Q: use of static variable in singleton why not public

**Singleton pattern requires to have a single object throughout the system**, which can only be done if the Variable is global and static. Now making it static restricts the ability to create multiple instances of the class. Because if it wasn’t static you’d be creating a new instance of the class every time you invoked the singleton method on the singleton class. *Static ensures there is only one instance of the variable across all instances of the class. If it wasn’t static, the singleton would behave like a factory and keep creating new instances*.

In Java, the Singleton pattern will ensure that there is only one instance of a classis created in the Java Virtual Machine. It is used to provide global point of access to the object. In terms of practical use Singleton patterns are used in logging, caches, thread pools, configuration settings, device driver objects.

* Static member: This contains the instance of the singleton class.
* Private constructor: This will prevent anybody else to instantiate the Singleton class.
* Static public method: This provides the global point of access to the Singleton object and returns the instance to the client calling class.

<https://javarevisited.blogspot.com/2017/02/10-points-about-volatile-modifier-and-field-in-java.html>

The **volatile** keyword helps as concurrency control tool in a multithreaded environment and provides the latest update in a most accurate manner.

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.

Use **volatile** modifier if you just need to synchronize access to shared variable whose value is set by one thread and queried by other. It provides a low-cost **alternative** to synchronized keyword or lock interface introduced in **Java** 5 without atomicity and mutual exclusion.

**What is volatile keyword in Java**

When we use volatile keyword with a variable, all the threads read it’s value directly from the memory and don’t cache it. This makes sure that the value read is the same as in the memory.

**Static Variable:** If two Threads (suppose Thread 1 and Thread 2) are accessing the same object and updating a variable which is declared as static then it means Thread 1 and Thread 2 can make their own local copy of the same object(including static variables) in their respective cache, so updating by Thread 1 to the static variable in its local cache wont reflect in the static variable for Thread 2 cache . Static variables are used in the Object Context where updating by one object would reflect in all the other objects of the same class but not in the Thread context where updating of one thread to the static variable will reflect the changes immediately to all the threads (in their local cache).

**Volatile variable:** If two Threads (suppose Thread 1 and Thread 2) are accessing the same object and updating a variable which is declared as volatile then it means Thread 1 and Thread 2 can make their own local cache of the Object except the variable which is declared as a volatile. So the volatile variable will have only one main copy which will be updated by different threads and updating by one thread to the volatile variable will immediately reflect to the other Thread. So the volatile variable is used in the Thread context.

Q: Can we use variable which work as volatile?

A boolean value that may be updated atomically

Q: Difference in arraylist and treeset when to use both? **Arraylist is faster**

The TreeSet is better if you add or remove items often as sorting a list again and again would be slow. tree structures need more memory

Q: Difference in synchronize & Reentrant lock.

Synchronization can’t use with constructor. Compile time error

No two threads can enter the same synchronized block **with the same lock**

when you call start() method it creates new Thread and executes code declared in the run() while directly calling run() method doesn’t create any new thread and execute code on the same calling thread.  
  
Q: How will you awake a blocked thread in Java?  
Ans: by throwing Interrupted Exception

**1) Difference between green thread and native thread in Java?**

**2) What is context switching in multi-threading?**

**3) Difference between deadlock and live lock, deadlock and starvation?**

**4) What thread-scheduling algorithm is used in Java?**

**5) How do you handle an unhandled exception in the thread?**

**6) Why Executor framework is better than creating and managing thread by the application?**

**7) Difference between Executor and Executors in Java?**

**8) How to find which thread is taking maximum CPU in windows and Linux server?**

**What is difference between user Thread and daemon Thread?**

When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.

New>runnable>running> Waiting, Blocked and Dead

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

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

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

When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource. Check this post to learn more about [thread wait, notify and notifyAll](https://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example).

**Why Thread sleep() and yield() methods are static?**

Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state.

**How can we achieve thread safety in Java?**

thread safety in java – synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and.

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

Thread class **setDaemon(true)** can be used to create daemon thread in java. We need to call this method before calling start() method else it will throw IllegalThreadStateException.

**What is ThreadLocal?**

Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it’s variables, so if the variable is not thread safe, we can use synchronization but if we want to avoid synchronization, we can use ThreadLocal variables.  
Every thread has it’s own ThreadLocal variable and they can use it’s get() and set() methods to get the default value or change it’s value local to Thread. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread.

Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation

**What is Thread Pool? How can we create Thread Pool in Java?**

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed.

A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue. java.util.concurrent.Executors provide implementation of java.util.concurrent.Executor interface to create the thread pool in java.

**What is atomic operation? What are atomic classes in Java Concurrency API?**

Atomic operations are performed in a single unit of task without interference from other operations. Atomic operations are necessity in multi-threaded environment to avoid data inconsistency.

int++ is not an atomic operation. So by the time one threads read it’s value and increment it by one, other thread has read the older value leading to wrong result.

To solve this issue, we will have to make sure that increment operation on count is atomic, we can do that using Synchronization but Java 5 java.util.concurrent.atomic provides wrapper classes for int and long that can be used to achieve this atomically without usage of Synchronization.

**What is Executors Class?**

Executors class provide utility methods for Executor, ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes.

Executors class can be used to easily create Thread Pool in java, also this is the only class supporting execution of Callable implementations.

**Reentrant Lock** is mutual exclusive lock, similar to implicit locking provided by **synchronized** keyword in **Java**, with extended feature like fairness, which can be used to provide lock to longest waiting thread. Lock is acquired by lock() method and held by Thread until a call to unlock() method

**Factory method**, **builder and prototype pattern** which uses Singleton pattern during the implementation.

Q: **Which classes in JDK uses singleton pattern?**  
**Answer**: java.lang.Runtime : In every Java application there is only one Runtime instance that allows the application to interface with the environment it is running. The **getRuntime** is equivalent to the **getInstance** () method of the singleton class.

**Why can’t we use a static class instead of singleton?**  
Answer:

* One of the key advantages of singleton over static class is that it can implement interfaces and extend classes while the static class cannot (it can extend classes, but it does not inherit their instance members). If we consider a static class, it can only be a nested static class as **top level class cannot be a static class**. Static means that it belongs to a class it is in and not to any instance. So, it cannot be a top-level class.
* Another difference is that static class will have all its member as static only unlike Singleton.
* Another advantage of Singleton is that it can be lazily loaded whereas static will be initialized whenever it is first loaded.
* Singleton object stores in Heap but, static object stores in stack.
* We can clone the object of Singleton but, we can not clone the static class object.
* Singleton can use the Object Oriented feature of polymorphism but static class cannot.

**FIND** is for searching files and directories at system level.

**GREP** is for searching a pattern inside a file.

find / -name "passwd"

ps -ef | grep “java”

Note: that **constructors** cannot be **synchronized** — using the **synchronized** keyword with a **constructor** is a syntax error. **Synchronizing constructors** doesn't make sense, because only the thread that creates an object should have access to it while it is being constructed.

1. **How do you analyze out of memory issues in production?**
2. **What is difference between stack and heap dump.**
3. Both are ways that [Java allocates memory](http://net-informations.com/faq/net/stack-heap.htm) and both are stored in the RAM. However, to make things easier to remember, heap is used for dynamic memory allocation, while stack is for static allocations.
4. On the stack, memory allocation happens when the program is compiled. Meanwhile, on the heap, it begins when the program is run.
5. kill -3 JAVA\_PID command for heap dump
6. -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dir/subdir

**Memory Heap Dumps:** Heap dumps contains:

* Snapshot of JVM Heap at that moment of time.
* Shows live objects in heap along with references between objects.
* Analysing memory issues in an application
  + Used to determine memory usage patterns
  + Used in identifying memory leak suspects
  + does not contain allocation information, i.e. what created the objects

**Encapsulation, Abstraction, Inheritance**

**Encapsulation vs Data Abstraction**

* [Encapsulation](http://contribute.geeksforgeeks.org/encapsulation-in-java/) is data hiding (information hiding) while Abstraction is detail hiding (implementation hiding).
* While encapsulation groups together data and methods that act upon the data, data abstraction deals with exposing the interface to the user and hiding the details of implementation.

**DRY (don't repeat yourself)** means don't write duplicate code, instead use [Abstraction](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html) to abstract common things in one place. If you have a block of code in more than two places consider making it a separate method, or if you use a hard-coded value more than one time make them [public final constant](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html).

**Q If a class is serializable but its superclass in not, what will be the state of the instance variables inherited from super class after deserialization?**

Ans) The values of the instance variables inherited from superclass will be reset to the values they were given during the original construction of the object as the non-serializable super-class constructor will run.

# Q: [Is a deserialised object the same instance as the original](https://stackoverflow.com/questions/29051809/is-a-deserialised-object-the-same-instance-as-the-original)

# Ans:

1. How stack dump and heap dump is structured?
2. In which folder heap dump logs are stored in server?
3. What is the garbage collection strategy used in your project?
4. There are deadlock errors in production logs, server is hung, how do you debug and resolve the issue?
5. Why spring AOP is heavy weight?
6. What is alternative of spring AOP?
7. Have you written from scratch any rest client?
8. Have you configured any projects in Jenkins on your own?

Wiz lab and login as [publicpierre@gmail.com](mailto:publicpierre@gmail.com) password ilovejava2017

1. How to prevent Singleton Pattern from Reflection, Serialization and Cloning?
2. How to use singleton object in serialization and if after deserialization, will singleton object be as it is?
3. What is use of JNDI and its advantage over jdbc?
4. **When singleton is not singleton?**

A **singleton** wouldn't work as a **singleton** if a given class is loaded by multiple class-loaders. Since a single class can exist (or can be loaded) in multiple class loaders, it's quite possible to have "multiple" instances of a "supposedly" **singleton** class for a given JVM instance.

distributed systems such as those using EJBs, Jini, and RMI

When two class loaders load a class, you actually have two copies of the class, and each one can have its own singleton instance.

1. How to write many to many annotation in Hibernate
2. Why string is immutable in java?

The hashcode of **string** is frequently used in **Java**. For example, in a HashMap. Being **immutable** guarantees that hashcode will always the same, so that it can be cashed without worrying the changes. That means, there is no need to calculate hashcode every time it is used.

1. **What is factory design pattern, Real time example?**

Factory Design pattern is based on [**Encapsulation**](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html)object oriented concept. Factory method is used to create different object from factory often refereed as Item and it encapsulate the creation code. So instead of having object creation code on client side we encapsulate inside **Factory method in Java**. One of the best examples of factory pattern in Java is BorderFactory Class of Swing API.

examples of factory method design pattern from JDK is:

**valueOf()** method which returns object created by factory equivalent to value of parameter passed.

**getInstance()** method which creates instance of Singleton class.

**newInstance()** method which is used to create and return new instance from factory method every time called.

**getType() and newType() equivalent of getInstance() and newInstance() factory method but used when factory method resides in separate class.**

1. **What is HashMap race condition and how to resolve it?**
2. Can we use setter and constructor both for a bean class at time?
3. What is immutable object? How does it help on writing concurrent application?
4. How will you awake a blocked thread in java?
5. What are some common problems you have faced in multi-threading environment? How did you resolve it?
6. Difference between Executor and Executors in Java?
7. How to find which thread is taking maximum CPU in windows and Linux server?
8. Difference between green thread and native thread in Java?
9. Difference between thread and process?
10. What is context switching in multi-threading?
11. Difference between deadlock and live lock, deadlock and starvation?
12. What thread-scheduling algorithm is used in Java?
13. What is thread-scheduler in Java?
14. How do you handle un-handled exception in thread?
15. What is thread-group, why its advised not to use thread-group in Java?
16. Why Executor framework is better than creating and managing thread by application?
17. Why wait, notify and notify All are defined in Object class in Java?
18. Difference between Runnable and Thread in Java
19. Why wait and notify needs to be called from synchronized method?
20. Race condition between wait () and notify () in Java that could exists if we don't call them inside synchronized method or block
21. Invokeandwait vs invoke Later in Java Swing
22. Why you should not mix static and non-static synchronized method in Java
23. **How to stop Thread in Java?**
24. **What happens on hashmap in Java if the size of the hashmap exceeds a given threshold defined by load factor?**

If the size of the Map exceeds a given threshold defined by load-factor e.g. If load factor is .75 it will act to re-size the map once it filled 75%. Like other collection classes like Array List, Java hashmap re-size itself by creating a new bucket array of size twice of previous size of hashmap, and then start putting every old element into that new bucket array. This process is called **rehashing** because it also applies hash function to find new bucket location.

1. Which jars is used to connect hibernate jdbc with spring?
2. How wait and notify are used in a code?
3. How to use volatile?
4. How we know all threads are executed and what is the mechanism implemented behind executor service. shutdown
5. What happens when wait is called internal implementation and notify?
6. How reentrant lock is implemented
7. In web service, what is payload, and how to handle a null when we are getting in request?
8. How HashMap works in jdk 1.8
9. What is session and session factory in hibernate
10. How executor frameworks internally
11. How to submit multiple threads in executor?
12. One prototype bean is injected in singleton bean. How singleton bean will behave?
13. What is difference between session and global session.
14. What are the new packages, classes and interfaces added in jdk1.8
15. How will you use lambda expression for list
16. Serialization: Class A extends B, does variable of class B also serialized
17. Internal hashmap working procedure
18. Internal linked HashMap working procedure.
19. What are the REST API standerd?
20. What is the diff b/w PUT/POST, when to use put and when to use POST
21. How many type locking
22. What are the diff b/w runnable and callable?
23. What are the way to create thread using Executer?
24. How to avoid deadlock
25. Markup interface
26. What is serialization how java works
27. What is Java garbage collector architect?
28. Connection pooling code
29. How link hash map work internally
30. What is fail fast and fail safe
31. What is JVM architecture
32. What is week and strong reference
33. Markup interface
34. What is serialization how java works
35. How you will short hash map’s value
36. If one huge file inserting into DB how you will insert what will be your approach to optimize code
37. On what basis, you will go for SOAP and REST
38. Apart from 200 what code you do received in REST call
39. What is difference between array list and link list
40. What is double locking in thread?
41. how you will write function to display in order and tree order tree
42. how you will write function to display BFS and DFS order tree
43. How you will call abstract constructer
44. If single client calling multiple REST service from AJAX then how you will take response from server as you required some information from server 1 or some information server 2 and so on.
45. If you are calling 2 service from AJAX then How you will insure that one service called then you need to call second service call

1. "If you have client A, middleware B and Server C.

Server C can send multiple response to B and B can response to client A, But Server C has limitation that can sent only 10 Records in single response to middleware B

1. **Can we make array volatile in Java?**  
   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?**  
   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 several 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) {

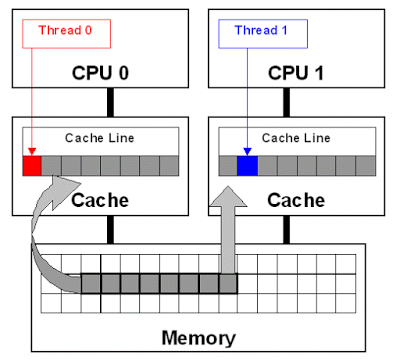
While (condition does not hold)

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

... // Perform action appropriate to condition

}

**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:

[](http://2.bp.blogspot.com/-Tze9foqpb74/VepwCzXHGCI/AAAAAAAADtM/i4KQDaefqk4/s1600/False+Sharing+in+Multi-threaded+application.gif)

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?**   
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?**    
Thread-local variables are variables confined to a thread, it’s like thread's own copy which is not shared between multiple threads. Java provides a thread local 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?**    
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?**    
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?**   
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?**    
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.

**Date types and Basic Java Interview Questions  
17) What is the right data type to represent a price in Java?**   
Big decimal 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.  
**19) How do you convert bytes to long in Java?**   
  
**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.  
  
**21) There are two classes B extends A and C extends B, Can we cast B into C e.g. C = (C) B;**   
  
**22) Which class contains clone method? Cloneable or Object?**    
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 variables 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*

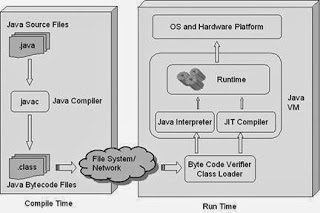
B += a; *// ok*

**25) Can I store a double value in a long variable without casting?**   
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.  
**26) What will this return 3\*0.1 == 0.3? True or false?**

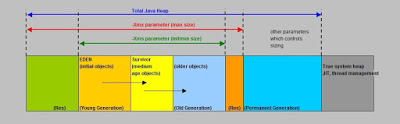
The short answer is false because some floating-point numbers cannot be represented exactly.  
  
**27) Which one will take more memory, an int or Integer?**  
An Integer object will take more memory an Integer is an object and it store meta data overhead about the object and int is primitive type so its takes less space.  
  
**28) Why is String Immutable in Java?**    
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.   
  
**30) What is constructor chaining in Java?**    
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.

**JVM Internals and Garbage Collection Interview Questions**

[Java Performance books](http://javarevisited.blogspot.com/2014/07/top-5-java-performance-tuning-books.html).   
**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?**    
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 weak reference and soft reference in Java?**   
Though both weak reference and soft reference helps garbage collector and memory efficient, weak reference becomes eligible for garbage collection as soon as last strong reference is lost but soft reference even though it cannot prevent GC, it can delay it until JVM absolutely need memory.  
  
**35) How do weak HashMap works?**  
Weak HashMap works like a normal HashMap but uses weak reference 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?**   
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?**   
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?**   
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?**    
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.

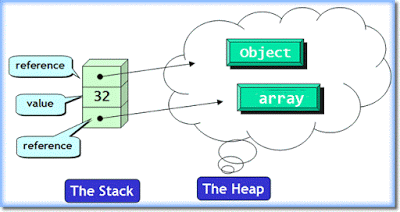
[](http://2.bp.blogspot.com/-ls3yC0U7ouo/VhDqX-3OUbI/AAAAAAAAD40/Zcsc5uCaGq0/s1600/JVM+JRE+JDK.jpg)

**40) Explain Java Heap space and Garbage collection?**    
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.

[](http://3.bp.blogspot.com/-DqV12_uIeZ4/VhDqtPCVIVI/AAAAAAAAD48/uqWZB0BgZUI/s1600/java_heaps_memory.jpg)

**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?**

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.

[](http://1.bp.blogspot.com/-NZeVo83YJAA/VhDrDO0oWtI/AAAAAAAAD5E/mEek8Ll7NfU/s1600/Difference+between+stack+and+heap+memory+in+Java.gif)

**Basic Java concepts Interview Questions**

**44) What's the difference between "a == b" and "a.equals(b)"?**    
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.  
  
**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 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.

**Java Collections Framework Interview Questions**  
  
**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 linked HashMap and priority queue in Java?**   
Priority queue guarantees that lowest or highest priority element always remain at the head of the queue, but linked HashMap maintains the order on which elements are inserted. When you iterate over a priority queue, iterator doesn't guarantee any order but iterator of linked HashMap does guarantee the order on which elements are inserted.  
  
**51) Difference between arraylist and linked list in Java?**    
arrraylist is backed by array data structure, supports random access and linkedlist is backed by linked list data structure and doesn't support 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.  
  
**53) How do you print Array in Java?**    
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) Linked list 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 tree map in Java?**   
A Red Black tree is used to implement tree map in Java.

**56) What is the difference between Hash table and HashMap?**   
There are many differences between these two classes, some of them are following:  
a) Hash table is a legacy class and present from JDK 1, hash map was added later.  
B) Hash table is synchronized and slower but HashMap is not synchronized and faster.  
C) Hash table doesn't allow null keys but HashMap allows one null key.  
  
**57) How hash set works internally in Java?**    
**Hash set is internally implemented using an HashMap**. Since a Map needs key and value, a default value is used for all keys. Like HashMap, hash set doesn't allow duplicate keys and only one null key, I mean you can only store one null object in hash set.  
  
**58) Write code to remove elements from array list while iterating?**    
 Key here is to check whether candidate uses array list’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 to 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 :

// from arraylist.java JDK 1.7

Private static final int DEFAULT\_CAPACITY = 10;

//from hashmap.java JDK 7

Static final int DEFAULT\_INITIAL\_CAPACITY = 1 **<<** 4; // aka 16

**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, that’s 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.

**Java IO and NIO Interview questions**

IO is very important from Java interview point of view. You should have a good knowledge of old Java IO, NIO, and NIO2 alsong with some operating system and disk IO fundamentals. Here are some frequently asked questions form Java IO.  
  
66) In my Java program, I have three sockets? How many threads I will need to handle that?  
  
  
**71) The difference between direct buffer and non-direct buffer in Java?   
  
72) What is the memory mapped buffer in Java?   
  
73) What is TCP NO DELAY socket option?  
  
74) What is the difference between TCP and UDP protocol?   
  
75) The difference between byte buffer and string buffer in Java?**   
  
**Java Best Practices Interview question**

**76) What best practices you follow while writing multi-threaded code in Java?**    
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 Collections 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?**    
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.  
  
**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?**    
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?**    
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.

**Date, Time and Calendar Interview questions in Java**  
**82) Does simple date format is safe to use in the multi-threaded program?**    
No, unfortunately, date format and all its implementations including simple date format is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. Confining simple date format object into a thread local 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?**    
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.  
  
**84) How do you show timezone in formatted date in Java?   
85) The difference between java.util.Date and java.sql.Date in Java?   
86) How to you calculate the difference between two dates in Java?   
  
87) How do you convert a String(YYYYMMDD) to date in Java?**   
  
**Unit testing junit Interview questions**

89) How do you test static method?   
You can use powermock library to test static methods in Java.  
90) How to do you test a method for an exception using junit?   
91) Which unit testing libraries you have used for testing Java programs?   
92) What is the difference between @Before and @beforeclass annotation?   
  
**Programming and Coding Questions**

**93) How to check if a String contains only numeric digits? (**[**solution**](http://java67.blogspot.com/2014/01/java-regular-expression-to-check-numbers-in-String.html)**)  
  
94) How to write LRU cache in Java using Generics?   
  
95) Write a Java program to convert bytes to long?   
  
96) How to reverse a String in Java without using string buffer? (**[**solution**](http://java67.blogspot.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.htm)**)  
  
97) How to find the word with the highest frequency from a file in Java? (**[**solution**](http://java67.blogspot.com/2015/10/java-program-to-find-repeated-words-and-count.html)**)  
  
98) How do you check if two given String are anagrams? (**[**solution**](http://javarevisited.blogspot.sg/2013/03/Anagram-how-to-check-if-two-string-are-anagrams-example-tutorial.html)**)  
  
99) How to print all permutation of a String in Java? (**[**solution**](http://javarevisited.blogspot.com/2015/08/how-to-find-all-permutations-of-string-java-example.html)**)  
  
100) How do you print duplicate elements from an array in Java? (**[**solution**](http://javarevisited.blogspot.com/2015/06/3-ways-to-find-duplicate-elements-in-array-java.html)**)  
  
101) How to convert String to int in Java? (**[**solution**](http://java67.blogspot.com/2015/08/2-ways-to-parse-string-to-int-in-java.html)**)**  
  
102) **How to swap two integers without using temp variable?** ([solution](http://java67.blogspot.com/2015/08/how-to-swap-two-integers-without-using.html))

**Java Interview questions from OOP and Design Patterns**  
  
**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?**   
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?**    
  
**107) What is Law of Demeter violation? Why it matters?**    
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?**  
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.   
  
**109) What is "dependency injection" and "inversion of control"? Why would someone use it?**   
  
**110) What is an abstract class? How is it different from an interface? Why would you use it?**   
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?**([answer](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
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?**   
The difference between Adapter and Proxy patterns is in their intent. Since both Adapter and Proxy pattern encapsulate the class which 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?**   
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?**   
  
**124) When do you use Flyweight pattern?**   
  
Flyweight pattern allows you to share object to support large numbers without creating too many objects. 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.

**Miscellaneous Java Interview Questions**

It contains XML Processing in Java Interview question, JDBC Interview question, Regular expressions Interview questions, Java Error and Exception Interview Questions, Serialization,  
  
**125) The difference between nested static class and top level class?**   
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?**   
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?**    
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?**    
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?**   
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.  
  
**132) Tell me 5 features introduced in JDK 1.8?**   
**Lambda expression**, which allows you pass an anonymous function as object.

* **Stream API**, take advantage of multiple cores of modern CPU and allows you to write succinct code.
* **Date and Time API**, finally you have a solid and easy to use date and time library right into JDK
* **Extension methods**, now you can have static and default method into your interface
* **Repeated annotation**, allows you apply the same annotation multiple times on a type

**133) What is the difference between Maven and ANT in Java?**  
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.

## Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception clonenotsupportedexception being thrown.

## 134. Upgrading Java Classes with Backward-Compatible Serialization?

**135.What is deff between keyset() and entryset of Map.**

Java.util.Map<K,V> interface provides three methods keyset(), values() and entryset() for retrieving keys, values and key-value pairs respectively

Map<String, String> map = new hashmap<String, String>();

    Map.put("1", "Jan");

    Map.put("2", "Feb");

    Map.put("3", "Mar");

//Using the entryset you can get all the keys and values in one hit. This saves having to perform the map lookup.

    For (Map.Entry<String, String> entry : map.entryset()) {

        System.out.println("Key : " + entry.getkey() + " Value : "+ entry.getvalue());

    }

  // keyset() inefficient because you have to fetch the keys and perform a lookup in the map with the key.

    For (String key : map.keyset()) {

        System.out.println("Key : " + key.tostring() + " Value : " + map.get(key));

    }

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.

### ->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.

### ->A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

->public and abstract are the only applicable modifiers for method declaration in an interface.

-> Constructor cannot be inherited, though a derived class can call the base class constructor

* Constructors use this to refer to another constructor in the same class with a different parameter list.
* Constructors use super to invoke the super class’s constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

-> Static block which exactly executed exactly once when the class is first loaded into JVM. Before going to the main method the static block will execute.

**Can a Byte object be cast to a double value?**

No, an object cannot be cast to a primitive value.

**When can an object reference be cast to an interface reference?**

An object reference is cast to an interface reference when the object implements the referenced interface.

**What is casting?**

There are two types of casting, casting between primitive numeric types and casting between object references. Casting between numeric types is used to convert larger values, such as double values, to smaller values, such as byte values. Casting between object references is used to refer to an object by a compatible class, interface, or array type reference.

Up and Down costing

Class Car {}  
class Car 1 extends Car {}  
class Car 2 extends Car {}  
  
Car c = new Car 1 (); // a Car 1 object is up cast to a Car  
Car b = (Car 1) c; // a Car 1 object is downcast from a Cars  
  
we cannot up cast a Car 2 to a Car and then downcast it to a Car 1. We also cannot downcast a Car to a Car 1. In the working example, above, the c references a Car 1, so you can downcast the Car 1.

### Difference between PATH and CLASSPATH in Java

Here are some of the common difference between PATH vs CLASSPATH in Java

Path is used define where the system can find the executables (.exe) files and classpath is used to specify the location .class files.  
  
1) Main difference between PATH and CLASSPATH is that PATH is an environment variable which is used to locate JDK binaries like "java" or "javac" command used to run java program and compile java source file. On the other hand CLASSPATH environment variable is used by System or Application Class Loader to locate and load compile Java byte codes stored in .class file.  
  
2) In order to set PATH in Java you need to include JDK\_HOME/bin directory in PATH environment variable while in order to set CLASSPATH in Java you need to include all those directory where you have put either your .class file or JAR file which is required by your Java application.  
  
3) Another significant difference between PATH and CLASSPATH is that PATH cannot be overridden by any Java settings but CLASSPATH can be overridden by providing command line option -classpath or -cp to both "java" and "javac" commands or by using Class-Path attribute in Manifest file inside JAR archive.

Autoboxing and Unboxing

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called unboxing.

Arraylist<Integer> intlist = new arraylist<Integer>();  
intlist.add(1); //autoboxing - primitive to object  
intlist.add(2); //autoboxing  
        
threadlocal<Integer> intlocal = new threadlocal<Integer>();  
intlocal.set(4); //autoboxing  
  
int number = intlist.get(0); // unboxing  
int local = intlocal.get(); // unboxing in Java

Int i = 0;

I = new Integer(5); // auto-unboxing

Integer i2 = 5; // autoboxing

Conversion: String to double

String text = "12.34"; // example String

Double value = Double.parsedouble(text);

Double to string

Double d = 0.11;  
 String s = Double.tostring(d);

Convert int to string:

Int x = 1;  
      
    String y = Integer.tostring(x);

Constructor chaining in java?

Calling another constructor in the same class from another constructor is called constructor chaining

Ex: Object();

Animal calls super()

Horse calls super()

Main() calls new Horse()

Public class mychaining {

    Public mychaining(){

        System.out.println("In default constructor...");

    }

    Public mychaining(int i){

        This();

        System.out.println("In single parameter constructor...");

    }

    Public mychaining(int i,int j){

        This(j);

        System.out.println("In double parameter constructor...");

    }

    Public static void main(String a[]){

        Mychaining ch = new mychaining(10,20);

    }

}

[**What are all the different ways to create an object in Java?**](http://stackoverflow.com/questions/95419/what-are-all-the-different-ways-to-create-an-object-in-java)

There are four different ways to create objects in java:

A. Using new keyword this is the most common way to create an object in java. Almost 99% of objects are created in this way.

Myobject object = new myobject();

B. Using Class.forname() If we know the name of the class & if it has a public default constructor we can create an object in this way.

Myobject object = (myobject) Class.forname("subin.rnd.myobject").newinstance();

C. Using clone() The clone() can be used to create a copy of an existing object.

Myobject anotherobject = new myobject();

Myobject object = anotherobject.clone();

D. Using object deserialization Object deserialization is nothing but creating an object from its serialized form.

Objectinputstream instream = new objectinputstream(aninputstream );

Myobject object = (myobject) instream.readobject();

**Q: Suppose you are using a map in your program, how would you count the number of times the program calls the put() and get() functions?**

Answer: keep the count variables static. We know static variables have only one copy for all objects of the class so the total count would be reflected in count variables.

**Q: Explain what object reflection is in Java and why it is useful.**

Object Reflection is a feature in Java which provides a way to get reflective information about Java classes and objects, such as:

1. Getting information about methods and fields present inside the class at run time.

2. Creating a new instance of a class.

3. Getting and setting the object fields directly by getting field reference, regardless of what the access modifier is.

Import java.lang.reflect.\*;

Public class Sample {

Public static void main(String args[]) {

Try {

Class c = Class.forname(“java.sql.Connection”);

Method m[] = c.getdeclaredmethods();

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

System.out.println(m[i].tostring());

}

} catch (Throwable e) {

System.err.println(e);

}}}

Constructors are special methods defined in a class that create and return an object of the class in which they're defined. Constructors have the same name as the name of a class and they don't specify a return type, not even a void. Constructors come in two flavors: user-defined constructors and default constructors.

**Can you override private or static method in Java ?** No (private and static methods can’t be override)

**2) If a method throws nullpointerexception in super class, can we override it with a method which throws runtimeexception?**

Can very well throw super class of runtimeexception in overridden method but you cannot do same if its checked Exception.

**3) What will happen if we put a key object in a hashmap which is already there?**

If you put the same key again than it will replace the old mapping because hashmap doesn't allow duplicate keys

**4) What is the issue with following implementation of compareto() method in Java**

Public int compareto(Object o){

Employee emp = (Employee) emp;

Return [this.id](http://this.id/) - [o.id](http://o.id/);

}

Where id is an integer number ?

If id is negative than subtraction may overflow and produce incorrect result

**5)How do you ensure that N thread can access N resources without deadlock?**

Key point here is order, if you acquire resources in a particular order and release resources in reverse order you can prevent deadlock.

**6) What is difference between string buffer and string builder in Java ?**

Stringbuffer methods are synchronized while stringbuilder is non synchronized.

**7) Can you access non static variable in static context?**

No you cannot access static variable in non static context in Java

**8) Why wait and notify is declared in Object class instead of Thread?**

Locking is done on per object basis and not thread.

10) How to create thread-safe singleton in Java using double checked locking?

**11) Why character array is better than String for Storing password in Java?**

1) Since Strings are immutable in Java if you store password as plain text it will be available in memory until Garbage collector clears it and since String are used in String pool for reusability there is pretty high chance that it will be remain in memory for long duration, which pose a security threat. Since any one who has access to memory dump can find the password in clear text and that's another reason you should always used an encrypted password than plain text. Since Strings are immutable there are no way contents of Strings can be changed because any change will produce new String, while if you char[] you can still set all his element as blank or zero. So Storing password in character array clearly mitigates security risk of stealing password.

2) Java itself recommends using getpassword() method of jpasswordfield which returns a char[] and deprecated gettext() method which returns password in clear text stating security reason. Its good to follow advice from Java team and adhering to standard rather than going against it.

3) With String there is always a risk of printing plain text in log file or console but if use Array you won't print contents of array instead its memory location get printed. Though not a real reason but still make sense.

String strpassword="Unknown";

Char[] charpassword= new char[]{'U','n','k','w','o','n'};

System.out.println("String password: " + strpassword);

System.out.println("Character password: " + charpassword);

String password: Unknown

Character password: [C@110b053

That's all on why character array is better choice than String for storing passwords in Java.  Though using char[] is not just enough you need to erase content to be more secure

**12) Why String is immutable in Java or How Substring can cause memory leak in Java**

1) Imagine stringpool facility without making string immutable , it’s not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many [reference variables](http://javarevisited.blogspot.sg/2012/02/difference-between-instance-class-and.html) , so if any one of them change the value others will be automatically gets affected i.e. Let’s say  
String A = "Test"  
String B = "Test"   
Now String B called "Test".touppercase() which change the same object into "TEST" , so A will also be "TEST" which is not desirable.  
  
2)String has been widely used as parameter for many Java classes e.g. For opening network connection, you can pass hostname and port number as string , you can pass database URL as string for opening database connection, you can [open any file in Java](http://javarevisited.blogspot.sg/2012/07/read-file-line-by-line-java-example-scanner.html) by passing name of file as argument to File I/O classes.  
  
In case, if String is not immutable, this would lead serious security threat , I mean some one can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access of those file. Because of immutability, you don't need to worry about those kind of threats. This reason also gel with, Why String is final in Java, by making java.lang.String final, Java designer ensured that no one overrides any behavior of String class.  
  
3)Since String is immutable it can safely shared between many threads ,which is very important for multithreaded programming and to avoid any [synchronization issues in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), Immutability also makes String instance [thread-safe in Java](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), means you don't need to synchronize String operation externally. Another important point to note about String is [memory leak caused by substring](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html), which is not a thread related issues but something to be aware of.  
  
4) Another reason of Why String is immutable in Java is to allow String to cache its hashcode , being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in [hashmap in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html).  This one is also suggested by  Jaroslav Sedlacek in comments below. In short because String is immutable, no one can change its contents once created which guarantees [hashcode](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html)of String to be same on multiple invocation.  
  
5) Another good reason of Why String is immutable in Java suggested by Dan Bergh Johnsson on comments is: The absolutely most important reason that String is immutable is that it is used by the [class loading mechanism](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.diskerasingwriter"  
  
 **What is serial version id? What would happen if you don't define this?**

 Serial version id is an ID which is stamped on object when it get serialized usually hashcode of object . Serialversionuid is used for version control of an object. You can specify serialversionuid in your class file also.  Consequence of not specifying serialversionuid is that when you add or modify any field in class then already serialized class will not be able to recover because serialversionuid generated for new class and for old serialized object will be different. Java serialization process relies on correct serialversionuid for recovering state of serialized object and throws java.io.invalidclassexception in case of serialversionuid mismatch.

**Q: If class A does not implement Serializable but a subclass B implements Serializable, will the fields of class A be serialized when B is serialized?**

A: Only the fields of Serializable objects are written out and restored. The object may be restored only if it has a no-arg constructor that will initialize the fields of non-serializable super types. If the subclass has access to the state of the super class it can implement writeobject and readobject to save and restore that state.

So, if the subclass has access to the fields of its non-Serializable superclass(es) it can use the writeobject and readobject protocol to implement serialization. Otherwise, there will be fields that won't be possible to serialize.

## Serializing an Object:

The objectoutputstream class is used to serialize an Object. The following serializedemo program instantiates an Employee object and serializes it to a file.

When the program is done executing, a file named employee.ser is created. The program does not generate any output, but study the code and try to determine what the program is doing.

Note: When serializing an object to a file, the standard convention in Java is to give the file a .serextension.

Import java.io.\*;

Public class serializedemo

{

Public static void main(String [] args)

{

Employee e = new Employee();

E.name = "A";

E.address = "aaaaa";

E.SSN = 11122333;

E.number = 101;

Try

{

Fileoutputstream fileout =

New fileoutputstream("employee.ser");

Objectoutputstream out =

New objectoutputstream(fileout);

Out.writeobject(e);

Out.close();

Fileout.close();

}catch(ioexception i)

{

I.printstacktrace();

}

}

}

## Deserializing an Object:

The following deserializedemo program deserializes the Employee object created in the serializedemo program. Study the program and try to determine its output:

Import java.io.\*;

Public class deserializedemo

{

Public static void main(String [] args)

{

Employee e = null;

Try

{

Fileinputstream filein =

New fileinputstream("employee.ser");

Objectinputstream in = new objectinputstream(filein);

E = (Employee) in.readobject();

In.close();

Filein.close();

}catch(ioexception i)

{

I.printstacktrace();

Return;

}catch(classnotfoundexception c)

{

System.out.println("Employee class not found");

C.printstacktrace();

Return;

}

System.out.println("Deserialized Employee...");

System.out.println("Name: " + e.name);

System.out.println("Address: " + e.address);

System.out.println("SSN: " + e.SSN);

System.out.println("Number: " + e.number);

}

}

This would produce following result:

Deserialized Employee...

Name: A

Address:aaaa

SSN: 0

Number:101

Here are following important points to be noted:

* The try/catch block tries to catch a class not found exception, which is declared by the readobject() method. For a JVM to be able to deserialize an object, it must be able to find the bytecode for the class. If the JVM can't find a class during the deserialization of an object, it throws a class not found exception.
* Notice that the return value of readobject() is cast to an Employee reference.
* The value of the SSN field was 11122333 when the object was serialized, but because the field is transient, this value was not sent to the output stream. The SSN field of the deserialized Employee object is 0.

**4) While serializing you want some of the members not to serialize? How do you achieve it?**

If you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

**5) What will happen if one of the members in the class doesn't implement Serializable interface?**

If you try to serialize an object of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a‘notserializableexception’ will be thrown at [runtime](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) and this is why I always put a serializablealert (comment section in my code) , one of the [code comment best practices](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html), to instruct developer to remember this fact while adding a new field in a Serializable class.

**6) Why String, Integer and other wrapper classes are considered good keys ?**

String, Integer and other wrapper classes are natural candidates of hashmap key, and String is most frequently used key as well because [String is immutable and final](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html),and overrides equals and hashcode() method. Other wrapper class also shares similar property. Immutabiility is required, in order to prevent changes on fields used to calculate hashcode() because if key object return different hashcode during insertion and retrieval than it won't be possible to get object from hashmap. Immutability is best as it offers other advantages as well like [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), If you can  keep your hashcode same by only making certain fields [final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), then you go for that as well. Since equals() and hashcode() method is used during reterival of value object from hashmap, its important that key object correctly override these methods and follow contact. If unequal object return different hashcode than chances of collision will be less which subsequently improve performance of hashmap.

**2) Can we use any custom object as key in hashmap ?**

You can use any Object as key in Java hashmap provided it follows equals and hashcode contract and its hashcode should not vary once the object is inserted into [Map](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html). If custom object is Immutable than this will be already taken care because you cannot change it once created.

**3) Can we use concurrenthashmap in place of Hashtable ?**

Since we know Hashtable is synchronized but concurrenthashmap provides better concurrency by only locking portion of map determined by concurrency level. Concurrenthashmap is certainly introduced as Hashtable and can be used in place of it but Hashtable provide stronger thread-safety than concurrenthashmap

"Have you used hashmap before" or  "What is hashmap? Why do we use it “

[Hashmap accept null while Hashtable doesn't](http://www.blogger.com/), [hashmap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), hashmap is fast and so on along with basics like its stores key and value pairs etc.

"Do you Know how hashmap works in Java” or "How does get () method of hashmap works in Java"

"hashmap works on principle of hashing, we have put(key, value) and get(key) method for storing and retrieving Objects from hashmap. When we pass Key and Value object  to put() method on Java hashmap, hashmap implementation calls [hashcode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html)on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that hashmap in Java stores both key and value object asmap.Entry in bucket which is essential to understand the retrieving logic.

"What will happen if two different objects have same hashcode?”

"Since hashcode is same, bucket location would be same and collision will occur in hashmap, Since hashmap use linkedlist to store object, this entry (object of Map.Entry comprise key and value) will be stored in [linkedlist](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html).

**How will you retrieve Value object if two Keys will have same hashcode?”**

We will call keys.equals() method to identify correct node in Linked List and return associated value object for that key in Java Hash Map   
  
"What happens On hashmap in Java if the size of the hashmap  exceeds a given threshold defined by load factor ?".

If the size of the Map exceeds a given threshold defined by load-factor e.g. If load factor is .75 it will act to re-size the map once it filled 75%. Similar to other collection classes like [arraylist](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html),  Java hashmap re-size itself by creating a new bucket array of size twice of previous size of hashmap , and then start putting every old element into that new bucket array. This process is called rehashing because it also applies hash function to find new bucket location.

Fidelity looking for a person who has hands on experience in Spring, web services and Flex which I do not have

1. How to put object into Hashtable?

2. How to retrieve object from Hashtable in Java?

3. How to reuse Hashtable by using clear()?

4. How to check if Hastable contains a particular value?

5. How to check if Hashtable contains a particular key?

6. How to traverse Hashtable in Java?

7. How to check if Hashtable is empty in Java?

8. How to Copy content of Hashtable into hashmap?

9. How to find size of Hashtable in Java?

10. How to get all values form hashtable in Java?

11. How to get all keys from hashtable in Java?

Import java.util.Collection;

Import java.util.Enumeration;

Import java.util.Hashtable;

Import java.util.Set;

Public class hashtabledemo {

Public static void main(String args[]) {

// Creating Hashtable for example

Hashtable companies = new Hashtable();

// Java Hashtable example to put object into Hashtable

// put(key, value) is used to insert object into map

Companies.put("Google", "United States");

Companies.put("Nokia", "Finland");

Companies.put("Sony", "Japan");

// Java Hashtable example to get Object from Hashtable

// get(key) method is used to retrieve Objects from Hashtable

Companies.get("Google");

// Hashtable containskey Example

// Use containskey(Object) method to check if an Object exits as key in

// hashtable

System.out.println("Does hashtable contains Google as key: "

+ companies.containskey("Google"));

// Hashtable containsvalue Example

// just like containskey(), containsvalue returns true if hashtable

// contains specified object as value

System.out.println("Does hashtable contains Japan as value: "

+ companies.containsvalue("Japan"));

// Hashtable enumeration Example

// hashtabl.elements() return enumeration of all hashtable values

Enumeration enumeration = companies.elements();

While (enumeration.hasmoreelements()) {

System.out

.println("hashtable values: " + enumeration.nextelement());

}

// How to check if Hashtable is empty in Java

// use isempty method of hashtable to check emptiness of hashtable in

// Java

System.out.println("Is companies hashtable empty: "

+ companies.isempty());

// How to find size of Hashtable in Java

// use hashtable.size() method to find size of hashtable in Java<

//////////////////////////////////////////////////////////////////////////////

 if i want to make a class serializable, what are the things that we need to do for making the same

Implement serializable interface,fileoutputstream,objectoutputstream,writeobject.

 inner classes...when do we need an inner class..

Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are simply called static nested classes. Non-static nested classes are called inner classes.

A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class. As a member of the outerclass, a nested class can be declared private, public,protected, or package private. (Recall that outer classes can only be declared public or package private.)

## Why Use Nested Classes?

There are several compelling reasons for using nested classes, among them:

* It is a way of logically grouping classes that are only used in one place.
* It increases encapsulation.
* Nested classes can lead to more readable and maintainable code.

Logical grouping of classes—If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

Increased encapsulation—Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself be hidden from the outside world.

More readable, maintainable code—Nesting small classes within top-level classes places the code closer to where it is used.

## Static Nested Classes

As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class — it can use them only through an object reference.

**Note:** A static nested class interacts with the instance members of its outer class (and other classes) just like any other top-level class. In effect, a static nested class is behaviourally a top-level class that has been nested in another top-level class for packaging convenience.

1. When do you prefer to use static methods?
2. Worked with any collection objects.
3. Can we add duplicate keys in Hashmap?
4. What kind of collection I can use for automated sorting?
5. Use of finalize method
6. what are the different ways of object creation? - new, clone, de serialization??Create an object in serialized way
7. how will you implement serializable class?
8. wrapping an object? How will you put an object?
9. where static data members are used?
10. can a static variable be serialized? NO
11. if i want to make a class serializable, what are the things that we need to do for making the same
12. difference between wait and sleep 🡪
13. how can I break the waiting process...
14. how can you make a method thread safe?
15. is there any performance issue with synchronized method.

# Difference between java.sql.Date and java.util.Date

1. Java.sql.Date stores only day, month and year.  
   2. Java.util.Date stores day, month, year along with time information upto milliseconds accuracy.  
   The sql versions are namely:  
   1. Java.sql.Date - having only date information  
   2. Java.sql.Time - having only time information  
   3. Java.sql.timestamp - having date + time information with nanoseconds precision

**Default Capacity** of any Collection say **hashset** is **16**. It means when you create a hashset using its **default constructor** then first hashset will be created to hold 16 elements or you can say that **memory space** is allocated to hold **16 elements**.  
  
**Load factor**   
  
By default load factor is **0.75**. It means when the **75 of the capacity** will be filled and you add new element then **capacity will be increased** (most probably doubled).  
  
For example **16 default capacity** and **0.75 load factor** till 12th element addition the capacity will **remain 16**but when you add 13th element then first capacity will be increased to 32 and element will be added.

If the number of entries reaches threshold (capacity \* load factor), rehashing is done automatically. That means too small load factor can incur frequent rehashing as entries grow.

Interrupted exception will be thrown when a sleeping or waiting

Thread is interrupted.

To successfully store and retrieve objects from a hash table, the objects used as keys must implement the hash Code method and the equals method.

An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. Note that the hash table is open: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased by calling the rehash method.

Rehash

Protected void rehash()

Increases the capacity of and internally reorganizes this hashtable, in order to accommodate and access its entries more efficiently. This method is called automatically when the number of keys in the hashtable exceeds this hashtable's capacity and load factor.

Finalize() is never run more than once on any object

Before an object is garbage collected, the runtime system calls its finalize() method. The intent is for finalize() to release system resources such as open files or open sockets before getting collected.

Convert int to string

Int aint = 1;  
      
 String astring = Integer.tostring(aint);

String to int.

String astring = "78";  
    int aint = Integer.parseint(astring);

String to double

String astring = "78";  
    double adouble = Double.parsedouble(astring);

Throws a numberformatexception when the String can't be converted to an int type.

Int to Double

Int i = 10;  
double d = Double.parsedouble(Integer.tostring(i));

Double.intvalue() converts a Double to an int

# What are differences among throw, throws, and Throwable?

* In Java, all error's and execption's class are drieved from java.lang.Throwable class. It is the top of the hierarchy of classes of error and exceptions. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.
* Throws is a post-method modifier and specifies which execptions may be thrown by the method. If they are checked exceptions, the compiler will guarantee the code invoking that method must catch these checked exceptions.
* Throw statement is used to throw an error or exceptions. Throw statement requires a single argument: a instance of any subclass of the Throwable class or Throwable class. Executingthrow statement triggers the JVM to throw this exception and causes an exception to occur.
* Final class myimmutableclass   
  {   
  // instance var are made private & final to restrict the access
* Private final int count;   
  private final double value;
* // Constructor where we can provide the constant value   
  public myimmutableclass(int paramcount,double paramvalue)   
  {   
     count = paramcount;   
     value = paramvalue;   
  }
* // provide only methods which return the instance var   
  // & not change the values
* Public int getcount() {   
     return count; }
* Public double getvalue() {   
    return value; } }
* // class testimmutable   
  public class testimmutable {   
    public static void main(String[] args)   {   
     myimmutableclass obj1 = new myimmutableclass(3,5);
* System.out.println(obj1.getcount());   
     System.out.println(obj1.getvalue());
* // there is no way to change the values of count & value-   
    // no method to call besides getxx, no subclassing, no public access to var -> Immutable } }
* The possible use of immutable classes would be a class containing a price list represented for a set of products.
* **First - security** <http://www.javafaq.nu/java-article1060.html>
* The main reason why String made immutable was security. We have a file open method with login check. We pass a String to this method to process authentication which is necessary before the call will be passed to OS. If String was mutable it was possible somehow to modify its content after the authentication check before OS gets request from program then it is possible to request any file. So if you have a right to open text file in user directory but then on the fly when somehow you manage to change the file name you can request to open "passwd" file or any other. Then a file can be modified and it will be possible to login directly to OS.
* **Second - Memory efficiency** <http://hikrish.blogspot.com/2006/07/why-string-class-is-immutable.html>
* JVM internally maintains the "String Pool". To achieve the memory efficiency, JVM will refer the String object from pool. It will not create the new String objects. So, whenever you create a new string literal, JVM will check in the pool whether it already exists or not. If already present in the pool, just give the reference to the same object or create the new object in the pool. There will be many references point to the same String objects, if someone changes the value, it will affect all the references. So, sun decided to make it immutable

Thread Safe

Object Identity:

An identity in object-oriented programming, object-oriented design and object-oriented analysis describes the property of objects that distinguishes them from other objects.  
  
Identity of objects allows objects to be treated as black boxes. The object need not expose its internal structure. It can still be referred to, and its other properties can be accessed via its external behaviour associated with the identity. The identity provides a mechanism for referring to such parts of the object that are not exposed in the interface. Thus, identity is the basis for polymorphism in object-oriented programming.  
  
Identity allows comparison of references. Two references can be compared whether they are equal or not. Due to the identity property, this comparison has special properties. If the comparison of references indicates that the references are equal, then it's clear that the two objects pointed by the references are the same object. If the references do not compare equal, then it's not necessarily guaranteed that the identity of the objects behind those references is different. The object identity of two objects of the same type is the same, if every change to either object is also a change to the other object.

How to Make a Class Serializable

1. Implement the Serializableinterface.
2. Make sure that instance-level, locally defined state is serialized properly.
3. Make sure that superclass state is serialized properly.
4. Override equals( )and hashcode( ).

Using Serialization

Serialization is a mechanism built into the core Java libraries for writing a graph of objects into a stream of data. This stream of data can then be programmatically manipulated, and a deep copy of the objects can be made by reversing the process. This reversal is often called deserialization.

There are three main uses of serialization:

As a persistence mechanism

If the stream being used is file output stream, then the data will automatically be written to a file.

As a copy mechanism

If the stream being used is bytearrayoutputstream, then the data will be written to a byte array in memory. This byte array can then be used to create duplicates of the original objects.

As a communication mechanism

If the stream being used comes from a socket, then the data will automatically be sent over the wire to the receiving socket, at which point another program will decide what to do.

If we have a serializable class, we can save it to a file or make a copy of it simply by changing the way we use the output of the serialization mechanism.

To serialize an object, create an instance of objectoutputstream and call the writeobject( )method; to read in a serialized object, create an instance of objectinputstream and call the readobject( )object.

Serializing an object involves doing two things:

1. Creating an objectouptutstream and
2. Calling writeobject( )with a single "top-level" instance.

Fileoutputstream ser = new fileoutputstream ("C:\\temp\\test");  
objectoutputstream obj = new objectoutputstream(ser);  
obj.writeobject(serializableobject);

**1.What is the flaw with the Stack class?**   
**2.Explain the Java Class loader mechanism. How would you prevent someone from overriding the java.lang.Object class?   
3.Can interfaces contain inner classes?   
4.Explain how you would get a Thread Deadlock with a code example?   
5.What happens when you try to serialize a class whose base class does not implement serializable?**   
**6.What happens if you try to pass an int variable to a function that accepts only:a) long b) Double() c) Number() ?   
7.How can you prevent flow of control to a finally { } clause?**   
I write two classes : I write a static method in the base class and override that static method in the child class. Will this code compile?   
**8.Write a program that prints a.b.c.a.b.c...and so on 100 times using threads?   
9.What happens if you try to start a thread that’s already been started?**   
**How does garbage collection work?**  
  
What frameworks would you use to develop your dream application?

14) What is immutable object? How does it help on writing concurrent application?

15) What are some common problems you have faced in multi-threading environment? How did you resolve it?

Memory-interference, race conditions, [deadlock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html), live lock and starvation are example of some problems comes in multi-threading and concurrent programming. There is no end of problem if you get it wrong and they will be hard to detect and debug. This is mostly experienced based interview question on java thread instead of fact based.

How do you traverse through a collection using its Iterator?

To use an iterator to traverse through the contents of a collection, follow these steps:

* Obtain an iterator to the start of the collection by calling the collections iterator() method.
* Set up a loop that makes a call to hasnext(). Have the loop iterate as long as hasnext() returns true.
* Within the loop, obtain each element by calling next().

How to obtain Array from an arraylist ?

Array can be obtained from an arraylist using toarray() method on arraylist.

List arraylist = new arraylist();  
 arraylist.add(â);

Objectâ  a[] = arraylist.toarray();

**How do you decide when to use hashmap and when to use treemap ?**

For inserting, deleting, and locating elements in a Map, the hashmap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then treemap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a hashmap, then convert the map to a treemap for sorted key traversal.

Throw: Used to trigger an exception. The exception will be caught by the nearest try-catch clause that can catch that type of exception. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.

To throw a user-defined exception within a block, we use the throw command:

Throw new myexception("I always wanted to throw an exception!");

### What is the Java volatile keyword?

Essentially, volatile is used to indicate that a **variable's value will be modified by different** [threads](http://www.javamex.com/tutorials/threads/).

Declaring a volatile Java variable means:

* The value of this variable will **never be cached thread-locally**: all reads and writes will go straight to "main memory";
* Access to the variable **acts as though it is enclosed in a** [synchronized block](http://www.javamex.com/tutorials/synchronization_concurrency_synchronized1.shtml), synchronized on itself.

:: Volatile keyword in Java is used as an indicator to Java compiler and  [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html)that do not cache value of this variable and always read it from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html). So if you want to share any variable in which read and writes operation is atomic by implementation e.g. Read and write in int or Boolean variable you can declare them as volatile variable

A class can contain any of the following variable types.

Local variables. Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

Instance variables. Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that class.

Class variables. Class variables are variables declared with in a class, outside any method, with the static keyword.

JDBC connection steps:

* Loading Driver
* Establishing Connection
* Executing Statements
* Getting Results
* Closing Database Connection
* Class.forname(driver);  
    con = drivermanager.getconnection(url+db, user, pass);  
    Statement st = con.createstatement();  
    resultset res = st.executequery("SELECT \* FROM  employee");

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

* Information for the compiler — Annotations can be used by the compiler to detect errors or suppress warnings.
* Compile-time and deployment-time processing — Software tools can process annotation information to generate code, XML files, and so forth.
* Runtime processing — some annotations are available to be examined at runtime.

## Where Annotations Can Be Used

Annotations can be applied to declarations: declarations of classes, fields, methods, and other program elements. When used on a declaration, each annotation often appears, by convention, on its own line.

As of the Java SE 8 release, annotations can also be applied to the use of types. Here are some examples:

* Class instance creation expresson:
* New @Interned myobject();
* Type cast:
* Mystring = (@nonnull String) str;
* Implements clause:
* Class unmodifiablelist<T> implements
* @Readonly List<@Readonly T> { ... }
* Thrown exception declaration:
* Void monitortemperature() throws
* @Critical temperatureexception { ... }

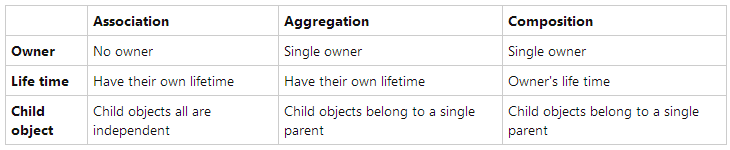
This form of annotation is called a type annotation.

## Maven's Objectives

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

* Making the build process easy
* Providing a uniform build system
* Providing quality project information
* Providing guidelines for best practices development
* Allowing transparent migration to new features

Java.lang.Object  
   |  
   +--java.lang.Throwable  
         |  
         +--java.lang.Exception  
         |     |  
         |     +--java.lang.classnotfoundexception  
         |     |  
         |     +--java.io.ioexception  
         |     |     |  
         |     |     +--java.io.filenotfoundexception  
         |     |  
         |     +--java.lang.runtimeexception  
         |           |  
         |           +--java.lang.nullpointerexception  
         |           |  
         |           +--java.lang.indexoutofboundsexception  
         |                 |  
         |                 +--java.lang.arrayindexoutofboundsexception  
         |  
         +--java.lang.Error  
               |  
               +--java.lang.virtualmachineerror  
                     |  
                     +--java.lang.outofmemoryerror



### When an Object becomes Eligible for Garbage Collection

An Object becomes eligible for Garbage collection or GC if its not reachable from any live threads or any static references in other words you can say that an object becomes eligible for garbage collection if its all references are null. Cyclic dependencies are not counted as reference so if Object A has reference of object B and object B has reference of Object A and they don't have any other live reference then both Objects A and B will be eligible for Garbage collection.   
Generally an object becomes eligible for garbage collection in Java on following cases:  
1) All references of that object explicitly set to null e.g. Object = null  
2) Object is created inside a block and reference goes out scope once control exit that block.  
3) Parent object set to null, if an object holds reference of another object and when you set container object's reference null, child or contained object automatically becomes eligible for garbage collection.  
4) If an object has only live references via weakhashmap it will be eligible for garbage collection.

Outofmemoryerror in Java is a subclass of java.lang.virtualmachineerror and JVM throws java.lang.outofmemoryerror when it ran out of memory in heap. Outofmemoryerror in Java can come any time in heap mostly while you try to create an object and there is not enough space in heap to allocate that object.  
  
"java.lang.outofmemoryerror: permgen" is memory leak through Class loaders and it’s very often surfaced in webserver and application server like tomcat, webshere, glassfish or weblogic. In Application server different class loaders are used to load different web application so that you can deploy and undeploy one application without affecting other application on same server, but while undeploying if container somehow keeps reference of any class loaded by application class loader than that class and all other related class will not be garbage collected and can quickly fill the permgen space if you deploy and undeploy your application many times Another reason of outofmemoryerror in permgen space is if any thread started by application doesn't exist when you undeploy your application.

Fastest Way of reading file use Mapped byte buffers is the fastest way:

Fileinputstream f = new fileinputstream( name );

Filechannel ch = f.getchannel( );

Mappedbytebuffer mb = ch.map( ch.mapmode.READ\_ONLY,

0L, ch.size( ) );

Byte[] barray = new byte[SIZE];

Long checksum = 0L;

Int nget;

While( mb.hasremaining( ) )

{

Nget = Math.min( mb.remaining( ), SIZE );

Mb.get( barray, 0, nget );

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

Checksum += barray[i];

}

**Q: Can we pass null in tree set...?** It will throw null pointer exception

**Q: Do we need to implement cloneable interface to call clone method?**

Yes, or else exception will be thrown. Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception clonenotsupportedexception being thrown.

**Q: why we need clone method. Couldn’t I create a constructor and just passing in the same object, and do the copying variable by variable?**

Like ...

Public class myobject{

Private int membervar;...

Myobject( myobject obj ){

This.membervar = obj.membervar;...}...}

The clone() method copies the whole object’s memory in one operation. This is much faster than using the new keyword. Object creation with the new keyword is expensive, so if you need to create lots of objects with the same type, performance will be better if you create one object and clone new ones from it.

Creating object receiving from a remote source

When an object is sent through a network, the object needs to be recreated at the receiving host.

**Object Serialization**

The term Object Serialization refers to the act of converting the object to a byte stream. The Byte stream can be stored on the file system, or can be sent through a network. At the later time the object can be re-created from that stream of bytes. The only requirement

Is that the same class must be available at both times, when the object is serialized and When the object is re-created. If that happens in different servers, then the same class must be

Available on both servers. Same class means that the same version of the class must be Available, otherwise the object won’t be able to be re-created. This is a maintenance problem to Those applications where java serialization is used to persist object or sent the object through

The network. When a class is modified, there could be a problem re-creating those objects that were serialized Using an earlier version of the class.

Normally, a minor change to the class can cause the serialization to fail. You can still allow the Class to be loaded by defining the serialization version id:

Private static final long serialversionuid = 42L;

Static variables can be used as data sharing amongst objects of the same class.

Any unreferenced objects are automatically erased from memory.

You can instantiate an abstract class in 2 ways:

Abstract class abstractclass {}

1. Extending the Abstract class:

Class concreteclass extends abstractclass {}

Abstractclass c = new concreteclass();

2. Using Anonymous Class:

Abstractclass c = new abstractclass(){};

Swipe two numbers without using 3rd var

A=7 b=5

A=a+b=12

B=a-b=12-5=7

A=a-b=12-7=5

**Q.** Can you write code that will check if a given string input is a numeric?  
**A.** You could use exception handling for this. If it throws an exception, it is not a numeric value.

|  |  |
| --- | --- |
| 2  3  4  5  6 | Public boolean isinteger( String input )  {     Try    {        Integer.parseint(input);        Return true;   }   Catch(Exception e)   {  Return false;  }  } |

Q. How can you drop a raw egg onto a concrete floor without cracking it?

 A. Concrete floors are very hard to crack!   
Q. If it took eight men ten hours to build a wall, how long would it take four men to build it?                                                                                          A. No time at all it is already built.   
Q.If you had three apples and four oranges in one hand and four apples and three oranges in the other hand, what would you have?   
A. Very large hands.  
Q. How can you lift an elephant with one hand?   
A. It is not a problem, since you will never find! An elephant with one hand.   
  
Q. How can a man go eight days without sleep?  
A. No Probs , He sleeps at night.   
  
Q. If you throw a red stone into the blue sea what it will become?  
A. It will become Wet or Sink as simple as that.  
  
Q. What looks like half apple ?  
A : The other half.   
  
Q. What can you never eat for breakfast ?  
A : Lunch and Dinner.  
  
Q. What happened when wheel was invented ?  
A : It caused a revolution.  
  
Q. Bay of Bengal is in which state?  
A : Liquid   
  
  
Question 1:  
You are driving along in your car on a wild, stormy night, it’s raining heavily, when suddenly you pass by a bus stop, and you see three people waiting for a bus:  
  
\* An old lady who looks as if she is about to die.  
\* An old friend who once saved your life.  
\* The perfect partner you have been dreaming about.  
  
Which one would you choose to offer a ride to, knowing very well that  
there could only be one passenger in your car? This is a moral/ethical dilemma that was once actually used as part of a job application.  
  
He simply answered:  
“I would give the car keys to my Old friend and let him take the lady to the hospital. I would stay behind and wait for the bus with the partner of my dreams.”  
Sometimes, we gain more if we are able to give up our stubborn thought limitations. Never forget to “Think Outside of the Box.”  
  
Question 2:   What will you do if I run away with your sister?                           The candidate who was selected answered ” I will not get a better match for my sister than you sir”  
  
Question 3:Interviewer (to a student girl candidate) – What is one morning you woke up & found that you were pregnant.                                               Girl - I will be very excited and take an off, to celebrate with my husband. Normally an unmarried girl will be shocked to hear this, but she managed it well. Why I should think it in the wrong way, she said later when asked.  
  
Question 4:  
Interviewer: He ordered a cup of coffee for the candidate. Coffee arrived kept before the candidate, and then he asked what is before you?  
Candidate: Instantly replied “Tea”  
He got selected.  
You know how and why did he say “TEA” when he knows very well that coffee was kept before.  
(Answer: The question was “What is before you (U – alphabet) Reply was “TEA” ( T – alphabet)  
Alphabet “T” was before Alphabet “U”  
  
Question6;  
Interviewer said “I shall either ask you ten easy questions or one  
really difficult question.  
  
Think well before you make up your mind!” The boy thought for a  
while and said, “my choice is one really difficult question.”  
  
“Well, good luck to you, you have made your own choice! Now tell me  
this. “What comes first, Day or Night?”  
  
The boy was jolted into reality as his admission depends on the  
correctness of his answer, but he thought for a while and  
said, “It’s the DAY sir!”  
  
“How” the interviewer asked,  
  
“Sorry sir, you promised me that you will not ask me a SECOND  
difficult question!”

## What is a Semaphore

Semaphore is a technique used to control access to common resource for competing multiple processes. Semaphore maintains a counter which keeps track of the number of resources available. When a process requests access to resource, semaphore checks the variable count and if it is less than total count then grants access and subsequently reduces the available count.

Semaphore is just a gatekeeper guarding the resources. If available grants access and otherwise asks the processes to wait.

* When the resource count is arbitrary then this is called counting semaphore.
* If resource count is only one and the state value is restricted to on/off, then it is called binary semaphore. This is slightly different from mutex.

**What is Thread Factory?**

Public interface **thread factory**

An object that creates new threads on demand. Using thread factories removes hardwiring of calls to [new Thread](http://docs.oracle.com/javase/6/docs/api/java/lang/Thread.html#Thread(java.lang.Runnable)), enabling applications to use special thread subclasses, priorities, etc.

The simplest implementation of this interface is just:

Class simplethreadfactory implements threadfactory {

Public Thread newthread(Runnable r) {

Return new Thread(r);

} }

While static methods lock on the class object of the class, non-static methods lock on the instance on which they're called (by default, synchronized(anyotherlock) is also possible). Since they lock on different objects, they can run in "parallel".

What is Race Condition in multithreading?

## *Race conditions occur when two threads operate on same object without proper synchronization and there operation interleaves on each other. Example of Race condition is incrementing a counter since increment is not an atomic operation and can be further divided into three steps like read, update and write. If two threads tries to increment count at same time and if they read same value because of interleaving of read operation of one thread to update operation of another thread, one count will be lost when one thread overwrite increment done by other thread. Atomic operations are not subject to race conditions because those operations cannot be interleaved.*

**What is the use of concurrent hash map in java? What are the benefits of it? How does it work?**

1)The point is to provide an implementation of hashmap that is thread safe. Multiple threads can read from and write to it without the chance of receiving out-of-date or corrupted data. ConcurrentHashMap provides its own synchronization, so you do not have to synchronize accesses to it explicitly.

2)Another feature of ConcurrentHashMap is that it provides the putifabsent method, which will atomically add a mapping if the specified key does not exist.

3)ConcurrentHashMap only locked certain portion of Map while hashmap lock full map while doing iteration. ConcurrentHashMap do not allow null keys or null values while hashmap allows null keys.

## *Consider the following code:*

## *Concurrenthashmap<String, Integer> mymap = new concurrenthashmap<String, Integer>();*

## *// some stuff*

## *If (!Mymap.contains("key")) { Mymap.put("key", 3);}*

## *This code is not thread safe, because another thread could add a mapping for "key" between the call to contain and the call to put. The correct implementation would be: Mymap.putifabsent("key", 3);*

## How ConcurrentHashMap is implemented in Java

ConcurrentHashMap is introduced as an alternative of hashmap and provided all functions supported by hashmap with additional feature called "concurrency level", which allows ConcurrentHashMap to partition Map. Concurrenthashmap allows multiple readers to read concurrently without any [blocking](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html). This is achieved by partitioning Map into different parts based on concurrency level and locking only a portion of Map during updates. Default concurrency level is 16, and accordingly Map is divided into 16 part and each part is governed with different lock. This means, 16 threads can operate on Map simultaneously, until they are operating on different part of Map. This makes ConcurrentHashMap high performance despite keeping thread-safety intact.  Though, it comes with caveat. Since update operations like put(), remove(), putall() or clear() is not synchronized, **concurrent retrieval may not reflect most recent change on Map**.

In case of putall() or clear(), which operates on whole Map, concurrent read may reflect insertion and removal of only some entries. Another important point to remember is iteration over CHM, [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) returned by keyset of ConcurrentHashMap are weekly consistent and they only reflect state of ConcurrentHashMap and certain point and may not reflect any recent change. Iterator of ConcurrentHashMap keyset area also [fail-safe](http://javarevisited.blogspot.in/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and doesn’t throw concurrentmodificationexception.

Default concurrency level is 16 and can be changed, by providing a number which make sense and work for you while creating ConcurrentHashMap. Since concurrency level is used for internal sizing and indicate number of concurrent update without contention, so, if you just have few writers or thread to update Map keeping it low is much better. ConcurrentHashMap also uses Re-entrant Lock to internally lock its segments.

## ConcurrentHashMap putifabsent example in Java

Concurrenthashmap examples are similar to [Hashtable examples](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html), we have seen earlier, but worth knowing is use of putifabsent() method. Many times we need to insert entry into Map, if its not present already, and we wrote following kind of code:

Synchronized (map){

**If** (map**.**get(key) **==** *null*){

**Return** map**.**put(key, value);

  } **else**{

**Return** map**.**get(key);

  }

}

Though this code will work fine in [hashmap and Hashtable](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html), This won't work in ConcurrentHashMap; because, during put operation whole map is not locked, and while one thread is putting value, other thread's get() call can still return null which result in one thread overriding value inserted by other thread. Of course, you can wrap whole code in [synchronized block](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html) and make it [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) but that will only make your code single threaded. Concurrenthashmap provides putifabsent(key, value) which does same thing but atomically and thus eliminates above race condition.

## When to use concurrenthashmap in Java

Concurrenthashmap is best suited when you have multiple readers and few writers. If writers outnumber reader, or writer is equal to reader, than performance of concurrenthashmap effectively reduces to [synchronized map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) or [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html). Performance of CHM drops, because you got to lock all portion of Map, and effectively each reader will wait for another writer, operating on that portion of Map. Concurrenthashmap is a good choice for caches, which can be initialized during application start up and later accessed my many request processing threads. As javadoc states, CHM is also a [good replacement of Hashtable](http://javarevisited.blogspot.sg/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html) and should be used whenever possible, keeping in mind, that CHM provides slightly weeker form of synchronization than Hashtable.

### Summary

Now we know What is concurrenthashmap in Java and when to use concurrenthashmap, it’s time to know and revise some important points about CHM in Java.

1. Concurrenthashmap allows concurrent read and thread-safe update operation.

2. During update operation, concurrenthashmap only lock a portion of Map instead of whole Map.

3. Concurrent update is achieved by internally dividing Map into small portion which is defined by concurrency level.

4. Choose concurrency level carefully as a significant higher number can be waste of time and space and lower number may introduce thread contention in case writers overnumber concurrency level.

5. All operations of concurrenthashmap are [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html).

6. Since concurrenthashmap implementation doesn't lock whole Map, there is chance of read overlapping with update operations like put() and remove(). In that case result returned by get() method will reflect most recently completed operation from there start.

7. Iterator returned by concurrenthashmap is weekly consistent, [fail safe](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and never throw concurrentmodificationexception. In Java.

8. Concurrenthashmap doesn't allow null as key or value.

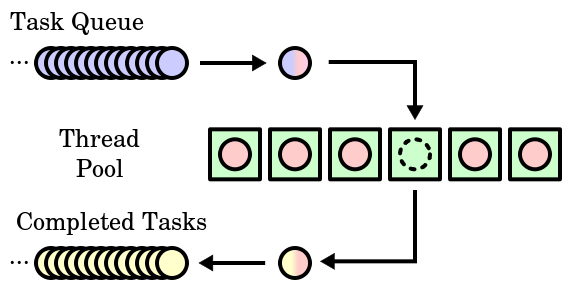
9. You can use concurrenthashmap in place of [Hashtable](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) but with caution as CHM doesn't lock whole Map.

10. During putall() and clear() operations, concurrent read may only reflect insertion or deletion of some entries.

## [What is the use of a Thread pool in Java?](http://stackoverflow.com/questions/3286626/what-is-the-use-of-a-thread-pool-in-java)

The **thread pool**[**pattern**](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)) (also **replicated workers** or **worker-crew model**) is where a number of [threads](http://en.wikipedia.org/wiki/Thread_(computer_science)) are created to perform a number of [tasks](http://en.wikipedia.org/wiki/Task_(computers)), which are usually organized in a [queue](http://en.wikipedia.org/wiki/Queue_(data_structure)).

A thread pool is a group of threads initially created that waits for jobs and executes them. The idea is to have the threads always existing, so that we won't have to pay overhead time for creating them every time. They are appropriate when we know there's a stream of jobs to process, even though there could be some time when there are no jobs.



A sample thread pool (green boxes) with waiting tasks (blue) and completed tasks (yellow)

A thread pool manages the pool of worker threads; it contains a queue that keeps tasks waiting to get executed. A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.**java.util.concurrent.Executors** provide implementationof **java.util.concurrent.Executor** interface to create the thread pool in java.

|  |  |  |
| --- | --- | --- |
|  | | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Public class workerthread implements Runnable { | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Private String command; | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Public workerthread(String s){ | |
|  | This.command=s; |

|  |  |  |
| --- | --- | --- |
|  | } | |
|  |  |

|  |  |
| --- | --- |
|  | @Override |
|  | Public void run() { | |

|  |  |  |
| --- | --- | --- |
|  | System.out.println(Thread.currentthread().getname()+' Start. Command = '+command); | |
|  | Processcommand(); |

|  |  |  |
| --- | --- | --- |
|  | System.out.println(Thread.currentthread().getname()+' End.'); | |
|  | } |

|  |  |
| --- | --- |
|  |  |
|  | Private void processcommand() { | |

|  |  |
| --- | --- |
|  | Try { |
|  | Thread.sleep(5000); | |

|  |  |  |
| --- | --- | --- |
|  | } catch (interruptedexception e) { | |
|  | E.printstacktrace(); |

|  |  |  |
| --- | --- | --- |
|  | } | |
|  | } |

|  |  |
| --- | --- |
|  |  |
|  | @Override | |

|  |  |  |
| --- | --- | --- |
|  | Public String tostring(){ | |
|  | Return this.command; |

|  |  |  |
| --- | --- | --- |
|  | } | |
|  | } |

Here is the test program where we are creating fixed thread pool from Executors framework.

|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Import java.util.concurrent.executorservice; | |
|  | Import java.util.concurrent.Executors; |

|  |  |
| --- | --- |
|  |  |
|  | Public class simplethreadpool { | |

|  |  |
| --- | --- |
|  |  |
|  | Public static void main(String[] args) { | |

|  |  |  |
| --- | --- | --- |
|  | Executorservice executor = Executors.newfixedthreadpool(5); | |
|  | For (int i = 0; i < 10; i++) { |

|  |  |  |
| --- | --- | --- |
|  | Runnable worker = new workerthread('' + i); | |
|  | Executor.execute(worker); |

|  |  |
| --- | --- |
|  | } |
|  | Executor.shutdown(); | |

|  |  |  |
| --- | --- | --- |
|  | While (!Executor.isterminated()) { | |
|  | } |

|  |  |  |
| --- | --- | --- |
|  | System.out.println('Finished all threads'); | |
|  | } |

|  |  |
| --- | --- |
|  |  |
|  | } | |

In above program, we are creating fixed size thread pool of 5 worker threads. Then we are submitting 10 jobs to this pool, since the pool size is 5, it will start working on 5 jobs and other jobs will be in wait state, as soon as one of the job is finished, another job from the wait queue will be picked up by worker thread and get’s executed.

**Executors** class provide simple implementation of **executorservice** using **threadpoolexecutor** but threadpoolexecutor provides much more feature than that. We can specify the number of threads that will be alive when we create threadpoolexecutor instance and we can limit the size of thread pool and create our own **rejectedexecutionhandler** implementation to handle the jobs that can’t fit in the worker queue.

The algorithm used to determine when to create or destroy threads will have an impact on the overall performance:

1) Create too many threads, and resources are wasted and time also wasted creating any unused threads

2) Destroy too many threads and more time will be spent later creating them again

3) Creating threads too slowly might result in poor client performance (long wait times)

4) Destroying threads too slowly may starve other processes of resources

Another advantage of using a thread pool over creating a new thread for each task is thread creation and destruction overhead is negated, which may result in better [performance](http://en.wikipedia.org/wiki/Performance_tuning) and better system [stability](http://en.wikipedia.org/wiki/Stability_Model). Creating and destroying a thread and its associated resources is an expensive process in terms of time. An excessive number of threads will also waste memory, and context-switching between the runnable threads also damages performance

A thread pool implementation is provided in the form of the [threadpoolexecutor](http://www.javamex.com/tutorials/threads/ThreadPoolExecutor.shtml) class, part of the Java concurrency framework introduced in Java 5;

Q: [How can I check if a thread is inside a synchronized block or method?](http://stackoverflow.com/questions/5084793/how-can-i-check-if-a-thread-is-inside-a-synchronized-block-or-method)

Ans: use Thread.holdslock(...) To check if the thread holds a specific lock.

Q: If two threads hit 2 different synchronized methods in an object at the same time will they both continue?

No. Only one method can acquire a lock.

[**Create threads in java to run in background**](http://stackoverflow.com/questions/12551514/create-threads-in-java-to-run-in-background)

Executorservice executor = Executors.newcachedthreadpool();

**Thread dump in java**

A thread dump is a list of all the Java threads that are currently active in a Java Virtual Machine (JVM).

A stack trace is a dump of the current execution stack that shows the method calls running on that thread from the bottom up

Ps -el | grep java

How do you create a thread dump?

For the creating of a thread dump there is no uniform solution. It depends on the environment.

Under Windows: type ctrl-break in the black Java console. The Java console is the black DOS box that open on starting. A Java console open only if you use java.exe and not if you use javaw.exe.

Under Unix: call kill -3 <java\_process\_id> (e.g. Kill -3 5555). This will NOT kill your application. It will only write a stack trace to standard output.

Under Weblogic: Where you go to find the thread dump usually depends on Java implementation. For most vendors, you'll go to the "standard out" log file. In weblogic, it's often referred to as "weblogic.out", "nohup.out" or something you've created yourself by redirecting standard output to a file

**How can I create a daemon thread?**

Setdeamon(true)

Setdaemon(true/false) ·/b> This method is used to specify that a thread is daemon thread.

Public boolean isdaemon() · This method is used to determine the thread is daemon thread or not.

The following program demonstrates the Daemon Thread:

Note: Text content in the code blocks is automatically word-wrapped

Public class daemonthread extends Thread {

Public void run() {

System.out.println("Entering run method");

Try {

System.out.println("In run Method: currentthread() is"

+ Thread.currentthread());

While (true) {

Try {

Thread.sleep(500);

} catch (interruptedexception x) {

}

System.out.println("In run method: woke up again");

}

} finally {

System.out.println("Leaving run Method");

} }

Public static void main(String[] args) {

System.out.println("Entering main Method");

Daemonthread t = new daemonthread();

T.setdaemon(true);

T.start();

Try {

Thread.sleep(3000);

} catch (interruptedexception x) {

}

System.out.println("Leaving main method");

} }

"What is deadlock ?"  
**Answer:** when two or more threads waiting for each other to release lock and get stuck for infinite time, situation is called deadlock. It will only happen in case of multitasking.  
  
How do you detect deadlock in Java?  
Answer: Nested synchronized block or calling one synchronized method from other or trying to get lock on different object then there is good chance of deadlock if developer is not very careful.  
  
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.  
  
Other way is to use jconsole , jconsole will show you exactly which threads are get locked and on which object.  
  
Write code which will result in deadlock ?  
  
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");  
}  
}  
}  
  
public void method2(){  
synchronized(Integer.class){  
System.out.println("Aquired lock on Integer.class object");  
  
synchronized (String.class) {  
System.out.println("Aquired lock on String.class object");  
}  
}  
}  
  
If method1() and method2() both will be called by two or many threads , there is a good chance of deadlock because if thead 1 aquires lock on Sting object while executing method1() and thread 2 acquires lock on Integer object while executing method2() both will be waiting for each other to release lock on Integer and String to proceed further which will never happen.  
  
How to fix deadlock ? Or How to avoid deadlock in Java ?  
  
**Answer:** reason for deadlock is not multiple threads but the way they access lock , if you provide an ordered access then problem will be resolved , here is the fixed version.  
Public void method1(){  
synchronized(Integer.class){  
System.out.println("Aquired lock on Integer.class object");  
  
synchronized (String.class) {  
System.out.println("Aquired lock on String.class object");  
}  
}  
}  
  
public void method2(){  
synchronized(Integer.class){  
System.out.println("Aquired lock on Integer.class object");  
  
synchronized (String.class) {  
System.out.println("Aquired lock on String.class object");  
}  
}  
}  
  
   
  
Now there would not be any deadlock because both method is accessing lock on Integer and String object in same order . So if thread A acquires lock on Integer object , thread B will not proceed until thread A releases Integer lock , same way thread A will not be blocked even if thread B holds String lock because now thread B will not expect thread A to release Integer lock to proceed further.

Difference between wait and sleep.

A wait can be "woken up" by another process calling notify on the monitor which is being waited on whereas a sleep cannot. Also a wait (and notify) must happen in a block synchronized on the monitor object whereas sleep does not:

Synchronized(LOCK) {

Thread.sleep(1000); // LOCK is held

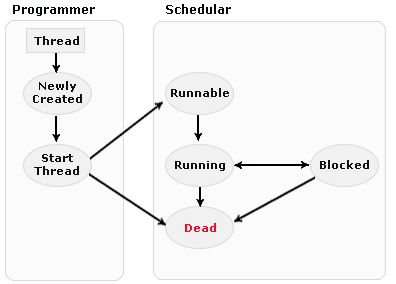
}

Synchronized(LOCK) {

LOCK.wait(); // LOCK is not held

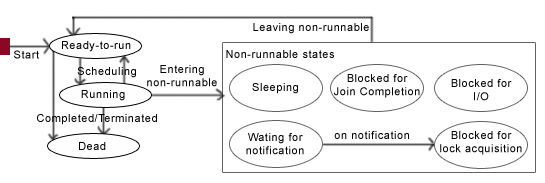
}

**Different states of a thread are :**



1. **New state –** After the creations of Thread instance the thread is in this state but before the start() method invocation. At this point, the thread is considered not alive.
2. **Runnable (Ready-to-run) state –** A thread start its life from Runnable state. A thread first enters runnable state after the invoking of start() method but a thread can return to this state after either running, waiting, sleeping or coming back from blocked state also. On this state a thread is waiting for a turn on the [processor](http://www.roseindia.net/java/thread/life-cycle-of-threads.shtml).
3. **Running state –** A thread is in running state that means the thread is currently executing. There are several ways to enter in Runnable state but there is only one way to enter in Running state: the scheduler select a thread from runnable pool.
4. **Dead state –** A thread can be considered dead when its run() method completes. If any thread comes on this state that means it cannot ever run again.
5. **Blocked -** A thread can enter in this state because of waiting the resources that are hold by another thread.

**Different states implementing Multiple-Threads are:**



As we have seen different states that may be occur with the single thread. A running thread can enter to any non-runnable state, depending on the circumstances. A thread cannot enters directly to the running state from non-runnable state, firstly it goes to runnable state. Now lets understand the some non-runnable states which may be occur handling the multithreads.

* **Sleeping –** On this state, the thread is still alive but it is not runnable, it might be return to runnable state later, if a particular event occurs. On this state a thread sleeps for a specified amount of time. You can use the method **sleep( )** to stop the running state of a thread.  
          
       **Static void sleep(long millisecond) throws interruptedexception**
* **Waiting for Notification –** A thread waits for notification from another thread. The thread sends back to runnable state after sending notification from another thread.  
           
       **Final void wait(long timeout) throws interruptedexception  
       final void wait(long timeout, int nanos) throws interruptedexception  
       final void wait() throws interruptedexception**
* **Blocked on I/O –** The thread waits for completion of blocking operation. A thread can enter on this state because of waiting I/O resource. In that case the thread sends back to runnable state after availability of resources.
* **Blocked for joint completion –** The thread can come on this state because of waiting the completion of another thread.
* **Blocked for lock acquisition –** The thread can come on this state because of waiting to acquire the lock of an object.

## Why do we have wait,notify and notifyall in object class and not in thread class?

**1) You have thread T1, T2 and T3, how will you ensure that thread T2 run after T1 and thread T3 run after T2?**

This thread interview questions is mostly asked in first round or phone screening round of interview and purpose of this multi-threading question is to check whether candidate is familiar with concept of *"join"* method or not. Answer of this multi-threading questions is simple it can be achieved by using **join** method of Thread class.

**What use of connection pooling?**

* Whenever our web app needs an access to database it connects to the database. It will be disadvantageous if it connect every time a user sends a request for a particular data.**so the server maintains a pool of connection instances. So if a request comes it will take a instance from the pool and serve.** Because the database connection is tedious task so it is better to maintain a pool of connection instances. It makes the access faster.
* Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the Web to a resource, the resource accesses a data source. Most user requests do not incur the overhead of creating a new connection because the datasource may locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the resource returns the connection to the connection pool for reuse. Again, the overhead of a disconnect is avoided. Each user request incurs a fraction of the cost of connection or disconnecting. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant because the existing connections are reused

**What is the advantage of new Lock interface over synchronized block in Java?**

The major advantage of lock interfaces on multi-threaded and concurrent programming is they provide two separate lock for reading and writing which enables you to write high performance data structure like [concurrenthashmap](http://javarevisited.posterous.com/difference-between-concurrenthashmap-and-coll) and [conditional blocking](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html). I would strongly suggest reading **Locks** before appearing for any *java multi-threading interview* because now days Its heavily used to build cache for electronic trading system on client and exchange connectivity space.

**4) Write code to implement blocking queue in Java?**

If he uses [wait() and notify() method](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html) to implement blocking queue, Once interviewee successfully writes it you can ask him to write it again using new java 5 concurrent classes etc.

**5) Write code to solve the Produce consumer problem in Java?**

**6) Write a program which will result in deadlock? How will you fix deadlock in Java?**

**7) What is atomic operation? What are atomic operations in Java?**

**8) What is volatile keyword in Java? How to use it? How is it different from synchronized method in Java?**

**9) What is race condition? How will you find and solve race condition?**

**10) How will you take thread dump in Java? How will you analyze Thread dump?**

In UNIX you can use **kill -3** and then thread dump will print on log on windows you can use **"CTRL+Break".** Rather simple and focus thread interview question but can get tricky if he ask how you analyze it. Thread dump can be useful to analyze deadlock situations as well.

**11) Why we call start() method which in turns calls run() method, why not we directly call run() method ?**

When you call start() method it creates new Thread and execute code declared in run() while directly calling run() method doesn’t create any new thread and execute code on same calling thread.

**12) How will you awake a blocked thread in java?**

If thread is blocked due to result of calling wait(), sleep() or join() method you can interrupt the thread and it will awake by throwing interruptedexception.

**13) What is difference between cyclicbarriar and countdownlatch in Java ?**

New java thread interview questions mostly to check familiarity with JDK 5 concurrent packages. One difference is that you can reuse cyclicbarrier once barrier is broken but you can not reuse contdownlatch.

[Difference between Runnable and Thread in java](http://javarevisited.blogspot.com/2012/01/difference-thread-vs-runnable-interface.html)

[How to Stop Thread in Java](http://javarevisited.blogspot.com/2011/10/how-to-stop-thread-java-example.html)

[Why wait and notify methods are declared in Object Class?](http://javarevisited.blogspot.com/2012/02/why-wait-notify-and-notifyall-is.html)

[How to Solve Producer Consumer Problem in Java](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html)

[Why wait and notify needs to called from Synchronized Context?](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html)

[Difference between concurrenthashmap and Hashtable in Java](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html)

Since **concurrenthashmap introduced concept of segmentation** , how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.

In Summary **concurrenthashmap only locked certain portion of Map** while Hashtable lock full map while doing iteration.

[Difference between invokeandwait and invokelater in java Swing.](http://javarevisited.blogspot.com/2011/09/invokeandwait-invokelater-swing-example.html)

[Difference between wait and sleep in Java](http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html)

How can I ensure thread safety using array list.

By using copyonwritearraylist  
  
<http://javapapers.com/spring/dependency-injection-di-with-spring/>

What is re-entrant?

In [computing](http://en.wikipedia.org/wiki/Computing), a [computer program](http://en.wikipedia.org/wiki/Computer_program) or [subroutine](http://en.wikipedia.org/wiki/Subroutine) is called **reentrant** if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution. The interruption could be caused by an internal action such as a jump or call, or by an external action such as a [hardware interrupt](http://en.wikipedia.org/wiki/Hardware_interrupt) or [signal](http://en.wikipedia.org/wiki/Signal_(computing)). Once the reentered invocation completes, the previous invocations will resume correct execution.

What is **Semaphores?**

**Semaphores** are often used to restrict the number of threads than can access some (physical or logical) resource.

 Semaphores are devices used to help with synchronization. If multiple processes share a common resource, they need a way to be able to use that resource without disrupting each other. You want each process to be able to read from and write to that resource uninterrupted.

A semaphore will either **allow or disallow access to the resource**, depending on how it is set up. One example setup would be a semaphore which allowed any number of processes to read from the resource, but only one could ever be in the process of writing to that resource at a time. Semaphores are commonly use for two purposes: to **share a common** [memory](http://ecomputernotes.com/fundamental/input-output-and-memory/what-are-the-different-types-of-ram-explain-in-detail) **space and to share access to files**. Semaphores are one of the techniques for interprocess communication (IPC).

**What do you mean by thread safe class?**

There are three ways to construct thread-safe Java class which has some state:

1) Make it truly immutable

2) Make field number volatile.

3) Use a synchronized block

**Volatile:**

1. *Volatile keyword in Java is only application to variable* and using volatile keyword with class and method is illegal.

2. Volatile keyword in Java guarantees that value of **volatile variable** will always be read from main memory and not from Thread's local cache.

3. In Java reads and writes are [atomic](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) for all variables declared using **Java volatile keyword** (including long and double variables).

4. Using Volatile keyword in Java on variables reduces the risk of memory consistency errors, because any write to a volatile variable in Java establishes a happens-before relationship with subsequent reads of that same variable.

5. From Java 5 changes to a volatile variable are always visible to other threads. What’s more it also means that when a thread reads a volatile variable in java, it sees not just the latest change to the volatile variable but also the side effects of the code that led up the change.

6. Reads and writes are atomic for reference variables are for most primitive variables (all types except long and double) even without use of volatile keyword in Java.

7. An access to a volatile variable in Java never has chance to block, since we are only doing a simple read or write, so unlike a synchronized block we will never hold on to any lock or wait for any [lock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html).

8. Java volatile variable that is an object reference may be null.

**Difference between synchronized and volatile keyword in Java**

Remember volatile is not a replacement of synchronized keyword but can be used as an alternative in certain cases.

1. Volatile keyword in java is a field modifier, while [synchronized modifies code blocks and methods](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html).

2. Synchronized obtains and releases lock on monitor’s java volatile keyword doesn't require that.

3. Threads in Java can be blocked for waiting any monitor in case of synchronized, that is not the case with volatile  in Java.

4. [Synchronized method affects performance](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) more than volatile in Java.

5. Since volatile  in Java only synchronizes the value of one variable between Thread memory  and "main" memory  while synchronized synchronizes the value of all variable between thread memory and "main" memory and locks and releases a monitor to boot. Due to this reason [synchronized keyword in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) is likely to have more overhead than volatile.

6. We can’t synchronize on null object but your volatile variable in java could be null.

7. From Java 5 Writing into a volatile field has the same memory effect as a monitor release, and reading from a volatile field has the same memory effect as a monitor acquire

Public class mysingleton implements Serializable{  
 private static mysingleton myinstance;

 Private mysingleton(){

 }  
  static{  
    myinstance =new mysingleton();  
 }  
 public static mysingleton getinstance(){  
    return mysingleton.myinstance;  
 }  
}

If another class X get the instance of the single and writes it to a file and at a later point deserializes it to obtain another instance we would have two instances which is against the Singleton principle.  
-----------------

The best way to do this is to use the enum singleton pattern:

Public enum mysingleton {  
  INSTANCE;  
}  
This guarantees the singleton-ness of the object and provides serializability for you in such a way that you always get the same instance.

More generally, you can provide a readresolve() method like so:

Protected Object readresolve() {  
  return myinstance;  
}

The rules for serializing an enum instance differ from those for serializing an "ordinary" serializable object: the serialized form of an enum instance consists only of its enum constant name, along with information identifying its base enum type. Deserialization behavior differs as well--the class information is used to find the appropriate enum class, and the Enum.valueof method is called with that class and the received constant name in order to obtain the enum constant to return.

Suppose that your application has instantiated the singleton in the normal course of events, and then it deserializes some object graph that includes a copy of a previous instance of the singleton. What can it do?

* If it deserializes the singleton normally, it violates "singleton-ness".
* If it doesn't then the application cannot access the singleton's previous state.

The solution with enum won't work with Singletons managed by Spring, EJB, Guice or any other DI framework. It works only with enums, only because enum is treated specially by the serialization algorithm.

Firstly, *singletons don't need serialization*, because if you deserialized it, and then deserialized singleton != yoursingleton.getinstance(), it would mean that you have two instances of your singleton, which means that yoursingleton isn't singleton at all, which may lead to unpredictable bugs.

However sometimes you need to serialize non-singleton which contains a reference to singleton. The solution is easy:

Class nonsingleton implements Serializable {  
    private transient yoursingleton singleton = yoursingleton.getinstance();  
    ...  
}

With ng:

@Configurable  
class nonsingleton implements Serializable {  
    @Autowired  
    private transient yoursingleton singleton;  
    ...  
}

**What is Serialization in Java**

[](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html)

Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running [Java virtual machine](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html); the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable,java.io.Externalizable, objectinputstream and objectoutputstream etc. Java programmers are free to use default Serialization mechanism which Java uses based upon structure of class but they are also free to use there own custom binary format, which is often advised as Serialization best practice, Because serialized binary format becomes part of Class's exported API and it can potentially break [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) provided by private and [package-private fields](http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html). This pretty much answer the question *What is Serialization in Java*.

**How to make a Java class Serializable?**

Making a class Serializable in Java is very easy, Your Java class just needs to implements java.io.Serializable interface and JVM will take care of serializing object in default format. Decision to making a [Class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) Serializable should be taken concisely because though near term cost of making a Class Serializable is low, long term cost is substantial and it can potentially limit your ability to further modify and change its implementation because like any public API, serialized form of an object becomes part of public API and when you change structure of your class by implementing addition interface, adding or removing any field can potentially break default serialization, this can be minimized by using a custom binary format but still requires lot of effort to ensure backward compatibility. One example of How Serialization can put constraints on your ability to change class is serialversionuid. If you don't explicitly declare serialversionuid then JVM generates its based upon structure of class which depends upon interfaces a class implements and several other factors which is subject to change. Suppose you implement another interface than [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) will generate a different serialversionuid for new version of class files and when you try to load old object object serialized by old version of your program you will get invalidclassexception.

**1) What is the difference between Serializable and Externalizable interface in Java?**

This is most frequently asked question in Java serialization interview. Here is my version Externalizable provides us writeexternal() and readexternal() method which gives us flexibility to control java serialization mechanism instead of relying on Java's default serialization. Correct implementation of Externalizable interface can [improve performance of application](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) drastically.

**2) How many methods Serializable has? If no method then what is the purpose of Serializable interface?**

Serializable interface exists in java.io  package and forms core of java serialization mechanism. It doesn't have any method and also called [Marker Interface in Java](http://javarevisited.blogspot.sg/2012/01/what-is-marker-interfaces-in-java-and.html). When your class implements java.io.Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

**3) What is serialversionuid? What would happen if you don't define this?**

One of my favorite question interview question on Java serialization. Serialversionuid is an ID which is stamped on object when it get serialized usually [hashcode of object](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html), you can use tool *serialver*to see serialversionuidof a serialized object . Serialversionuid is used for version control of object. You can specify serialversionuidin your [class file](http://javarevisited.blogspot.sg/2012/05/10-points-about-class-file-in-java.html) also.  Consequence of not specifying  serialversionuid is that when you add or modify any field in class then already serialized class will not be able to recover because serialversionuid generated for new class and for **old serialized object** will be different. Java serialization process relies on correct **serialversionuid**for recovering state of serialized object and throws java.io.invalidclassexception in case of serialversionuid mismatch.

**4) While serializing you want some of the members not to serialize? How do you achieve it?**

Another frequently asked Serialization interview question. This is sometime also asked as what is the use of [transient variable](http://javarevisited.blogspot.sg/2011/09/transient-keyword-variable-in-java.html), does transient*and*[*static variable*](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) gets serialized or not etc. So if you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

**5) What will happen if one of the members in the class doesn't implement Serializable interface?**

One of the easy question about Serialization process in Java. If you try to *serialize an object* of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a **‘**notserializableexception’ will be thrown at [runtime](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) and this is why I always put a serializablealert (comment section in my code) , one of the [code comment best practices](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html), to instruct developer to remember this fact while adding a new field in a Serializable class.

**6) If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?**

Java serialization process  only continues in object hierarchy till the class is Serializable i.e. Implements Serializable [interface in Java](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html)  and values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during *deserialization* process. Once the [constructor chaining](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) will started it wouldn't be possible to stop that , hence even if classes higher in hierarchy implements Serializable interface , there constructor will be executed. As you see from the statement this Serialization interview question looks very tricky and tough but if you are familiar with key concepts its not that difficult.

**7) Can you Customize Serialization process or can you override default Serialization process in Java?**

The answer is yes you can. We all know that for serializing an objectobjectoutputstream.writeobject*(savethisobject) is invoked*and for reading object objectinputstream.readobject() is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization and deserialization here by doing any kind of pre or post processing task. Important point to note is making these methods [private](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html) to avoid being inherited, [overridden or overloaded](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Since only Java Virtual Machine can call private method integrity of your class will remain and **Java Serialization** will work as normal. In my opinion this is one of the best question one can ask in any Java Serialization interview, a good follow-up question is why should you provide custom serialized form for your object?

**8) Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?**

One of the tricky interview question in Serialization in Java. If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you can not unimplemented an interface its not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. **To avoid java serialization** you need to implement writeobject() andreadobject() method in your Class and need to throw notserializableexception from those method. This is another benefit of *customizing java serialization* process as described in above .

**9) Which methods are used during Serialization and deserialization process in java?**

This is very common interview question in Serialization  basically interviewer is trying to know; Whether you are familiar with usage of readobject(), writeobject(), readexternal() and writeexternal () or not. Java Serialization is done by java.io.objectoutputstream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call objectoutputstream.writeobject(savethisobject) and to deserialize that object we call objectinputstream.readobject() method. Call to writeobject() method trigger serialization process in java. One important thing to note about readobject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an [Object](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) which needs to be casted on correct type.

**10) Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?**

It depends on whether class has its own [serialversionuid](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) or not. As we know from above question that if we don't provide serialversionuid in our code java compiler will generate it and normally it’s [equal to hashcode of object](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html). By adding any new field there is chance that new serialversionuid generated for that class version is not the same of already serialized object and in this case Java Serialization API will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html)java.io.invalidclassexception and this is the reason *its recommended to have your own serialversionuid* in code and make sure to keep it same always for a single class.

**11) What are the compatible changes and incompatible changes in Java Serialization Mechanism?**

The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object***.****As per Java Serialization specification adding any field or method comes under compatible change*and changing class hierarchy or UN-implementing Serializable interfaces some under **non compatible changes**. For complete list of compatible and non compatible changes I would advise reading Java serialization specification.

**12) Can we transfer a Serialized object vie network?**

*Yes you can transfer a Serialized object via network* because java serialized object remains in form of bytes which can be transmitter via network. You can also store serialized object in Disk or database as Blob.

**13) Which kind of variables is not serialized during Java Serialization?**

This question asked sometime differently but the purpose is same whether Java developer knows specifics about [***static and transient variable***](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) or not. Since *static variables belong to the class* and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object’s serialized state. After this question sometime interviewer ask a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those object?

**Difference between inheritance and composition**

**Class child extends parent extends denote inheritance.**

**Parent parent denotes composition**

**Why we can’t create object of abstract class?**

Since abstract class can have two types of method implemented and non-implemented method without any body. So in case if allow to create object of abstract class then calling non implemented methods through class object will throw an error because it’s not yet implemented. That’s why object creation defers to subclass which extends it.

And more over objects have states and behaviours. Example: A dog has states-colour, name, and breed as well as behaviours -wagging, barking, and eating. An object is an instance of a class.

**Oops Concepts**

A) **Association** is a relationship where all object have their own lifecycle and there is no owner. Ex. Multiple students can associate with a single teacher and single student can associate with multiple teachers, but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

B) **Aggregation** is a specialized form of Association where all object have their own lifecycle but there is ownership and child object can not belong to another parent object. Let’s take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department, the teacher object will not be destroyed. We can think about “has-a” relationship

Difference between Interface and Abstract Class

1. Main difference is methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implement default behaviour.

2. Variables declared in a Java interface is by default final. An abstract class may contain non-final variables.

3. Members of a Java interface are public by default. A Java abstract class can have the usual flavours of class members like private, protected, etc.

4. Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”.

5. An interface can extend another Java interface, an abstract class can extend another Java class and implement multiple Java interfaces.

6. A Java class can implement multiple interfaces but it can extend only one abstract class.

7. Interface is absolutely abstract and cannot be instantiated; A Java abstract class also cannot be instantiated, but can be invoked if a main() exists.

8. In comparison with java abstract classes, java interfaces are slow as it requires extra indirection.

Object (noun): A material thing that can be seen and touched.

Use a factory pattern if you have a super class and number of sub classes and based on some data provided, you have to return the object of one of the sub classes.

**When to use abstract class and Interface?**

Abstract class has non-abstract method which makes difference. When we want to implement own customized methods but interface does not allow.

**What is Abstraction?**

Abstraction refers to the act of representing essential features without including the background details or explanations.

**Abstraction** in [**Object Oriented Programming**](http://java9s.com/topic/object-oriented-programming)helps to hide the irrelevant details of an object. **Abstraction** is separating the functions and properties that logically can be separated to a separate entity which the main type depends on.

**Abstraction is achieved by Composition.**

## Abstraction has three advantages:

1. By using **abstraction**, we can separate the things that can be grouped to another type.
2. Frequently changing properties and methods can be grouped to a separate type so that the main type need not undergo changes. This adds strength to the **OOAD** principle -"**Code should be open for Extension but closed for Modification**".
3. Simplifies the representation of the**domain models**.

**Object -** Objects have states and behaviours. Example: A dog has states-colour, name, and breed as well as behaviours -wagging, barking, and eating. An object is an instance of a class.

**Class -** A class can be defined as a template/ blue print that describe the behaviours/states that object of its type support.

A class is a blue print from which individual objects are created.

**What is Encapsulation?**

Encapsulation is a technique used for hiding the properties and behaviours of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

**What is the difference between abstraction and encapsulation?**

* **Abstraction** focuses on the outside view of an object (i.e. The interface) **Encapsulation** (information hiding) prevents clients from seeing it’s inside view, where the behaviour of the abstraction is implemented.
* **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
* **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

**What is Inheritance?**

* Inheritance is the process by which objects of one class acquire the properties of objects of another class.
* A class that is inherited is called a super class.
* The class that does the inheriting is called a subclass.
* Inheritance is done by using the keyword extends.
* The two most common reasons to use inheritance are:
  + To promote code reuse
  + To use polymorphism

**Advantage of inheritance?**

One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses

Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass

Reusability -- facility to use public methods of base class without rewriting the same

Extensibility -- extending the base class logic as per business logic of the derived class

Data hiding -- base class can decide to keep some data private so that it cannot be altered by the derived class

Overriding--With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

**Disadvantage of inheritance?**

**1**. One of the main disadvantages of inheritance in Java is the increased time/effort it takes the program to jump through all the levels of overloaded classes. If a given class has ten levels of abstraction above it, then it will essentially take ten jumps to run through a function defined in each of those classes

2. Main disadvantage of using inheritance is that the two classes (base and inherited class) get tightly coupled. This means one cannot be used independent of each other.

3. Also with time, during maintenance adding new features both base as well as derived classes are required to be changed. If a method signature is changed then we will be affected in both cases (inheritance & composition)

4. If a method is deleted in the "super class" or aggregate, then we will have to re-factor in case of using that method.Here things can get a bit complicated in case of inheritance because our programs will still compile, but the methods of the subclass will no longer be overriding superclass methods. These methods will become independent methods in their own right.

Why isn't possible to overload a function just by changing the return type?

You can't do it in Java, and you can't do it in C++. The rationale is that the return value alone is not sufficient for the compiler to figure out which function to call:

Public int foo() {...}

Public float foo() {..}

...

Foo(); // which one?

**Overloading:**

* *Overloaded methods MUST change the argument list*
* *Overloaded methods CAN change the return type—Why? Because ex. Add method can return int, float, double and long according to need*
* *Overloaded methods CAN change the access modifier*
* *Overloaded methods CAN declare new or broader checked exceptions*
* *A method can be overloaded in the same class or in a subclass*

**Overriding:**

* *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can’t override a method marked public and make it protected).*
* *You cannot override a method marked final*
* *You cannot override a method marked static*

Method overriding allows you to write flexible and extensible code in Java because you can introduce new functionality with minimal code change.

If you are extending [abstract class](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html) or implementing interface than you need to override all abstract method unless your class is not abstract. Abstract method can only be used by using method overriding.

**Encapsulation** is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.

The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code.

Example:

Let us look at an example that depicts encapsulation:

/\* File name : encaptest.java \*/

Public class encaptest{

Private String name;

Private String idnum;

Private int age;

Public int getage(){

Return age;

}

Public String getname(){

Return name;

}

Public String getidnum(){

Return idnum;

}

Public void setage( int newage){

Age = newage;

}

Public void setname(String newname){

Name = newname;

}

Public void setidnum( String newid){

Idnum = newid;

}

}

The public methods are the access points to this class's fields from the outside java world. Normally these methods are referred as getters and setters. Therefore any class that wants to access the variables should access them through these getters and setters.

The variables of the encaptest class can be access as below::

/\* File name : runencap.java \*/

Public class runencap{

Public static void main(String args[]){

Encaptest encap = new encaptest();

Encap.setname("James");

Encap.setage(20);

Encap.setidnum("12343ms");

System.out.print("Name : " + encap.getname()+

" Age : "+ encap.getage());

}

}

This would produce following result:

Name : James Age : 20

Benefits of Encapsulation:

* The fields of a class can be made read-only or write-only.
* A class can have total control over what is stored in its fields.
* The users of a class do not know how the class stores its data. A class can change the data type of a field, and users of the class do not need to change any of their code.

**Unchecked**   
[arrayindexoutofboundsexception](http://docs.oracle.com/javase/7/docs/api/java/lang/ArrayIndexOutOfBoundsException.html)   
[classcastexception](http://docs.oracle.com/javase/7/docs/api/java/lang/ClassCastException.html)   
[illegalargumentexception](http://docs.oracle.com/javase/7/docs/api/java/lang/IllegalArgumentException.html)   
[illegalstateexception](http://www.coderanch.com/how-to/java/IllegalStateException)   
[nullpointerexception](http://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html)   
[numberformatexception](http://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html)   
[assertionerror](http://docs.oracle.com/javase/7/docs/api/java/lang/AssertionError.html)   
[exceptionininitializererror](http://docs.oracle.com/javase/7/docs/api/java/lang/ExceptionInInitializerError.html)   
[stackoverflowerror](http://docs.oracle.com/javase/7/docs/api/java/lang/StackOverflowError.html)   
[noclassdeffounderror](http://docs.oracle.com/javase/7/docs/api/java/lang/NoClassDefFoundError.html)   
  
**Checked**   
Exception   
[ioexception](http://docs.oracle.com/javase/7/docs/api/java/io/IOException.html)   
[filenotfoundexception](http://docs.oracle.com/javase/7/docs/api/java/io/FileNotFoundException.html)   
[parseexception](http://docs.oracle.com/javase/7/docs/api/java/text/ParseException.html)   
[classnotfoundexception](http://docs.oracle.com/javase/7/docs/api/java/lang/ClassNotFoundException.html)   
[clonenotsupportedexception](http://docs.oracle.com/javase/7/docs/api/java/lang/CloneNotSupportedException.html)   
[instantiationexception](http://docs.oracle.com/javase/7/docs/api/java/lang/InstantiationException.html)   
[interruptedexception](http://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html)   
[nosuchmethodexception](http://docs.oracle.com/javase/7/docs/api/java/lang/NoSuchMethodException.html)   
[nosuchfieldexception](http://docs.oracle.com/javase/7/docs/api/java/lang/NoSuchFieldException.html)

**Class:** A class defines the properties and behaviour (variables and methods) that is shared by all its objects. It is a blue [print](http://roseindia.net/java/learn-java-in-a-day/oops.shtml) for the creation of objects.

**Object:** Object is the basic entity of object oriented programming language. Class itself does nothing but the real functionality is achieved through their objects. Object is an instance of the class. It takes the properties (variables) and uses the behaviour (methods) defined in the class.

**Encapsulation, Inheritance and Polymorphism are main pillars of oops.** These have been described below :

**Encapsulation:** Encapsulation is the process of binding together the methods and data variables as a single entity. This keeps both the data and functionality code safe from the outside world. It hides the data within the class and makes it available only through the methods.Java provides different accessibility scopes (public, protected, private, default) to hide the data from outside.

Use setter, getter to achieve encatulation.

**Inheritance:** Inheritance allows a class (subclass) to acquire the properties and behaviour of another class (superclass**). In java, a class can inherit only one class (superclass) at a time** but a class can have any number of subclasses. It helps to reuse, customize and enhance the existing code. So it helps to write a code accurately and reduce the development time. Java uses extends [keyword](http://roseindia.net/java/learn-java-in-a-day/oops.shtml" \t "undefined)

[http://kona.kontera.com/javascript/lib/imgs/grey_loader.gif](http://roseindia.net/java/learn-java-in-a-day/oops.shtml" \t "undefined)

To extend a class.

|  |
| --- |
| **Class**A{   **public void**fun1(**int**x){      System.out.println("int in A");   }} **class**B **extends**A{   **public void**fun2(**int**x,int y){      **fun1(6);**  // prints "int in A"       System.out.println("int in B");     }}  **Public class**C{   **public static void**main(String[] args){      B obj= **new**B();      obj.fun2(2);       }} |

In the above example, class B extends class A and so acquires properties and behaviour of class A. So, we can call method of A in class B.

**Polymorphism:** Polymorphism allows one interface to be used for a set of actions i.e. One name may refer to different functionality. Polymorphism allows a object to accept different requests of a client (it then properly interprets the request like choosing appropriate method) and responds  according to the current state of the runtime [system](http://roseindia.net/java/learn-java-in-a-day/oops.shtml), all without bothering the user.

There are two types of polymorphism:

1. **Compile-time polymorphism**
2. **Runtime Polymorphism**

In **compiletime** Polymorphism, method to be invoked is determined at the compile time. Compile time polymorphism is supported through the **method overloading** concept in java.

Method overloading means having multiple methods with same name but with different signature (number, type and order of parameters).

|  |
| --- |
| **Class**A{   **public void**fun1(**int**x){     System.out.println("int");   }   **public void**fun1(**int**x,int y){     System.out.println("int and int");   }} **public class**B{   **public static void**main(String[] args){     A obj= A(); // Here compiler decides that fun1(int)  is to be called and "int" will be printed.     Obj.fun1(2);  // Here compiler decides that fun1(int,int)  is to be called and "int and int" will be printed.        Obj.fun1(2,3);          } } |

In **rumtime** polymorphism, the method to be invoked is determined at the run time. The example of run time polymorphism is **method overriding**. When a subclass contains a method with the same name and signature as in the super class then it is called as method overriding.

|  |
| --- |
| **Class**A{   **public void**fun1(**int**x){      System.out.println("int in A");   }   **public void**fun1(**int**x,int y){      System.out.println("int and int");   }} **class**C **extends**A{   **public void**fun1(**int**x){      System.out.println("int in C");   }} **public class**D{   **public static void**main(String[] args){      A obj;      obj= **new**A(); // line 1      obj.fun1(2);  // line 2 (prints "int in A")             obj=**new**C();  // line 3      obj.fun1(2);  // line 4 (prints "int in C")   }} |

In the above program, **obj** has been declared as A type. In line 1, object of class A is assigned. Now in the next line, fun1(int) of class A will be called. In line 3, obj has been assigned the object of class C so fun1(int) of class C will be invoked in line 4. Now we can understand that same name of the method invokes different functions, defined in different classes, according to the current type of variable **"obj"**. This binding of  method code to the method call is decided at run time.

**What is Enum in Java**  
Enum in Javais a keyword, a feature which is used to represent fixed number of well known values in Java, For example Number of days in Week, Number of planets in Solar system etc. **Enumeration (Enum) in Java** was introduced in JDK 1.5 and it is one of my favorite features of J2SE 5 among [Autoboxing and unboxing](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html) , [Generics](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html), [varargs](http://javarevisited.blogspot.sg/2011/09/variable-argument-in-java5-varargs.html)and [static import](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html). Java Enum as type is more suitable on certain cases for example representing state of Order as NEW, PARTIAL FILL, FILL or CLOSED. Enumeration(Enum) was not originally available in Java though it was available in other language like C and C++ but eventually Java realized and introduced Enum on JDK 5 (Tiger) by **keyword Enum**. In this **Java Enum tutorial** we will see different*enum example in Java* and learn using Enum in Java. Focus of this Java Enum tutorial will be on different features provided by Enum in Java and how to use them. If you have used Enumeration before in C or C++ than you will not be uncomfortable with Java Enum but in my opinion Enum in Java is more rich and versatile than in any other language. One of the common use of Enum which emerges is [Using Enum to write Singleton in Java](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html), which is by far easiest way to implement Singleton and handles several issues related to [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), [Serialization](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) automatically.

**How to represent enumerable value without Java enum**

[java enum example, enum in java tutorial](http://javarevisited.blogspot.com/2011/08/convert-string-to-integer-to-string.html)Since **Enum in Java** is only available from **Java 1.5** its worth to discuss how we used to represent enumerable values in Java prior JDK 1.5 and without it. I use public static [final constant](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) to replicate enum like behavior. Let’s see an Enum example in Java to understand the concept better. In this example we will use US Currency Coin as enumerable which has values like PENNY (1) NICKLE (5), DIME (10), and QUARTER (25).

**Class** currencydenom {

**Public** **static** **final** **int** *PENNY* = 1;

**Public** **static** **final** **int** *NICKLE* = 5;

**Public** **static** **final** **int** *DIME* = 10;

**Public** **static** **final** **int** *QUARTER* = 25; }

**Class** Currency {

**Int** currency; //currencydenom.PENNY,currencydenom.NICKLE,

                 // currencydenom.DIME,currencydenom.QUARTER

}  
 Though this can server our purpose it has some serious limitations:  
  
**1) No Type-Safety**: First of all it’s not [type-safe](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html); you can assign any valid int value to currency e.g. 99 though there is no coin to represent that value.  
  
**2) No Meaningful Printing**: printing value of any of these constant will print its numeric value instead of meaningful name of coin e.g. When you print NICKLE it will print "5" instead of "NICKLE"  
**3) No namespace:** to access the currencydenom constant we need to prefix class name e.g. Currencydenom.PENNY instead of just using PENNY though this can also be achieved by using [static import in JDK 1.5](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html)  
  
**Java Enum** is answer of all this limitation. Enum in Java is type-safe, provides meaningful String names and has there own namespace. Now let's see same example using Enum in Java:

**Public** **enum** Currency {PENNY, NICKLE, DIME, QUARTER};

Here Currency is our **enum** and PENNY, NICKLE, DIME, QUARTER are **enum constants**. Notice **curly braces around enum constants** because Enum are type like [class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html)and[interface in Java](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html). Also we have followed similar naming convention for enum like class and interface (first letter in Caps) and since *Enum constants are implicitly static final* we have used all caps to specify them like Constants in Java.

**What is Enum in Java**

Now back to primary questions **“What is Enum in java”** simple *answer Enum is a keyword in java* and on more detail term Java Enum is type like class and interface and can be used to define a set of Enum constants. Enum constants are [implicitly static and final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) and you can not change there value once created. Enum in Java provides type-safety and can be used inside switch statment like int variables. Since enum is a keyword you can not use as variable name and since its only introduced in JDK 1.5 all your previous code which has enum as variable name will not work and needs to be re-factored.

**Benefits of Enums in Java:**

1) **Enum is type-safe** you can not assign anything else other than predefined Enum constants to an Enum variable. It is compiler error to assign something else unlike the public static final variables used in Enum int pattern and Enum String pattern.  
  
2) Enum has its own name-space.  
  
3) Best feature of Enum is **you can use Enum in Java inside Switch statement** like int or char primitive data type.we will also see example of using java enum in switch statement in this java enum tutorial.  
  
4) Adding new constants on Enum in Java is easy and you can add new constants without breaking existing code.  
  
**Important points about Enum in Java**

1) **Enums in Java are type-safe** and has there own name-space. It means your enum will have a type for example "Currency" in below example and you can not assign any value other than specified in Enum Constants.

**Public** **enum** Currency {*PENNY*, *NICKLE*, *DIME*, *QUARTER*};

Currency coin = Currency.PENNY;

Coin = 1; //compilation error    
  
2**) Enum in Java are reference type**like [class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html)or [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html)and you can define constructor, methods and variables inside java Enum which makes it more powerful than Enum in C and C++ as shown in next example of Java Enum type.  
  
3) You can **specify values of enum constants at the creation time** as shown in below example:  
**public** **enum** Currency {*PENNY*(1), *NICKLE*(5), *DIME*(10), *QUARTER*(25)};  
But for this to work you need to define a member variable and a constructor because PENNY (1) is actually [calling a constructor](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) which accepts int value , see below example.

**Public** **enum** Currency {

*PENNY*(1), *NICKLE*(5), *DIME*(10), *QUARTER*(25);

**Private** **int** value;

**Private** Currency(**int** value) {

**This**.value = value;

        }

};     
**Constructor of enum in java** must be [**private**](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html)any other access modifier will result in compilation error. Now to get the value associated with each coin you can define a public getvalue() method inside java enum like any normal java class. Also semi colon in the first line is optional.  
  
  
4) Enum constants are implicitly [static](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html)and [final](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html)and can not be changed once created. For example below code of java enum will result in compilation error:

Currency.PENNY = Currency.DIME;

The final field enumexamples.Currency.PENNY cannot be re assigned.  
    
5) **Enum in java can be used as an argument on switch statment** and with "case:" like int or char primitive type. This feature of java enum makes them very useful for switch operations. Let’s see an example of how to use java enum inside switch statement:  

   Currency uscoin = Currency.DIME;

**Switch** (uscoin) {

**Case** PENNY:

                    System.out.println("Penny coin");

**Break**;

**Case** NICKLE:

                    System.out.println("Nickle coin");

**Break**;

**Case** DIME:

                    System.out.println("Dime coin");

**Break**;

**Case** QUARTER:

                    System.out.println("Quarter coin");

    }  
    
from JDK 7 onwards you can also [String in Switch case in Java](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html) code.  
  
6) Since **constants defined inside Enum in Java are final you can safely compare them using "==" equality operator** as shown in following example of  Java Enum:

Currency uscoin = Currency.DIME;

**If**(uscoin == Currency.DIME){

  System.*out*.println("enum in java can be compared using ==");

}

By the way comparing objects using == operator is not recommended, Always use [equals() method](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) or [compareto() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) to compare Objects.

7) Java compiler automatically generates static values() method for every enum in java. Values() method returns array of Enum constants in the same order they have listed in Enum and you can use values() to [iterate](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html)over values of Enum  in Java as shown in below example:

**For**(Currency coin: Currency.values()){

        System.*out*.println("coin: " + coin);

}  
  
And it will print:

**Coin: PENNY**

**Coin: NICKLE**

**Coin: DIME**

**Coin: QUARTER**  
                  
Notice the order its exactly same **with defined order in enums**.  
  
8) In Java Enum can override methods also. Let’s see an example of overriding tostring() method **inside Enum in Java** to provide **meaningful description** for enums constants.

**Public** **enum** Currency {

  ........

  @Override

**Public** String tostring() {

**Switch** (**this**) {

**Case** PENNY:

              System.out.println("Penny: " + value);

**Break**;

**Case** NICKLE:

              System.out.println("Nickle: " + value);

**Break**;

**Case** DIME:

              System.out.println("Dime: " + value);

**Break**;

**Case** QUARTER:

              System.out.println("Quarter: " + value);

        }

**Return** **super**.tostring();

 }

};          
And here is how it looks like when displayed:

Currency uscoin = Currency.*DIME*;

System.out.println(uscoin);

**Output:**

**Dime: 10**  
  
9) Two new collection classes **enummap and enumset** are added into collection package to **support Java Enum**. These classes are high performance implementation of [Map and Set interface in Java](http://javarevisited.blogspot.sg/2012/07/create-read-only-list-map-set-example-java.html)and we should use this whenever there is any opportunity.  
  
10**) You can not create instance of enums by using new operator** in Java because constructor of Enum in Java can only be private and Enums constants can only be created inside Enums itself.  
  
11) Instance of Enum in Java is created when any Enum constants are first called or referenced in code.  
  
12) **Enum in Java can implement the interface** and override any method like normal class It’s also worth noting that Enum in java implicitly implement both [Serializable](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html)and [Comparable](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html)interface. Let's see and example of **how to implement interface using Java Enum**:

**Public** **enum** Currency **implements** Runnable{

  PENNY(1), NICKLE(5), DIME(10), QUARTER(25);

**Private** **int** value;

  ............

  @Override

**Public** **void** run() {

  System.out.println("Enum in Java implement interfaces");

   }

}  
  
13) **You can define abstract methods inside Enum in Java** and can also provide different implementation for different instances of enum in java.  Let’s see an *example of using*[*abstract method*](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html)*inside enum in java*

**Public** **enum** Currency **implements** Runnable{

          PENNY(1) {

                  @Override

**Public** String color() {

**Return** "copper";

                  }

          }, NICKLE(5) {

                  @Override

**Public** String color() {

**Return** "bronze";

                  }

          }, DIME(10) {

                  @Override

**Public** String color() {

**Return** "silver";

                  }

          }, QUARTER(25) {

                  @Override

**Public** String color() {

**Return** "silver";

                  }

          };

**Private** **int** value;

**Public** **abstract** String color();

**Private** Currency(**int** value) {

**This**.value = value;

          }

          ..............

  }         
In this example since every coin will have different color we made the color() method abstract and let each instance of Enum to define   there own color. You can get color of any coin by just calling color() method as shown in below example of java enum:

System.out.println("Color: " + Currency.DIME.color());  
    
**Enum Java valueof example**  
One of my reader pointed out that I have not mention about valueof method of enum in Java, which is used to convert String to enum in java.  Here is what he has suggested, thanks @ Anonymous  
“You could also include **valueof() method of enum** in java which is added by compiler in any enum along with values() method. **Enum valueof()** is a static method which takes a string argument and can be used to convert a String into enum. One think though you would like to keep in mind is that valueof(String) method of enum will throw "**Exception in thread "main" java.lang.illegalargumentexception: No enum const class**" if you supply any string other than enum values.  
  
Another of my reader suggested about ordenal() and name() utility method of java enum Ordinal method of Java Enum returns position of a Enum constant as they declared in enum while name()of Enum returns the exact string which is used to create that particular Enum constant.” Name() method can also be used for [converting Enum to String in Java](http://javarevisited.blogspot.sg/2011/12/convert-enum-string-java-example.html).

**What is its basic use?**

JNDI allows distributed applications to look up services in an abstract, resource-independent way.

**When it is used?**

The most common use case is to set up a database connection pool on a Java EE application server. Any application that's deployed on that server can gain access to the connections they need using the JNDI name "java:comp/env/foobarpool" without having to know the details about the connection.

This has several advantages:

1. If you have a deployment sequence where apps move from devl->int->test->prod environments, you can use the same JNDI name in each environment and hide the actual database being used. Applications don't have to change as they migrate between environments.
2. You can minimize the number of folks who need to know the credentials for accessing a production database. Only the Java EE app server needs to know if you use JNDI.

Java: Static and Dynamic Class loading

There are 2 ways to load your classes in java.

**Static Class Loading** : Using new operator you load classes statically in java.

Public static void main(String[] args){

Man m = new Man();

}

In static class loading **noclassdeffoundexception** can be thrown if the loaded class is not available in run time.

**Dynamic Class Loading**: This can be achieved by invoking the Class loader functions programmatically. Typically we use**class.forname(String classname)** to get the class first and then we will call the **newinstance()** method on the returned class to get the instance.

For example if you want to get a instance of **Man** class using dynamic class loading; you have to

Man man = null;

Class manclass = Class.forname("com.javaj2ee.Man");

Man = (Man) manclass.newinstance();

Man.dosomething();

Here in dynamic class loading **classnotfoundexception** can be thrown if given Class loader doesn’t find the given Class name (com.javaj2ee.Man) in the classpath.Class Loader looks for a given class in following sequence:

* The **forname(..)** Method in the class called **Class**
* The **findsystemclass(..)** Method in the class class called **classloader**
* The **loadclass(..)** Method in the class called **classloader**

You can instantiate an abstract class in 2 ways:

Abstract class abstractclass {}

**1. Extending the Abstract class:**

Class concreteclass extends abstractclass {}

Abstractclass c = new concreteclass();

**2. Using Anonymous Class:**

Abstractclass c = new abstractclass(){};

Abstract class my {

Public void mymethod() {

System.out.print("Abstract");

}

}

Class poly extends my {

Public static void main(String a[]) {

My m = new my() {};

M.mymethod();

}

}

Above abstract class is not creating the instance of your abstract class here. Rather we are creating an instance of an *anonymous subclass* of our abstract class. And then we are invoking the method on our *abstract class* reference pointing to *subclass object*.

This behaviour is clearly listed in [JLS - Section # 15.9.1](http://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html#jls-15.9): -

If the class instance creation expression ends in a class body, then the class being instantiated is an anonymous class. Then:

* If T denotes a class, then an anonymous direct subclass of the class named by T is declared. It is a compile-time error if the class denoted by T is a final class.
* If T denotes an interface, then an anonymous direct subclass of Object that implements the interface named by T is declared.
* In either case, the body of the subclass is the classbody given in the class instance creation expression.
* **The class being instantiated is the anonymous subclass.**

////////////////

Emphasis mine.

Also, in [JLS - Section # 12.5](http://docs.oracle.com/javase/specs/jls/se7/html/jls-12.html#jls-12.5), you can read about the *Object Creation Process*. I'll quote one statement from that here: -

Whenever a new class instance is created, memory space is allocated for it with room for all the instance variables declared in the class type and all the instance variables declared in each superclass of the class type, including all the instance variables that may be hidden.

Just before a reference to the newly created object is returned as the result, the indicated constructor is processed to initialize the new object using the following procedure:

To practically see that the class being instantiated is an *Anonymous subclass*, you just need to compile both your classes. Suppose you put those classes in two different files:

**My.java:**

Abstract class My {

Public void mymethod() {

System.out.print("Abstract");

}

}

**Poly.java:**

Class Poly extends My {

Public static void main(String a[]) {

My m = new My() {};

M.mymethod();

}

}

Now, compile both your source files:

Javac My.java Poly.java

Now in the directory where you compiled the source code, you will see the following class files:

My.class

Poly$1.class // Class file corresponding to anonymous subclass

Poly.class

See that class - Poly$1.class. It's the class file created by the compiler corresponding to the anonymous subclass you instantiated using the below code:

New My() {};

So, it's clear that there is a different class being instantiated. It's just that, that class is given a name only after compilation by the compiler.

In general, all the anonymous subclasses in your class will be named in this fashion:

Poly$1.class, Poly$2.class, Poly$3.class, ... So on

Those numbers denote the order in which those anonymous classes appear in the enclosing class.

1. An abstract class has a protected constructor (by default) allowing derived types to initialize it.
2. The main reasoning behind the preference for abstract base classes is versioning, because you can always add a new virtual member to an abstract base class without breaking existing clients. That's not possible with interfaces.
3. An abstract class is also good if you want to be able to declare non-public members. In an interface, all methods must be public.
4. If you think you will need to add methods in the future, then an abstract class is a better choice. Because if you add new method headings to an interface, then all the classes that already implement that interface will have to be changed to implement the new methods. That can be quite a hassle.

<http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html>

# [How HashMap works in Java](http://javarevisited.blogspot.in/2011/02/how-hashmap-works-in-java.html)

HashMap in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time O(1) provided we know the key. In hashing, hash functions are used to link key and value in HashMap. Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key) method. When we call put method, hashcode() method of the key object is called so that hash function of the map can find a bucket location to store value object, which is actually an index of the internal array, known as the table. HashMap internally stores mapping in the form of **Map.Entry** object which contains both key and value object. When you want to retrieve the object, you call [the get() method](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) and again pass the key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve the object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location. If there is only one object then it is returned and that's your value object which you have stored earlier. Things get little [tricky](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html) when collisions occur.

Since the internal array of HashMap is of fixed size, and if you keep storing objects, at some point of time hash function will return same bucket location for two different keys, this is called collision in HashMap. In this case, a linked list is formed at that bucket location and a new entry is stored as next node.  
  
If we try to retrieve an object from this linked list, we need an extra check to search correct value, this is done by equals() method. Since each node contains an entry, HashMap keeps comparing entry's key object with the passed key using equals() and when it return true, Map returns the corresponding value.  
Since searching inlined list is O(n) operation, in worst case hash collision reduce a map to linked list. This issue is recently addressed in Java 8 by replacing linked list to the tree to search in O(logN) time. By the way, you can easily verify how HashMap works by looking at the code of HashMap.java

**Difference between ConcurrentHashMap and Hashtable**

So what is the difference between Hashtable and ConcurrentHashMap, both can be used in the multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be locked for a longer duration.

Since ConcurrentHashMap introduced the concept of segmentation, how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.

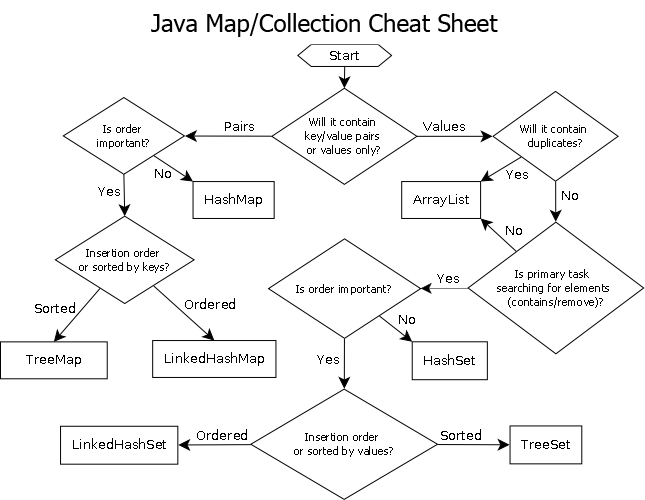
In Summary, ConcurrentHashMap only locked certain portion of Map while Hashtable locks full map while doing iteration.

**The difference between ConcurrentHashMap and Collections.synchronizedMap**

ConcurrentHashMap is designed for concurrency and improve performance while HashMap which is non-synchronized by nature can be synchronized by applying a wrapper using synchronized Map.   
  
ConcurrentHashMap does not allow null keys or null values while synchronized HashMap allows one null key.

The **toString() method** returns the string representation of the object. If you print any object, **java** compiler internally invokes the **toString() method** on the object. So**overriding** the **toString() method**, returns the desired output, it can be the state of an object etc. depends on your implementation

1. how wait and notify are used in a code?
2. how to use volatile?
3. how we know all threads are executed and what is the mechanism implemented behind executorservice.shutdown
4. what happens when wait is called internal implemententation and notify?
5. how reentrant lock is implemented
6. in web service, what is payload, and how to handle a null when we are getting in request?
7. how hashmap works in jdk 1.8
8. what is session and sessionFactory in hibernate
9. how executor frameworks internally
10. how to submit multiple threads in executor?
11. one prototype bean is injected in singleton bean. how singleton bean will behave?
12. what is difference between session and global session.
13. what are the new packages, classes and interfaces added in jdk1.8
14. how will you use lambda expression for list



## References

[Http://www.tutorialspoint.com/java/java\_files\_io.htm](http://www.tutorialspoint.com/java/java_files_io.htm)

[Http://www.tutorialspoint.com/javaexamples/java\_arrays.htm](http://www.tutorialspoint.com/javaexamples/java_arrays.htm)

[Http://www.techferry.com/articles/spring-annotations.html](http://www.techferry.com/articles/spring-annotations.html)

[Http://www.oracle.com/technetwork/articles/java/singleton-1577166.html](http://www.oracle.com/technetwork/articles/java/singleton-1577166.html)

[Http://lingpipe-blog.com/2010/05/04/upgrading-java-classes-backward-compatible-serialization/](http://lingpipe-blog.com/2010/05/04/upgrading-java-classes-backward-compatible-serialization/)

http://techcommunity.softwareag.com/web/guest/pwiki/-/wiki/Main/Performance+Issues+-+how+to+take+various+dumps+for+analysis