immutable?**  
Well, Immutability offers several advantages including thread-safety, ability to cache and result in more readable multithreading code. See [here](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html)to learn how to make object Immutable. Once again, this question can also go into more detail and depending on your answer, can bring several other questions e.g. when you mention Spring is Immutable, be ready with some reasons on [Why String is Immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html).  
  
**4. Which Design Patterns have You Used in Your Java Project?**  
Always expect some design patterns related question for Core Java Interview of senior developer position. It's a better strategy to mention any GOF design pattern rather than Singleton or MVC, which almost every other Java developer use it.   
Your best bet can be [Decorator pattern](http://java67.blogspot.sg/2013/07/decorator-design-pattern-in-java-real-life-example-tutorial.html) or may be [Dependency Injection Pattern](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html), which is quite popular in Spring Framework. It's also good to mention only the design patterns which you have *actually* used in your project and knows it's tradeoffs.   
It's common that once you mention a particular design pattern say Factory or Abstract Factory, Interviewer's next question would be, *have you used this pattern in your project?* So be ready with proper example and why you choose a particular pattern. You can also see [this](http://java67.blogspot.com/2012/09/top-10-java-design-pattern-interview-question-answer.html) article for more advanced design pattern questions from Java interviews.   
  
**5.  Do you know about Open Closed Design Principle or Liskov Substitution Principle?**  
Design patterns are based on [object-oriented design principles](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html), which I strongly felt every object-oriented developer and the programmer should know, or, at least, have a basic idea of what are these principles and how they help you to write better object oriented code. If you don't know the answer to this question, you can politely say No, as it's not expected from you to know the answer to every question, but by answering this question, you can make your claim stronger as many experienced developers fail to answer basic questions like this. See [Clean Code](http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882?tag=javamysqlanta-20) to learn more about object-oriented and SOLID design principles. **6. Which Design Pattern Will You Use to Shield Your Code From a Third Party library Which Will Likely to be Replaced by Another in Couple of Months?**  
This is just one example of  the scenario-based design pattern interview question. In order to test the practical experience of Java developers with more than 5 years experience, companies ask this kind of questions.  You can expect more real-world design problems in different formats, some with more detail explanation with context, or some with only intent around.   
One way to shield your code from third party library is to [code against an interface rather than implementation](http://javarevisited.blogspot.com/2014/11/why-use-interface-in-java-or-object-oriented-programming.html) and then use dependency injection to provide a particular implementation. This kind of questions is also asked quite frequently to experienced and senior *Java developers with 5 to 7 years of experience*.  
 **7)  How  do you prevent SQL Injection in Java Code?**  
This question is more asked to J2EE and Java EE developers than core Java developers, but, it is still a good question to check the JDBC and Security skill of experienced Java programmers.  
You can [use PreparedStatement to avoid SQL injection](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html) in Java code. Use of the PreparedStatement for executing SQL queries not only provides better performance but also shield your Java and J2EE application from SQL Injection attack.   
On a similar note, If you are working more on Java EE or J2EE side, then you should also be familiar with other security issues including *Session Fixation attack* or *Cross Site Scripting* attack and how to resolve them. These are some fields and questions where a good answer can make a lot of difference on your selection.   
**8) Tell me about different Reference types available in Java, e.g. WeakReference, SoftReference or PhantomReference? and Why should you use them?**  
Well, they are different reference types coming from java.lang.ref package and provided to assist Java Garbage Collector in a case of low memory issues. If you wrap an object with [WeakReference](http://javarevisited.blogspot.com/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html) than it will be eligible for garbage collected if there are o strong reference. They can later be reclaimed by Garbage collector if JVM is running low on memory.  
The java.util.WeakHashMap is a special Map implementation, whose keys are the object of WeakReference, so if only Map contains the reference of any object and no other, those object can be garbage collected if GC needs memory. See [Java Performance The Definitive Guide](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20) learn more about how to deal with performance issues in Java.

**9) How does get method of HashMap works in Java?**  
Yes, this is still one of the most popular core Java questions for senior developer interviews. You can also expect this question on telephonic round, followed by lot's of follow-up questions as discussed on my post [how does HashMap work in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html).   
The short answer to this question is that HashMap is based upon hash table data structure and uses hashCode() method to calculate hash code to find the bucket location on underlying array and equals() method to search the object in the same bucket in case of a collision. See [here](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) to learn a more about how does get() method of HashMap works in Java.   
  
**10) Which Two Methods HashMap key Object Should Implement?**  
This is one of the follow-up questions I was saying about in previous questions. Since working of HashMap is based upon hash table data structure, any object which you want to use as key for HashMap or any other hash based collection e.g. Hashtable, or ConcurrentHashMap must implement [equals()](http://java67.blogspot.com/2012/11/difference-between-operator-and-equals-method-in.html) and [hashCode()](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) method.

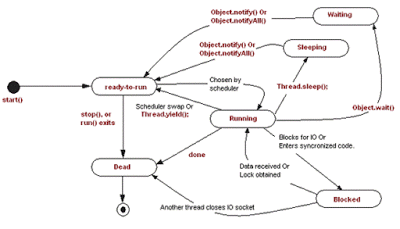
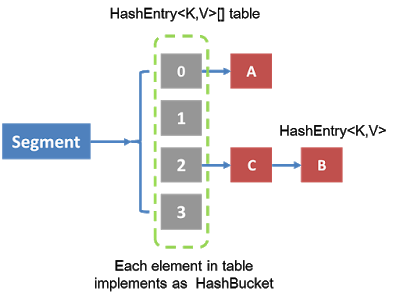
The hashCode() is used to find the bucket location i.e. index of underlying array and equals() method is used to find the right object in linked list stored in the bucket in case of a collision. By the way, from Java 8, HashMap also started using tree data structure to store the object in case of collision to reduce worst case performance of HashMap from O(n) to O(logN). See the article for learning more about [how does HashMap handless collision in Java](http://javarevisited.blogspot.com/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html).   
  
**11) Why Should an Object Used As the Key should be Immutable?**   
This is another follow-up of previous core Java interview questions. It's good to test the depth of technical knowledge of candidate by asking more and more question on the same topic. If you know about Immutability, you can answer this question by yourself. The short answer to this question is key should be immutable so that hashCode() method  always return the same value.

Since hash code returned by hashCode() method depends on upon the content of object i.e. values of member variables. If an object is [mutable](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) than those values can change and so is the hash code. If the same object returns different hash code once you inserted the value in HashMap, you will end up searching in different bucket location and will not able to retrieve the object. That's why a key object should be immutable. It's not a rule enforced by the compiler but you should take care of it as an experienced programmer. See the article for more [advanced Java Collection interview questions](http://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html).

**12) How does ConcurrentHashMap achieves its Scalability?**   
Sometimes this multithreading + collection interview question is also asked as, the difference between ConcurrentHashMap and Hashtable in Java. The problem with [synchronized HashMap](http://java67.blogspot.com/2015/02/how-to-synchronize-hashmap-in-java-with.html) or [Hashtable](http://java67.blogspot.com/2012/08/difference-between-hashmap-and-concurrentHashMap-java-collection.html) was that whole Map is locked when a thread performs any operation with Map.   
The java.util.ConcurrentHashMap class solves this problem by using *lock stripping* technique, where the whole map is locked at different segments and only a particular segment is locked during the write operation, not the whole map. The ConcurrentHashMap also achieves it's scalability by allowing lock-free reads as read is a thread-safe operation.  See [here](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html) for more advanced multi-threading and concurrency questions in Java.   
  
**13) How do you share an object between threads? or How to pass an object from one thread to another?**  
There are multiple ways to do that e.g. Queues, Exchanger etc, but BlockingQueue using [Producer Consumer pattern](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html) is the easiest way to pass an object from thread to another.  
  
**14) How do find if your program has a deadlock?**  
By taking thread dump using kill -3, using JConsole or VisualVM), I suggest to prepare this core java interview question in more detail, as Interviewer definitely likes to go with more detail e.g. they will press with questions like, have you really done that in your project or not?

**15) How do you avoid deadlock while coding?**  
By ensuring locks are acquire and released in an ordered manner, see [here](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html)for detail answer of this question.

**1) What is the difference between start and run method in Java Thread?**  
This thread interview question is also ask as if start() method eventually call run() method then why do you need to call start() method, why not call run() method directly. well, reason is that because start method creates a new thread and call the code written inside the run method on a new thread while calling run method executes that code on the same thread. You can also see the article [start vs run method in Java](http://javarevisited.blogspot.sg/2012/03/difference-between-start-and-run-method.html) for more details.  
  
**2) Write code to avoid deadlock in Java where N threads are accessing N shared resources?**  
This is a classic Java multithreading interview questions, which appears on almost every list of Java thread questions. This question is based on risk and issues faced by parallel programs without proper synchronization or incorrect synchronization. This question explores the concept of looking and best practices on acquiring and releasing the lock on shared resource. By the way, it's been covered in many places as well and I suggest reading  [How to prevent deadlock in Java](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html), not only for detail answer of this Java multithreading question but also to learn how to prevent deadlock in Java.  
  
**3) Which one is better to implement thread in Java ? extending Thread class or implementing Runnable?**  
Well, this is another frequently asked questions on any Java thread interview. Essentially these are two way to implement Thread in Java, by extending java.lang.Thread class or by implementing java.lang.Runnable interface.  
By extending the class you are using your chance to extend one any only one class as Java does not support multiple inheritances, by implementing a Runnable interface you can still extend another class. So extending Runnable or even Callable is a better choice. You can also see [Runnable vs Thread class in Java](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) for more answers on this questions.  
Given its simplicity and fact-based nature, this question mostly appears on either telephonic round or initial screening rounds. Key points to mention, while answering this question includes multiple inheritance at the class level and separation of defining a task and execution of a task. Runnable only represent a task, while Thread represent both tasks and it's execution.  
  
**4) What is Busy Spinning? Why will you use Busy Spinning as wait strategy?**([answer](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html))  
This is one of the advanced concurrency interview questions in Java and only asked to experienced and senior Java developers, with lots of concurrent coding experience under the belt. By the way, he concept of *busy spinning* is not new, but its usage with multi-core processor has risen recently The busy waiting is a wait strategy, where one thread wait for a condition to become true, but instead of calling wait or sleep method and releasing CPU, it just spins. This is particularly useful if the condition is going to be true quite quickly i.e. in a millisecond or microsecond.  
The advantage of not releasing CPU is that all cached data and instruction remain unaffected, which may be lost, had this thread is suspended on one core and brought back to another thread. If you can answer this question, that rest assure of a good impression.  
  
**5) What is the difference between CountDownLatch and CyclicBarrier in Java?**([answer](http://java67.blogspot.com/2012/08/difference-between-countdownlatch-and-cyclicbarrier-java.html))  
The [CountDownLatch](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html)and [CyclicBarrier in Java](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html)are two important concurrency utility which is added on Java 5 Concurrency API. Both are used to implement scenario, where one thread has to wait for other thread before starting processing but there is a difference between them.  
The key point to mention, while answering this question is that CountDownLatch is not reusable once the count reaches to zero, while CyclicBarrier can be reused even after the barrier is broken.  
  
**6) What is the difference between wait and sleep in Java?**   
One more *classic Java multithreading question* from the telephonic round of interviews. The key point to mention while answering this question is to mention that wait will release the lock and must be called from the synchronized context, while sleep will only pause the thread for some time and keep the lock.  
By the way, both methods throw IntrupptedException and can be interrupted, which can lead to some follow-up questions like, can we awake a sleeping or waiting for a thread in Java? You can also read a detailed answer on my post of same title .

  
  
**7) How do you solve producer consumer problem in Java?**  
One of my favorite questions during any Java multithreading interview, Almost half of the concurrency problems can be categorized in the producer-consumer pattern. There are basically two ways to solve this problem in Java, One by [using wait and notify method](http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html) and other by using [BlockingQueue](http://javarevisited.blogspot.com/2014/06/synchronousqueue-example-in-java.html) in Java.  later is easy to implement and a good choice if you are coding in Java 5.  
The key points to mention, while answering this question is thread-safety and blocking nature of BlockingQueue and how that helps, while writing concurrent code. You can also expect lots of follow-up questions including, what happen if you have multiple producer or multiple consumers, what will happen if a producer is faster than consumer thread or vice-versa. You can also see this link for example of [how to code producer-consumer design in Java using blocking queue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html)  
  
**8) Why is ConcurrentHashMap faster than Hashtable in Java?**([answer](http://javarevisited.blogspot.com/2016/05/what-is-difference-between-synchronized.html))  
ConcurrentHashMap is introduced as an alternative of Hashtable in Java 5, it is faster because of its design. ConcurrentHashMap divides the whole map into different segments and only lock a particular segment during the update operation, instead of [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html), which locks whole Map.  
The ConcurrentHashMap also provides lock-free read, which is not possible in Hashtable, because of this and lock striping, ConcurrentHashMap is faster than Hashtable, especially when a number of the reader is more than the number of writers.  
In order to better answer this popular Java concurrency interview questions, I suggest reading my post about the [internal working of ConcurrentHashMap in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html).  
****

**9) What is the difference between submit() and execute() method of Executor and ExecutorService in Java?**

The main difference between submit and execute method from ExecutorService interface is that former return a result in the form of a Future object, while later doesn't return a result. By the way, both are used to submit a task to thread pool in Java but one is defined in Executor interface,while other is added into ExecutorService interface. This multithreading interview question is also asked in the first round of Java interviews.  
  
**10) How do you share data between two threads in Java?**   
One more Java multithreading question from the telephonic round of interview. You can share data between thread by using shared object or shared data structures like Queue. Depending upon, what you are using, you need to provide the thread-safety guarantee, and one way of providing thread-safety is using synchronized keyword.  
If you use concurrent collection classes from Java 5 e.g. [BlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html), you can easily share data without being bothered about thread safety and inter-thread communication. I like this thread question, because of it's simplicity and effectiveness. This also leads further follow-up questions on issues which arise due to sharing data between threads e.g. race conditions.  
  
**11) What is ReentrantLock in Java? Have you used it before?**   
ReentrantLock is an alternative of synchronized keyword in Java, it is introduced to handle some of the limitations of synchronized keywords. Many concurrency utility classes and concurrent collection classes from Java 5, including ConcurrentHashMap uses ReentrantLock, to leverage optimization.  
The ReentrantLock mostly uses an atomic variable and faster CAS operation to provides better performance. Key points to mention are the [difference between ReentrantLock and synchronized keyword in Java](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html), which includes the ability to acquire lock interruptibly, timeout feature while waiting for lock etc. ReentrantLock also gives the option to create fair lock in Java.Once again a very good Java concurrency interview question for experienced Java programmers.   
Books like [Java Programming Interview exposed](http://www.amazon.com/Java-Programming-Interviews-Exposed-Markham/dp/1118722868?tag=javamysqlanta-20) also helps you to prepare better and answer to the point.

**12) What is ReadWriteLock in Java? What is the benefit of using ReadWriteLock in Java?**   
This is usually a follow-up question of previous Java concurrency questions. The  ReadWriteLock is again based upon the concept of lock striping, one of the advance thread-safety mechanism which advocates separating locks for reading and writing operations (see [Java Concurrency in Practice Bundle by Heinz Kabutz](https://learning.javaspecialists.eu/courses/concurrency-in-practice-bundle?affcode=92815_johrd7r8) for more details).  
If you have noticed before, reading operation can be done without locking if there is no writer and that can hugely improve the performance of any application. The ReadWriteLock leverage this idea and provide policies to allow maximum concurrency level. Java Concurrency API also provides an implementation of this concept as ReentrantReadWriteLock.  
Depending upon Interviewer and experience of the candidate, you can even expect to provide your own implementation of ReadWriteLock, so be prepare for that as well.

1. Guess the output of below program.

|  |  |
| --- | --- |
| class A{  void m1() {System.out.println("In m1 A");}  }  class B extends A{  void m1() {System.out.println("In m1 B");}  void m2() {System.out.println("In m2 B");}  }  public class Test {  public static void main(String[] args) {  A a=new B();  a.m2();}} | **What will be the output?** **A. In m2 B** **B. Compile time error** **C. Runtime error** |
| Answer: B. Compile time error There will be compile time error.Even though we are assigning B’s object to A’s reference we can call only methods which are in A from A’s reference. |

2. Guess the output of below program.

|  |  |
| --- | --- |
| class A{  void m1() throws ArrayIndexOutOfBoundsException {  System.out.println("In m1 A"); **}**  **}**  class B extends A{  void m1() throws IndexOutOfBoundsException {  System.out.println("In m1 B"); **}**  **}**  public class Test {  public static void main(String[] args) {  A a=new B();  a.m1(); **}**  **}** | **What will be the output?** **A. In m1 B** **B. Compile time error** **C. Runtime error** |
| Answer : A. In m1 B This will work fine as ArrayIndexOutOfBoundsException and IndexOutOfBoundsException are Runtime exceptions and there is no rule for runtime exceptions while method overriding. |

3. Guess the output of below program.

|  |  |
| --- | --- |
| import java.io.IOException;  class A{  void m1() throws IOException {  System.out.println("In m1 A"); **}}**  class B extends A{  void m1() throws Exception {  System.out.println("In m1 B");**}}**  public class Test {  public static void main(String[] args) {  A a=new B();  try {  a.m1();  } catch (IOException e) {  e.printStackTrace();**}}}** | **What will be the output?** **A. In m1 B** **B. Compile time error** **C. Runtime error** |
| **Answer** : B. Compile time error As IOException and Exception are checked exception, so you can not broaden the scope of Exception while method overriding. |

4. What will happen in case of below program?

|  |  |
| --- | --- |
| class A  {  synchronized void m1() {  System.out.println("In m1 A");  }  void m2() {  System.out.println("In m2 A");  }  } | There are two threads T1 and T2. T1 is accessing m1 method. Will T2 be able to access m2 method on the same instance at the same time? |
| Answer: Yes, T2 will be able to access m2 as it does not require lock to access m2 method |

5. Guess the output of below program.

|  |  |
| --- | --- |
| import java.util.HashSet;  public class Customer {  String name;  int age;  Customer(String name,int age) {  this.name=name;  this.age=age;}  public static void main(String[] args) {  Customer c1= new Customer("John",20);  Customer c2= new Customer("John",20); HashSet<Customer>customerSet=new HashSet<>();  customerSet.add(c1);  customerSet.add(c2);  System.out.println(customerSet.size());  }  // getters and setters  } | Output: Output will be 2 as we did not implement hashcode and equals method in Customer class. |

9. How to decide young generation and old generation size for your application?

It depends on nature of application.  
If you have lots of temporary objects then there will be lot of minor gc. You can provide arguments XX:NewRatio=1 to distribute 50% to young generation and 50% to old.  
By default, NewRatio=2 hence young Generation is 1/3 of total heap.  
Similarly, If you have too many long-lived objects, then you might need to increase size of tenure space by putting high value of NewRatio.

10. What is garbage collection in java?

Garbage collection is the process of identifying used and unused objects on java heap and removing unused object from the heap.  
A live object means object is still being referred to some part of program. Unused object means object is not being referred by any part of program and is eligible for garbage collection.

12. What is difference between Collection.synchronizedMap(map) and ConcurrentHashMap?

When you make map thread safe by using Collection.synchronizedMap(map), it locks whole map object, but ConcurrentHashMap does not lock the whole map, it just locks part of it(Segment).

13. What will happen when you run below code

|  |  |
| --- | --- |
| public class IterateMapMain {  public static void main(String args[])  {  HashMap<String,String> map=new HashMap<String,String>();  map.put("India","Delhi");  map.put("Japan","Tokyo");  map.put("France","Paris");  map.put("Russia","Moscow");  System.out.println("Iterating java Iterator");  Iterator<String> countryKeySetIterator=map.keySet().iterator();  while(countryKeySetIterator.hasNext()){  String countryKey=countryKeySetIterator.next();  map.put("Nepal", "KathMandu");  System.out.println(countryKey); }  System.out.println("-----------------------------");  }   } |  |

Output : You will get below output  
Iterating java IteratorException in thread “main” java.util.ConcurrentModificationException  
at java.base/java.util.HashMap$HashIterator.nextNode(HashMap.java:1489)  
at java.base/java.util.HashMap$KeyIterator.next(HashMap.java:1512)  
at org.arpit.java2blog.IterateMapMain.main(IterateMapMain.java:24)Japan  
Whenever you try to change count of hashmap while iterating, it will throw java.util.ConcurrentModificationException because we call iterator.next,it checks for modCount and initial count, if it does not match it will throw ConcurrentModificationException.

16. What is double level locking in singleton design pattern?

|  |  |
| --- | --- |
| public static Singleton getInstance(){  if (\_instance == null) { // Single Checked  synchronized (Singleton.class){  if (\_instance == null)// Double checked{  \_instance = new Singleton();}}  } return \_instance;} | Let’s say two threads(T1 and T2) checked for null and both reached at synchronized (Singleton.class) . T1 gets the lock and create instance of Singleton and return.Now T2 enters in synchronized block, as we have checked for null again,it will not create object again. |

1.   Can we override static method in java?

No, you cannot override static method in java. You can only hide them.

2.   Can you overload main method in java?

Yes, you can overload main method in java.

3. Can we override private methods in java?

You cannot override private methods in java as it is visible to that class only.

5.   Can you list down some of important method from object class?

Important methods of object classes are:

* hashcode : It returns hash value of the object
* equals : It compares the object references
* wait : It causes current thread to wait until notify or notifyAll is not called
* notify : Wakes up single thread which is waiting for lock
* notifyAll: Wakes up all threads which is waiting for lock
* toString : Provides String representation of the object
* clone : This method is used to clone the object
* finalize: This method is called when object is being garbage collected.

6.   Which two methods should you override while putting the custom object as Key in HashMap?

You need to override [hashcode and equals method](https://www.java2blog.com/hashcode-and-equals-method-in-java/) in custom class while putting objects of custom class in HashMap.

8. Can we have abstract class without having any abstract method in it?

Yes, you can have abstract class without having any abstract method.

9. Have you heard about transient variable? When will you use it?

Transient variables are used Serialization. If you don’t want to make variable serializable, you can make it transient variable.

10. Can you call start method twice in java?

No, you can not call Start method twice. It will throw illegal state exception.

13. Can we have static method in the interface?

Yes, we can have [static method in the interface](https://www.java2blog.com/interface-default-methods-in-java-8/) from Java 8.

14. Can you declare constructor final?

No, You can not declare constructor final.

### 15. What is the difference between StringBuffer and StringBuilder?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **StringBuffer** | **StringBuilder** |
| Thread-safe | StringBuffer is thread safe. Two threads can not call methods of StringBuffer simultaneously. | StringBuilder is not thread safe, so two threads can call methods of StringBuilder simultaneously. |
| Performance | It is less performance efficient as it is thread-safe | It is more performance efficient as it is not thread-safe. |

### 16. What is Java ClassPath?

ClassPath is environment variable which java virtual machine (JVM) uses to locate all classes which is used by the program.  
For example: jre/lib/rt.jar has all java classes and you also need to include jar files or class file which is being used by program.

17. You have a list of Custom objects? How can you sort them?

You need to use [Comparable](https://www.java2blog.com/comparable-in-java/) or [Comparator](https://www.java2blog.com/comparator-in-java/) interface to sort list of custom objects.

18. What is volatile in java?

If you mark any variable volatile then this variable will be read from main memory rather than CPU cache so each thread will have updated value in the variable.

19. What are two different ways to call garbage collector?

System.gc() OR Runtime.getRuntime().gc().

20. What is marker interface in java? Can you provide some examples of marker interface?

Marker interfaces are those interfaces which do not have any method in it.

Examples of marker interfaces are : Serializable and Cloneable.

21. How many objects will be created below:

|  |
| --- |
| String str1= new String("John");  String str2= new String("John"); |

Three objects will be created here, two in heap memory and one in String constant pool.

**1) Can we make array volatile in Java?**  
This is one of the tricky Java multi-threading questions you will see in senior Java developer Interview. Yes, you can make an array volatile in Java but only the reference which is pointing to an array, not the whole array. What I mean, if one thread changes the reference variable to points to another array, that will provide a volatile guarantee, but if multiple threads are changing individual array elements they won't be having happens before guarantee provided by the volatile modifier.  
  
**2) Can volatile make a non-atomic operation to atomic?**  
This another good question I love to ask on volatile, mostly as a follow-up of the previous question. This question is also not easy to answer because volatile is not about atomicity, but there are cases where you can use a volatile variable to make the operation atomic.  
One example I have seen is having a long field in your class. If you know that a long field is accessed by more than one thread e.g. a counter, a price field or anything, you better make it volatile. Why? because reading to a long variable is not atomic in Java and done in two steps, If one thread is writing or updating long value, it's possible for another thread to see half value (fist 32-bit). While reading/writing a volatile long or double (64 bit) is atomic.  
  
**3) What are practical uses of volatile modifier?**  
One of the practical use of the volatile variable is to make reading double and long atomic. Both double and long are 64-bit wide and they are read in two parts, first 32-bit first time and next 32-bit second time, which is non-atomic but volatile double and long read is atomic in Java. Another use of the volatile variable is to provide a memory barrier, just like it is used in Disrupter framework. Basically, Java Memory model inserts a write barrier after you write to a volatile variable and a read barrier before you read it. Which means, if you write to volatile field then it's guaranteed that any thread accessing that variable will see the value you wrote and anything you did before doing that right into the thread is guaranteed to have happened and any updated data values will also be visible to all threads, because the memory barrier flushed all other writes to the cache.

**4) What guarantee volatile variable provides?**   
volatile variables provide the guarantee about ordering and visibility e.g. volatile assignment cannot be re-ordered with other statements but in the absence of any synchronization instruction compiler, JVM or JIT are free to reorder statements for better performance. volatile also provides the happens-before guarantee which ensures changes made in one thread is visible to others. In some cases volatile also provide atomicity e.g. reading 64-bit data types like long and double are not atomic but read of volatile double or long is atomic.  
  
**5) Which one would be easy to write? synchronization code for 10 threads or 2 threads?**  
In terms of writing code, both will be of same complexity because synchronization code is independent of a number of threads. Choice of synchronization though depends upon a number of threads because the number of thread present more contention, so you go for advanced synchronization technique e.g. lock stripping, which requires more complex code and expertise.  
  
**6) How do you call wait() method? using if block or loop? Why?**   
wait() method should always be called in loop because it's possible that until thread gets CPU to start running again the condition might not hold, so it's always better to check condition in loop before proceeding. Here is the standard idiom of using wait and notify method in Java:

// The standard idiom for using the wait method

synchronized (obj) {

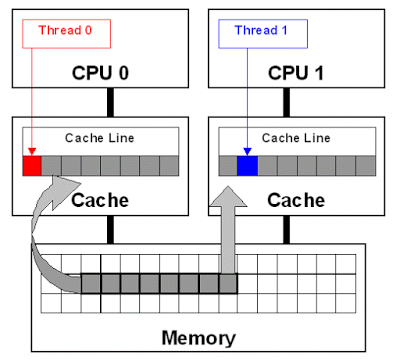
while (condition does not hold)

obj.wait(); // (Releases lock, and reacquires on wakeup)

... // Perform action appropriate to condition

}

See [Effective Java Item 69](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) to learn more about why wait method should call in the loop.  
  
**7)  What is false sharing in the context of multi-threading?**  
false sharing is one of the well-known performance issues on multi-core systems, where each process has its local cache. false sharing occurs when threads on different processor modify variables that reside on same cache line as shown in the following image:



False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory. Like many concurrency issues, the primary way to avoid false sharing is careful code review and aligning your data structure with the size of a cache line.  
  
**8) What is busy spin? Why should you use it?**  
Busy spin is one of the technique to wait for events without releasing CPU. It's often done to avoid losing data in CPU cached which is lost if the thread is paused and resumed in some other core. So, if you are working on low latency system where your order processing thread currently doesn't have any order, instead of sleeping or calling wait(), you can just loop and then again check the queue for new messages. It's only beneficial if you need to wait for a very small amount of time e.g. in micro seconds or nano seconds. [LMAX Disrupter](http://lmax-exchange.github.io/disruptor/) framework, a high-performance inter-thread messaging library has a BusySpinWaitStrategy which is based on this concept and uses a busy spin loop for EventProcessors waiting on the barrier.  
  
**9) How do you take thread dump in Java?**  
You can take a thread dump of Java application in Linux by using **kill -3 PID**, where PID is the process id of Java process. In Windows, you can press **Ctrl + Break**. This will instruct JVM to print thread dump in standard out or err and it could go to console or log file depending upon your application configuration. If you have used Tomcat then when  
  
**10) is Swing thread-safe?**([answer](http://javarevisited.blogspot.sg/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html))  
No, Swing is not thread-safe. You cannot update Swing components e.g. JTable, JList or JPanel from any thread, in fact, they must be updated from GUI or AWT thread. That's why swings provide invokeAndWait() and invokeLater() method to request GUI update from any other threads. This methods put update request in AWT threads queue and can wait till update or return immediately for an asynchronous update. You can also check the detailed answer to learn more.  
  
**11) What is a thread local variable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html))  
Thread-local variables are variables confined to a thread, its like thread's own copy which is not shared between multiple threads. Java provides a ThreadLocal class to support thread-local variables. It's one of the many ways to achieve thread-safety. Though be careful while using thread local variable in manged environment e.g. with web servers where worker thread out lives any application variable. Any thread local variable which is not removed once its work is done can potentially cause a memory leak in Java application.

**12) Write wait-notify code for producer-consumer problem?** ([answer](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html))  
Please see the answer for a code example. Just remember to call wait() and notify() method from synchronized block and test waiting for condition on the loop instead of if block.  
  
**13) Write code for thread-safe Singleton in Java?** ([answer](http://javarevisited.blogspot.in/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))  
Please see the answer for a code example and step by step guide to creating thread-safe singleton class in Java. When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads. Using Java enum as Singleton class is one of the easiest ways to create a thread-safe singleton in Java.  
**14) The difference between sleep and wait in Java?**([answer](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html))  
Though both are used to pause currently running thread, sleep() is actually meant for short pause because it doesn't release lock, while wait() is meant for conditional wait and that's why it release lock which can then be acquired by another thread to change the condition on which it is waiting.  
  
**15) What is an immutable object? How do you create an Immutable object in Java?**([answer](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html" \t "_blank))  
Immutable objects are those whose state cannot be changed once created. Any modification will result in a new object e.g. String, Integer, and other wrapper class. Please see the answer for step by step guide to creating Immutable class in Java.  
  
**16) Can we create an Immutable object, which contains a mutable object?**  
Yes, its possible to create an Immutable object which may contain a mutable object, you just need to be a little bit careful not to share the reference of the mutable component, instead, you should return a copy of it if you have to. Most common example is an Object which contain the reference of java.util.Date object.  
 **17) What is the right data type to represent a price in Java?**([answer](http://javarevisited.blogspot.sg/2012/02/java-mistake-1-using-float-and-double.html))  
BigDecimal if memory is not a concern and Performance is not critical, otherwise double with predefined precision.  
  
**18) How do you convert bytes to String?** ([answer](http://javarevisited.blogspot.sg/2014/08/2-examples-to-convert-byte-array-to-String-in-Java.html))  
you can convert bytes to the string using string constructor which accepts byte[], just make sure that right character encoding otherwise platform's default character encoding will be used which may or may not be same.  
  
**20) Can we cast an int value into byte variable? what will happen if the value of int is larger than byte?**  
Yes, we can cast but int is 32 bit long in java while byte is 8 bit long in java so when you cast an int to byte higher 24 bits are lost and a byte can only hold a value from -128 to 128.  
  
**22) Which class contains clone method? Cloneable or Object?** ([answer](http://javarevisited.blogspot.sg/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html))  
java.lang.Cloneable is marker interface and doesn't contain any method clone method is defined in the object class. It is also knowing that clone() is a native method means it's implemented in C or C++ or any other native language.

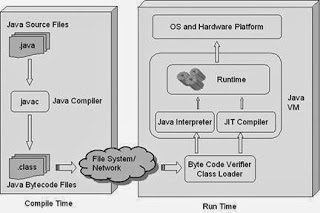
**23) Is ++ operator is thread-safe in Java?**   
 No it's not a thread safe operator because its involve multiple instructions like reading a value, incriminating it and storing it back into memory which can be overlapped between multiple threads.  
  
**24) Difference between a = a + b and a += b ?**   
The += operator implicitly cast the result of addition into the type of variable used to hold the result. When you add two integral variable e.g. variable of type byte, short, or int then they are first promoted to int and them addition happens. If result of addition is more than maximum value of a then a + b will give compile time error but a += b will be ok as shown below

byte a = 127;

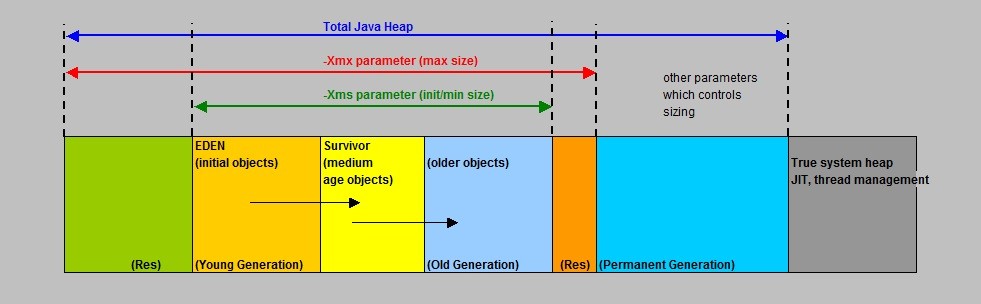
byte b = 127;

b = a + b; *// error : cannot convert from int to byte*

b += a; *// ok*

**25) Can I store a double value in a long variable without casting?**([answer](http://java67.blogspot.com/2014/11/how-to-convert-double-to-long-in-java-example.html))  
No, you cannot store a double value into a long variable without casting because the range of double is more  that long and you we need to type cast. It's not dificult to answer this question but many develoepr get it wrong due to confusion on which one is bigger between double and long in Java.  
  
**26) What will this return 3\*0.1 == 0.3? true or false?**  
This is one of the really tricky questions. Out of 100, only 5 developers answered this question and only of them have explained the concept correctly. The short answer is false because some floating point numbers can not be represented exactly.  
  
**27) Which one will take more memory, an int or Integer?**  
An Integer object will take more memory an Integer is the an object and it  store meta data overhead about the object and int is primitive type so its takes less space.  
  
**28) Why is String Immutable in Java?** ([answer](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html))  
One of my favorite Java interview question. The String is Immutable in java because java designer thought that string will be heavily used and making it immutable allow some optimization easy sharing same String object between multiple clients. See the link for the more detailed answer. This is a great question for Java programmers with less experience as it gives them food for thought, to think about how things works in Java, what Jave designers might have thought when they created String class etc.  
  
**29) Can we use String in the switch case?** ([answer](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html))  
Yes from Java 7 onward we can use String in switch case but it is just syntactic sugar. Internally string hash code is used for the switch. See the detaiedl answer for more explanation and discussion.  
  
**30) What is constructor chaining in Java?** ([answer](http://java67.blogspot.sg/2012/12/how-constructor-chaining-works-in-java.html))  
When you call one constructor from other than it's known as constructor chaining in Java. This happens when you have multiple, overloaded constructor in the class.  
  
**31) What is the size of int in 64-bit JVM?**  
The size of an int variable is constant in Java, it's always 32-bit irrespective of platform. Which means the size of primitive int is same in both 32-bit and 64-bit Java virtual machine.  
  
**32) The difference between Serial and Parallel Garbage Collector?** ([answer](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html))  
Even though both the serial and parallel collectors cause a stop-the-world pause during Garbage collection. The main difference between them is that a serial collector is a default copying collector which uses only one GC thread for garbage collection while a parallel collector uses multiple GC threads for garbage collection.  
  
**33) What is the size of an int variable in 32-bit and 64-bit JVM?**  
The size of int is same in both 32-bit and 64-bit JVM, it's always 32 bits or 4 bytes.  
  
**34) A difference between WeakReference and SoftReference in Java?**([answer](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html))  
Though both WeakReference and SoftReference helps garbage collector and memory efficient, WeakReference becomes eligible for garbage collection as soon as last strong reference is lost but SoftReference even thought it can not prevent GC, it can delay it until JVM absolutely need memory.  
  
**35) How do WeakHashMap works?**  
WeakHashMap works like a normal HashMap but uses WeakReference for keys, which means if the key object doesn't have any reference then both key/value mapping will become eligible for garbage collection.  
  
**36) What is -XX:+UseCompressedOops JVM option? Why use it?**([answer](http://javarevisited.blogspot.com/2012/06/what-is-xxusecompressedoops-in-64-bit.html))  
When you go migrate your Java application from 32-bit to 64-bit JVM, the heap requirement suddenly increases, almost double, due to increasing size of ordinary object pointer from 32 bit to 64 bit. This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP.  
  
**37) How do you find if JVM is 32-bit or 64-bit from Java Program?**([answer](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html))  
You can find that by checking some system properties like sun.arch.data.model or os.arch  
  
**38) What is the maximum heap size of 32-bit and 64-bit JVM?**([answer](http://javarevisited.blogspot.sg/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html))  
Theoretically, the maximum heap memory you can assign to a 32-bit JVM is 2^32 which is 4GB but practically the limit is much smaller. It also varies between operating systems e.g. form 1.5GB in Windows to almost 3GB in Solaris. 64-bit JVM allows you to specify larger heap size, theoretically 2^64 which is quite large but practically you can specify heap space up to 100GBs. There are even JVM e.g. Azul where heap space of 1000 gigs is also possible.  
  
**39) What is the difference between JRE, JDK, JVM and JIT?** ([answer](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html))  
JRE stands for Java run-time and it's required to run Java application. JDK stands for Java development kit and provides tools to develop Java program e.g. Java compiler. It also contains JRE. The JVM stands for Java virtual machine and it's the process responsible for running Java application. The JIT stands for Just In Time compilation and helps to boost the performance of Java application by converting Java byte code into native code when the crossed certain threshold i.e. mainly hot code is converted into native code.  


**40) Explain Java Heap space and Garbage collection?** ([answer](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html))  
When a Java process is started using java command, memory is allocated to it. Part of this memory is used to create heap space, which is used to allocate memory to objects whenever they are created in the program. Garbage collection is the process inside JVM which reclaims memory from dead objects for future allocation.



**41) Can you guarantee the garbage collection process?**  
No, you cannot guarantee the garbage collection, though you can make a request using System.gc() or Runtime.gc() method.  
  
**42) How do you find memory usage from Java program? How much percent of the heap is used?**  
You can use memory related methods from java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java.  By using these methods, you can find out how many percents of the heap is used and how much heap space is remaining. Runtime.freeMemory() return amount of free memory in bytes, Runtime.totalMemory()returns total memory in bytes and Runtime.maxMemory() returns maximum memory in bytes.  
  
**43) What is the difference between stack and heap in Java?**([answer](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html))  
Stack and heap are different memory areas in the JVM and they are used for different purposes. The stack is used to hold method frames and local variables while objects are always allocated memory from the heap. The stack is usually much smaller than heap memory and also didn't shared between multiple threads, but heap is shared among all threads in JVM.

**44) What's the difference between "a == b" and "a.equals(b)"?** ([answer](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html))  
The a = b does object reference matching if both a and b are an object and only return true if both are pointing to the same object in the heap space, on the other hand, a.equals(b) is used for logical mapping and its expected from an object to override this method to provide logical equality. For example, String class overrides this equals() method so that you can compare two Strings, which are the different object but contains same letters.  
  
**45) What is a.hashCode() used for? How is it related to a.equals(b)?**([answer](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
hashCode() method returns an int hash value corresponding to an object. It's used in hash based collection classes e.g Hashtable, HashMap, LinkedHashMap and so on. It's very tightly related to equals() method. According to Java specification, two objects which are equal to each other using equals() method must have same hash code.  
  
**46) Difference between final, finalize and finally?**([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-final-finally-and-finalize-java.html))  
The final is a modifier which you can apply to variable, methods and classes. If you make a variable final it means its value cannot be changed once initialized. finalize is a method, which is called just before an object is a garbage collected, giving it last chance to resurrect itself, but the call to finalize is not guaranteed. finally is a keyword which is used in exception handling along with try and catch. the finally block is always executed irrespective of whether an exception is thrown from try block or not.  
  
**47) What is a compile time constant in Java? What is the risk of using it?**  
public static final variables are also known as a compile time constant, the public is optional there. They are replaced with actual values at compile time because compiler know their value up-front and also knows that it cannot be changed during run-time. One of the problem with this is that if you happened to use a public static final variable from some in-house or third party library and their value changed later than your client will still be using old value even after you deploy a new version of JARs. To avoid that, make sure you compile your program when you upgrade dependency JAR files.  
  
**48) The difference between List, Set, Map, and Queue in Java?** ([answer](http://java67.blogspot.sg/2013/01/difference-between-set-list-and-map-in-java.html))  
The list is an ordered collection which allows duplicate. It also has an implementation which provides constant time index based access, but that is not guaranteed by List interface. Set is unordered collection which  
  
**49) Difference between poll() and remove() method?**  
Both poll() and remove() take out the object from the Queue but if poll() fails then it returns null but if remove fails it throws Exception.  
  
**50) The difference between LinkedHashMap and PriorityQueue in Java?**([answer](http://javarevisited.blogspot.sg/2013/10/what-is-priorityqueue-data-structure-java-example-tutorial.html))  
PriorityQueue guarantees that lowest or highest priority element always remain at the head of the queue, but LinkedHashMap maintains the order on which elements are inserted. When you iterate over a PriorityQueue, iterator doesn't guarantee any order but iterator of LinkedHashMap does guarantee the order on which elements are inserted.  
  
**51) Difference between ArrayList and LinkedList in Java?** ([answer](http://java67.blogspot.sg/2012/12/difference-between-arraylist-vs-LinkedList-java.html))  
The obvious difference between them is that ArrrayList is backed by array data structure, supprots random access and LinkedList is backed by linked list data structure and doesn't supprot random access. Accessing an element with the index is O(1) in ArrayList but its O(n) in LinkedList. See the answer for more detailed discussion.  
  
**52) What is a couple of ways that you could sort a collection?** ([answer](http://java67.blogspot.sg/2012/07/sort-list-ascending-descending-order-set-arraylist.html))  
You can either use the Sorted collection like TreeSet or TreeMap or you can sort using the ordered collection like a list and using Collections.sort() method.  
  
**53) How do you print Array in Java?** ([answer](http://java67.blogspot.sg/2014/03/how-to-print-array-in-java-example-tutorial.html))  
You can print an array by using the Arrays.toString() and Arrays.deepToString() method. Since array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but Arrays.toString() will print each element.

**54) LinkedList in Java is doubly or singly linked list?**   
It's a doubly linked list, you can check the code in JDK. In Eclipse, you can use the [shortcut](http://javarevisited.blogspot.com/2010/10/eclipse-tutorial-most-useful-eclipse.html), Ctrl + T to directly open this class in Editor.  
  
**55) Which kind of tree is used to implement TreeMap in Java**  
A Red Black tree is used to implement TreeMap in Java.

**56) What is the difference between Hashtable and HashMap**  
There are many differences between these two classes, some of them are following:  
a) Hashtable is a legacy class and present from JDK 1, HashMap was added later.  
b) Hashtable is synchronized and slower but HashMap is not synchronized and faster.  
c) Hashtable doesn't allow null keys but HashMap allows one null key.  
See the answer for more differences between HashMap and Hashtable in Java.  
  
**57) How HashSet works internally in Java**  
HashSet is internally implemented using an HashMap. Since a Map needs key and value, a default value is used for all keys. Similar to HashMap, HashSet doesn't allow duplicate keys and only one null key, I mean you can only store one null object in HashSet.  
  
  
**58) Write code to remove elements from ArrayList while iterating?**    
 Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Here is the [sample code](http://java67.blogspot.com/2015/10/how-to-solve-concurrentmodificationexception-in-java-arraylist.html) which uses right way o remove elements from ArrayList while looping over and avoids ConcurrentModificationException.  
  
**59) Can I write my own container class and use it in the for-each loop?**  
Yes, you can write your own container class. You need to implement the Iterable interface if you want to loop over advanced for loop in Java, though. If you implement Collection then you by default get that property.  
  
**60) What is default size of ArrayList and HashMap in Java?**   
As of Java 7 now, default size of ArrayList is 10 and default capacity of HashMap is 16, it must be power of 2. Here is code snippet from ArrayList  and HashMap class :  
  
**61) Is it possible for two unequal objects to have the same hashcode?**  
Yes, two unequal objects can have same hashcode that's why collision happen in a hashmap.  
the equal hashcode contract only says that two equal objects must have the same hashcode it doesn't say anything about the unequal object.  
  
**62) Can two equal object have the different hash code?**  
No, thats not possible according to hash code contract.  
  
**63) Can we use random numbers in the hashcode() method**

No, because hashcode of an object should be always same. See the answer to learning more about things to remember while overriding hashCode() method in Java.  
  
**64) What is the difference between Comparator and Comparable in Java?**   
The Comparable interface is used to define the  natural order of object while Comparator is used to define custom order. Comparable can be always one, but we can have multiple comparators to define customized order for objects.  
  
**65) Why you need to override hashcode, when you override equals in Java?**    
 Because equals have code contract mandates to override equals and hashcode together .since many container class like HashMap or HashSet depends on hashcode and equals contract.  
  
**76) What best practices you follow while writing multi-threaded code in Java?** ([answer](http://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html))  
Here are couple of best practices which I follow while writing concurrent code in Java:  
a) Always name your thread, this will help in debugging.  
b) minimize the scope of synchronization, instead of making whole method synchronized, only critical section should be synchronized.  
c) prefer volatile over synchronized if you can can.  
e) use higher level concurrency utilities instead of waitn() and notify for inter thread communication e.g. BlockingQueue, CountDownLatch and Semeaphore.  
e) Prefer concurrent collection over synchronized collection in Java. They provide better scalability.  
  
**77) Tell me few best practices you apply while using Collections in Java?**  
Here are couple of best practices I follow while using Collectionc classes from Java:  
a) Always use the right collection e.g. if you need non-synchronized list then use ArrayList and not Vector.  
b) Prefer concurrent collection over synchronized collection because they are more scalable.  
c) Always use interface to a represent and access a collection e.g. use List to store ArrayList, Map to store HashMap and so on.  
d) Use iterator to loop over collection.  
e) Always use generics with collection.  
  
**78) Can you tell us at least 5 best practice you use while using threads in Java?** ([answer](http://java67.blogspot.com/2014/01/10-points-about-thread-and-javalangthread-in-java.html))  
This is similar to the previous question and you can use the answer given there. Particularly with thread, you should:  
a) name your thread  
b) keep your task and thread separate, use Runnable or Callable with thread pool executor.  
c) use thread pool  
d) use volatile to indicate compiler about ordering, visibility, and atomicity.  
e) avoid thread local variable because incorrect use of ThreadLocal class in Java can create a memory leak.  
Look there are many best practices and I give extra points to the developer which bring something new, something even I don't know. I make sure to ask this question to Java developers of 8 to 10 years of experience just to gauge his hands on experience and knowledge.  
  
**79) Name 5 IO best practices?**   
IO is very important for performance of your Java application. Ideally you should avoid IO in critical path of your application. Here are couple of Java IO best practices you can follow:

a) Use buffered IO classes instead of reading individual bytes and char.

b) Use classes from NIO and NIO2

c) Always close streams in finally block or use try-with-resource statements.

d) use memory mapped file for faster IO.

If a Java candidate doesn't know about IO and NIO, especially if he has at least 2 to 4 years of experience, he needs some reading.  
  
  
**80) Name 5 JDBC best practices your follow?** ([answer](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html))  
Another good Java best practices for experienced Java developer of 7 to 8 years experience. Why it's important? because they are the ones which set the trend in the code and educate junior developers. There are many best practices and you can name as per your confort and conviniece. Here are some of the more common ones:  
a) use batch statement for inserting and updating data.  
b) use PreparedStatement to avoid SQL exception and better performance.  
c) use database connection pool  
d) access resultset using column name instead of column indexes.  
e) Don't generate dynamic SQL by concatenating String with user input.  
  
**81) Name couple of method overloading best practices in Java?** ([answer](http://javarevisited.blogspot.sg/2013/01/java-best-practices-method-overloading-constructor.html))  
Here are some best practices you can follow while overloading a method in Java to avoid confusion with auto-boxing:  
a) Don't overload method where one accepts int and other accepts Integer.  
b) Don't overload method where number of argument is same and only order of argument is different.  
c) Use varargs after overloaded methods has more than 5 arguments.  
  
**82) Does SimpleDateFormat is safe to use in the multi-threaded program?** ([answer](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html))  
No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a ThreadLocal variable. If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **joda-time** library.  
  
**83) How do you format a date in Java? e.g. in the ddMMyyyy format?** ([answer](http://javarevisited.blogspot.com/2011/09/convert-date-to-string-simpledateformat.html))  
You can either use SimpleDateFormat class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. dd-MM-yyyy or ddMMyyyy.  
  
89) How do you test static method 🡺You can use PowerMock library to test static methods in Java.  
  
**103) What is the interface? Why you use it if you cannot write anything concrete on it?**  
The interface is used to define API. It tells about the contract your classes will follow. It also supports abstraction because a client can use interface method to leverage multiple implementations e.g. by using List interface you can take advantage of [random access of ArrayList](http://javarevisited.blogspot.com/2015/07/java-arraylist-tutorial.html) as well as flexible insertion and deletion of LinkedList. The interface doesn't allow you to write code to keep things abstract but from Java 8 you can declare static and default methods inside interface which are concrete.  
  
**104) The difference between abstract class and interface in Java?**([answer](http://javarevisited.blogspot.sg/2013/05/difference-between-abstract-class-vs-interface-java-when-prefer-over-design-oops.html))  
There are multiple differences between abstract class and interface in Java, but the most important one is Java's restriction on allowing a class to extend just one class but allows it to implement multiple interfaces. An abstract class is good to define default behavior for a family of class, but the interface is good to define Type which is later used to leverage Polymorphism. Please check the answer for a more thorough discussion of this question.  
  
**105) Which design pattern have you used in your production code? apart from Singleton?**  
This is something you can answer from your experience. You can generally say about dependency injection, factory pattern, decorator pattern or observer pattern, whichever you have used. Though be prepared to answer follow-up question based upon the pattern you choose.  
  
**106) Can you explain Liskov Substitution principle 🡺** This is one of the toughest questions I have asked in Java interviews. Out of 50 candidates, I have almost asked only 5 have managed to answer it. I am not posting an answer to this question as I like you to do some research, practice and spend some time to understand this confusing principle well.  
  
**107) What is Law of Demeter violation? Why it matters 🡺**Believe it or not, Java is all about application programming and structuring code. If  you have good knowledge of common coding best practices, patterns and what not to do than only you can write quality code.  Law of Demeter suggests you "talk to friends and not stranger", hence used to reduce coupling between classes.  
**108) What is Adapter pattern? When to use it?**  
Another frequently asked Java design pattern questions. It provides interface conversion. If your client is using some interface but you have something else, you can write an Adapter to bridge them together. This is good for Java software engineer having 2 to 3 years experience because the question is neither difficult nor tricky but requires knowledge of OOP design patterns.  
  
**110) What is an abstract class? How is it different from an interface? Why would you use it**  
One more classic question from Programming Job interviews, it is as old as chuck Norris. An abstract class is a class which can have state, code and implementation, but an interface is a contract which is totally abstract. Since I have answered it many times, I am only giving you the gist here but you should read the article linked to answer to learn this useful concept in much more detail.  
  
**111) Which one is better constructor injection or setter dependency injection**  
Each has their own advantage and disadvantage. Constructor injection guaranteed that class will be initialized with all its dependency, but setter injection provides flexibility to set an optional dependency. Setter injection is also more readable if you are using an XML file to describe dependency. Rule of thumb is to use constructor injection for mandatory dependency and use setter injection for optional dependency.  
  
**112) What is difference between dependency injection and factory design pattern**  
Though both patterns help to take out object creation part from application logic, use of dependency injection results in cleaner code than factory pattern. By using dependency injection, your classes are nothing but POJO which only knows about dependency but doesn't care how they are acquired. In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern. Please see the answer for a more detailed discussion on this topic.  
  
**113) Difference between Adapter and Decorator pattern 🡺**Though the structure of Adapter and Decorator pattern is similar, the difference comes on the intent of each pattern. The adapter pattern is used to bridge the gap between two interfaces, but Decorator pattern is used to add new functionality into the class without the modifying existing code.  
  
**114) Difference between Adapter and Proxy Pattern 🡺** Similar to the previous question, the difference between Adapter and Proxy patterns is in their intent. Since both Adapter and Proxy pattern encapsulate the class which actually does the job, hence result in the same structure, but Adapter pattern is used for interface conversion while the Proxy pattern is used to add an extra level of indirection to support distribute, controlled or intelligent access.  
  
**115) What is Template method pattern 🡺** Template pattern provides an outline of an algorithm and lets you configure or customize its steps. For examples, you can view a sorting algorithm as a template to sort object. It defines steps for sorting but let you configure how to compare them using Comparable or something similar in another language. The method which outlines the algorithms is also known as template method.

**116) When do you use Visitor design pattern**  
The visitor pattern is a solution of problem where you need to add operation on a class hierarchy but without touching them. This pattern uses double dispatch to add another level of indirection.  
  
**117) When do you use Composite design pattern**  
Composite design pattern arranges objects into tree structures to represent part-whole hierarchies. It allows clients treat individual objects and container of objects uniformly. Use Composite pattern when you want to represent part-whole hierarchies of objects.

**118) The difference between Inheritance and Composition**  
Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies.  
  
**119) Describe overloading and overriding in Java**  
Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity. Though you can overload a method in the same class, but you can only override a method in child classes. Inheritance is necessary for overriding.  
  
**120) The difference between nested public static class and a top level class in Java**  
You can have more than one nested public static class inside one class, but you can only have one top-level public class in a Java source file and its name must be same as the name of Java source file.  
  
**121) Difference between Composition, Aggregation and Association in OOP**  
If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation. In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.  
  
**122) Give me an example of design pattern which is based upon open closed principle**  
This is one of the practical questions I ask experienced Java programmer. I expect them to know about OOP design principles as well as patterns. Open closed design principle asserts that your code should be open for extension but closed for modification. Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code.  There are several design patterns which are based upon open closed design principle e.g. [Strategy pattern](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) if you need a new strategy, just implement the interface and configure, no need to modify core logic. One working example is Collections.sort() method which is based on Strategy pattern and follows the open-closed principle, you don't modify sort() method to sort a new object, what you do is just implement Comparator in your own way.

**123) Difference between Abstract factory and Prototype design pattern?**   
This is the practice question for you, If you are feeling bored just reading and itching to write something, why not write the answer to this question. I would love to see an example the, which should answer where you should use the Abstract factory pattern and where is the Prototype pattern is more suitable.  
  
**124) When do you use Flyweight pattern 🡺** This is another popular question from the design pattern. Many Java developers with 4 to 6 years of experience know the definition but failed to give any concrete example. Since many of you might not have used this pattern, it's better to look examples from JDK. You are more likely have used them before and they are easy to remember as well. Now let's see the answer. Flyweight pattern allows you to share object to support large numbers without actually creating too many objects. In order to use Flyweight pattern, you need to make your object Immutable so that they can be safely shared. String pool and pool of Integer and Long object in JDK are good examples of Flyweight pattern.

**125) The difference between nested static class and top level class 🡺** One of the fundamental questions from Java basics. I ask this question only to junior Java developers of 1 to 2 years of experience as it's too easy for an experience Java programmers. The answer is simple, a public top level class must have the same name as the name of the source file, there is no such requirement for nested static class. A nested class is always inside a top level class and you need to use the name of the top-level class to refer nested static class e.g. HashMap.Entry is a nested static class, where HashMap is a top level class and Entry is nested static class.  
  
**126) Can you write a regular expression to check if String is a number 🡺** If you are taking Java interviews then you should ask at least one question on the regular expression. This clearly [differentiates an average programmer with a good programmer](http://javarevisited.blogspot.com/2015/05/how-to-differentiate-between-average.html). Since one of the traits of a good developer is to know tools, regex is the best tool for searching something in the log file, preparing reports etc. Anyway, answer to this question is, a numeric String can only contain digits i.e. 0 to 9 and + and - sign that too at start of the String, by using this information you can write following regular expression to check if given String is number or not  
  
**127) The difference between checked and unchecked Exception in Java?**([answer](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html))  
checked exception is checked by the compiler at compile time. It's mandatory for a method to either handle the checked exception or declare them in their throws clause. These are the ones which are a sub class of Exception but doesn't descend from RuntimeException. The unchecked exception is the descendant of RuntimeException and not checked by the compiler at compile time. This question is now becoming less popular and you would only find this with interviews with small companies, both investment banks and startups are moved on from this question.

**128) The difference between throw and throws in Java?** ([answer](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html))  
the throw is used to actually throw an instance of java.lang.Throwable class, which means you can throw both Error and Exception using throw keyword e.g.

throw new IllegalArgumentException("size must be multiple of 2")

On the other hand, throws is used as part of method declaration and signals which kind of exceptions are thrown by this method so that its caller can handle them. It's mandatory to declare any unhandled checked exception in **throws** clause in Java. Like the previous question, this is another frequently asked Java interview question from errors and exception topic but too easy to answer.

**129) The difference between Serializable and Externalizable in Java**   
This is one of the frequently asked questions from Java Serialization. The interviewer has been asking this question since the day Serialization was introduced in Java, but yet only a few good candidate can answer this question with some confidence and practical knowledge. Serializable interface is used to make Java classes serializable so that they can be transferred over network or their state can be saved on disk, but it leverages default serialization built-in JVM, which is expensive, fragile and not secure. Externalizable allows you to fully control the Serialization process, specify a custom binary format and add more security measure.  
  
**130) The difference between DOM and SAX parser in Java**   
Another common Java question but from XML parsing topic. It's rather simple to answer and that's why many interviewers prefers to ask this question on the telephonic round. DOM parser loads the whole XML into memory to create a tree based DOM model which helps it quickly locate nodes and make a change in the structure of XML while SAX parser is an event based parser and doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and not suitable to parse large XML files.  
  
**131) Tell me 3 features introduced on JDK 1.7**   
This is one of the good questions I ask to check whether the candidate is aware of recent development in Java technology space or not. Even though JDK 7 was not a big bang release like JDK 5 or JDK 8, it still has a lot of good feature to count on e.g. try-with-resource statements, which free you from closing streams and resources when you are done with that, Java automatically closes that. Fork-Join pool to implement something like the Map-reduce pattern in Java. Allowing String variable and literal into switch statements. Diamond operator for improved type inference, no need to declare generic type on the right-hand side of variable declaration anymore, results in more readable and succinct code. Another worth noting feature introduced was improved exception handling e.g. allowing you to catch multiple exceptions in the same catch block.  
  
**132) Tell me 5 features introduced in JDK 1.8**

1. Lambda expression, which allows you pass an anonymous function as object.

2. Stream API, take advantage of multiple cores of modern CPU and allows you to write succinct code.

3. Date and Time API, finally you have a solid and easy to use date and time library right into JDK

4. Extension methods, now you can have static and default method into your interface

5. Repeated annotation, allows you apply the same annotation multiple times on a type  
  
**133) What is the difference between Maven and ANT in Java?** ([answer](http://javarevisited.blogspot.sg/2015/01/difference-between-maven-ant-jenkins-and-hudson.html))  
Another great question to check the all round knowledge of Java developers. It's easy to answer questions from core Java but when you ask about setting things up, building Java artifacts, many Java software engineer struggles. Coming back to the answer of this question, Though both are build tool and used to create Java application build, Maven is much more than that. It provides standard structure for Java project based upon "convention over configuration" concept and automatically manage dependencies (JAR files on which your application is dependent) for Java application. Please see the answer for more differences between Maven and ANT tool.

**Question 1: What’s wrong using HashMap in the multi-threaded environment? When does the get() method go to an infinite loop?**

Well, nothing is wrong, depending on how you use it. For example, if you initialize the HashMap just by one thread and then all threads are only reading from it, then it’s perfectly fine. One example of this is a Map which contains configuration properties.The real problem starts when at-least one of that thread is updating HashMap i.e. adding, changing or removing any key value pair. Since put() operation can cause re-sizing and which can further lead to infinite loop, that’s why either you should use [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) or [ConcurrentHashMap](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html), later is better.

**Question 2. Does overriding the hashCode() method have any performance implication?** 🡺 This is a good question and open to all, as per my knowledge a poor hash code function will result in the frequent collision in HashMap which eventually increases the time for adding an object into Hash Map.From Java 8 onwards though, collision will not impact performance as much as it does in earlier versions, because after a threshold the linked list will be replaced by the binary tree, which will give you O(logN) performance in the worst case, as compared to O(n) of linked list.

**Question 3: Do all properties of an Immutable Object need to be final 🡺**Not necessarily, as stated above you can achieve same functionality by making the member non-final but private and not modifying them except in a constructor. Don’t provide setter methods for them and if it is a mutable object, then don’t ever leak any reference for that member. Remember making a reference variable final, only ensures that it will not be reassigned to a different value, but you can still change individual properties of object, pointed by that reference variable. This is one of the key point, Interviewer like to hear from candidates.

**Question 4: How does the substring() method inside String works 🡺**Another good Java interview question, I think the answer is not sufficient, but here it is “Substring creates a new object out of source string by taking a portion of original string”.This question was mainly asked to see if the developer is familiar with the risk of memory leaks, which a sub-string can create. Until Java 1.7, substring holds the reference of the original character array, which means even a sub-string of 5 character long, can prevent 1GB character array from garbage collection, by holding a strong reference.

This issue is fixed in Java 1.7, where the original character array is not referenced anymore, but that change also made the creation of substring a bit more costly in terms of time. Earlier it was on the range of O(1), which could be O(n) in worst case on Java 7.

**Question** 5: Can you write a critical section code for the singleton 🡺This core Java question is a followup of the previous question and expecting the candidate to write Java singleton using double checked locking. Remember to use the volatile variable to make Singleton thread-safe.

**Question 6: How do you handle error condition while writing stored procedure or accessing stored procedure from java 🡺** This is one of the tough Java interview questions and its open for all, my friend didn’t know the answer so he didn’t mind telling me. My take is that stored procedure should return an error code if some operation fails but if stored procedure itself fails than catching SQLException is the only choice.

**Question 7 : What is difference between Executor.submit() and Executer.execute() methods ?**  🡺This question is from my list of Top 15 Java multi-threading question answers. It’s getting popular day by day because of huge demand of Java developers with good concurrency skills. The answer is that former returns an object of Future which can be used to find result from worker thread.There is a difference when looking at exception handling. If your tasks throw an exception and if it was submitted with executing this exception will go to the uncaught exception handler (when you don’t have provided one explicitly, the default one will just print the stack trace to System.err).If you submitted the task with submit any thrown exception, [checked exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) or not, is then part of the task’s return status. For a task that was submitted with submitting and that terminates with an exception, the Future.get() will re-throw this exception, wrapped in an ExecutionException.

**Question 8:  What is the difference between factory and abstract factory pattern?** 🡺Abstract Factory provides one more level of abstraction. Consider different factories each extended from an Abstract Factory and responsible for the creation of different hierarchies of objects based on the type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for the creation of objects in that genre

.

**Question 9: What is a Singleton? Is it better to make the whole method synchronized or only critical section synchronized 🡺** Singleton in Java is a class with just one instance in the whole Java application, for example, java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced Enum its very easy.

**Question 10:  Can you write code for iterating over HashMap in Java 4 and Java 5**🡺 Tricky one but he managed to write using while and a for loop. Actually there are four ways to iterate over any Map in Java, one involves using keySet() and iterating over key and then using get() method to retrieve values, which is bit expensive.Second method involves using entrySet() and iterating over them either by using for each loop or while with Iterator.hasNext() method. This one is a better approach because both key and value objects are available to you during Iteration and you don’t need to call the get() method for retrieving value, which could give O(n) performance in case of huge linked list at one bucket.

**Question 11 : When do you override hashCode() and equals() 🡺** Whenever necessary, especially if you want to do equality check based upon business logic rather than object equality, e.g. two employee objects are equal if they have the same emp\_id, despite the fact that they are two different objects, created by different part of the code.Also overriding both these methods are must if you want to use them as key in HashMap. Now as part of the equals-hashcode contract in Java, when you override equals, you must override hashcode as well, otherwise your object will not break invariant of classes e.g. Set, Map which relies on equals() method for functioning properly.

**Question 12 :. What will be the problem if you don’t override hashCode() method ? 🡺**If you don’t override the equals method, then the contract between equals and hashcode will not work, according to which, two objects which are equal by equals() must have the same hashcode. In this case, an other object may return different hashCode and will be stored on that location, which breaks invariant of HashMap class, because they are not supposed to allow duplicate keys.When you add object using put() method, it iterate through all Map.Entry object present in that bucket location, and update value of previous mapping, if Map already contains that key. This will not work if hashcode is not overridden.

**Question 13 : Is it better to synchronize critical sections of getInstance() method or the whole getInstance() method?** 🡺 The answer is only the critical section, because if we lock the whole method, then every time some some one call this method, it will have to wait even though we are not creating any object. In other words, synchronization is only needed, when you create object, which happens only once. Once object has created, there is no need for any synchronization. In fact, that’s very poor coding in terms of performance, as synchronized method reduce performance up to 10 to 20 times. there are several ways to create a thread-safe singleton in Java, which you can also mention as part of this question or any follow-up.

**Question 14: Where does equals() and hashCode() method comes in the picture during the get() operation?** 🡺 This core Java interview question is a follow-up of previous Java question and the candidate should know that once you mention hashCode, people are most likely ask, how they are used in HashMap. When you provide a key object, first it’s hashcode method is called to calculate bucket location. Since a bucket may contain more than one entry as linked list, each of those Map.Entry object is evaluated by using equals() method to see if they contain the actual key object or not.

**Questions 15: How do you avoid a deadlock in Java 🡺** You can avoid deadlock by breaking the circular wait condition. In order to do that, you can make an arrangement in the code to impose the ordering on acquisition and release of locks.If lock will be acquired in a consistent order and released in just opposite order, there would not be a situation where one thread is holding a lock which is acquired by other and vice-versa. See the detailed answer for the code example and more detailed explanation.

**Question 16:  What is the difference between creating String as new() and literal 🡺**When we create the string with new() Operator, it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

String str = new String("Test") does not put the object str in String pool, we need to call String.intern() method which is used to put them into String pool explicitly. It’s only when you create String object as a String literal e.g. String s = "Test" that Java automatically puts that into the String pool. By the way there is a catch here Since we are passing arguments as “Test”, which is a String literal, it will also create another object as “Test” on string pool. This is the one point, which has gone unnoticed until knowledgeable readers of Javarevisited blog suggested it.

**Question 17: What is an Immutable Object? Can you write an Immutable Class?** 🡺 Immutable classes are Java classes whose objects can not be modified once created. Any modification in Immutable object results in the new object. For example, [String is immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html). Mostly Immutable classes are also final in Java, in order to prevent sub classes from overriding methods, which can compromise Immutability. You can achieve the same functionality by making member as non-final but private and not modifying them except in constructor. Apart form obvious, you also need to make sure that, you should not expose the internals of Immutable object, especially if it contains a mutable member. Similarly, when you accept the value for the mutable member from client e.g. java.util.Date, use [clone() method](http://javarevisited.blogspot.sg/2013/09/how-clone-method-works-in-java.html) keep a separate copy for yourself, to prevent the risk of malicious client modifying mutable reference after setting it. The Same precaution needs to be taken while returning value for a mutable member, return another separate copy to the client, never return original reference held by Immutable class. You can see my post How to create an Immutable class in Java for step by step guide and code examples.

**Question 18: Give the simplest way to find out the time a method takes for execution without using any profiling tool?** 🡺 Read the system time just before the method is invoked and immediately after thr method returns. Take the time difference, which will give you the time taken by a method for execution.

long start = System.currentTimeMillis ();

method ();

long end = System.currentTimeMillis (); System.out.println (“Time taken for execution is ” + (end – start));

Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method which is big enough, in the sense the one which is doing considerable amount of processing

**Question 19: Which two methods you need to implement to use an Object as key in HashMap?** 🡺 In order to use any object as Key in HashMap or Hashtable, it must implement equals and hash-code methods in Java. Read How HashMap works in Java for a detailed explanation on how equals and hash code method is used to put and get an object from HashMap.

**1)  What is Thread in Java?**   
The thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound task. For example, if one thread takes 100 milliseconds to do a job, you can use 10 thread to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level, and it's also one of the strong selling points.  
  
**2)  What is the difference between Thread and Process in Java?**  
The thread is a subset of Process, in other words, one process can contain multiple threads. Two process runs on different memory space, but all threads share same memory space. Don't confuse this with stack memory, which is different for the different thread and used to store local data to that thread.   
  
**3)  How do you implement Thread in Java?**   
At the language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it needs a task to execute, which is an instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run() method either by extending Thread class or just implementing Runnable interface.   
  
**4)  When to use Runnable vs Thread in Java**

As we know we can implement thread either by extending Thread class or implementing Runnable interface, Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. Which means, it's better to implement Runnablethen extends Thread if you also want to extend another class   
  
**6)  What is the difference between start() and run() method of Thread class?**

start() method is used to start newly created thread, while start() internally calls run() method, there is difference calling run() method directly. When you invoke run() as normal method, its called in the same thread, no new thread is started, which is the case when you call start() method.   
  
**7)  What is the difference between Runnable and Callable in Java?**   
Both Runnable and Callable represent task which is intended to be executed in a separate thread. Runnable is there from JDK 1.0 while Callable was added on JDK 1.5. Main difference between these two is that Callable's call() method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return Future object, which can hold the result of computation.   
  
**8)  What is the difference between CyclicBarrier and CountDownLatch in Java?**  
Though both CyclicBarrier and CountDownLatch wait for number of threads on one or more events, the main difference between them is that you can not re-use CountDownLatch once count reaches to zero, but you can reuse same CyclicBarrier even after barrier is broken.     
  
**9)  What is Java Memory model?**

Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-before relationship. This relationship defines several rules which allows programmers to anticipate and reason behaviour of concurrent Java programs. For example, happens-before relationship guarantees :

* Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.
* An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.
* A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.
* A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from Thread.join() or by Thread.isAlive() returning false, also known as Thread start rule.
* A thread calling interrupt on another thread happens-before the interrupted thread detects the interrupt( either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.
* The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.
* If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

**10) What is volatile variable in Java?**  
volatile is a special modifier, which can only be used with instance variables. In concurrent Java programs, changes made by multiple threads on instance variables is not visible to other in absence of any synchronizers e.g. synchronized keyword or locks. Volatile variable guarantees that a write will happen before any subsequent read: as stated: *"volatile variable rule"* in previous question.

**11) How does volatile variable works in Java?**

**A.** volatile is a keyword . Volatile guarantee not just limited to the variable but also all the variables [two threads](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) see known as “happens before” relationship. Another important aspect of making a variable volatile is that compiler **will not reorder the variable** when switching from client to server configuration or while performing optimization.

 the volatile keyword in Java is used as an indicator to Java compiler and Thread that do not cache value of this variable and always read it from [main memory](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html)  
The Java volatile keyword cannot be used with method or class and it can only be used with a variable.  
Java volatile keyword also guarantees visibility and ordering, after Java 5 write to any volatile variable happens before any read into the volatile variable

use of volatile keyword also prevents compiler or JVM from the reordering of code or moving away them from synchronization barrier.

Example :

|  |  |
| --- | --- |
| public class Singleton{  private static volatile Singleton \_instance; //volatilevariable  public static Singleton getInstance(){  if(\_instance == null){  synchronized(Singleton.class){  if(\_instance == null)  \_instance = new Singleton(); }  }  return \_instance; } | If you look at the code carefully you will be able to figure out: 1) We are only creating instance one time 2) We are creating instance lazily at the time of the first request comes |

If we do not make the \_instance variable volatile than the Thread which is creating instance of Singleton is not able to communicate other thread, that instance has been created until it comes out of the Singleton block, so if Thread A is creating Singleton instance and just after creation lost the CPU, all other thread will not be able to see value of \_instance as not null and they will believe its still [null](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html).

**When to use :**

1) You can use Volatile variable if you want to read and write long and [double](http://javarevisited.blogspot.sg/2011/10/convert-double-to-string-example.html) variable atomically. long and double both are [64 bit](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html) data type and by default writing of long and double is not atomic and platform dependence. Many platform perform write in long and double variable 2 step, writing 32 bit in each step, due to this its possible for a Thread to see 32 bit from two different write. You can avoid this issue by making long and double variable volatile in Java  
2) A volatile variable can be used as an alternative way of achieving [synchronization in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) in some cases, like Visibility. with volatile variable, it's guaranteed that all reader thread will see updated value of the volatile variable once write operation completed, without volatile keyword different reader thread may see different values.

3) volatile variable can be used to inform the compiler that a particular field is subject to be accessed by multiple threads, which will prevent the compiler from doing any reordering or any kind of optimization which is not desirable in a multi-threaded environment. Without volatile variable compiler can re-order the code, free to cache value of volatile variable instead of always reading from main memory. like following example without volatile variable may result in an [infinite loop](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)

4) Another place where a volatile variable can be used is to fixing double checked locking in Singleton pattern  
  
**11) What is thread-safety? is Vector a thread-safe class?**    
Thread-safety is a property of an object or code which guarantees that if executed or used by multiple threads in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modify state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.  
  
**12) What is race condition in Java? Given one example?**   
Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As the name suggests, a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behavior of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between threads. One example of race condition is out-of-order processing.   
  
**13) How to stop a thread in Java?**

I always said that Java provides rich APIs for everything but ironically Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. stop(), suspend() and resume() which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads. Programmers mainly rely on the fact that thread stops automatically as soon as they finish execution of run() or call() method. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or interrupt threads to abruptly cancel tasks.   
  
**14) What happens when an Exception occurs in a thread?**   
In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for ts UncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.  
  
**15) How do you share data between two thread in Java?**   
You can share data between threads by using shared object, or concurrent data structure like BlockingQueueIt implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.

**16) What is the difference between notify and notifyAll in Java?**   
This is another tricky questions from core Java interviews, since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There notify()method doesn't provide any way to choose a particular thread, that's why its only useful when you know that there is only one thread is waiting. On the other hand, notifyAll() sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further.  
  
**17) Why wait, notify and notifyAll are not inside thread class?**   
One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object.

**18) What is ThreadLocal variable in Java**  
ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, ThreadLocal variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using ThreadLocal. Since that class is expensive, its not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds with one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and Second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is ThreadLocalRandom class, which reduces number of instances of expensive-to-create Random object in multi-threading environment. See this [answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html) to learn more about thread local variables in Java.  
  
**19) What is FutureTask in Java?**  
FutureTask represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A FutureTask object can be used to wrap a Callable or Runnable object. Since FutureTask also implements Runnable, it can be submitted to an Executor for execution.  
  
**20) What is the difference between the interrupted() and isInterrupted() method in Java?**   
Main difference between interrupted() and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the [static method](http://java67.blogspot.com/2012/11/what-is-static-class-variable-method.html) Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted() method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt  
  
**21) Why wait and notify method are called from synchronized block?**   
Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A more subtle reason is to avoid the race condition between wait and notify calls.   
  
**22) Why should you check condition for waiting in a loop**  
Its possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.  
  
**23) What is the difference between synchronized and concurrent collection in Java?**   
Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like ConcurrentHashMap, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table. See this [answer](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-synchronized.html) for more differences between synchronized and concurrent collection in Java.

**24) What is the difference between Stack and Heap in Java?**   
Why does someone this question as part of multi-threading and concurrency? because Stack is a memory area which is closely associated with threads. To answer this question, both stack and heap are specific memories in Java application. Each thread has their own stack, which is used to store local variables, method parameters and call stack. Variable stored in one Thread's stack is not visible to other. On another hand, the heap is a common memory area which is shared by all threads. Objects whether local or at any level is created inside heap. To improve performance thread tends to cache values from heap into their stack, which can create problems if that variable is modified by more than one thread, this is where volatile variables come into the picture. volatile suggest threads read the value of variable always from main memory.

**25) What is thread pool? Why should you thread pool in Java?**  
Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issues, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of threads) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks). See this [article](http://javarevisited.blogspot.com/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html) to learn more about thread pools in Java to prepare detailed answer of this question.  
  
**26) Write code to solve Producer Consumer problem in Java?**([answer](http://java67.blogspot.com/2015/12/producer-consumer-solution-using-blocking-queue-java.html))  
Most of the threading problem you solved in the real world are of the category of Producer consumer pattern, where one thread is producing task and another thread is consuming that. You must know how to do inter thread communication to solve this problem. At the lowest level, you can use wait and notify to solve this problem, and at a high level, you can leverage Semaphore or BlockingQueue to implement Producer consumer pattern, as shown in this [tutorial](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html).  
  
**27) How do you avoid deadlock in Java? Write Code?**

Deadlock is a condition in which two threads wait for each other to take action which allows them to move further. It's a serious issue because when it happen your program hangs and doesn't do the task it is intended for. In order for deadlock to happen, following four conditions must be true:  
**Mutual Exclusion :** At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.

**Hold and Wait:** A process is currently holding, at least, one resource and requesting additional resources which are being held by other processes.

**No Pre-emption:** The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.

**Circular Wait:**A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.  
The easiest way to avoid deadlock is to prevent *Circular wai*t, and this can be done by acquiring locks in a particular order and releasing them in reverse order so that a thread can only proceed to acquire a lock if it held the other one.   
  
**28) What is the difference between livelock and deadlock in Java?**   
A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.  
  
**29) How do you check if a Thread holds a lock or not**  
I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object.   
  
**30) How do you take thread dump in Java?**   
There are multiple ways to take thread dump of Java process depending upon operating system. When you take thread dump, JVM dumps state of all threads in log files or standard error console. In windows you can use Ctrl + Break key combination to take thread dump, on Linux you can use kill -3 command for same. You can also use a tool called jstack for taking thread dump, it operate on process id, which can be found using another tool called jps.  
  
**31) Which JVM parameter is used to control stack size of a thread?**   
This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this [list of JVM options](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html) to learn more about this parameter.  
**32) What is the difference between synchronized and ReentrantLock in Java?**   
There were days when the only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you can not extend lock beyond a method or block boundary, you can not give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities. See [this article](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html) learn about those capabilities and some more differences between synchronized vs ReentrantLock in Java.  
  
**33) There are three threads T1, T2, and T3? How do you ensure sequence T1, T2, T3 in Java?** Sequencing in multi-threading can be achieved by different means but you can simply use the join() method of thread class to start a thread when another one has finished its execution. To ensure three threads execute you need to start the last one first e.g. T3 and then call join methods in reverse order e.g. T3 calls T2. join and T2 calls T1.join, these ways T1 will finish first and T3 will finish last. To learn more about join method,   
  
**34) What does yield method of Thread class do?**   
Yield method is one way to request current thread to relinquish CPU so that other thread can get a chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for the same thread to get CPU back and start its execution again.   
  
**35) What is the concurrency level of ConcurrentHashMap in Java**)  
ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. To learn more about concurrency level and internal resizing,   
  
**36) What is Semaphore in Java?** Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool.

**37) What happens if you submit a task when the queue of the thread pool is already filled?**   
This is another tricky question on my list. Many programmers will think that it will block until a task is cleared but its true. ThreadPoolExecutor's submit() method throws RejectedExecutionException if the task cannot be scheduled for execution.  
  
**38) What is the difference between the submit() and execute() method thread pool in Java?**   
Both methods are ways to submit a task to thread pools but there is a slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly, it does not return anything. Its return type is void. On other hand submit() is an overloaded method, it can take either Runnable or Callable task and can return Future object which can hold the pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods.   
  
**39) What is blocking method in Java?**   
A blocking method is a method which blocks until the task is done, for example, accept() method of ServerSocket blocks until a client is connected. here blocking means control will not return to the caller until the task is finished. On the other hand, there is an asynchronous or non-blocking method which returns even before the task is finished.   
  
**40) Is Swing thread-safe? What do you mean by Swing thread-safe?** You can simply this question as No, Swing is not thread-safe, but you have to explain what you mean by that even if the interviewer doesn't ask about it. When we say swing is not thread-safe we usually refer its component, which can not be modified in multiple threads. All update to GUI components has to be done on AWT thread, and Swing provides synchronous and asynchronous callback methods to schedule such updates. You can also read my article to learn more about [swing and thread-safety](http://javarevisited.blogspot.com/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html) to better answer this question. Even next two questions are also related to this concept.  
  
**41) What is the difference between invokeAndWait and invokeLater in Java?** These are two methods Swing API provides Java developers for updating GUI components from threads other than Event dispatcher thread. InvokeAndWait() synchronously update GUI component, for example, a progress bar, once progress is made, the bar should also be updated to reflect that change. If progress is tracked in a different thread, it has to call invokeAndWait() to schedule an update of that component by Event dispatcher thread. On another hand, invokeLater() is an asynchronous call to update components.   
  
**42) Which method of Swing API are thread-safe in Java?**   
This question is again related to swing and thread-safety though components are not thread-safe there is a certain method which can be safely called from multiple threads. I know about repaint(), and revalidate() being thread-safe but there are other methods on different swing components e.g. setText() method of JTextComponent, insert() and append()method of JTextArea class.  
  
**43) How to create an Immutable object in Java?**   
This question might not look related to multi-threading and concurrency, but it is. Immutability helps to simplify already complex concurrent code in Java. Since immutable object can be shared without any synchronization its very dear to Java developers. Core value object, which is meant to be shared among thread should be immutable for performance and simplicity. Unfortunately there is no @Immutable annotation in Java, which can make your object immutable, hard work must be done by Java developers. You need to keep basics like initializing state in constructor, no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object.   
  
**44) What is ReadWriteLock in Java?**   
In general, read write lock is the result of lock stripping technique to improve the performance of concurrent applications. In Java, ReadWriteLock is an interface which was added in Java 5 release. A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive. If you want you can implement this interface with your own set of rules, otherwise you can use ReentrantReadWriteLock, which comes along with JDK and supports a maximum of 65535 recursive write locks and 65535 read locks.  
  
  
**45) What is busy spin in multi-threading?**   
Busy spin is a technique which concurrent programmers employ to make a thread wait on certain condition. Unlike traditional methods e.g. wait(), sleep() or yield() which all involves relinquishing CPU control, this method does not relinquish CPU, instead it the just runs empty loop. Why would someone do that? to preserve CPU caches. In a multi-core system, it's possible for a paused thread to resume on a different core, which means rebuilding cache again. To avoid cost of rebuilding cache, programmer prefer to wait for much smaller time doing busy spin.  
  
**46) What is the difference between the volatile and atomic variable in Java?**   
This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.  
  
**47) What happens if a thread throws an Exception inside synchronized block?** This is one more tricky question for average Java programmer, if he can bring the fact about whether lock is released or not is a key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a [finally block](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html).  
  
**48) What is double checked locking of Singleton?**  
This is one of the very popular question on Java interviews, and despite its popularity, chances of candidate answering this question satisfactory is only 50%. Half of the time, they failed to write code for double checked locking and half of the time they failed how it was broken and fixed on Java 1.5. This is actually an old way of creating thread-safe singleton, which tries to optimize performance by only locking when Singleton instance is created first time, but because of complexity and the fact it was broken for JDK 1.4,  I personally don't like it. Anyway, even if you not prefer this approach its good to know from interview point of view. Since this question deserve a detailed answer, I have answered in a separate post,   
  
**49) How to create thread-safe Singleton in Java**  
This question is actually follow-up of the previous question. If you say you don't like double checked locking then Interviewer is bound to ask about alternative ways of creating thread-safe Singleton class. There are actually man, you can take advantage of class loading and static variable initialization feature of JVM to create instance of Singleton, or you can leverage powerful enumeration type in Java to create Singleton  
  
**50) List down 3 multi-threading best practice you follow?**   
This is my favorite question because I believe that you must follow certain best practices while writing concurrent code which helps in performance, debugging and maintenance. Following are three best practices, I think an average Java programmer should follow:

**Always give meaningful name to your thread**This goes a long way to find a bug or trace an execution in concurrent code. OrderProcessor, QuoteProcessor or TradeProcessor is much better than Thread-1. Thread-2 and Thread-3. The name should say about task done by that thread. All major framework and even JDK follow this best practice.

**Avoid locking or Reduce scope of Synchronization**  
Locking is costly and context switching is even costlier. Try to avoid synchronization and locking as much as possible and at a bare minimum, you should reduce critical section. That's why I prefer synchronized block over synchronized method because it gives you absolute control on the scope of locking.

**Prefer Synchronizers over wait and notify**  
Synchronizers like CountDownLatch, Semaphore, CyclicBarrier or Exchangersimplifies coding. It's very difficult to implement complex control flow right using wait and notify. Secondly, these classes are written and maintained by best in business and there is good chance that they are optimized or replaced by better performance code in subsequent JDK releases. By using higher level synchronization utilities, you automatically get all these benefits.

**Prefer Concurrent Collection over Synchronized Collection**  
This is another simple best practice which is easy to follow but reap good benefits. Concurrent collection are more scalable than their synchronized counterpart, that's why its better to use them while writing concurrent code. So next time if you need map, think about ConcurrentHashMap before thinking Hashtable.

**51) How do you force to start a Thread in Java?**   
This question is like how do you force garbage collection in Java, there is no way though you can make a request using System.gc() but it's not guaranteed. On Java multi-threading there is absolutely no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.  
  
**52) What is the fork-join framework in Java?**   
The fork-join framework, introduced in JDK 7 is a powerful tool available to Java developer to take advantage of multiple processors of modern day servers. 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. One significant advantage of The fork/joinframework is that 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.

**53) What is the difference between calling wait() and sleep() method in Java multi-threading**

Though both wait and sleep introduce some form of pause in Java application, they are the tool for different needs. Wait method is used for inter-thread communication, it relinquishes lock if waiting for a condition is true and wait for notification when due to an action of another thread waiting condition becomes false. On the other hand sleep() method is just to relinquish CPU or stop execution of current thread for specified time duration. Calling sleep method doesn't release the lock held by the current thread

**1. Why String is immutable in java?**

**A. 1.** tring pool requires string to be immutable otherwise shared reference can be changed from anywhere

2. security because string is shared on different area like file system, networking connection, database connection , having immutable string allows you to be secure and safe because no one can change reference of string once it gets created. if string had been mutable anyone can surpass the security be logging in someone else name and then later modifying file belongs to other

**2) Why multiple inheritances are not supported in Java?**

**A.** Short answer is because of diamond pattern, diamond pattern creates ambiguity and make problem for compiler. Anyway java supports multiple inheritances via interfaces. I think more convincing reason for not supporting multiple inheritance is complexity involved in constructor chaining, casting etc. rather than diamond problem

**3) How to detect deadlock and fix it?**

**A .** When two or more threads waiting for each other to release lock and get stuck for infinite time, situation is called deadlock. Other way is to find it when you actually get locked while running the application, try to take thread dump, in Linux you can do this by command **"kill -3”,** this will print status of all the thread in application log file and you can see which thread is locked on which object.

**Code for deadlock**

|  |
| --- |
| **public class DeadLockDemo {**  **public void method1() {**  **synchronized (String.class) {**  **System.out.println("Aquired lock on String.class object");**  **synchronized (Integer.class) {**  **System.out.println("Aquired lock on Integer.class object");**  **} }}**  **synchronized (Integer.class) {**  **System.out.println("Aquired lock on Integer.class object");**  **synchronized (String.class) {**  **System.out.println("Aquired lock on String.class object");**  **} } }}** |

**4) When a singleton is not singleton in Java?**

**A.** Many cases like Serialization, RMI, Class loader loading multiple instance etc.

**5) Difference between**[**noclassdeffounderror**](http://javarevisited.blogspot.com/2011/06/noclassdeffounderror-exception-in.html)**and classnotfoundexception?**

**A.** java.lang.NoClassDefFoundError occurs when a particular class was present during Compile time but not available during run time by any reason, while j[ava.lang.ClassNotFoundException](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html)comes when class loaded explicitly during runtime.

**6) Why equals() method must be compatible with compareTo in java ?**

A. [compareTo method in Java](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) must be compatible with equals method in Java i.e. if two objects are equal via equals method compareTo method must return “0” for them, failing this may result in some subtle bug when you store those objects in collection class like [arraylist in java](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html).

**7)**[**Explain race condition in Java hashmap**](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html)**?**

**A.** Race conditions in Java are always tricky to find and solve. Not many people know about it that hashmap could run into race condition if it would be modified by two threads simultaneous and one thread tries to re-size or rehash the map because of capacity crossing threshold value. Since hashmap maintains a linked list of element in bucket and while copying from one hashmap to other or old to new order of linked list got reversed, which could result in infinite loop if two threads are doing resizing at same time?

Otherwise some set e.g. TreeSet and TreeMap which uses compareTo will not able to detect duplicates and allow duplicate inside set.

**Race conditions** occurs when two thread operate on same object without proper synchronization and there operation interleaves on each other.

**8) Can we call static method with null object?**

**A.** Yes we can call because static method is bound at compile time and only type of variable is used for [static binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) not the value of object. It’s not a good practice to [call static method by instance](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html), always call static method in Java by using class name like Math.max().

**10) How to detect memory leak in Java.**

**A.** you can use profile and memory dump to find memory leak in Java. JConsole can also help to provide graph of memory usage which can show pattern for memory leak.

1. What is the performance effect of a large number of import statements which are not used?  
   **Answer:** They are ignored if the corresponding class is not used.
2. Give a scenario where hotspot will optimize your code?  
   **Answer**: If we have defined a variable as static and then initialized this variable in a static block then the Hotspot will merge the variable and the initialization in a single statement and hence reduce the code.
3. What will happen if an exception is thrown from the finally block?  
   **Answer:** The program will exit if the exception is not catched in the finally block.
4. How does decorator design pattern works in I/O classes?  
   **Answer:**  The various classes like BufferedReader , BufferedWriter workk on the underlying stream classes. Thus Buffered\* class will provide a Buffer for Reader/Writer classes.
5. If I give you an assignment to design Shopping cart web application, how will you define the architecture of this application. You are free to choose any framework, tool or server?  
   **Answer:**  Usually I will choose a MVC framework which will make me use other design patterns like Front Controller, Business Delegate, Service Locater, DAO, DTO, Loose Coupling etc. Struts 2 is very easy to configure and comes with other plugins like Tiles, Velocity and Validator etc. The architecture of Struts becomes the architecture of my application with various actions and corresponding JSP pages in place.
6. What is a deadlock in Java? How will you detect and get rid of deadlocks?  
   **Answer:**  Deadlock exists when two threads try to get hold of a object which is already held by another object.
7. Why is it better to use hibernate than JDBC for database interaction in various Java applications?  
   **Answer:**  Hibernate provides an OO view of the database by mapping the various classes to the database tables. This helps in thinking in terms of the OO language then in RDBMS terms and hence increases productivity.
8. How can one call one constructor from another constructor in a class?  
   **Answer:**  Use the this() method to refer to constructors.
9. What is the purpose of intern() method in the String class?  
   **Answer:**  It helps in moving the normal string objects to move to the String literal pool
10. How will you make your web application to use the https protocol?  
    **Answer:**  This has more to do with the particular server being used  than the application itself. Here is how it can be done on tomcat:

**[What is immutable object in Java? Can you change values of a immutable object?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-immutable-object-in-java--can-you-change-values-of-a-immutable-object-" \o "What is immutable object in Java? Can you change values of a immutable object?)**

A Java object is considered immutable when its state cannot change after it is created. Use of immutable objects is widely accepted as a sound strategy for creating simple, reliable code. Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state. java.lang.String and java.lang.Integer classes are the Examples of immutable objects from the Java Development Kit. Immutable objects simplify your program due to following characteristics :

* Immutable objects are simple to use test and construct.
* Immutable objects are automatically thread-safe.
* Immutable objects do not require a copy constructor.
* Immutable objects do not require an implementation of clone.
* Immutable objects allow hashCode to use lazy initialization, and to cache its return value.
* Immutable objects do not need to be copied defensively when used as a field.
* Immutable objects are good Map keys and Set elements (Since state of these objects must not change while stored in a collection).
* Immutable objects have their class invariant established once upon construction, and it never needs to be checked again.
* Immutable objects always have "failure atomicity" (a term used by Joshua Bloch) : if an immutable object throws an exception, it's never left in an undesirable or indeterminate state.

**[How to create a immutable object in Java? Does all property of immutable object needs to be final?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "how-to-create-a-immutable-object-in-java--does-all-property-of-immutable-object-needs-to-be-final-" \o "How to create a immutable object in Java? Does all property of immutable object needs to be final?)**

To create a object immutable You need to make the class final and all its member final so that once objects gets crated no one can modify its state. You can achieve same functionality by making member as non final but private and not modifying them except in constructor. Also its NOT necessary to have all the properties final since you can achieve same functionality by making member as non final but private and not modifying them except in constructor.

**[What is difference between String, StringBuffer and StringBuilder? When to use them?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-difference-between-string--stringbuffer-and-stringbuilder--when-to-use-them-" \o "What is difference between String, StringBuffer and StringBuilder? When to use them?)**

The main difference between the three most commonly used String classes as follows.

* StringBuffer and StringBuilder objects are mutable whereas String class objects are immutable.
* StringBuffer class implementation is synchronized while StringBuilder class is not synchronized.
* Concatenation operator "+" is internally implemented by Java using either StringBuffer or StringBuilder.

Criteria to choose among String, StringBuffer and StringBuilder

* If the Object value will not change in a scenario use String Class because a String object is immutable.
* If the Object value can change and will only be modified from a single thread, use a StringBuilder because StringBuilder is unsynchronized(means faster).
* If the Object value may change, and can be modified by multiple threads, use a StringBuffer because StringBuffer is thread safe(synchronized).

**[Why String class is final or immutable?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-string-class-is-final-or-immutable-" \o "Why String class is final or immutable?)**

It is very useful to have strings implemented as final or immutable objects. Below are some advantages of String Immutability in Java

* Immutable objects are thread-safe. Two threads can both work on an immutable object at the same time without any possibility of conflict.
* Security: the system can pass on sensitive bits of read-only information without worrying that it will be altered
* You can share duplicates by pointing them to a single instance.
* You can create substrings without copying. You just create a pointer into an existing base String guaranteed never to change. Immutability is the secret that makes Java substring implementation very fast.
* Immutable objects are good fit for becoming Hashtable keys. If you change the value of any object that is used as a hash table key without removing it and re-adding it you will lose the object mapping.
* Since String is immutable, inside each String is a char[] exactly the correct length. Unlike a StringBuilder there is no need for padding to allow for growth.
* If String were not final, you could create a subclass and have two strings that look alike when "seen as Strings", but that are actually different.

**[Is Java Pass by Reference or Pass by Value?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "is-java-pass-by-reference-or-pass-by-value-" \o "Is Java Pass by Reference or Pass by Value?)**

The Java Spec says that everything in Java is pass-by-value. There is no such thing as "pass-by-reference" in Java. The difficult thing can be to understand that Java passes "objects as references" passed by value. This can certainly get confusing and I would recommend reading this article from an expert: <http://javadude.com/articles/passbyvalue.htm> Also read this interesting thread with example on StackOverflow : [Java Pass By Ref or Value](http://stackoverflow.com/questions/40480/is-java-pass-by-reference)

**[What is OutOfMemoryError in java? How to deal with java.lang.OutOfMemeryError error?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-outofmemoryerror-in-java--how-to-deal-with-java-lang-outofmemeryerror--error-" \o "What is OutOfMemoryError in java? How to deal with java.lang.OutOfMemeryError error?)**

This Error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector. **Note:** Its an Error (extends java.lang.Error) not Exception. Two important types of OutOfMemoryError are often encountered

1. **java.lang.OutOfMemoryError: Java heap space**

The quick solution is to add these flags to JVM command line when Java runtime is started:

-Xms1024m -Xmx1024m

1. **java.lang.OutOfMemoryError: PermGen space**

The solution is to add these flags to JVM command line when Java runtime is started:

-XX:+CMSClassUnloadingEnabled-XX:+CMSPermGenSweepingEnabled

**Long Term Solution**: Increasing the Start/Max Heap size or changing Garbage Collection options may not always be a long term solution for your Out Of Memory Error problem. Best approach is to understand the memory needs of your program and ensure it uses memory wisely and does not have leaks. You can use a Java memory profiler to determine what methods in your program are allocating large number of objects and then determine if there is a way to make sure they are no longer referenced, or to not allocate them in the first place.

**[What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-the-use-of-the-finally-block--is-finally-block-in-java-guaranteed-to-be-called--when-finally-block-is-not-called-" \o "What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?)**

Finally is the block of code that executes always. The code in finally block will execute even if an exception is occurred. Finally block is NOT called in following conditions

* If the JVM exits while the try or catch code is being executed, then the finally block may not execute. This may happen due to System.exit() call.
* if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.
* If a exception is thrown in finally block and not handled then remaining code in finally block may not be executed.

**[Why there are two Date classes; one in java.util package and another in java.sql?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-there-are-two-date-classes--one-in-java-util-package-and-another-in-java-sql-" \o "Why there are two Date classes; one in java.util package and another in java.sql?)**

From the JavaDoc of java.sql.Date:

A thin wrapper around a millisecond value that allows JDBC to identify this as an SQL DATE value. A milliseconds value represents the number of milliseconds that have passed since January 1, 1970 00:00:00.000 GMT. To conform with the definition of SQL DATE, the millisecond values wrapped inside a java.sql.Date instance must be 'normalized' by setting the hours, minutes, seconds, and milliseconds to zero.

**Explanation**: A java.util.Date represents date and time of day, a java.sql.Date only represents a date (the complement of java.sql.Date is java.sql.Time, which only represents a time of day, but also extends java.util.Date).

**[What is Marker interface? How is it used in Java?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-marker-interface--how-is-it-used-in-java-" \o "What is Marker interface? How is it used in Java?)**

The marker interface is a design pattern, used with languages that provide run-time type information about objects. It provides a way to associate metadata with a class where the language does not have explicit support for such metadata. To use this pattern, a class implements a marker interface, and code that interact with instances of that class test for the existence of the interface. Whereas a typical interface specifies methods that an implementing class must support, a marker interface does not do so. The mere presence of such an interface indicates specific behavior on the part of the implementing class. There can be some hybrid interfaces, which both act as markers and specify required methods, are possible but may prove confusing if improperly used. Java utilizes this pattern very well and the example interfaces are

* java.io.Serializable - Serializability of a class is enabled by the class implementing the java.io.Serializable interface. The Java Classes that do not implement Serializable interface will not be able to serialize or deserializ their state. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.
* java.rmi.Remote - The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a "remote interface", an interface that extends java.rmi.Remote are available remotely.
* java.lang.Cloneable - A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class. Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.
* javax.servlet.SingleThreadModel - Ensures that servlets handle only one request at a time. This interface has no methods.
* java.util.EvenListener - A tagging interface that all event listener interfaces must extend.

The "instanceof" keyword in java can be used to test if an object is of a specified type. So this keyword in combination with Marker interface can be used to take different actions based on type of interface an object implements.

**[Why main() in java is declared as public static void main? What if the main method is declared as private?](https://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-main---in-java-is-declared-as-public-static-void-main-" \o "Why main() in java is declared as public static void main?)**

Public - main method is called by JVM to run the method which is outside the scope of project therefore the access specifier has to be public to permit call from anywhere outside the application static - When the JVM makes are call to the main method there is not object existing for the class being called therefore it has to have static method to allow invocation from class. void - Java is platform independent language therefore if it will return some value then the value may mean different to different platforms so unlike C it can not assume a behavior of returning value to the operating system. If main method is declared as private then - Program will compile properly but at run-time it will give "Main method not public." error.

## What are available drivers in JDBC?

## **What are the types of statements in JDBC?**

the JDBC API has 3 Interfaces, (1. Statement, 2. PreparedStatement, 3. CallableStatement ). The key features of these are as follows: **Statement**

* This interface is used for executing a static SQL statement and returning the results it produces.
* The object of Statement class can be created using Connection.createStatement() method.

**PreparedStatement**

* A SQL statement is pre-compiled and stored in a PreparedStatement object.
* This object can then be used to efficiently execute this statement multiple times.
* The object of PreparedStatement class can be created using Connection.prepareStatement() method. This extends Statement interface.

**CallableStatement**

* This interface is used to execute SQL stored procedures.
* This extends PreparedStatement interface.
* The object of CallableStatement class can be created using Connection.prepareCall() method.

## **What is a stored procedure? How to call stored procedure using JDBC API?**

Stored procedure is a group of SQL statements that forms a logical unit and performs a particular task. Stored Procedures are used to encapsulate a set of operations or queries to execute on database. Stored procedures can be compiled and executed with different parameters and results and may have any combination of input/output parameters. Stored procedures can be called using CallableStatement class in JDBC API. Below code snippet shows how this can be achieved.

CallableStatement cs = con.prepareCall("{call MY\_STORED\_PROC\_NAME}");

ResultSet rs = cs.executeQuery();

## What is Connection pooling? What are the advantages of using a connection pool?

Connection Pooling is a technique used for sharing the server resources among requested clients. It was pioneered by database vendors to allow multiple clients to share a cached set of connection objects that provides access to a database.   
Getting connection and disconnecting are costly operation, which affects the application performance, so we should avoid creating multiple connection during multiple database interactions. A pool contains set of Database connections which are already connected, and any client who wants to use it can take it from pool and when done with using it can be returned back to the pool.   
Apart from performance this also saves you resources as there may be limited database connections available for your application.

## How to do database connection using JDBC thin driver ?

This is one of the most commonly asked questions from JDBC fundamentals, and knowing all the steps of JDBC connection is important.

import java.sql.\*;

class JDBCTest {

public static void main (String args []) throws Exception

{

//Load driver class

Class.forName ("oracle.jdbc.driver.OracleDriver");

//Create connection

Connection conn = DriverManager.getConnection

("jdbc:oracle:thin:@hostname:1526:testdb", "scott", "tiger");

// @machineName:port:SID, userid, password

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("select 'Hi' from dual");

while (rs.next())

System.out.println (rs.getString(1)); // Print col 1 => Hi

stmt.close();

}}

## **What does Class.forName() method do?**

Method forName() is a static method of java.lang.Class. This can be used to dynamically load a class at run-time. Class.forName() loads the class if its not already loaded. It also executes the static block of loaded class. Then this method returns an instance of the loaded class. So a call to Class.forName('MyClass') is going to do following   
- Load the class MyClass.  
- Execute any static block code of MyClass.  
- Return an instance of MyClass.  
  
JDBC Driver loading using Class.forName is a good example of best use of this method. The driver loading is done like this 

Class.forName("org.mysql.Driver");

All JDBC Drivers have a static block that registers itself with DriverManager and DriverManager has static initializer method registerDriver() which can be called in a static blocks of Driver class. A MySQL JDBC Driver has a static initializer which looks like this: 

## **Which one will you use Statement or PreparedStatement? Or Which one to use when** (Statement/PreparedStatement)? Compare PreparedStatement vs Statement.

By Java API definitions: **Statement** is a object used for executing a static SQL statement and returning the results it produces. **PreparedStatement**is a SQL statement which is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times. There are few advantages of using PreparedStatements over Statements

1. Since its pre-compiled, Executing the same query multiple times in loop, binding different parameter values each time is faster. (What does pre-compiled statement means? The prepared statement(pre-compiled) concept is not specific to Java, it is a database concept. Statement precompiling means: when you execute a SQL query, database server will prepare a execution plan before executing the actual query, this execution plan will be cached at database server for further execution.)
2. In PreparedStatement the setDate()/setString() methods can be used to escape dates and strings properly, in a database-independent way.
3. SQL injection attacks on a system are virtually impossible when using PreparedStatements.

## **What does setAutoCommit(false) do?**

A JDBC connection is created in auto-commit mode by default. This means that each individual SQL statement is treated as a transaction and will be automatically committed as soon as it is executed. If you require two or more statements to be grouped into a transaction then you need to disable auto-commit mode using below command

con.setAutoCommit(false);

Once auto-commit mode is disabled, no SQL statements will be committed until you explicitly call the commit method. A Simple transaction with use of autocommit flag is demonstrated below.

con.setAutoCommit(false);

PreparedStatement updateStmt =

con.prepareStatement( "UPDATE EMPLOYEE SET SALARY = ? WHERE EMP\_NAME LIKE ?");

updateStmt.setInt(1, 5000); updateSales.setString(2, "Jack");

updateStmt.executeUpdate();

updateStmt.setInt(1, 6000); updateSales.setString(2, "Tom");

updateStmt.executeUpdate();

con.commit();

con.setAutoCommit(true);

## [What is Java Collections API?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Java-Collections-API)

Java Collections framework API is a **unified architecture** for representing and manipulating collections. The API contains Interfaces, Implementations & Algorithm to help java programmer in everyday programming. In nutshell, this API does 6 things at high level

* Reduces programming efforts. - Increases program speed and quality.
* Allows interoperability among unrelated APIs.
* Reduces effort to learn and to use new APIs.
* Reduces effort to design new APIs.
* Encourages & Fosters software reuse.

To be specific, There are **six collection java interfaces**. The most basic interface is Collection. Three interfaces extend Collection: Set, List, and SortedSet. The other two collection interfaces, Map and SortedMap, do not extend Collection, as they represent mappings rather than true collections.

## [What is an Iterator?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "What-is-Iterator)

Some of the collection classes **provide traversal** of their contents via a java.util.Iterator interface. This interface allows you to walk through a collection of objects, operating on each object in turn. Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

## [What is the difference between java.util.Iterator and java.util.ListIterator?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Iterator-vs-ListIterator)

* **Iterator** : Enables you to traverse through a collection in the forward direction only, for obtaining or removing elements
* **ListIterator** : extends Iterator, and allows bidirectional traversal of list and also allows the modification of elements.

## [What is HashMap and Map?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "What-is-HashMap)

Map is Interface which is part of Java collections framework. This is to store Key Value pair, and Hashmap is class that implements that using hashing technique.

## [Difference between HashMap and HashTable? Compare Hashtable vs HashMap?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "HashMap-vs-HashTable)

Both Hashtable and HashMap provide key-value access to data. The Hashtable is one of the original collection classes in Java (also called as legacy classes). HashMap is part of the new Collections Framework, added with Java 2, v1.2. There are several differences between HashMap and Hashtable in Java as listed below

* The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn’t allow nulls).
* HashMap does not guarantee that the order of the map will remain constant over time. But one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.
* HashMap is non synchronized whereas Hashtable is synchronized.
* Iterator in the HashMap is fail-fast while the enumerator for the Hashtable isn't. So this could be a design consideration.

## [What does synchronized means in Hashtable context?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "synchronized-Hashtable)

Synchronized means only one thread can modify a hash table at one point of time. Any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

## [What is fail-fast property?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "fail-fast-property)

At high level - Fail-fast is a property of a system or software with respect to its response to failures. A fail-fast system is designed to immediately report any failure or condition that is likely to lead to failure. Fail-fast systems are usually designed to stop normal operation rather than attempt to continue a possibly-flawed process.   
When a problem occurs, a fail-fast system fails immediately and visibly. Failing fast is a non-intuitive technique: "failing immediately and visibly" sounds like it would make your software more fragile, but it actually makes it more robust. Bugs are easier to find and fix, so fewer go into production.   
In Java, Fail-fast term can be related to context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown.   
It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

## [Why doesn't Collection extend Cloneable and Serializable?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Collection-extend-Cloneable)

From Sun FAQ Page: Many Collection implementations (including all of the ones provided by the JDK) will have a public clone method, but it would be mistake to require it of all Collections.  
For example, what does it mean to clone a Collection that's backed by a terabyte SQL database? Should the method call cause the company to requisition a new disk farm? Similar arguments hold for serializable.   
If the client doesn't know the actual type of a Collection, it's much more flexible and less error prone to have the client decide what type of Collection is desired, create an empty Collection of this type, and use the addAll method to copy the elements of the original collection into the new one.   
  
**Note on Some Important Terms**

* Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.
* Fail-fast is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally”, a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn’t modify the collection "structurally”. However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

[How can we make Hashmap synchronized?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Hashmap-synchronized)

HashMap can be synchronized by *Map m = Collections.synchronizedMap(hashMap);*

[Where will you use Hashtable and where will you use HashMap?](https://www.blogger.com/blogger.g?blogID=637499528247154499#Hashtable-vs-Hashmap-use)  
There Are Multiple Aspects To This Decision:

* The basic difference between a Hashtable and an HashMap is that, Hashtable is synchronized while HashMap is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Hashtable. While if not multiple threads are going to access the same instance then use HashMap. Non synchronized data structure will give better performance than the synchronized one.
* If there is a possibility in future that - there can be a scenario when you may require to retain the order of objects in the Collection with key-value pair then HashMap can be a good choice. As one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable. Also if you have multiple thread accessing you HashMap then Collections.synchronizedMap() method can be leveraged. Overall HashMap gives you more flexibility in terms of possible future changes.

## [Difference between Vector and ArrayList? What is the Vector class?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Vector-vs-ArrayList)

Vector and ArrayList both classes are implemented using dynamically resizable arrays, providing fast random access and fast traversal. ArrayList and Vector class both implement the List interface. Both the classes are member of Java collection framework, therefore from an API perspective, these two classes are very similar. However, there are still some major differences between the two. Below are some key differences 

* Vector is a legacy class which has been retrofitted to implement the List interface since Java 2 platform v1.2
* Vector is synchronized whereas ArrayList is not. Even though Vector class is synchronized, still when you want programs to run in multithreading environment using ArrayList with Collections.synchronizedList() is recommended over Vector.
* ArrayList has no default size while vector has a default size of 10.
* The Enumerations returned by Vector's elements method are not fail-fast. Whereas ArraayList does not have any method returning Enumerations.

## [What is the Difference between Enumeration and Iterator interface?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Difference-between-Enumeration-Iterator)

Enumeration and Iterator are the interface available in java.util package. The functionality of Enumeration interface is duplicated by the Iterator interface. New implementations should consider using Iterator in preference to Enumeration. Iterators differ from enumerations in following ways:

1. Enumeration contains 2 methods namely hasMoreElements() & nextElement() whereas Iterator contains three methods namely hasNext(), next(),remove().
2. Iterator adds an optional remove operation, and has shorter method names. Using remove() we can delete the objects but Enumeration interface does not support this feature.
3. Enumeration interface is used by legacy classes. Vector.elements() & Hashtable.elements() method returns Enumeration. Iterator is returned by all Java Collections Framework classes. java.util.Collection.iterator() method returns an instance of Iterator.

## [Why Java Vector class is considered obsolete or unofficially deprecated? or Why should I always use ArrayList over Vector?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Why-vector-class-deprecated)

You should use ArrayList over Vector because you should default to non-synchronized access. Vector synchronizes each individual method. That's almost never what you want to do. Generally you want to synchronize a whole sequence of operations.   
Synchronizing individual operations is both less safe (if you iterate over a Vector, for instance, you still need to take out a lock to avoid anyone else changing the collection at the same time) but also slower (why take out a lock repeatedly when once will be enough)? Of course, it also has the overhead of locking even when you don't need to. It's a very flawed approach to have synchronized access as default.   
You can always decorate a collection using Collections.synchronizedList - the fact that Vector combines both the "resized array" collection implementation with the "synchronize every operation" bit is another example of poor design; the decoration approach gives cleaner separation of concerns.   
Vector also has a few legacy methods around enumeration and element retrieval which are different than the List interface, and developers (especially those who learned Java before 1.2) can tend to use them if they are in the code. Although Enumerations are faster, they don't check if the collection was modified during iteration, which can cause issues, and given that Vector might be chosen for its syncronization - with the attendant access from multiple threads, this makes it a particularly pernicious problem.   
Usage of these methods also couples a lot of code to Vector, such that it won't be easy to replace it with a different List implementation. Despite all above reasons Sun may never officially deprecate Vector class.

[What is an enumeration?](https://www.blogger.com/blogger.g?blogID=637499528247154499#What-enumeration)

An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It is a construct which collection classes return when you request a collection of all the objects stored in the collection. It allows sequential access to all the elements stored in the collection.

## [What is the difference between Enumeration and Iterator?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "Enumeration-vs-Iterator)

The functionality of Enumeration interface is duplicated by the Iterator interface. Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as using Iterator we can manipulate the objects also like adding and removing the objects. So Enumeration is used when ever we want to make Collection objects as Read-only.

## [Where will you use Vector and where will you use ArrayList?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "13)

The basic difference between a Vector and an ArrayList is that, vector is synchronized while ArrayList is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Vector. While if not multiple threads are going to access the same instance then use ArrayList. Non synchronized data structure will give better performance than the synchronized one.

## [What is the importance of hashCode() and equals() methods? How they are used in Java?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "14)

The java.lang.Object has two methods defined in it. They are

* public boolean equals(Object obj)
* public int hashCode().

These two methods are used heavily when objects are stored in collections. There is a contract between these two methods which should be kept in mind while overriding any of these methods. The Java API documentation describes it in detail. 

### The hashCode Method

The hashCode() method returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable or java.util.HashMap.   
The general contract of hashCode is: Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.   
If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.   
It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.   
As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. The equals(Object obj) method indicates whether some other object is "equal to" this one. 

### The equals Method

The equals method implements an equivalence relation on non-null object references:   
It is reflexive: for any non-null reference value x, x.equals(x) should return true.   
It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.   
It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.   
It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.   
For any non-null reference value x, x.equals(null) should return false. The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true). Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes. 

### A practical Example of hashcode() and equals():

This can be applied to classes that need to be stored in Set collections. Sets use equals() to enforce non-duplicates, and HashSet uses hashCode() as a first-cut test for equality. Technically hashCode() isn't necessary then since equals() will always be used in the end, but providing a meaningful hashCode() will improve performance for very large sets or objects that take a long time to compare using equals().  
  
[What is the difference between Sorting performance of Arrays.sort() vs Collections.sort() ? Which one is faster? Which one to use and when?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "15)

Many developers are concerned about the performance difference between java.util.Array.sort() java.util.Collections.sort() methods. Both methods have same algorithm the only difference is type of input to them. Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting.   
So this should be used when you are trying to sort a list. Arrays.sort is for arrays so the sorting is done directly on the array. So clearly it should be used when you have a array available with you and you want to sort it.  
  
[What is java.util.concurrent BlockingQueue? How it can be used?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "16)

Java has implementation of BlockingQueue available since Java 1.5. Blocking Queue interface extends collection interface, which provides you power of collections inside a queue. Blocking Queue is a type of Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.   
A typical usage example would be based on a producer-consumer scenario. Note that a BlockingQueue can safely be used with multiple producers and multiple consumers. An ArrayBlockingQueue is a implementation of blocking queue with an array used to store the queued objects. The head of the queue is that element that has been on the queue the longest time.   
The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.   
ArrayBlockingQueue requires you to specify the capacity of queue at the object construction time itself. Once created, the capacity cannot be increased. This is a classic "bounded buffer" (fixed size buffer), in which a fixed-sized array holds elements inserted by producers and extracted by consumers. Attempts to put an element to a full queue will result in the put operation blocking; attempts to retrieve an element from an empty queue will be blocked.  
  
[Set and List interface extend Collection, so Why doesn't Map interface extend Collection?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "17)

Though the Map interface is part of collections framework, it does not extend collection interface. This is by design, and the answer to this questions is best described in Sun's FAQ Page: This was by design. We feel that mappings are not collections and collections are not mappings. Thus, it makes little sense for Map to extend the Collection interface (or vice versa).   
If a Map is a Collection, what are the elements? The only reasonable answer is "Key-value pairs", but this provides a very limited (and not particularly useful) Map abstraction. You can't ask what value a given key maps to, nor can you delete the entry for a given key without knowing what value it maps to.   
Collection could be made to extend Map, but this raises the question: what are the keys? There's no really satisfactory answer, and forcing one leads to an unnatural interface. Maps can be viewed as Collections (of keys, values, or pairs), and this fact is reflected in the three "Collection view operations" on Maps (keySet, entrySet, and values).  
While it is, in principle, possible to view a List as a Map mapping indices to elements, this has the nasty property that deleting an element from the List changes the Key associated with every element before the deleted element. That's why we don't have a map view operation on Lists.  
  
[Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "18)

List interface has three main implementation classes 

* Vector
* ArrayList
* LinkedList

ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element.   
The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

## [What is the difference between ArrayList and LinkedList? (ArrayList vs LinkedList.)](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "19)

java.util.ArrayList and java.util.LinkedList are two Collections classes used for storing lists of object references **Here are some key differences:** 

* ArrayList uses primitive object array for storing objects whereas LinkedList is made up of a chain of nodes. Each node stores an element and the pointer to the next node. A singly linked list only has pointers to next. A doubly linked list has a pointer to the next and the previous element. This makes walking the list backward easier.
* ArrayList implements the RandomAccess interface, and LinkedList does not. The commonly used ArrayList implementation uses primitive Object array for internal storage. Therefore an ArrayList is much faster than a LinkedList for random access, that is, when accessing arbitrary list elements using the get method. Note that the get method is implemented for LinkedLists, but it requires a sequential scan from the front or back of the list. This scan is very slow. For a LinkedList, there's no fast way to access the Nth element of the list.
* Adding and deleting at the start and middle of the ArrayList is slow, because all the later elements have to be copied forward or backward. (Using System.arrayCopy()) Whereas Linked lists are faster for inserts and deletes anywhere in the list, since all you do is update a few next and previous pointers of a node.
* Each element of a linked list (especially a doubly linked list) uses a bit more memory than its equivalent in array list, due to the need for next and previous pointers.
* ArrayList may also have a performance issue when the internal array fills up. The arrayList has to create a new array and copy all the elements there. The ArrayList has a growth algorithm of (n\*3)/2+1, meaning that each time the buffer is too small it will create a new one of size (n\*3)/2+1 where n is the number of elements of the current buffer. Hence if we can guess the number of elements that we are going to have, then it makes sense to create a arraylist with that capacity during object creation (using construtor new ArrayList(capacity)). Whereas LinkedLists should not have such capacity issues.

[What is performance of various Java collection implementations/algorithms? What is Big 'O' notation for each of them ?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "BigONotationJavaCollections)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Collection interface | Collection class | performance | |  |  |  |
|  |  | Addition | remove |  |  |  |
| List | ArrayList | Normal O(1) , Resizing  and copying O(n) | O(n) |  |  |  |
| LinkedList | O(n) | O(n) |  |  |  |
| Set | HashSet | O(1) | O(1) |  |  |  |
| TreeSet | log(n) | log(n) |  |  |  |
| LinkedHashSet | -- | -- |  |  |  |
| Map | TreeMap | log(n) | log(n) |  |  |  |
| HashMap | is O(1) | is O(1) |  |  |  |
| LinkedHashMap | -- | -- |  |  |  |
| HashTable | -- | -- |  |  |  |

## [What is synchronization in respect to multi-threading in Java?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-synchronization-in-respect-to-multi-threading-in-java-" \o "What is synchronization in respect to multi-threading in Java?)

With respect to multi-threading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one Java thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to erroneous behavior or program.

[Explain different way of using thread?](https://www.blogger.com/blogger.g?blogID=637499528247154499#explain-different-way-of-using-thread-)

A Java thread could be implemented by using Runnable interface or by extending the Thread class. The Runnable is more advantageous when you are going for multiple inheritance.

## [What is the difference between Thread.start() & Thread.run() method?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-the-difference-between-thread-start-----thread-run---method-" \o "What is the difference between Thread.start() & Thread.run() method?)

Thread.start() method (native method) of Thread class actually does the job of running the Thread.run() method in a thread. If we directly call Thread.run() method it will execute in same thread, so does not solve the purpose of creating a new thread.

## [Why do we need run() & start() method both. Can we achieve it with only run method?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "why-do-we-need-run-----start---method-both--can-we-achieve-it-with-only-run-method-" \o "Why do we need run() & start() method both. Can we achieve it with only run method?)

We need run() & start() method both because JVM needs to create a separate thread which can not be differentiated from a normal method call. So this job is done by start method native implementation which has to be explicitly called. Another advantage of having these two methods is we can have any object run as a thread if it implements Runnable interface. This is to avoid Java’s multiple inheritance problems which will make it difficult to inherit another class with Thread.

## [What is ThreadLocal class? How can it be used?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-threadlocal-class--how-can-it-be-used-" \o "What is ThreadLocal class? How can it be used?)

Below are some key points about ThreadLocal variables

* A thread-local variable effectively provides a separate copy of its value for each thread that uses it.
* ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread
* In the case when multiple threads access a ThreadLocal instance, a separate copy of the Threadlocal variable is maintained for each thread.
* Common use is seen in DAO pattern where the DAO class can be a singleton but the Database connection can be maintained separately for each thread. (Per Thread Singleton)

## [When InvalidMonitorStateException is thrown? Why?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "when-invalidmonitorstateexception-is-thrown--why-" \o "When InvalidMonitorStateException is thrown? Why?)

This exception is thrown when you try to call wait()/notify()/notifyAll() any of these methods for an Object from a point in your program where u are NOT having a lock on that object.(i.e. u r not executing any synchronized block/method of that object and still trying to call wait()/notify()/notifyAll()) wait(), notify() and notifyAll() all throw IllegalMonitorStateException. since This exception is a subclass of RuntimeException so we r not bound to catch it (although u may if u want to). and being a RuntimeException this exception is not mentioned in the signature of wait(), notify(), notifyAll() methods.

## [What is the difference between sleep(), suspend() and wait() ?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-the-difference-between-sleep----suspend---and-wait----" \o "What is the difference between sleep(), suspend() and wait() ?)

Thread.sleep() takes the current thread to a "Not Runnable" state for specified amount of time. The thread holds the monitors it has acquired. For example, if a thread is running a synchronized block or method and sleep method is called then no other thread will be able to enter this block or method. The sleeping thread can wake up when some other thread calls t.interrupt on it. Note that sleep is a static method, that means it always affects the current thread (the one executing sleep method).  
A common mistake is trying to call t2.sleep() where t2 is a different thread; even then, it is the current thread that will sleep, not the t2 thread. thread.suspend() is deprecated method. Its possible to send other threads into suspended state by making a suspend method call. In suspended state, a thread keeps all its monitors and can not be interrupted. This may cause deadlocks, therefore, it has been deprecated. object.wait() call also takes the current thread into a "Not Runnable" state, just like sleep(), but with a slight change. Wait method is invoked on a lock object, not thread.  
Here is the sequence of operations you can think

* A thread T1 is already running a synchronized block with a lock on object - let's say "lockObject"
* Another thread T2 comes to execute the synchronized block and find that it's already acquired.
* Now T2 calls lockObject.wait() method for waiting on the lock to be release by the T1 thread.
* T1 thread finishes all its synchronized block work.
* The T1 thread calls lockObject.notifyAll() to notify all waiting threads that it done using the lock.
* Since the T2 thread is first in the queue of waiting it acquires the lock and starts processing.

[What happens when I make a static method as synchronized?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-happens-when-i-make-a-static-method-as-synchronized-" \o "What happens when I make a static method as synchronized?)

Synchronized static methods have a lock on the class "Class", so when a thread enters a synchronized static method, the class itself gets locked by the thread monitor and no other thread can enter any static synchronized methods on that class. This is unlike instance methods, as multiple threads can access "same synchronized instance methods" at the same time for different instances.

## [Can a thread call a non-synchronized instance method of an Object when a synchronized method is being executed?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "can-a-thread-call-a-non-synchronized-instance-method-of-an-object-when-a-synchronized-method-is-being-executed--" \o "Can a thread call a non-synchronized instance method of an Object when a synchronized method is being executed ?)

Yes, a Non-synchronized method can always be called without any problem. In fact, Java does not do any check for a non-synchronized method. The Lock object check is performed only for synchronized methods/blocks. In case the method is not declared synchronized Jave will call even if you are playing with shared data. So you have to be careful while doing such thing. The decision of declaring a method as synchronized has to be based on critical section access.

[What is a deadlock?](https://www.blogger.com/blogger.g?blogID=637499528247154499#what-is-a-deadlock-)

Deadlock is a situation where two or more threads are blocked forever, waiting for each other. This may occur when two threads, each having a lock on one resource, attempt to acquire a lock on the other's resource. Each thread would wait indefinitely for the other to release the lock unless one of the user processes is terminated. In terms of Java API, thread deadlock can occur in following conditions:  
When two threads call Thread.join() on each other.

* When two threads use nested synchronized blocks to lock two objects and the blocks lock the same objects in different order.

## [What is Starvation? and What is a Livelock?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-starvation--and-what-is-a-livelock-" \o "What is Starvation? and What is a Livelock?)

Starvation and livelock are much less common a problem than deadlock but are still problems that every designer of concurrent software is likely to encounter.  
**LiveLock**

Livelock occurs when all threads are blocked or are otherwise unable to proceed due to unavailability of required resources, and the non-existence of any unblocked thread to make those resources available. In terms of Java API, thread livelock can occur in following conditions:

* When all the threads in a program execute Object.wait(0) on an object with zero parameters. The program is live-locked and cannot proceed until one or more threads call Object.notify() or Object.notifyAll() on the relevant objects. Because all the threads are blocked, neither call can be made.
* When all the threads in a program are stuck in infinite loops.

### Starvation

Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods of "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequently synchronized access to the same object will often be blocked. Starvation occurs when one thread cannot access the CPU because one or more other threads are monopolizing the CPU. In Java, thread starvation can be caused by setting thread priorities inappropriately. A lower-priority thread can be starved by higher-priority threads if the higher-priority threads do not yield control of the CPU from time to time.  
  
[How to find a deadlock has occurred in Java? How to detect a Deadlock in Java?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "how-to-find-a-deadlock-has-occurred-in-java-" \o "How to find a deadlock has occurred in Java?)

Earlier versions of Java had no mechanism to handle/detect deadlock. Since JDK 1.5 there are some powerful methods added in the java.lang.management package to diagnose and detect deadlocks. The java.lang.management.ThreadMXBean interface is management interface for the thread system of the Java virtual machine. It has two methods which can leverage to detect deadlock in a Java application.

* findMonitorDeadlockedThreads() - This method can be used to detect cycles of threads that are in deadlock waiting to acquire object monitors. It returns an array of thread IDs that are deadlocked waiting on the monitor.
* findDeadlockedThreads() - It returns an array of thread IDs that are deadlocked waiting on the monitor or ownable synchronizers.

## 

## [What is an immutable object? How does it help in writing a concurrent application?](https://www.blogger.com/blogger.g?blogID=637499528247154499#what-is-immutable-object--how-does-it-help-on-writing-concurrent-application-)

An object is considered immutable if its state cannot change after it is constructed. Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code. Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state. Examples of immutable objects from the JDK include String and Integer. Immutable objects greatly simplify your multi-threaded program, since they are

* Simple to construct, test, and use.
* Automatically thread-safe and have no synchronization issues.

To create an object immutable You need to make the class final and all its member final so that once objects get created no one can modify its state. You can achieve the same functionality by making member as non-final but private and not modifying them except in constructor.

## [What is a thread leak? What does it mean in Java?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-thread-leak-" \o "What is thread leak?)

Thread leak is when an application does not release references to a thread object properly. Due to this, some Threads do not get garbage collected and the number of unused threads grows with time. Thread leak can often cause serious issues on a Java application since over a period of time too many threads will be created but not released and may cause applications to respond slow or hang.

## [What is thread pool? Why should we use thread pools?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "what-is-thread-pool--why-should-we-use-thread-pools-" \o "What is thread pool? Why should we use thread pools?)

A thread pool is a collection of threads on which task can be scheduled. Instead of creating a new thread for each task, you can have one of the threads from the thread pool pulled out of the pool and assigned to the task. When the thread is finished with the task, it adds itself back to the pool and waits for another assignment. One common type of thread pool is the fixed thread pool. This type of pool always has a specified number of threads running; if a thread is somehow terminated while it is still in use, it is automatically replaced with a new thread. Below are key reasons to use a Thread Pool

* Using thread pools minimizes the JVM overhead due to thread creation. Thread objects use a significant amount of memory, and in a large-scale application, allocating and de-allocating many thread objects creates a significant memory management overhead.
* You have control over the maximum number of tasks that are being processed in parallel (= number of threads in the pool).

Most of the executor implementations in java.util.concurrent use thread pools, which consist of worker threads. This kind of thread exists separately from the Runnable and Callable tasks it executes and is often used to execute multiple tasks.

## [Can we synchronize the run method? If yes then what will be the behavior?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "can-we-synchronize-the-run-method--if-yes-then-what-will-be-the-behavior-" \o "Can we synchronize the run method? If yes then what will be the behavior?)

Yes, the run method of a runnable class can be synchronized. If you make the run method synchronized then the lock on the runnable object will be occupied before executing the run method. In case we start multiple threads using the same runnable object in the constructor of the Thread then it would work. But until the 1st thread ends the 2nd thread cannot start and until the 2nd thread ends the next cannot start as all the threads depend on the lock on the same object.

[Can we synchronize the constructor of a Java Class?](https://www.blogger.com/blogger.g?blogID=637499528247154499" \l "can-we-synchronize-the-constructor-of-a-java-class-" \o "Can we synchronize the constructor of a Java Class?)

As per Java Language Specification, constructors cannot be synchronized because other threads cannot see the object being created before the thread creating it has finished it. There is no practical need of a Java Objects constructor to be synchronized since it would lock the object being constructed, which is normally not available to other threads until all constructors of the object finish.

## [Define Serialization? What do you mean by Serialization in Java?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "define-serialization--what-do-you-mean-by-serialization-in-java-" \o "Define Serialization? What do you mean by Serialization in Java?)

Serialization is a mechanism by which you can save or transfer the state of an object by converting it to a byte stream. This can be done in java by implementing Serialiazable interface. Serializable is defined as a marker interface which needs to be implemented for transferring an object over a network or persistence of its state to a file. Since its a marker interface, it does not contain any methods. Implementation of this interface enables the conversion of object into byte stream and thus can be transferred. The object conversion is done by the JVM using its default serialization mechanism.

## [Why is Serialization required? What is the need to Serialize?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "why-is-serialization-required--what-is-the-need-to-serialize-" \o "Why is Serialization required? What is the need to Serialize?)

Serialization is required for a variety of reasons. It is required to send across the state of an object over a network by means of a socket. One can also store an object’s state in a file. Additionally, manipulation of the state of an object as streams of bytes is required. The core of Java Serialization is the Serializable interface. When Serializable interface is implemented by your class it provides an indication to the compiler that java Serialization mechanism needs to be used to serialize the object.

## [What is the Difference between Externalizable and Serializable Interfaces?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-is-the-difference-between-externalizable-and-serializable-interfaces-" \o "What is the Difference between Externalizable and Serializable Interfaces?)

This is one of top serialization questions that is asked in many big companies to test your in-depth understanding of serialization. Serializable is a marker interface therefore you are not forced to implement any methods, however Externalizable contains two methods readExternal() and writeExternal() which must be implemented.   
  
Serializable interface provides a inbuilt serialization mechanism to you which can be in-efficient at times. However Externilizable interface is designed to give you greater control over the serialization mechanism. The two methods provide you immense opportunity to enhance the performance of specific object serialization based on application needs.   
  
Serializable interface provides a default serialization mechanism, on the other hand, Externalizable interface instead of relying on default Java Serialization provides flexibility to control this mechanism.   
One can drastically improve the application performance by implementing the Externalizable interface correctly. However there is also a chance that you may not write the best implementation, so if you are not really sure about the best way to serialize, I would suggest your stick to the default implementation using Serializable interface.

[When will you use Serializable or Externalizable interface? and why?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html#when-will-you-use-serializable-or-externalizable-interface--and-why-)

Most of the times when you want to do a selective attribute serialization you can use Serializable interface with transient modifier for variables not to be serialized. However, use of Externalizable interface can be really effective in cases when you have to serialize only some dynamically selected attributes of a large object.   
Lets take an example, Some times when you have a big Java object with hundreds of attributes and you want to serialize only a dozen dynamically selected attributes to keep the state of the object you should use Externalizable interface writeExternal method to selectively serialize the chosen attributes.  
In case you have small objects and you know that most or all attributes are required to be serialized then you should be fine with using Serializable interface and use of transient variable as appropriate.

## [What are the ways to speed up Object Serialization? How to improve Serialization performance?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-the-ways-to-speed-up-object-serialization--how-to-improve-serialization-performance-" \o "What are the ways to speed up Object Serialization? How to improve Serialization performance?)

The default Java Serialization mechanism is really useful, however it can have a really bad performance based on your application and business requirements. The serialization process performance heavily depends on the number and size of attributes you are going to serialize for an object. Below are some tips you can use for speeding up the marshaling and un-marshaling of objects during Java serialization process.

* Mark the unwanted or non Serializable attributes as transient. This is a straight forward benefit since your attributes for serialization are clearly marked and can be easily achieved using Serialzable interface itself.
* Save only the state of the object, not the derived attributes. Some times we keep the derived attributes as part of the object however serializing them can be costly. Therefore consider calcualting them during de-serialization process.
* Serialize attributes only with NON-default values. For examples, serializing a int variable with value zero is just going to take extra space however, choosing not to serialize it would save you a lot of performance. This approach can avoid some types of attributes taking unwanted space. This will require use of Externalizable interface since attribute serialization is determined at runtime based on the value of each attribute.
* Use Externalizable interface and implement the readExternal and writeExternal methods to dynamically identify the attributes to be serialized. Some times there can be a custom logic used for serialization of various attributes.

## [What is a Serial Version UID (serialVersionUID) and why should I use it? How to generate one?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-is-a-serialversionuid-and-why-should-i-use-it--how-to-generate-one-" \o "What is a serialVersionUID and why should I use it? How to generate one?)

The serialVersionUID represents your class version, and you should change it if the current version of your class is not backwards compatible with its earlier versions. This is extract from Java API Documentation

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

Most of the times, we probably do not use serialization directly. In such cases, I would suggest to generate a default serializable uid by clicking the quick fix option in eclipse.

## [What would happen if the SerialVersionUID of an object is not defined?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-would-happen-if-the-serialversionuid-of-an-object-is-not-defined-" \o "What would happen if the SerialVersionUID of an object is not defined?)

If you don't define serialVersionUID in your serilizable class, Java compiler will make one by creating a hash code using most of your class attributes and features. When an object gets serialized, this hash code is stamped on the object which is known as the SerialVersionUID of that object. This ID is required for the version control of an object. SerialVersionUID can be specified in the class file also. In case, this ID is not specified by you, then Java compiler will regenerate a SerialVersionUID based on updated class and it will not be possible for the already serialized class to recover when a class field is added or modified. Its recommended that you always declare a serialVersionUID in your Serializable classes.

## [Does setting the serialVersionUID class field improve Java serialization performance?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "does-setting-the-serialversionuid-class-field-improve-java-serialization-performance-" \o "Does setting the serialVersionUID class field improve Java serialization performance?)

Declaring an explicit serialVersionUID field in your classes saves some CPU time only the first time the JVM process serializes a given Class. However the gain is not significant, In case when you have not declared the serialVersionUID its value is computed by JVM once and subsequently kept in a soft cache for future use.

## [What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-the-alternatives-to-serialization--if-serialization-is-not-used--is-it-possible-to-persist-or-transfer-an-object-using-any-other-approach-" \o "What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?)

In case, Serialization is not used, Java objects can be serialized by many ways, some of the popular methods are listed below:

* Saving object state to database, this is most common technique used by most applications. You can use ORM tools (e.g. hibernate) to save the objects in a database and read them from the database.
* Xml based data transfer is another popular mechanism, and a lot of XML based web services use this mechanism to transfer data over network. Also a lot of tools save XML files to persist data/configurations.
* JSON Data Transfer - is recently popular data transfer format. A lot of web services are being developed in JSON due to its small footprint and inherent integration with web browser due to JavaScript format.

## [What are transient variables? What role do they play in Serialization process?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-transient-variables--what-role-do-they-play-in-serialization-process-" \o "What are transient variables? What role do they play in Serialization process?)

The transient keyword in Java is used to indicate that a field should not be serialized. Once the process of de-serialization is carried out, the transient variables do not undergo a change and retain their default value. Marking unwanted fields as transient can help you boost the serialization performance. Below is a simple example where you can see the use of transient keyword.

## 

## [Why does serialization NOT save the value of static class attributes? Why static variables are not serialized?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html#why-does-serialization-not-save-the-value-of-static-class-attributes--why-static-variables-are-not-serialized-)

The Java variables declared as static are not considered part of the state of an object since they are shared by all instances of that class. Saving static variables with each serialized object would have following problems

* It will make redundant copy of same variable in multiple objects which makes it in-efficient.
* The static variable can be modified by any object and a serialized copy would be stale or not in sync with current value.

## [How to Serialize a collection in java? How to serialize a ArrayList, Hashmap or Hashset object in Java?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "how-to-serialize-a-collection-in-java--how-to-serialize-a-arraylist--hashmap-or-hashset-object-in-java-" \o "How to Serialize a collection in java? How to serialize a ArrayList, Hashmap or Hashset object in Java?)

All standard implementations of collections List, Set and Map interface already implement java.io.Serializable. All the commonly used collection classes like java.util.ArrayList, java.util.Vector, java.util.Hashmap, java.util.Hashtable, java.util.HashSet, java.util.TreeSet do implement Serializable. This means you do not really need to write anything specific to serialize collection objects. However you should keep following things in mind before you serialize a collection object - Make sure all the objects added in collection are Serializable. - Serializing the collection can be costly therefore make sure you serialize only required data isntead of serializing the whole collection. - In case you are using a custom implementation of Collection interface then you may need to implement serialization for it.

## [Is it possible to customize the serialization process? How can we customize the Serialization process?](https://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "is-it-possible-to-customize-the-serialization-process--how-can-we-customize-the-serialization-process-" \o "Is it possible to customize the serialization process? How can we customize the Serialization process?)

Yes, the serialization process can be customized. When an object is serialized, objectOutputStream.writeObject (to save this object) is invoked and when an object is read, ObjectInputStream.readObject () is invoked. What most people do not know is that Java Virtual Machine provides you with an option to define these methods as per your needs. Once this is done, these two methods will be invoked by the JVM instead of the application of the default serialization process. Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

private void writeObject(java.io.ObjectOutputStream out)

throws IOException

private void readObject(java.io.ObjectInputStream in)

throws IOException, ClassNotFoundException;

private void readObjectNoData()

throws ObjectStreamException;

## [What is the difference between NoClassDefFoundError and ClassNotFoundException? When NoClassDefFoundError and ClassNotFoundException are thrown?](https://www.fromdev.com/2012/09/Java-Path-Classpath-Questions-Answers.html" \l "what-is-the-difference-between-noclassdeffounderror-and-classnotfoundexception-" \o "What is the difference between NoClassDefFoundError and ClassNotFoundException?)

NoClassDefFoundError and ClassNotFoundException are very closely related and often confused with each other by many developers. Below is the description of each from the Java API Specifications

### ClassNotFoundException

Thrown when an application tries to load in a class through its string name using:

* The forName method in class Class.
* The findSystemClass method in class ClassLoader.
* The loadClass method in class ClassLoader.

but the definition of the class with the specified name could not be found due to following reasons  
The specified name class does not exist.

* The specified name class is not in the classpath
* The specified name class is not visible to the classloader.

NoClassDefFoundError

Thrown if the Java Virtual Machine or a ClassLoader instance tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the new expression) and no definition of the class could be found.  
The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found.

### Key Differences

* The NoClassDefFoundError is thrown when the source was successfully compiled, but during runtime, the required class files were not found. This may be a case when some dependency jar files were not included or not in classpath.  
  A ClassNotFoundException is thrown when the reported class is not found by the ClassLoader or not visible to the Classloader.
* Another important distinction between these two is, NoClassDefFoundError is a sub class of java.lang.Error and the ClassNotFoundException is a subclass of java.lang.Exception.
* NoClassDefFoundError is a critical error for JVM since its having problems finding a class it expected to find.   
  On the other hand, the ClassNotFoundException is an Exception. Use of reflection api can be error-prone and there is no compile-time check to validate reflection call is loading right classes, so there can be situations when some classes may not be found.

### Some scenario when ClassNotFoundException may occur Scenario 1 (Use of reflection) - You will see ClassNotFoundException when you are using reflection to load classes at runtime, and the class you are trying to load does not exist or not in classpath. Scenario 2 (Multiple Classloaders being used) - You will see ClassNotFoundException when a class is being loaded from another class which was already loaded in a parent classloader and the class from the child classloader is not visible.

## **What Is Scalability?**

Scalability is the ability of a system, network, or process to handle a growing amount of load by adding more resources. The adding of resource can be done in two ways

* **Scaling Up 🡺** This involves adding more resources to the existing nodes. For example, adding more RAM, Storage or processing power.
* **Scaling Out🡺**This involves adding more nodes to support more users.

Any of the approaches can be used for scaling up/out a application, however the cost of adding resources (per user) may change as the volume increases. If we add resources to the system It should increase the ability of application to take more load in a proportional manner of added resources.

## What Is A Cluster?

A cluster is group of computer machines that can individually run a software. Clusters are typically utilized to achieve high availability for a server software.   
Clustering is used in many types of servers for high availability.

* **App Server Cluster 🡺**An app server cluster is group of machines that can run a application server that can be reliably utilized with a minimum of down-time.
* **Database Server Cluster 🡺**An database server cluster is group of machines that can run a database server that can be reliably utilized with a minimum of down-time.

**Why Do You Need Clustering?**

Clustering is needed for achieving high availability for a server software. The main purpose of clustering is to achieve 100% availability or a zero down time in service.   
A typical server software can be running on one computer machine and it can serve as long as there is no hardware failure or some other failure.  
By creating a cluster of more than one machine, we can reduce the chances of our service going un-available in case one of the machine fails.  
Doing clustering does not always guarantee that service will be 100% available since there can still be a chance that all the machine in a cluster fail at the same time. However it in not very likely in case you have many machines and they are located at different location or supported by their own resources.

## **What Is Middle Tier Clustering?**

Middle tier clustering is just a cluster that is used for service the middle tier in a application. This is popular since many clients may be using middle tier and a lot of heavy load may also be served by middle tier that requires it be to highly available.  
Failure of middle tier can cause multiple clients and systems to fail, therefore its one of the approaches to do clustering at the middle tier of a application.  
In java world, it is really common to have EJB server clusters that are used by many clients. In general any application that has a business logic that can be shared across multiple client can use a middle tier cluster for high availability.  
  
**What Is Load Balancing?**

Load balancing is simple technique for distributing workloads across multiple machines or clusters.   
The most common and simple load balancing algorithm is Round Robin. In this type of load balancing the request is divided in circular order ensuring all machines get equal number of requests and no single machine is overloaded or underloaded.   
  
**The Purpose of load balancing is to**

* Optimize resource usage (Avoid overload and under-load of any machines.)
* Achieve Maximum Throughput
* Minimize response time

**Most common load balancing techniques in web based applications are**

1. Round robin
2. Session affinity or sticky session
3. IP Address affinity

What Is Sticky Session (session Affinity) Load Balancing? What Do You Mean By 'session Affinity'?

Sticky session or a session affinity technique another popular load balancing technique that requires a user session to be always served by a allocated machine.   
  
Why Sticky Session?

In a load balanced server application where user information is stored in session it will be required to keep the session data available to all machines. This can be avoided by always serving a particular user session request from one machine.  
  
How It Is Done?

The machine is associated with a session as soon as the session is created. All the requests in a particular session are always redirected to the associated machine. This ensures the user data is only at one machine and load is also shared.

In Java world, this is typically done by using jsessionid cookie. The cookie is sent to the client for the first request and every subsequent request by client must be containing that same cookie to identify the session.  
What Are The Issues With Sticky Session?

There are few issues that you may face with this approach

* The client browser may not support cookies, and your load balancer will not be able to identify if a request belongs to a session. This may cause strange behavior for the users who use no cookie based browsers.
* In case one of the machine fails or goes down, the user information (served by that machine) will be lost and there will be no way to recover user session.