1. Marshalling – Convert a Java object into a XML file.
2. Unmarshalling – Convert XML content into a Java Object.

<http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html>

**HashMap vs Hashtable in Java**

Both HashMap and Hashtable implements Map interface but there are some significant difference  
  
1.The HashMap class is roughly equivalent to Hashtable, except that it is non synchronized and permits nulls. (HashMap allows null values as key and value whereas [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html)doesn't allow nulls).  
  
2. One of the major differences between HashMap and Hashtable is that HashMap is [non synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap can not be shared between multiple threads without proper synchronization. Java 5 introduces [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html)which is an alternative of Hashtable and provides better scalability than Hashtable in Java.

3. One more notable difference between Hashtable and HashMap is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don't need synchronization and HashMap is only used by one thread, it out perform Hashtable in Java.  
4. HashMap does not guarantee that the order of the map will remain constant over time.

1)Synchronized means only one [Thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html)can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a Hashtable will have to acquire a lock on the object while others will [wait for lock](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) to be released.  
  
**Difference between ConcurrentHashMap and Collections.synchronizedMap and Hashtable in Java**

both can be used in multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be locked for longer duration. 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.

The synchronized collections classes, Hashtable and Vector, and the synchronized wrapper classes, **Collections.synchronizedMap and Collections.synchronizedList,** provide a basic conditionally thread-safe implementation of Map and List. However, several factors make them unsuitable for use in highly concurrent applications  for example their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent

ConcurrentHashMap is better suited for situation where you have multiple readers and one

Writer or fewer writers since Map gets locked only during write operation. If you have equaled number of reader and writer than [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) will perform in line of hashtable or synchronized hashMap.

**Difference between ArrayList and Vector in Java**1) Vector and ArrayList are index based and backed up by an array internally.

2) Both ArrayList and Vector maintains the insertion order of element. Means you can assume that you will get the object in the order you have inserted if you iterate over ArrayList or Vector.

3) Both [Iterator](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html)and ListIterator returned by ArrayList and Vector are [fail-fast](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html).

4) ArrayList and Vector also allows null and duplicates.

First and foremost difference between Vector and ArrayList is that Vector is synchronized and ArrayList is not, ArrayList is way faster than Vector.

Whenever Vector crossed the threshold specified it increases itself by value specified in capacityIncrement field while you can increase size of ArrayList by calling ensureCapacity () method.

**Difference between List and Set in Java Collection**1) Fundamental difference between List and Set in Java is allowing duplicate elements. List in Java allows duplicates while Set doesn't allow any duplicate. If you insert duplicate in Set it will replace the older value. Any implementation of Set in Java will only contains unique elements.

2) Another significant difference between List and Set in Java is order. List is an Ordered Collection while Set is an unordered Collection. List maintains insertion order of elements, means any element which is inserted before will go on lower index than any element which is inserted after. Set in Java doesn't  maintain any order. Though Set provide another alternative called SortedSet which can store Setelements in specific Sorting order defined by [Comparable and Comparator](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) methods of Objects stored in Set.  
  
**What is difference between Synchronized Collection and ConcurrentCollection?**Java5 has added several new ConcurrentCollection classes e.g. **ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue** etc, which has made Interview questions on Java Collection even trickier. Java Also provided way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.**synchronizedMap**() Utility function.**One Significant difference is that ConccurentCollections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization.**

**HashCode and Equals method in Java object – A pragmatic concept**

Java.lang.Object has methods called hasCode() and equals(). In some case these methods are overridden to perform certain purpose. it is necessary to override  both method.

**hashCode():**this method provides the has code of an object. the default implementation of hashCode() provided by Object is derived by mapping the **memory address to an integer value**.

public native int hashCode(); -- It indicates that hashCode is the native implementation

**equals()**

This particular method is used to make equal comparison between two objects. There are two types of comparisons in Java. One is using “= =” operator and another is “equals()”. .equals()” refers to equivalence relations. So in broad sense you say that two objects are equivalent they satisfy the “equals()” condition.

This method checks if some other object passed to it as an argument is *equal* to the object on which this method is invoked.

**why it is necessary to override this method. :**

There is one reason that if want to compare two objects based upon the equals() method. Although in a very simple class like “Emp”, you can achieve without overriding hashCode() method. But if you do this , you are going to violate the contract for the methods hashCode() and hashCode() of the object class. The similar case is for the method equals(). So funcational point is that if want to compare two objects based upon the equals() method you have to override both hashCode() and equals() methods.

1. What is immutable object? Can you write immutable object?  
  
Immutable classes are Java classes whose objects can not be modified once created. Any modification in Immutable object result in new object. For example is [String is immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html). Mostly Immutable are also final in Java, in order to prevent sub class from overriding methods in Java which can compromise Immutability. You can achieve same functionality by making member as non [final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) but private and not modifying them except in constructor.

**2. Does all property of immutable object needs to be final?**

Not necessary as stated above you can achieve same functionality by making member as non final but [private](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html)and not modifying them except in [constructor](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html).

A **final**[**class**](http://en.wikipedia.org/wiki/Class_(computer_science)) cannot be subclassed.  A **final**[**method**](http://en.wikipedia.org/wiki/Method_(computer_science)) cannot be [overridden](http://en.wikipedia.org/wiki/Method_overriding) or hidden by subclasses. Several classes in Java are final e.g. String, Integer and other wrapper classes.  
All variable declared inside java interface are implicitly final.

**3. What is the difference between creating String as new() and literal?**

When we create string with new() Operator, it’s created in [heap](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html)and not added into string pool while String created using literal are created in String pool itself which exists in [PermGen area of heap](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html).

String s = new String("Test");  
   
does not  put the object in String pool , we need to call String.intern() method which is used to put  them into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" Java automatically put that into String pool.

**4. How does substring () inside String works?**

# [How SubString method works in Java - Memory Leak Fixed in JDK 1.7](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html)

**Substring method**from String classis one of most used method in Java, and it's also part of an interesting [String interview question](http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html)  e.g. How substring works in Java or sometime asked as how does substring creates memory leak in Java. In order to answer these questions, you knowledge of  implementation details is required. Recently one of my friend was drilled on **substring method in Java** during a Java interview, he was using substring() method from long time, and of course all of us has used this, but what surprises him was interviewer's obsession on Java substring, and deep dive till the implementation level. Though String is a special class in Java, and subject of many interview questions e.g. [Why char array is better than String for storing password](http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html) . In this case it was, substring method, which took center stage. Most of us rather just use *substring*(..), and than forgot. Not every Java programmer go into code, and see how exactly it's working. To get a feel of how his interview was let's start .

Question starts with normal chit chat, and Interviewer ask,  "**Have you used substring method in Java**", and my friend proudly said Yes, lot many times, which brings a smile on interviewer's face. He says well, that’s good. Next question was **Can you explain what does substring do?** My friend got an opportunity to show off his talent, and how much he knows about Java API;  He said substring method is used to get parts of String in Java. It’s defined injava.lang.String class, and it's an [overloaded method](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html). One version of substring method takes just beginIndex, and returns part of String started from beginIndex till end, while other takes two parameters, beginIndex and endIndex, and returns part  of String starting from beginIndex to endIndex-1. He also stressed that every time you call  substring() method in Java,  it will return a new String because [String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html).

Next question was, **what will happen if beginIndex is equal to length in substring(int beginIndex)**, no it won't throw IndexOutOfBoundException instead it will return [empty String](http://javarevisited.blogspot.com/2013/02/5-ways-to-check-if-string-is-empty-in-java-examples.html). Same is the case when beginIndex and endIndex is equal, in case of second method. It will only throw StringIndexBoundException when beginIndex is negative, larger than endIndex or larger than length of String.

So far so good, my friend was happy and interview seems going good, until Interviewee asked him**,  Do you know how substring works in Java**? Most of Java developers fail here, because they don't know how exactly substring method works, until they have not seen the code of java.lang.String. If you look substring method inside String class, you will figure out that it calls String (int offset, int count, char value []) [constructor](http://javarevisited.blogspot.com/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) to create new String object. What is interesting here is, value[], which is the same character array used to represent original string. So **what's wrong with this**?

In case If you have still not figured it out, If the original string is very long, and has array of size 1GB, no matter how small a substring is, it will hold 1GB array.  This will also stop original string to be [garbage collected](http://javarevisited.blogspot.com/2012/10/10-garbage-collection-interview-question-answer.html), in case if doesn't have any live reference. This is clear case of memory leak in Java, where memory is retained even if it's not required. That's how *substring method* creates **memory leak**.

Obviously next question from interviewer would be,  **how do you deal with this problem?** Though you can not go, and change Java substring method, you can still make some work around, in case you are creating substring of significant longer String. Simple solution is to trim the string, and keep size of character array according to length of substring. Luckily java.lang.String has constructor to do this, as shown in below example.

*// comma separated stock symbols from NYSE*

String listOfStockSymbolsOnNYSE = getStockSymbolsForNYSE();

*//calling String(string) constructor*

String apple = **new** String(

listOfStockSymbolsOnNYSE.substring(appleStartIndex, appleEndIndex)

);

If you look code on java.lang.String class, you will see that this [constructor](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) trim the array, if it’s bigger than String itself.

**public** String(String original) {

...

**if** (originalValue.length > size) {

*// The array representing the String is bigger than the new*

*// String itself. Perhaps this constructor is being called*

*// in order to trim the baggage, so make a copy of the array.*

**int** off = original.offset;

v = Arrays.copyOfRange(originalValue, off, off+size);

} **else** {

*// The array representing the String is the same*

*// size as the String, so no point in making a copy.*

v = originalValue;

}

...

}

Another way to solve this problem is to call intern() method on substring, which will than fetch an [existing string](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html) from pool or add it if necessary. Since the String in the pool is a real string it only take space as much it requires. It’s also worth noting that sub-strings are not internalized, when you call intern() method on original String. Most developer successfully answers first three questions, which is related to usage of substring, but they get stuck on last two, How substring creates memory leak or How substring works. It's not completely there fault, because what you know is that every time substring() returns new String which is not exactly true, since it’s backed by same [character array](http://javarevisited.blogspot.com/2012/02/how-to-convert-char-to-string-in-java.html).

**5. Which two method you need to implement for key Object in HashMap ?**

In order to use any object as Key in HashMap, it must implements equals and hashcode method in Java. Read [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for detailed explanation on how equals and hashcode method is used to put and get object from HashMap. You can also see my post [5 tips to correctly override equals in Java](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) to learn more about equals.

**6. Where does these  two method comes in picture during get operation?**

This core Java interview question is follow-up of previous Java question and candidate should know that once you mention [hashCode](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html), people are most likely ask How its used in HashMap. See [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for detailed explanation.

**7. How do you handle error condition while writing stored procedure or accessing stored procedure from java?**

This is one of the tough Java interview question and its open for all, my friend didn't know the answer so he didn't mind telling me. my take is that stored procedure should return error code if some operation fails but if stored procedure itself fail than catching [SQLException](http://javarevisited.blogspot.sg/2012/01/javasqlsqlexception-invalid-column.html)is only choice.

**8. What is difference between Executor.submit() and Executer.execute() method ?**

This Java interview question is from my list of [Top 15 Java multi-threading question answers](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html), Its getting popular day by day because of huge demand of Java developer with good concurrency skill. Answer of this Java interview question is that former returns an object of Future which can be used to find result from worker thread)

There is a difference when looking at exception handling. If your tasks [throws an exception](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) and if it was submitted with execute this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, [checked exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) or not, is then part of the task's return status. For a task that was submitted with submit and that terminates with an exception, the Future.get will re-throw this exception, wrapped in an ExecutionException.

**9. What is the difference between factory and abstract factory pattern?**

Abstract Factory provides one more level of [abstraction](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html). Consider different factories each extended from an Abstract Factory and responsible for creation of different hierarchies of objects based on the type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for creation of objects in that genre.

You can also refer [What is Factory method design pattern in Java](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html) to know more details.

**10. What is Singleton? is it better to make whole method synchronized or only critical section synchronized ?**

Singleton in Java is a class with just one instance in whole Java application, for example java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced [Enum](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html)its very easy. see my article [How to create thread-safe Singleton in Java](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) for more details on writing Singleton using enum and double checked locking which is purpose of this Java interview question.

**11. Can you write critical section code for singleton?**

This core Java question is followup of previous question and expecting candidate to write Java singleton using double checked locking. Remember to use [volatile variable](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html)to make Singleton thread-safe. check [10 Interview questions on Singleton Pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) for more details and questions answers

**12. Can you write code for iterating over hashmap in Java 4 and Java 5 ?**

# Tricky one but he managed to write using while and for loop. 13. When do you override hashcode and equals() ? Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap. check this for writing equals method correctly [5 tips on equals in Java](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) 14. What will be the problem if you don't override hashcode() method ? You will not be able to recover your object from hash Map if that is used as key in HashMap.  See here  [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for detailed explanation. 15. Is it better to synchronize critical section of getInstance() method or whole getInstance() method ? Answer is critical section because if we lock whole method than every time some one call this method will have to wait even though we are not creating any object) 16. What is the difference when String is gets created using literal or new() operator ? When we create string with new() its created in heap and not added into string pool while String created using literal are created in String pool itself which exists in Perm area of heap. 17. Does not overriding hashcode() method has any performance implication ? This is a good question and open to all , as per my knowledge a poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map. 18. What’s wrong using HashMap in multithreaded environment? When get() method go to infinite loop ? Another good question. His answer was during concurrent access and re-sizing. 19. Give a simplest way to find out the time a method takes for execution without using any profiling tool? this questions is suggested by @Mohit Read the system time just before the method is invoked and immediately after method returns. Take the time difference, which will give you the time taken by a method for execution. To put it in code… long start = System.currentTimeMillis (); method (); long end = System.currentTimeMillis (); System.out.println (“Time taken for execution is ” + (end – start)); Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method which is big enough, in the sense the one which is doing considerable amout of processing [What is Encapsulation in Java and OOPS with Example](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html)

**Encapsulation in Java** or object oriented programming language is a concept which enforce protecting variables, functions from outside of class, in order to better manage that piece of code and having least impact or no impact on other parts of program duec to change in protected code. *Encapsulation in Java* is visible at different places and Java language itself provide many construct to encapsulate members. You can completely encapsulate a member be it a variable or method in Java by using private keyword and you can even achieve a lesser degree of encapsulation in Java by using other access modifier like protected or public. true value of encapsulation is realized in an environment which is prone to change a lot and we know that in software requirements changes every day at that time if you have your code well encapsulated you can better manage risk with change in requirement. Along with [abstaction in java](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html)and [polymorphism in Java](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html), Encapsulation is a must know concept. In this java tutorial  we will see **How to use encapsulation in Java**, advantage and disadvantage of Encapsulation and various design patterns and real life problems which makes use of Encapsulation object oriented concept. If you are looking for a quick guide on both OOPS and SOLID design principle in Java than you may find [**10 Object Oriented Design principles Java programmer should know**](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html) interesting.

## What is Encapsulation in Java

[http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)**Encapsulation** is nothing but protecting anything which is prone to change. rational behind encapsulation is that if any functionality which is well encapsulated in code i.e maintained in just one place and not scattered around code is easy to change. this can be better explained with a simple example of encapsulation in Java. we all know that constructor is used to create object in Java and constructor can accept argument. Suppose we have a class Loan has a constructor and than in various classes you have created instance of loan by using this [constructor](http://javarevisited.blogspot.com/2012/01/what-is-constructor-overloading-in-java.html). now requirements change and you need to include age of borrower as well while taking loan. Since this code is not well encapsulated i.e. not confined in one place you need to change everywhere you are calling this constructor i.e. for one change you need to modify several file instead of just one file which is more error prone and tedious, though it can be done with refactoring feature of advanced IDE wouldn't it be better if you only need to make change at one place ? Yes that is possible if we encapsulate Loan creation logic in one method say createLoan() and client code call this method and this method internally crate Loan object. in this case you only need to modify this method instead of all client code.

**Example of Encapsulation in Java**

**class** Loan{  
    **private** **int** duration;  //private variables examples of encapsulation  
    **private** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) loan;  
    **private** **String** borrower;  
    **private** **String** salary;

    //public constructor can break encapsulation instead use factory method  
    **private**Loan(**int** duration, **String** loan, **String** borrower, **String** salary){  
        **this**.duration = duration;  
        **this**.loan = loan;  
        **this**.borrower = borrower;  
        **this**.salary = salary;  
    }

    //no argument consustructor omitted here

   // create loan can encapsulate loan creation logic

**public** Loan createLoan(**String** loanType){  
    
     //processing based on loan type and than returning loan object

**return** loan;

    }

}

In this same example of *Encapsulation in Java* you see all member variables are made private so they are well encapsulated you can only change or access this variable directly inside this class. if you want to allow outside world to access these variables is better creating a getter and setter e.g. getLoan() that allows you to do any kind of validation, security check before return loan so it gives you complete control of whatever you want to do and single channel of access for client which is controlled and managed.

## Advantage of Encapsulation in Java and OOPS

Here are few advantages of using **Encapsulation** while writing code in Java or any Object oriented programming language:

1. Encapsulated Code is more flexible and easy to change with new requirements.

2. Encapsulation in Java makes unit testing easy.

3. Encapsulation in Java allows you to control who can access what.

4. Encapsulation also helps to write immutable class in Java which are a good choice in multi-threading

environment.

5. Encapsulation reduce coupling of modules and increase cohesion inside a module because all piece of one thing

are encapsulated in one place.

6. Encapsulation allows you to change one part of code without affecting other part of code.

**What should you encapsulate in code**

Anything which can be change and more likely to change in near future is candidate of Encapsulation. This also helps to write more specific and cohesive code. Example of this is object creation code, code which can be improved in future like sorting and searching logic.

## Design Pattern based on Encapsulation in Java

Many design pattern in Java uses encapsulation concept, one of them is [Factory pattern](http://javarevisited.blogspot.com/2011/12/factory-design-pattern-java-example.html) which is used to create objects. Factory pattern is better choice than new operator for creating object of those classes whose creation logic can vary and also for creating different implementation of same interface. BorderFactory class of JDK is a good example of encapsulation in Java which creates different types of Border and encapsulate creation logic of Border.[Singleton pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) also encapsulate how you create instance by providing getInstance() method. since object

is created inside one class and not from any other place in code you can easily change how you create object without

affect other part of code.

### Important points aboue encapsulation in Java.

1. "Whatever changes encapsulate it" is a famous design principle.

2. Encapsulation helps in loose coupling and high cohesion of code.

3. Encapsulation in Java is achieved using access modifier private, protected and public.

4. Factory pattern , Singleton pattern in Java makes good use of Encapsulation.