# OOPS:

The main idea behind OOP is to represent the data and logic with objects instead of actions or functions.

**Object:** Object is a real world entity that has state and behaviour. (Chair, bike, fan)

**Class:** A class is a user defined blueprint or prototype. (House design) Car is a class audi, nissan and Volvo are objects.

**Abstraction:** Abstraction is a process of hiding the implementation details and showing only functionality to the user. In Java, abstraction is achieved using abstract classes and interfaces.

**Encapsulation:** Encapsulation is all about wrapping variables and methods in one single unit. Encapsulation is also known as data hiding. We can create a fully encapsulated class in java by making all the data members of the class private. Encapsulation in java is achieved using private, protected and public keywords.

**Polymorphism:** The process of representing one form in multiple forms is known as Polymorphism.

**Overriding Rules:**

1. public, protected and default (in the same package) can be overridden and private methods final methods cannot be overridden.
2. The overriding method must have same return type or subtype (called co-variant return type).
3. If the overridden method is has default access, then the overriding method must be default, protected or public.
4. The overriding method must not throw new or broader checked exceptions. In other words, the overriding method may throw narrower checked exceptions, or any unchecked exceptions

IOException to FileNotFoundException

Exception to IOException

1. Overriding is applicable for method but not for variables.

**Inheritance: Inheritance** is a mechanism in which one object acquires all the properties and behaviors of parent object. Inheritance represents the **IS-A relationship, extends keyword** is used for inheritance.

**Aggregation:** If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Without existing container object there may be a chance of exiting contained object.

**Composition:** Without existing container object there is no chance of existing container object.

ENCAPSULATION: Encapsulation in java is a process of wrapping code and data together into a single unit. Also Encapsulation means data hiding in-order to make it safe from any modification.

EXAMPLE:

1. Television is loaded with different functionalities that we don't know because they are completely hidden.

2. As a driver you know how to start the car by pressing the start button and internal details of the starting operations are hidden from you.

POLYMORPHISM: The process of representing one form in multiple forms is known as Polymorphism.

EXAMPLE: Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviour’s.

INHERITANCE: Inheritance is a special type of relationship where a class acquires the inherent properties of its parent class along with this it also contains its own exclusive properties.

EXAMPLE: Father and Son Relationship. Here son inherits Father physical properties but father can't get son physical properties.

ABSTRACTION: Abstraction is the concept of exposing only the required essential characteristics and behaviour with respect to a co

EXAMPLE: As a smartphone user, we are not aware of internal complexity of smartphone like internal circuits etc., we a user we only use touchscreen and some buttons.

# JVM Architecture

1. **Class loader sub system:** JVM's class loader sub system performs 3 tasks
2. It loads .class file into memory.
3. It verifies byte code instructions.
4. It allots memory required for the program.

* The class loader sub system is responsible for loading the .class file (Byte code) into the JVM.
* Before loading the Byte code into the JVM it will verify where there the Byte code is valid      or   not. This verification will be done by Byte code verifier.
* If the Byte code is valid then the memory for the Byte code will be allocated in different areas.
* The different areas into which the Byte code is loaded are called as Run Time Data Areas. The various run time data areas are.
  1. **Loading:**

The Class loader reads the *.class* file, generate the corresponding binary data and save it in method area. For each *.class* file, JVM stores following information in method area. Fully qualified name of the loaded class and its immediate parent class. Whether *.class* file is related to Class or Interface or Enum Modifier, Variables and Method information etc. After loading *.class* file, JVM creates an object of type Class to represent this file in the heap memory. Please note that this object is of type Class predefined in *java.lang* package. This Class object can be used by the programmer for getting class level information like name of class, parent name, methods and variable information etc. To get this object reference we can use *getClass()*method of [Object](http://www.geeksforgeeks.org/object-class-in-java/) class.

1. **BootStrap ClassLoader:** Responsible for loading classes from the bootstrap classpath, nothing but rt.jar. Highest priority will be given to this loader.
2. **Extension ClassLoader:** Responsible for loading classes which are inside ext folder (jre\lib).
3. **Application ClassLoader:** Responsible for loading Application Level Classpath, path mentioned Environment Variable etc.

### **2.2 Linking:**

1. **Verify:** Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get verification error
2. **Prepare:** For all static variables memory will be allocated and assigned with default values. Static methods as well as static variables are stored in the PermGen section of the heap, now they have introduced a new memory space called METASPACE.
3. **Resolve:** All symbolic memory references are replaced with the original references from Method Area.

### **1.3 Initialization**

This is the final phase of Class Loading, here all [static variable](https://www.javainterviewpoint.com/use-of-static-keyword-in-java/)will be assigned with the original values and [static block](https://www.javainterviewpoint.com/java-static-import/) will be executed.

## **Runtime Data Area**

Runtime Data Area is divided into 5 major components

* 1. **Method area:** In method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.
  2. **Heap Area**: All the Objects and its corresponding instance variables and arrays will be stored here. Heap Area is also one per JVM since Method area and Heap area shares memory for multiple threads the data stored is not thread safe.
  3. **Stack Area:** For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. Stack area is thread safe since it is not a shared resource. Stack Frame is divided into three sub-entities such as
  4. **Local Variable Array** – Related to the method how many local variables are involved and the corresponding values will be stored here.
  5. **Operand stack** – If any intermediate operation is required to perform, operand stack act as runtime workspace to perform the operation.
  6. **Frame data** – All symbols corresponding to the method is stored here. In the case of any exception, the catch block information will be maintained in the frame data.
  7. **PC Registers**: Each thread will have separate PC Registers, to hold address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
  8. **Native Method stacks**: Native Method Stack holds native method information. For every thread, separate native method stack will be created.

1. **Execution Engine**

The bytecode which is assigned to the Runtime Data Area will be executed by the Execution Engine. The Execution Engine reads the byte code and executes one by one.

1. **Interpreter:** Reads the bytecode, interprets it and executes it one by one. The interpreter interprets the bytecode faster but executes slowly. The disadvantage of the interpreter is that when one method called multiple times, every time interpretation is required.
2. **JIT Compiler:** JIT Compiler neutralizes the disadvantage of the Interpreter (a single method called multiple times, each time interpretation is required), The Execution Engine will be using the help of Interpreter in converting but when it found repeated code it uses JIT compiler which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls which improve the performance of the system.
3. **Garbage Collector:** Garbage Collector is a part of Execution Engine, it collects/removes the unreferenced objects. Garbage Collection can be triggered by calling “System.gc()”, but the execution is not guaranteed. Garbage collector of JVM collects only those objects that are created by new keyword. So, if you have created any object without new, you can use finalize method to perform cleanup.



All static & instance variables are stored in PermGen space of heap memory.

As of Java 8, the PermGen space has been replaced with Metaspace. Metaspace by default auto increases its size (up to what the underlying OS provides), while PermGen always has a fixed maximum size. You can set a fixed maximum for Metaspace with JVM parameters, but you cannot make PermGen auto increase.

public void tricky(Point arg1, Point arg2)

{

arg1.x = 100;// we can modify the object it will reflect in caller method.

arg1.y = 100;

arg2 = arg1;// but we can't ressign the object it will not reflect in caller method.

}

1) How does get(Key key) method works internally in HashMap, and Hashtable in Java?

Here are steps, which happens, when you call get() method with key object to retrieve corresponding value from hash based collection

a) Key.hashCode() method is used to find the bucket location in backing array. (Remember HashMap is backed by array in Java) Though hashcode() is not used directly, but they are passed to internal hash() function.

b) In backing array or better known as bucket, key and values are stored in form of a nested class called Entry. If there is only one Entry at bucket location, than value from that entry is returned. Pretty straightforward right?

Things get little tricky, when Interviewer ask second question, What happens if two keys has same hashCode? If multiple keys has same hashCode, than during put() operation collision had occurred, which means multiple Entry object stored in a bucket location. Each Entry keep track of another Entry, forming a [linked list data structure](http://javarevisited.blogspot.com/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html) there. Now, if we need to retrieve value object in this situation, following steps will be followed:

1) Call hashCode() method of key to find bucket location.

2) Traverse thought linked list, comparing keys in each entries using keys.equals() until it return true.

So, we use equals() method of key object to find correct entry and then return value from that. Remember key.equals() method, and this is what Interviewer want to know. I have seen many programmer mentioning value.equals(), which may be due to interview nervousness, but that’s incorrect. Since you don't have value object passed to get() method, there is no question of calling equals and hashCode method on value object.

That's all on these two HashMap questions guys. Remember to mention about key.hashCode() and key.equals(), whenever some one ask how get method of HashMap or Hashtable works in Java. Value object is not used, it's just exist in Entry, so that get can return it.

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.

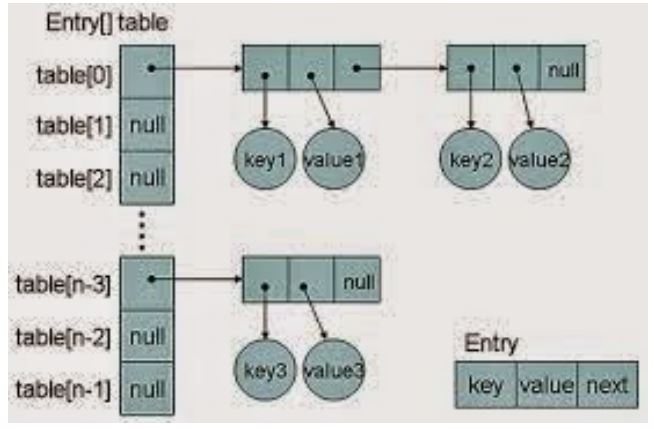
Since the internal array of HashMap is of fixed size, and if you keep storing objects, at some point of time hash function will return same bucket location for two different keys, this is called collision in HashMap. In this case, a linked list is formed at that bucket location and a new entry is stored 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 in lined 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 O(N) to the tree to search in O(logN) time. Java 8 HashMap uses Node and child class TreeNode to construct the red black tree to store the duplicate hashcode elements.

**HashMap works on the 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 as Map.Entry in a bucket which is essential to understand the retrieving logic.   
 If people fail to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in Java HashMap. This answer is very much acceptable and does make sense that interviewee has a fair bit of knowledge on how hashing works and how HashMap  works in Java. But this is just start of story and confusion increases when you put interviewee on scenarios faced by Java developers on day by day basis. Next question could be about collision detection and collision resolution in Java HashMap e.g.

HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but then you need to remind him that there are two Value objects are stored in same bucket , so they will say about [traversal in LinkedList](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html) until we find the value object , HashMap  stores both Key and Value in LinkedList node or as Map.Entry, we will call keys.equals() method to identify a correct node in LinkedList and return associated value object for that key in Java HashMap. Perfect this is the correct answer.



[final object](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) with proper equals() and hashcode() implementation would act as perfect Java HashMap  keys and **improve the performance of Java HashMap  by reducing collision**.

"**What happens On HashMap in Java if the size of the HashMap exceeds a given threshold defined by load factor?** Until you know how HashMap works exactly you won't be able to answer this question. If the size of the Map exceeds a given threshold defined by load-factor e.g. if the 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 the 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 the hash function to find new bucket location.

WHY MAP KEY SHOULD BE IMMUTABLE OBJECT?

If immutable, the object's hashcode won’t change and it allows caching the hashcode of different keys which makes the overall retrieval process very fast. Also for mutable objects, the hashCode() might be dependent on fields that could change, if this happens you won’t be able to find the key (and its value) in the HashMap since hashCode() returns different value.

## PROBLEM WITH MUTABLE KEYS IN HASHMAP:

If key’s hash code changes after the key-value pair (Entry) is stored in HashMap, the map will not be able to retrieve the Entry.

Key’s hashcode can change if the key object is mutable. Mutable keys in HahsMap can result in data loss.

HashMap key Mutable vs Immutable.

See <http://howtodoinjava.com/core-java/collections/how-to-design-a-good-key-for-hashmap/>

Following example will show you the problem with mutable keys in HashMap.3

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import java.util.HashMap;  import java.util.Map;  public class MutableDemo1 {      public static void main(String[] args) {          // HashMap          Map<MutableKey, String> map = new HashMap<>();          // Object created          MutableKey key = new MutableKey(10, 20);          // Insert entry.          map.put(key, "Robin");          // This line will print 'Robin'          System.out.println(map.get(key));          // Object State is changed after object creation.          // i.e. Object hash code will be changed.          key.setI(30);          // This line will print null as Map would be unable to retrieve the          // entry.          System.out.println(map.get(key));      }  } |

**Output: Robin**

**null**

How to fix it?

*Use immutable objects as keys in HashMap. String, Integer etc are considered as good keys for HashMap since objects of these classes are immutable.* We can write your own immutable class also. If mutable objects are used as keys in HashMap, then care should be taken such that change in the key object state should not change its hashcode.

**Can we use any custom object as a key in HashMap?**

This is an extension of previous questions. Of course 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 the custom object is Immutable than this will be already taken care because you cannot change it once created.

**"do you see any problem with resizing of HashMap  in Java"** , you might not be able to pick the context and then he will try to give you hint about multiple thread accessing the Java HashMap and potentially looking for **race condition on HashMap  in Java**.

So the answer is Yes there is potential [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) exists while resizing HashMap in Java, if two [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) at the same time found that now HashMap needs resizing and they both try to resizing. on the process of resizing of HashMap in Java, the element in the bucket which is stored in linked list get reversed in order during their migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at the head *to avoid tail traversing*. If race condition happens then you will end up with an infinite loop. Though this point, you can potentially argue that what the hell makes you think to use Has hMap  in multi-threaded environment to interviewer :)

A couple days ago a coworker and I were working on a piece of code that at first seemed like a mystery to us. We were working with a set. Every attempt at removing the value from the set resulted in false. After hours of looking at the code we discovered that the value was changing, which was the source of our problem. Actual problem was the new Hashcode for the object doesn't match the Hashcode the set used to store the value.

### Source Code with Problem and Results

1. public class SetTest {
2. public static void main(String[] args) {
3. //Both Hash code and equals method
4. // are properly implemented in the Person Class.
5. Set&ltPerson&gt set = new HashSet&ltPerson&gt();
7. Person p1 = new Person("Jain Lokesh", new Date());
8. set.add(p1);
9. Person p2 = new Person("Jain", new Date());
10. set.add(p2);
11. Person p3 = new Person("Lokesh", new Date());
12. set.add(p3);
14. System.out.println(set.size()); //Prints 3
16. //Changing the object values
17. p1.setName(null); p1.setDob(null);
18. p2.setName(null); p2.setDob(null);
19. p3.setName(null); p3.setDob(null);
21. set.add(p1); //P1 added Again
22. System.out.println(set.size()); //Prints 4
24. System.out.println(set.remove(p1)); //Returns true
25. System.out.println(set.remove(p2)); //Returns false
26. System.out.println(set.remove(p3)); //Returns false
27. System.out.println(set.size()); // Prints 3
28. }
29. }

**Key points to remember**:

* Hash set is backed by Hash Map
* Hash Map uses hashing algorithm to store object as a Key value for the Map.

**When hash value is calculated for the Hash map objects**:

* Hash value for an object is calculated at the time we add it into Hash Map as a key. And store this value in the Hash Map as a hashing bucket for this key.
* Any time we try to retrieve values from the Map against the key it does following operations:
* Calculate hash value from the object
* Find the bucket from the hash map for the objects hash value.
* If bucket contains multiple object it uses equals method to compare two objects and return proper key value.
* If bucket only contain one object it return this value directly without any another compare.

**Drawbacks of the mutable objects**:

* If object is updated via any other thread or method the behavior will be unexpected.
* We get unexpected behavior because Hash value for object will be changed even though user is changing same Object which is stored to Map.
* As the hash value changes for the object, we are not able to find the hashing bucket for this object.
* Once object changes some methods start giving unpredictable results.
* Hash Set remove method (Object can’t be removed).
* Hash Set add method (Object added again).
* Hash Set contains method (Returns false for the object)
* Hash Map put (Key will be added again)
* Hash Map remove (Object can’t be removed)
* Hash Map contains (Returns false for the object)

Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz44GTlQs3B>

HOW TO GET KEY FROM VALUE:

//finding key corresponding to value in hashtable - one to one mapping

String key**=** *null*;

String value**=**"Lumia";

**for**(Map**.**Entry entry**:** table**.**entrySet()){

**if**(value**.**equals(entry**.**getValue())){

key **=** entry**.**getKey();

**break**; //breaking because its one to one map

}

}

System**.**out**.**println("got key from value in hashtable key: "**+** key **+**" value: " **+** value);

//finding key corresponding to value in hashtable - one to many mapping

table**.**put("HTC", "Lumia");

Set keys **=** **new** HashSet();

**for**(Map**.**Entry entry**:** table**.**entrySet()){

**if**(value**.**equals(entry**.**getValue())){

keys**.**add(entry**.**getKey()); //no break, looping entire hashtable

}

}

System**.**out**.**println("keys : " **+** keys **+**" corresponding to value in hash table: "**+** value);

# [WHY STATIC CODE ANALYSIS IS IMPORTANT?](http://javarevisited.blogspot.in/2014/02/why-static-code-analysis-is-important.html)

In my opinion, [unit testing](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html), [code review](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) and **static code analysis** makes a nice combo, along with **continuous integration**.

Many organizations, including investment banks are making it mandatory to pass static code analysis test, penetration testing and security testing before you deploy your code in production. Static analysis tools like findbugs and fortify are getting popular

Pen testing is actually more realistic than static code analysis, because test cases are provided by user and they are more close to real world use case scenario, while static code analysis, only look for patterns, which can cause bugs. If there is no pattern, it doesn't mean no bugs, so ideally you need to do both pen testing and static code analysis to push your application in production.

Read more: <http://javarevisited.blogspot.com/2014/02/why-static-code-analysis-is-important.html#ixzz44MikzUAA>

# [TIPS TO IMPROVE PROGRAMMING SKILL AND BECOME BETTER PROGRAMMER, DEVELOPER](http://javarevisited.blogspot.in/2014/01/10-tips-to-improve-programming-skill-become-better-programmer.html)

Read more: <http://javarevisited.blogspot.com/2014/01/10-tips-to-improve-programming-skill-become-better-programmer.html#ixzz44MlosXSc>

# [WHY STRING IS IMMUTABLE OR FINAL IN JAVA](http://javarevisited.blogspot.in/2010/10/why-string-is-immutable-in-java.html)

Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since [HashMap works in the principle of hashing](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.

The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, [*String was made final*](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc. by extending and overriding behaviors. Another reason of *why String class is immutable* could die due to HashMap.

Read more: <http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html#ixzz44NFNX9y0>

GENERICS TUTORIAL

**Type-safety:**

Advantage of Generics in Java is **type-safety**. Collections prior to JDK1.5 are not type-safe because they accept Object type argument which allows them to catch all type of objects instead of only required the type of object.

**No Casting:**

With Generics you don’t need to cast object, Generics will automatically do that for you. For example here is the code for adding and retrieving an element in List with and without Generics in Java:

**List**  items = **new** **ArrayList**();  
items.add("chocolates");  
**String** item = (**String**) items.get(0)  
  
**List**<**String**> items = **new** **ArrayList**();  
items.add("biscuits");  
**String** item = items.get(0) *//no cast required*

Since no cast required, the result is clear and robust code.

**No ClassCastException:**

With Generics compiler ensures that correct types are added into Java collection classes and no cast is required while retrieving an element, so there is no risk of ClassCastException at [runtime](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html).

**Generics wild cards Example in Java**

There are generally two kinds of wildcards in Generics, Bounded and unbounded. Bounded wildcards can be written in two ways to denote upper bound and lower bound. <?> is called unbounded wildcards because it can accept any Type while <? extends T> and <? super T> are bounded wildcards. To know more about them see my post:  [Bounded vs Unbounded wildcards in Generics](http://javarevisited.blogspot.sg/2012/04/what-is-bounded-and-unbounded-wildcards.html) .  
Now let’s see example of different wildcards in Generics:

**<?>**

*"*?" denotes any unknown type, It can represent any Type at in code for. Use this wildcard if you are not sure about Type. for example, if you want to have an ArrayList which can work with any type than declare it as  "ArrayList<?> unknownList" and it can be assigned to any type of ArrayList as shown in following an example of generics in Java:

**ArrayList**<?> unknownList = **new** **ArrayList**<**Number**>();  
unknownList = **new** **ArrayList**<**Float**>();

**<? extends T>**

This is little restrictive than the previous one it will allow All Types which are either "T" or extends T means a subclass of T. for example List<? extends Number> can hold List<Number> or List<Integer>

**ArrayList**<? **extends** **Number**> numberList = **new** **ArrayList**<**Number**>();  
numberList = **new** **ArrayList**<**Integer**>();  
numberList = **new** **ArrayList**<**Float**>();

**<T super ?>**

This is just opposite of previous one, It will allow T and super classes of T, e.g. List<? super Integer>can hold List<Integer> or List<Number>.

**ArrayList**<? **super** **Integer**> numberList = **new** **ArrayList**<**Number**>();  
numberList = **new** **ArrayList**<**Integer**>();  
numberList = **new** **ArrayList**<**Float**>(); *//compilation error*

T – Used to denote type

E – Used to denote element

K – Keys

V - Values

N – for numbers

|  |  |
| --- | --- |
| Generic Term | Meaning |
| Set<E> | Generic Type , E is called formal parameter |
| Set<Integer> | Parametrized type , Integer is actual parameter here |
| <T extends Comparable> | Bounded type parameter |
| <T super Comparable> | Bounded type parameter |
| Set<?> | Unbounded wildcard |
| <? extends T> | Bounded wildcard type |
| <? Super T> | Bounded wildcards |
| Set | Raw type |
| <T extends Comparable<T>> | Recursive type bound |

Read more: <http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html#ixzz44ODaTfbA>

## WHAT IS HEAP SPACE IN JAVA?

When a Java program started Java Virtual Machine gets some memory from Operating System. Java Virtual Machine or JVM uses this memory for all its need and part of this memory is call java heap memory. Heap in Java generally located at bottom of address space and move upwards. whenever we create object using new operator or by any another means object is allocated memory from Heap and When object dies or garbage collected, memory goes back to Heap space in Java.

<http://www.journaldev.com/4098/java-heap-space-vs-stack-memory>

### OUTOFMEMORYERROR IN JAVA HEAP

When JVM starts JVM heap space is equal to the initial size of Heap specified by -Xms parameter, as application progress more objects get created and heap space is expanded to accommodate new objects. JVM also run garbage collector periodically to reclaim memory back from dead objects. JVM expands Heap in Java somewhere near to Maximum Heap Size specified by -Xmx and if there is no more memory left for creating new object in java heap , JVM throws java.lang.OutOfMemoryError and your application dies. Before throwing [OutOfMemoryError No Space in Java Heap](http://javarevisited.blogspot.sg/2011/09/javalangoutofmemoryerror-permgen-space.html), JVM tries to run garbage collector to free any available space but even after that not much space available on Heap in Java it results into OutOfMemoryError. To resolve this error you need to understand your application object profile i.e. what kind of object you are creating, which objects are taking how much memory etc. you can use profiler or heap analyzer to troubleshoot OutOfMemoryError in Java. "java.lang.OutOfMemoryError: Java heap space" error messages denotes that Java heap does not have sufficient space and cannot be expanded further while "java.lang.OutOfMemoryError: PermGen space" error message comes when the permanent generation of Java Heap is full, the application will [fail to load a class](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html) or to allocate an interned string.

## HOW TO INCREASE HEAP SIZE IN JAVA

Default size of Heap space in Java is 128MB on most of 32 bit Sun's [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) but its highly varies from JVM to JVM e.g. default maximum and start heap size for the 32-bit Solaris Operating System (SPARC Platform Edition) is -Xms=3670K and -Xmx=64M and Default values of heap size parameters on 64-bit systems have been increased up by approximately 30%.

By the way you can increase size of java heap space based on your application need and I always recommend this to avoid using default JVM heap values. if your application is large and lots of object created you can change size of heap space by using JVM options -Xms and -Xmx. Xms denotes starting size of Heap while -Xmx denotes maximum size of Heap in Java. There is another parameter called -Xmn which denotes Size of new generation of Java Heap Space. Only thing is you cannot change the size of Heap in Java dynamically, you can only provide Java Heap Size parameter while starting JVM. I have shared some more useful JVM options related to Java Heap space and Garbage collection on my post [10 JVM options Java programmer must know](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html), you may find useful.

### JAVA HEAP DUMP

Java Heap dump is a snapshot of Java Heap Memory at a particular time. This is very useful to analyze or troubleshoot any memory leak in Java or any java.lang.OutOfMemoryError. There are tools available inside JDK which helps you to take heap dump and there are heap analyzer available tool which helps you to analyze java heap dump. You can use "jmap" command to get java heap dump, this will create heap dump file and then you can use "jhat - Java Heap Analysis Tool" to analyze those heap dumps. You should also read [Java Performance The Definitive Guide By Scott Oaks](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20) to learn more about Java performance tuning and profiling. It is one of the must read Java performance book for any senior Java developers.

## POINTS ABOUT JAVA HEAP SPACE

1. Java Heap Memory is part of memory allocated to JVM by Operating System.
2. Whenever we create objects they are created inside Heap in Java.
3. Java Heap space is divided into three regions or generation for sake of garbage collection called New Generation, Old or tenured Generation or Perm Space. Permanent generation is garbage collected during full GC in hotspot JVM.
4. You can increase or change size of Java Heap space by using JVM command line option -Xms, -Xmx and -Xmn. don't forget to add word "M" or "G" after specifying size to indicate Mega or Gig. For example you can set java heap size to 258MB by executing following command java -Xmx256m HelloWord.
5. You can use either JConsole or Runtime.maxMemory(), Runtime.totalMemory(), Runtime.freeMemory() to query about Heap size programmatic in Java. See my post [How to find memory usage in Java program](http://javarevisited.blogspot.sg/2012/01/find-max-free-total-memory-in-java.html) for more details.
6. You can use command "jmap" to take Heap dump in Java and "jhat" to analyze that heap dump.
7. Java Heap space is different than Stack which is used to store call hierarchy and local variables.
8. Java Garbage collector is responsible for reclaiming memory from dead object and returning to Java Heap space.
9. Don’t panic when you get java.lang.OutOfMemoryError, sometimes it’s just matter of increasing heap size but if its recurrent then look for [memory leak in Java](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html).
10. Use Profiler and Heap dump Analyzer tool to understand Java Heap space and how much memory is allocated to each object.

Read more: <http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html#ixzz44SDup4FE>

# [HOW GARBAGE COLLECTION WORKS IN JAVA](http://javarevisited.blogspot.in/2011/04/garbage-collection-in-java.html)

Garbage collection works by employing several GC algorithm e.g. Mark and Sweep. There are different kinds of garbage collector available in Java to collect different area of heap memory e.g. you have serial, parallel and concurrent garbage collector in Java.

A new collector called G1 (Garbage first) are also introduced in JDK 1.7.  The first step to learning about GC is to understand when an object becomes eligible to garbage collection? Since JVM provides memory management, Java developers only care about creating an object, they don't care about cleaning up, that is done by the garbage collector, but it can only collect objects which have no live strong reference or it's not reachable from any thread.

## IMPORTANT POINTS ABOUT GARBAGE COLLECTION IN JAVA

* 1. Objects are created on the heap in Java irrespective of their scope e.g. local or member variable. while it's worth noting that class variables or static members are created in method area of [Java memory space](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) and both heap and method area is shared between different thread.
  2. Garbage collection is a mechanism provided by Java Virtual Machine to reclaim heap space from objects which are eligible for Garbage collection.
  3. Garbage collection relieves Java programmer from memory management which is an essential part of C++ programming and gives more time to focus on business logic.
  4. Garbage Collection in Java is carried by a daemon thread called Garbage Collector.
  5. Before removing an object from memory garbage collection thread invokes [finalize() method](http://javarevisited.blogspot.com/2012/03/finalize-method-in-java-tutorial.html) of that object and gives an opportunity to perform any sort of cleanup required.
  6. As a Java programmer you can’t force garbage collection in Java; it will only trigger if JVM thinks it needs a garbage collection based on Java heap size.
  7. There are methods like System.gc() and Runtime.gc() which is used to send request of Garbage collection to JVM but it’s not guaranteed that garbage collection will happen.
  8. If there is no memory space for creating a new object in Heap Java Virtual Machine throws OutOfMemoryError or [java.lang.OutOfMemoryError heap space](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html)

## WHEN AN OBJECT BECOMES ELIGIBLE FOR GARBAGE COLLECTION

An object becomes eligible for Garbage collection or GC if it's not reachable from any live threads or by 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 the reference so if object A has a reference to object B and object B has a reference to 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 to that object explicitly set to null e.g. object = null
2. The 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 the reference to 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 [lived weak references](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html) via WeakHashMap it will be eligible for garbage collection.

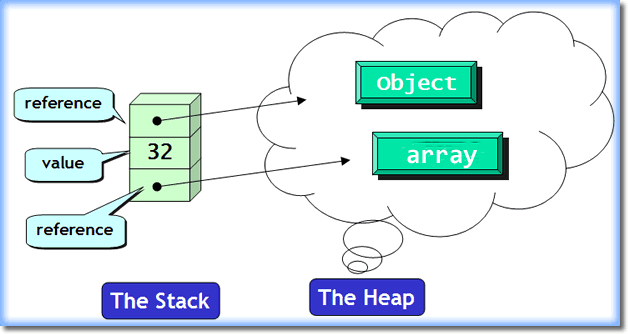
### SUMMARY ON GARBAGE COLLECTION IN JAVA

1. Java Heap is divided into three generation for the sake of garbage collection. These are a young generation, tenured or old generation, and Perm area.
2. New objects are created by young generation and subsequently moved to the old generation.
3. String pool is created in [PermGen area of Heap](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html#uds-search-results), garbage collection can occur in perm space but depends upon JVM to JVM. By the way from JDK 1.7 update, String pool is moved to heap area where objects are created.

Read more: <http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html#ixzz44ZMXe3D3>

## DIFFERENCE BETWEEN STACK VS HEAP IN JAVA

1. The main difference between heap and stack is that stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call while heap memory is used to store objects in Java. No matter, where the object is created in code e.g. as a member variable, local variable or class variable, they are always created inside heap space in Java.
2. Each [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has their own stack which can be specified using -Xss JVM parameter, similarly, you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of the heap and -Xmx is a maximum size of java heap. to learn more about JVM options see my post [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html).



1. If there is no memory left in the stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating an object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space. Read more about how to deal with java.lang.OutOfMemoryError in my post [2 ways to solve OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).
2. If you are using [Recursion](http://javarevisited.blogspot.com/2012/12/recursion-in-java-with-example-programming.html), on which method calls itself, you can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is a lot lesser than the size of heap memory in Java.
3. Variables stored in stacks are only visible to the owner Thread while objects created in the heap are visible to all thread. In other words, stack memory is kind of private memory of Java Threads while heap memory is shared among all threads.

Read more: <http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html#ixzz44ZTyJfno>

# [HOW TO CHECK IF STRING IS NOT NULL AND EMPTY IN JAVA?](http://javarevisited.blogspot.in/2016/01/how-to-check-if-string-is-not-null-and-empty-in-java-example.html)

**1st solution - using isEmpty() method**

isEmpty() method is available from Java SE 6 onwards

if(stirng != null && !string.isEmpty()){

System.out.println("String is not null and not empty");

}

**2nd solution - Using length() function**  
if(stirng != null && string.length() > 0){

System.out.println("String is not null and not empty");

}

**3rd solution - Using trim() method**  
if(stirng != null && string.trim().length() > 0){

System.out.println("String is not null and not empty");

}

# DIFFERENCE BETWEEN STRING LITERAL AND NEW STRING OBJECT IN JAVA

String strObject = new String("Java");

and

String strLiteral = "Java";

Both expression gives you String object, but there is subtle difference between them. When you create String object using new() operator, it always create a new object in [heap memory](http://java67.blogspot.sg/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html). On the other hand, if you create object using String literal syntax e.g. "Java", it may return an existing object from String pool (a cache of String object in Perm gen space, which is now moved to heap space in recent Java release), if it's already exists. Otherwise it will create a new string object and put in string pool for future re-use.

## WHAT IS STRING LITERAL AND STRING POOL

Since String is one of the most used type in any application, Java designer took a step further to optimize uses of this class. They know that Strings will not going to be cheap, and that's why they come up with an idea to cache all String instances created inside double quotes e.g. "Java". These double quoted literal is known as *String literal* and the cache which stored these String instances are known as *String pool*. In earlier version of Java, I think up-to Java 1.6 String pool is located in permgen area of heap, but in Java 1.7 updates its moved to main heap area. Earlier since it was in PermGen space, it was always a risk to create too many String object, because it’s a very limited space, default size 64 MB and used to store class metadata e.g. .class files. Creating too many String literals can cause [java.lang.OutOfMemory: permgen space](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html). Now because String pool is moved to a much larger memory space, it's much safer. By the way, don't misuse memory here, always try to minimize temporary String object e.g. "a", "b" and then "ab". Always use StringBuilder to deal with temporary String object.

### STRING INTERNING USING INTERN() METHOD

Java by default doesn't put all String object into String pool, instead they gives you flexibility to explicitly store any arbitrary object in String pool. You can put any object to String pool by calling intern() method of java.lang.String class. Though, when you create using *String literal notation* of Java, it automatically call intern() method to put that object into String pool, provided it was not present in the pool already. This is another *difference between string literal and new string*, because in case of new, interning doesn't happen automatically, until you call intern() method on that object. Also don't forget to use [StringBuffer](http://java67.blogspot.sg/2012/08/difference-between-string-and-stringbuffer-in-java.html) and [StringBuilder](http://java67.blogspot.sg/2014/05/difference-between-stringbuilder-and-StringBuffer-java.html) for string concatenation, they will reduce number.

# [WHAT IS AUTOBOXING AND UNBOXING IN JAVA](http://javarevisited.blogspot.in/2012/07/auto-boxing-and-unboxing-in-java-be.html)

Autoboxing and unboxing is introduced in Java 1.5 to automatically convert primitive type into Object or Wrapper class type called **autoboxing**, in opposite case is called **unboxing**.

**ArrayList**<**Integer**> intList = **new** **ArrayList**<**Integer**>();  
intList.add(1); *//autoboxing - primitive to object*  
intList.add(2); *//autoboxing*  
        
**int** number = intList.get(0); *// unboxing*  
**int** local = intLocal.get(); *// unboxing in Java*

*Or*

*Integer a = 2; //autoboxing - primitive to object*

*int s = a;// unboxing*

*or*

**public** **static** **Integer** show(**Integer** iParam){  
   **System**.out.println("autoboxing example - method invocation i: " + iParam);  
   **return** iParam;  
}

*//autoboxing and unboxing in method invocation*  
show(3); *//autoboxing*  
**int** result = show(3); *//unboxing because return type of method is Integer*

Read more: <http://javarevisited.blogspot.com/2012/07/auto-boxing-and-unboxing-in-java-be.html#ixzz44Ztoq1a2>

## COMPARING INTEGER OBJECT WITH == IN JAVA

Some of the JVM cache objects of some wrapper class e.g. Integer from -128 to 127 and return same object which if compare via “ ==” can return true but after this range this validity doesn't work and to make it worse this behavior is JVM dependent so better avoid this kind of check and use equals() method. e.g.

Integer i1 = **260**;

Integer i2 = **260**;

**if** (i1 == i2) {

System.out.println("i1 and i2 is equal");

} **else** {

System.out.println("i1 and i2 is not equal ");

}

Here you will most probably get **"i1 and i2 is not equal”** at least in my machine.  
Because in this case, unboxing operation is not involved. The literal 260 is boxed into two different Integer objects (again it varies between JVM to JVM), and then those objects are compared with ==. The result is false, as the two objects are different instances, with different memory addresses. Because both sides of the == expression contain objects, no unboxing occurs.

Integer i1 = **100**;

Integer i2 = **100**;

**if** (i1 == i2) {

System.out.println("i1 and i2 is equal");

} **else** {

System.out.println("i1 and i2 is not equal ");

}

Here, most probably you will get the text "i1 and i2 is equal". Because int values from -127 to 127 are in a range which most JVM will like to cache so the VM actually uses the same object instance (and therefore memory address) for both i1 and i2. As a result, == returns a true result.

IS STRING THREAD-SAFE IN JAVA?

If you are familiar with the concept of immutability and [thread-safety](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html) you can easily answer this String interview question in Java. Since [String is immutable](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html), it is thread-safe and it can be shared between multiple threads without external synchronization.

FINDING SUBSTRING IN STRING

String s = "abc xyz bbc anz";

System.***out***.println(s.contains("abc"));

System.***out***.println(s.indexOf("xyz") != -1);

[HOW DOES HASHMAP OR LINKEDHAHSMAP HANDLES COLLISIONS?](http://javarevisited.blogspot.in/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html)

A collision occurs when a hash function returns same bucket location for two different keys. Since all hash based Map class e.g. HashMap uses [equals() and hashCode() contract](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html) to find the bucket. HashMap calls the hashCode() method to compute the hash value which is used to find the bucket location.

1. HashMap handles collision by using linked list to store map entries ended up in same array location or bucket location.
2. From Java 8 onwards, HashMap, ConcurrentHashMap, and LinkedHashMap will use the balanced tree in place of [linked list](http://java67.blogspot.com/2016/01/how-to-implement-singly-linked-list-in-java-using-generics-example.html) to handle frequently hash collisions. The idea is to switch to the balanced tree once the number of items in a hash bucket grows beyond a certain threshold. This will improve the worst case get() method performance from O(n) to O(log n).
3. By switching from linked list to balanced tree for handling collision, the iteration order of HashMap will change. This is Ok because HashMap doesn't provide any guarantee on iteration order and any code which depends upon that are likely to break.
4. Legacy class Hashtable which exists in JDK from Java 1 will not use the balanced binary tree to handle frequent hash collision to keep its iteration order intact. This was decided to avoid breaking many legacy Java application which depends upon iteration order of Hashtable.
5. Apart from Hashtable, WeakHashMap and IdentityHashMap will also continue to use the linked list for handling collision even in the case of frequent collisions.
6. Collision in [HashMap](http://java67.blogspot.com/2016/01/how-to-initialize-hashmap-with-values-in-java.html) is possible because hash function uses hashCode() of key object and equals() and hashCode() contract doesn't guarantee different hashCode for different objects. Remember, they guarantee same hash code for the equal object but not the vice-versa.
7. A collision will occur on Hashtable or HashMap when hashCode() method of two different key objects will return same values.

# [FAIL-SAFE VS FAIL-FAST ITERATOR IN JAVA](http://javarevisited.blogspot.in/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html)

## ***Fail-fast Iterators in Java:***

As name suggest **fail-fast Iterators** fail as soon as they realized that *structure of Collection has been changed since iteration has begun*. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection. Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

## ***Fail-safe Iterator in java:***

Contrary to fail-fast Iterator, fail-safe iterator doesn't throw any Exception if Collection is modified structurally while one thread is Iterating over it because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator. Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException in Java.

[TOP 5 CONCURRENT COLLECTIONS FROM JDK 5 & 6 JAVA PROGRAMMER SHOULD KNOW](http://javarevisited.blogspot.in/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html)

java.util.concurrent package.

1. ConcurrentHashMap

ConcurrentHashMap is undoubtedly most popular collection class introduced in Java 5 and most of us are already using it. ConcurrentHashMap provides a concurrent alternative of [Hashtable or Synchronized Map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) classes with aim to support higher level of concurrency by implementing fined grained locking. Multiple reader can access the Map concurrently while a portion of Map gets locked for write operation depends upon concurrency level of Map. ConcurrentHashMap provides better scalability than there synchronized counterpart. [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) of ConcurrentHashMap are [fail-safe iterators](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) which doesn't throw ConcurrencModificationException thus eliminates another requirement of locking during iteration which result in further scalability and performance. ConcurrentHashMap does not allow NULL single key and values. HashMap allows single null key and value.

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 a different lock. This means, 16 thread can operate on Map simultaneously until they are operating on different part of Map. This makes ConcurrentHashMap high performance despite keeping thread-safety intact.

### **ConcurrentHashMap:**

* You should use ConcurrentHashMap when you need very high concurrency in your project.
* It is thread safe without synchronizing the whole map.
* Reads can happen very fast while write is done with a lock.
* There is no locking at the object level.
* The locking is at a much finer granularity at a hashmap bucket level.
* ConcurrentHashMap doesn’t throw a ConcurrencModificationException if one thread tries to modify it while another is iterating over it. Its fail safe.
* ConcurrentHashMap uses multitude of locks.

### **SynchronizedHashMap:**

* Synchronization at Object level.
* Every read/write operation needs to acquire lock.
* Locking the entire collection is a performance overhead.
* This essentially gives access to only one thread to the entire map & blocks all the other threads.
* It may cause contention.
* SynchronizedHashMap returns Iterator, which fails-fast on concurrent modification.

### **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 the update operation, ConcurrentHashMap only locks a portion of Map instead of whole Map.
3. The concurrent update is achieved by internally dividing Map into the small portion which is defined by concurrency level.
4. Choose concurrency level carefully as a significantly higher number can be a waste of time and space and the lower number may introduce thread contention in case writers over number 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 ConcurrencModificationException. 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, the concurrent read may only reflect insertion or deletion of some entries.

2. CopyOnWriteArrayList and CopyOnWriteArraySet

## **Difference between CopyOnWriteArrayList and Synchronized** **ArrayList in Java:**

1) First difference between Synchronized ArrayList and CopyOnWriteArrayList comes from the fact how they achieve thread safety. Synchronized List locks the whole list to provide synchronization and thread-safety, while CopyOnWriteArrayList doesn't lock the list and when a Thread writes into the list it simply replace the list by copying. This way it provides concurrent access to the list to multiple threads without locking and since read is a thread-safe operation and no two thread writes into list anytime. The add method will always create a copy of the existing array and do the modification on the copy and then finally update the volatile reference of the array to point to this new array. And that's why we have the name "CopyOnWriteArrayList" - makes copy when you write into it.. This also avoids the ConcurrentModificationException.

2) The second difference between them comes from the fact how their iterator behave. The Iterator returned from synchronized ArrayList is a fail fast but iterator returned by CopyOnWriteArrayList is a fail-safe iterator i.e. it will not throw ConcurrentModifcationException even when the list is modified when one thread is iterating over it. If you want to learn more about fail safe and fail-fast iterator, I suggest you read my earlier post understanding [difference between fail-safe and fail-fast iterator in Java](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html).

**3. BlockingQueue**

BlockingQueue is also one of better known collection class in Java 5. BlockingQueue makes it easy to implement [producer-consumer design pattern](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html) by providing inbuilt blocking support for put() and take() method. put() method will block if Queue is full while take() method will block if Queue is empty. Java 5 API provides two concrete implementation of BlockingQueue in form of [ArrayBlockingQueue and LinkedBlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html), both of them implement FIFO ordering of element. ArrayBlockingQueue is backed by Array and it’s bounded in nature while LinkedBlockingQueue is optionally bounded. Consider using BlockingQueue to solve producer Consumer problem in Java instead of writing your won [wait-notify code](http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html). In terms of throughput LinkedBlockingQueue provides higher throughput than ArrayBlockingQueue in Java. Java 5 also provides PriorityBlockingQueue, another implementation of BlockingQueue which is ordered on priority and useful if you want to process elements on order other than FIFO.

**Usage of BlockingQueue in Java:**

There can be many creative usage of BlockingQueue in Java given its flow control ability. Two of the most common ways I see programmer uses BlockingQueue is to implement Producer Consumer design pattern and implementing Bounded buffer in Java. It surprisingly made coding and inter thread communication over a shared object very easy.

If you are implementing [Producer Consumer design pattern in Java](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html), consider using BlockingQueue, it not only make coding easy but also performs better and provide better robustness and stability than writing your own BlockingQueue or using naked [wait and notify method](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html).

**4. Deque and BlockingDeque**

Deque interface is added in Java 6 and it extends Queue interface to support insertion and removal from both end of Queue referred as head and tail. Java6 also provides concurrent implementation of Deque like ArrayDeque and LinkedBlockingDeque. Deque Can be used efficiently to increase parallelism in program by allowing set of [worker thread](http://javarevisited.blogspot.sg/2013/01/threadlocal-memory-leak-in-java-web.html) to help each other by taking some of work load from other thread by utilizing Deque double end consumption property. So if all [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has their own set of task Queue and they are consuming from head; helper thread can also share some work load via consumption from tail.

**5. ConcurrentSkipListMap and ConcurrentSkipListSet**

Just like [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) provides a concurrent alternative of [synchronized HashMap](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html). ConcurrentSkipListMap and ConcurrentSkipListSet provide concurrent alternative for synchronized version of SortedMap and SortedSet. For example instead of using TreeMap or TreeSet wrapped inside synchronized Collection, You can consider using ConcurrentSkipListMap or ConcurrentSkipListSet from java.util.concurrent package. They also implement NavigableMap and NavigableSet to add additional navigation method we have seen in our post [How to use NavigableMap in Java](http://javarevisited.blogspot.sg/2013/01/what-is-navigablemap-in-java-6-example-submap-head-tail.html).

[HOW TO USE CALLABLE AND FUTURE IN JAVA](http://javarevisited.blogspot.in/2015/06/how-to-use-callable-and-future-in-java.html)

1. In Runnable interface, run cannot return a value or throw checked exception. Callable interface solves this problem.
2. Callable expects a main entry point called ‘call’ method and can return a value. Also it can throw an Exception.
3. Future represents the lifecycle of a task and provides methods to test whether a task is completed or been canceled, retrieve its result and cancel the task.

### IMPORTANT POINTS ABOUT CALLABLE AND FUTURE

1. Callable is a SAM type interface, so it can be used in lambda expression.
2. Callable has just one method call() which holds all the code needs to executed asynchronously.
3. In Runnable interface, there was no way to return the result of computation or throw checked exception but with Callable you can both return a value and can throw [checked exception](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html).
4. You can use get() method of Future to retrieve result once computation is done. You can check if computation is finished or not by using isDone() method.
5. You can cancel the computation by using Future.cancel() method.
6. get() is a blocking call and it blocks until computation is completed.

Read More: <http://javarevisited.blogspot.in/2015/06/how-to-use-callable-and-future-in-java.html>

3 WAYS TO FIND DUPLICATE ELEMENTS IN ARRAY

**Brute force method**

**for** (**int** i = 0; i < names.length; i++) {

**for** (**int** j = i + 1; j < names.length; j++) {

**if** (names[i].equals(names[j])) {

// got the duplicate element

}

}

}

**Using Set data structure**

**for** (String name : names) {

**if** (set.add(name) == **false**) {

// your duplicate element

}

}

**Using Hashtable or HashMap**

Map<String, Integer> nameAndCount = **new** HashMap<>();

// build hash table with count

**for** (String name : names) {

Integer count = nameAndCount.get(name);

**if** (count == **null**) {

nameAndCount.put(name, 1);

} **else** {

nameAndCount.put(name, ++count);

}

}

// Print duplicate elements from array in Java

Set<Entry<String, Integer>> entrySet = nameAndCount.entrySet();

**for** (Entry<String, Integer> entry : entrySet) {

**if** (entry.getValue() > 1) {

System.***out***.println("Duplicate element from array : "

+ entry.getKey());

}

}

## DIFFERENCE BETWEEN TREESET, LINKEDHASHSET AND HASHSET IN JAVA

TreeSet, LinkedHashSet and HashSet in Java are three Set implementation in collection framework and like many others they are also used to store objects. Main feature of TreeSet is sorting, LinkedHashSet is insertion order and HashSet is just general purpose collection for storing object. HashSet is implemented using [HashMap in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) while TreeSet is implemented using [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html).  TreeSet is a SortedSet implementation which allows it to keep elements in the sorted order defined by either [Comparable or Comparator interface](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html). Comparable is used for natural order sorting and Comparator for [custom order sorting](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html) of objects, which can be provided while creating instance of TreeSet. Anyway before seeing difference between TreeSet, LinkedHashSet and HashSet, let's see some similarities between them:

1. **Duplicates**: All three implements Set interface means they are not allowed to store duplicates.
2. **Thread safety:** HashSet, TreeSet and LinkedHashSet are not [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), if you use them in multi-threading environment where at least one Thread modifies Set you need to externally synchronize them.
3. **Fail-Fast Iterator**: Iterator returned by TreeSet, LinkedHashSet and HashSet are fail-fast Iterator. I.e. If Iterator is modified after its creation by any way other than Iterators remove() method, it will throw ConcurrentModificationException with best of effort. Read more about [fail-fast vs fail-safe Iterator](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) here

Now let’s see **difference between HashSet, LinkedHashSet and TreeSet in Java**:

**Performance** and **Speed**: First difference between them comes in terms of speed.  HashSet is fastest, LinkedHashSet is second on performance or almost similar to HashSet but TreeSet is bit slower because of sorting operation it needs to perform on each insertion. TreeSet provides guaranteed O(log(n)) time for common operations like add, remove and contains, while HashSet and LinkedHashSet offer constant time performance e.g. O(1) for add, contains and remove given hash function uniformly distribute elements in bucket.

**Ordering:** HashSet does not maintain any order while LinkedHashSet maintains insertion order of elements much like List interface and TreeSet maintains sorting order or elements.

**Internal Implementation:** HashSet is backed by a HashMap instance, LinkedHashSet is implemented using HashSet and LinkedList while TreeSet is backed up by NavigableMap in Java and by default it uses TreeMap.

**null**: Both HashSet and LinkedHashSet allows null but TreeSet doesn't allow null but TreeSet doesn't allow null and throw [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you will insert null into TreeSet. Since TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) of respective elements to compare them which throws NullPointerException while comparing with null, here is an example:

TreeSet cities

Exception in thread "main" java.lang.NullPointerException

at java.lang.String.compareTo(String.java:1167)

at java.lang.String.compareTo(String.java:92)

at java.util.TreeMap.put(TreeMap.java:545)

at java.util.TreeSet.add(TreeSet.java:238)

**Comparison:** HashSet and LinkedHashSet uses [equals() method in Java](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) for comparison but TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) for maintaining ordering. That's why compareTo() should be consistent to equals in Java. Failing to do so break general contact of Set interface i.e. it can permit duplicates.

**Why ConcurrentHashMap is faster than Hashtable in Java?**

ConcurrentHashMap is introduced as alternative of [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) in Java 5, it is faster because of it's design. ConcurrentHashMap divides whole map into different segments and only lock a particular segment during update operation, instead of Hashtable, which locks whole Map. ConcurrentHashMap also provides lock free read, which is not possible in Hashtable, because of this and lock striping, ConcurrentHashMap is faster than Hashtable, especially when number of reader is more than number of writers. In order to better answer this popular Java concurrency interview questions, I suggest reading my post about i[nternal working of ConcurrentHashMap in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html).

**Difference between wait and sleep in Java?**

Key point to mention while answering this question is to mention that wait will release lock and must be called from synchronized context, while sleep will only pause thread for some time and keep the lock. By the way, both throws IntrupptedException and can be interrupted, which can lead to some follow-up questions like, can we awake a sleeping or waiting thread in Java? You can also read detailed answer on my post of same title [here](http://java67.blogspot.com/2012/08/what-are-difference-between-wait-and.html).

**How do you solve producer consumer problem in Java?**

One of my favorite questions during any Java multithreading interview, almost half of the concurrency problems can be categorized in producer consumer pattern. There are basically two ways to solve this problem in Java, One by [using wait and notify method](http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html) and other by using BlockingQueue in Java. Later is easy to implement and good choice if you are coding in Java 5. Key points to mention, while answering this question is thread safety and blocking nature of BlockingQueue and how that helps, while writing concurrent code. You can also expect lots of follow-up questions including, what happen if you have multiple producer or multiple consumer, what will happen if producer is faster than consumer thread or vice-versa. You can also see this link for example of [how to code producer consumer design in Java using blocking queue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html).

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

By the way both are used to submit task to thread pool in Java. As you see from the JavaDoc execute(Runnable) does not return anything.

However, submit (Callable<T>) returns a Future object which allows a way for you to programmatically cancel the running thread later as well as get the T that is returned when the Callable completes. See [JavaDoc of Future](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) for more details

Future<?> future = executor.submit(longRunningJob);

//long running job is taking too long

future.cancel(true);

future.get()// to get value returned by Callable thread.

**How do you share data between two threads in Java?**

One more Java multithreading question from telephonic round of interview. You can share data between thread by using shared object or shared data structures like Queue. Depending upon, what you are using, you need to provide thread-safety guarantee, and one way of providing thread-safety is using synchronized keyword. If you use concurrent collection classes from Java 5 e.g. [BlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html), you can easily share data without being bothered about thread safety and inter thread communication. I like this thread question, because of its simplicity and effectiveness. This also leads further follow-up questions on issues which arises due to sharing data between threads e.g. race conditions.

**What is Difference between Iterator and Enumeration in Java?**

One of the classic interview Questions asked on Java collection framework, this is pretty old and programmer who has been working in Java for 4 to 6 years must have seen this question before. Well [Iterator and ListIterator in Java](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) is a new way to iterator collection in Java and provides ability to remove object while traversing while Enumeration doesn't allow you to remove object. See [Iterator vs Enumeration in Java](http://javarevisited.blogspot.sg/2010/10/what-is-difference-between-enumeration.html) for more differences between both of them.

## TOP 5 NEW FEATURES IN JDK 1.7

* 1. **String in Switch case**

You cannot use String in Switch or case statement prior to Java 7. Talk of allowing String in Switch case was growing ever since [Enum](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) was introduced in Java 5, though you can use [Enum in Switch](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html) case, you cannot do the same with String which is more common than Enum in Java programming language. From JDK 7 you can use String in Switch case, see [How to use String in Switch case in Java7](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html) for more details.

* 1. **Multiple Exception in one catch block**

This is the first feature which comes in my mind when people ask **what is new in Java 7 feature**. Can you name few new feature introduced in JDK 7 release? From JDK 7 you can catch multiple Exception in one catch block. This is called multi-catch block feature of Java 7. This is a significant improvement Java 7 made in Exception handling because it will reduce clutter in Exception handling code in Java. In order to learn [how to catch multiple Exception in once catch block in Java 7](http://javarevisited.blogspot.sg/2011/07/jdk7-multi-cache-block-example-tutorial.html), see that article.

Syntax: **catch** (**exception1 | exception2 e**) {

    }

* 1. **Automatic resource management or ARM blocks**

Similar to previous Java 7 feature, Automatic resource management (ARM blocks) also known as try with resource block is another significant improvement in area of Exception handling in Java. ARM blocks allows automatic closing of resource e.g. Files, Streams etc. opened inside try block. Resource needs to implement Closeable interface in Java. See [How to use ARM block in Java 7](http://javarevisited.blogspot.sg/2011/09/arm-automatic-resource-management-in.html) for more details.

Syntax: **try** (**resource1 ; resource2 e**) {

    }

Whatever resource we are using should be subtypes ofAutoCloseable otherwise will get compile time error. **java.lang.AutoCloseable**, interface has been added in API which contains single method **close() throws Exception** this interface is a parent of **java.io.closeable** interface so all the input and output devices inherit this property.

* 1. **Diamond operator <> for type inference**

**Why wait and notify called from synchronized method in Java?**

Another tough core Java question for wait and notify. They are called from [synchronized method or synchronized block](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) because wait and modify need monitor on Object on which wait or notify get called. See [Why wait and notify require synchronized context](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html) for complete answer of this *tough and tricky Java multi-threading question*.

**How do you prevent SQL Injection in Java Code?**

You can [use PreparedStatement to avoid SQL injection](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html) in Java code. Use of the PreparedStatement for executing SQL queries not only provides better performance but also shield your Java and J2EE application from SQL Injection attack.

**How does ConcurrentHashMap achieves its Scalability?**

The java.util.ConcurrentHashMap class solves this problem by using *lock stripping* technique, where the whole map is locked at different segments and only a particular segment is locked during the write operation, not the whole map. The ConcurrentHashMap also achieves its scalability by allowing lock-free reads as read is a thread-safe operation.  See [here](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html) for more advanced multi-threading and concurrency questions in Java.

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

# [DIFFERENCE BETWEEN TRUSTSTORE AND KEYSTORE IN JAVA - SSL](http://javarevisited.blogspot.in/2012/09/difference-between-truststore-vs-keyStore-Java-SSL.html)

Only *difference between trustStore and keyStore* is what they store and there purpose. In SSL handshake purpose of **trustStore is to verify credentials** and purpose of **keyStore is to provide credential**. KeyStore in Java stores private key and certificates corresponding to their public keys and require if you are SSL Server or SSL requires client authentication. Keytool is used to access keystore in Java and by using keytool you can list, add certificates from keystore.

### **Example of adding Certificate on Java KeyStore:**

Now let's see example of *adding certificates into key store in Java:*

1. Get Certificate: easier way is to point your browser to that URL and when certificate is presented save it on your

local folder or directory say in C:/certificates/test.cer

2. Now go to Security folder of your JRE installation directory. id you have JDK installed then it would be

something like C:/Program Files/Java//jdk1.6.0\_20/jre/lib/security

3 Execute following keytool command to insert certificate into keystore

keytool -import -keystore cacerts -file test.cer

Read more: <http://javarevisited.blogspot.com/2012/09/difference-between-truststore-vs-keyStore-Java-SSL.html#ixzz45sHUdkrf>

# [**HOW TO RESET ARRAYLIST IN JAVA**](http://javarevisited.blogspot.in/2015/09/how-to-reset-arraylist-in-java-clear-vs-removeAll-example.html)

There are two ways to reset an ArrayList in Java, by using clear() method or calling removeAll(). If your ArrayList is small enough e.g. contains only 10 or 100 elements then you can use any of these two methods without worrying too much, but, if you have a huge list of lots of objects e.g. an ArrayList containing 10M entries, then choice of clear() vs removeAll() can make a huge difference in performance of your Java application. Sometimes it's even better to create a new ArrayList instead of resetting the old one, especially if resetting takes a long time, but this also has a caveat, you need to make sure that old ArrayList is eligible for garbage collection, otherwise there is a huge risk of [java.lang.OutOfMemoryError: Java Heap Space](http://java67.blogspot.sg/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html).

Coming back to clear() vs removeAll() method, you should always use clear(), because it gives you O(n) performance, while removeAll(Collection c) is worse, it gives O(n^2) performance, that's why you see huge difference in time taken by clearing a large ArrayList by these two methods.

Read more: <http://javarevisited.blogspot.com/2015/09/how-to-reset-arraylist-in-java-clear-vs-removeAll-example.html#ixzz46RaPNsfz>

### JDK 1.7.0.\_40 UPDATE:

From above version of java ArrayList and HashMap is initialized with empty default size. Instead of ArrayList default size 10 and HashMap default size 16.

## How to Disable Submit Button Using JavaScript

You don't need to do a lot just add this.disabled='disabled' on onclick event handler of button like below:

<form action="submit.jsp" method="post" >

<input type="submit" name="SUBMIT" value="Submit Form" onclick="this.disabled='disabled'" />

</form>

This JavaScript code will disable submit button once clicked. Instead of this, which represent current element similar to [Java this keyword](http://javarevisited.blogspot.com/2012/01/this-keyword-java-example-tutorial.html), You can also use document.getElementById('id') but this is short and clear.

Now some browser has problem to **submit data from disabled button.** So, its better to call form.submit() from onclick itself to submit form data, which makes your code to onclick="this.disabled=true;this.form.submit(). Particularly, on Internet Explorer a disabled submit button doesn't submit form data to server. You can also change value or text of submit button from submit to "Submitting...." to indicate user that submit is in progress. Final JavaScript code should look like:

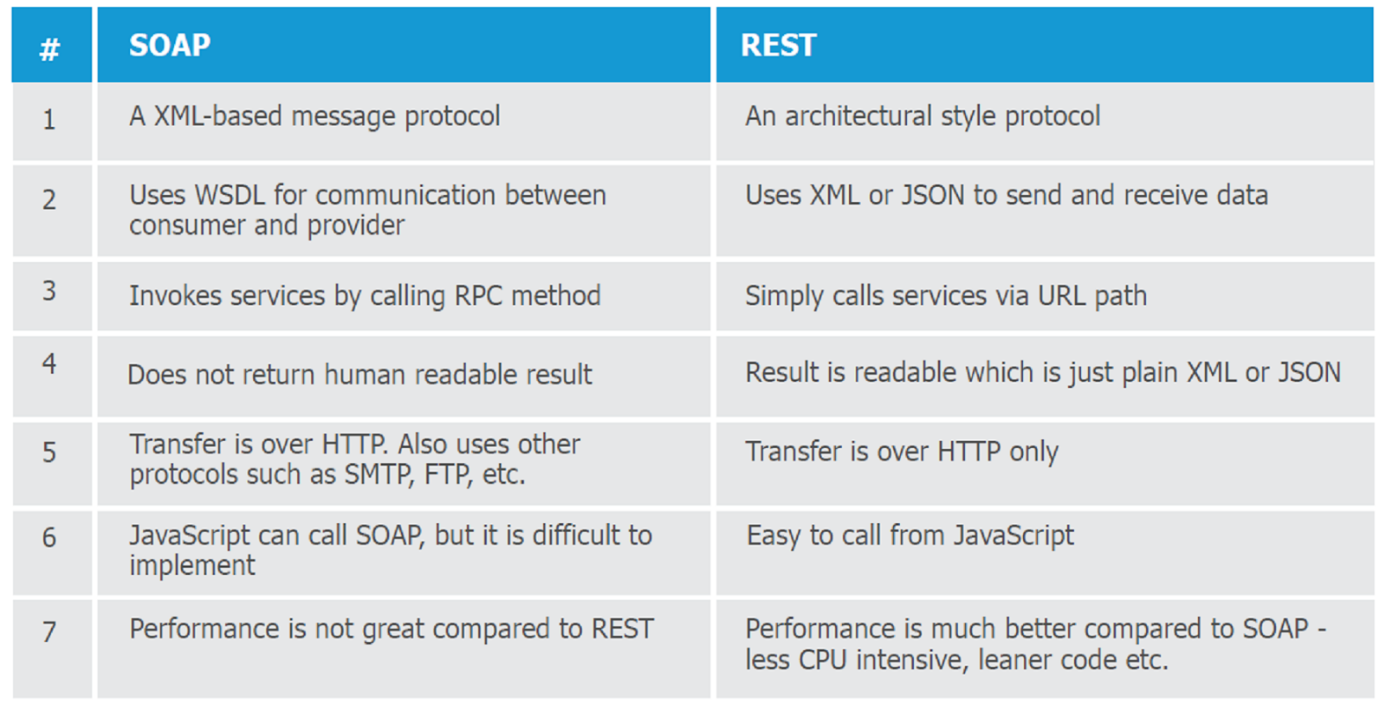
<form action="submit.jsp" method="post" >

<input type="submit" name="SUBMIT" value="Submit Form" onclick="this.value='Submitting ..';this.disabled='disabled'; this.form.submit();" />

</form>

That's all you need to disable submit button using HTML and JavaScript to prevent multiple submission.

## Difference between REST and SOAP in Java

****

**Short Form**  
REST stands for REpresntational State Transfer (REST) while SOAP Stands for Simple Object Access Protocol (SOAP).  
**Architecture style vs Protocol**

REST is an architectural style, on which RESTFul web services are built while SOAP is a standard devised to streamline communication between client and server in terms of format, structure and method.

**Use of HTTP Protocol**

REST takes full advantage of HTTP protocol, including methods e.g. [GET, POST, PUT, and DELETE](http://javarevisited.blogspot.sg/2012/03/get-post-method-in-http-and-https.html) to represent action e.g. from an application which provides data related to books, GET request can be used to retrieve books, POST can be used to upload data of a new book, and DELETE can be used to remove a book from library. On the other hand, SOAP uses XML messages to communicate with the server.

**Supported Format**

RESTful web service can return the response in various format e.g. JSON, XML and HTML, while by using SOAP web service you tie your response with XML because actual response is bundled inside a SOAP message which is always in XML format.

**Speed**

Processing a RESTful web service request is much faster than processing a SOAP message because you need to less parsing. Because of this reason RESTful, web services are ***faster*** than SOAP web service.

## Why REST is better than SOAP?

Now that you know some differences between REST and SOAP web services, let's summarize our reasons of *why REST is better choice* for modern day web service requirement:

1. REST can be consumed by any client e.g. Java, C++, Python client and even a web browser with Ajax and JavaScript.
2. REST is lightweight as compared to SOAP, it doesn't require CPU consuming XML parsing and it also consumes less bandwidth because unlike SOAP, REST doesn't require a SOAP header for every message.
3. SOAP is an old technology, all modern technical giant are using REST e.g. Google, Twitter, and Flickr.
4. REST is easy to learn, it’s just nouns and verbs. If you already know HTTP methods then it’s even easier.

# [How to Clone Collection in Java](http://javarevisited.blogspot.in/2014/03/how-to-clone-collection-in-java-deep-copy-vs-shallow.html)

Programmer often mistook copy constructors provided by various collection classes, as a mean to clone Collection e.g. List, Set, ArrayList, HashSet or any other implementation. What is worth remembering is that, copy constructor of Collection in Java only provides shallow copy and not deep copy, which means objects stored in both original List and cloned List will be same and point to same memory location in Java heap. One thing, which adds into this misconception is shallow copy of Collections with [Immutable Objects](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html). Since Immutable can't be changed, it’s Ok even if two collections are pointing to same object. This is exactly the case of String contained in pool, update on one will not affect other.

Problem arise, when we use Copy constructor of ArrayList to create a clone of List of Employees, where Employee is not immutable. In this case, if original collection modifies an employee, that change will also reflect into cloned collection.

Similarly if an employee is modified in cloned collection, it will also appeared as modified in original collection. This is not desirable, in almost all cases, clone should be independent of original object. Solution to avoid this problem is **deep cloning of collection**, which means recursively cloning object, until you reached to primitive or Immutable.

// deep cloning Collection in Java

Collection<Employee> org = **new** HashSet<>();

org.add(**new** Employee("Joe", "Manager"));

org.add(**new** Employee("Tim", "Developer"));

org.add(**new** Employee("Frank", "Developer"));

// creating copy of Collection using copy constructor

Collection<Employee> copy = **new** HashSet<>(org);

logger.debug("Original Collection {}", org);

logger.debug("Copy of Collection {}", copy );

Iterator<Employee> itr = org.iterator();

**while**(itr.hasNext()){

itr.next().setDesignation("staff");

}

logger.debug("Original Collection after modification {}", org);

logger.debug("Copy of Collection without modification {}", copy );

**Output:**

- Original Collection [*Joe:* Manager, *Frank:* Developer, *Tim:* Developer]

- Copy of Collection [*Joe:* Manager, *Frank:* Developer, *Tim:* Developer]

- Original Collection after modification [*Joe:* staff, *Frank:* staff, *Tim:* staff]

- Copy of Collection without modification [*Joe:* staff, *Frank:* staff, *Tim:* staff]

1) Let Employee implements Cloneable interface  
2) Add following clone() method into Employee class

@Override

**protected** **Employee** clone() {

**Employee** clone **=** **null**;

**try**{

clone **=** (**Employee**) **super**.clone();

}**catch**(**CloneNotSupportedException** e){

**throw** **new** **RuntimeException**(e); // won't happen

}

**return** clone

}

3) Instead of using Copy constructor use following code, to deep copy Collection in Java

**Collection**<**Employee**> copy **=** **new** **HashSet**<**Employee**>(org.size());

**Iterator**<**Employee**> iterator **=** org.iterator();

**while**(iterator.hasNext()){

copy.add(iterator.next().clone());

}

4) Running same code for modifying collection, will not result in different output:  
- Original Collection after modification  [Joe: staff, Tim: staff, Frank: staff]  
- Copy of Collection without modification [Frank: Developer, Joe: Manager, Tim: Developer]

Read more: <http://javarevisited.blogspot.com/2014/03/how-to-clone-collection-in-java-deep-copy-vs-shallow.html#ixzz46TD9w3NJ>

# [Binary Search vs Contains Performance in Java](http://javarevisited.blogspot.in/2014/03/binary-search-vs-contains-performance.html)

Contains(obj) method gives better performance than Collections.binarySearch(list,**obj**);

# [How to declare and initialize a List (ArrayList and LinkedList) with values in Java](http://javarevisited.blogspot.in/2012/12/how-to-initialize-list-with-array-in-java.html)

*//declaring and initializing array in one line*  
      **String**[] oldValues = **new** **String**[] {"list" , "set" , "map"};  
      **String**[] values = {"abc","bcd", "def"};   
      
      *//initializing list with array in java*  
      **List** init = **Arrays**.asList(values);  
      **System**.out.println("size: " + init.size()+" list: " + init);  
      
      *//initializing List in one line in Java*  
      **List** oneLiner = **Arrays**.asList("one" , "two", "three");  
      **System**.out.println("size: " + init.size()+" list: " + oneLiner);  
      
      *// List returned by Arrays.asList is fixed size  
      // and doesn't support add or remove*  
*// This will throw java.lang.UnsupportedOperationException*      oneLiner.add("four");   
 *// This also throws java.lang.UnsupportedOperationException*      *//oneLiner.remove("one");*

Solution: use copy constructor of ArrayList to perform add and remove operations.

List<String> copyCon = new ArrayList<String>(init);

Read more: <http://javarevisited.blogspot.com/2012/12/how-to-initialize-list-with-array-in-java.html#ixzz46WowtzLv>

## [What is PriorityQueue](http://javarevisited.blogspot.in/2013/10/what-is-priorityqueue-data-structure-java-example-tutorial.html)

PriorityQueue allows you to keep elements in a particular *order*, according to their natural order or custom order defined by [Comparator interface](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html) in Java.

Queue<Item> items = **new** PriorityQueue<Item>();

Item class should implement either Comparable or Comparator interfaces. The same order flow the Queue (FIFO).

1) PriorityQueue is not synchronized, if [thread-safety](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html) is requirement use BlockingPriorityQueue.

2) PriorityQueue doesn't allow null elements, if you try to add null, it will throw [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html).

## String Immutable:

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 a 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 the name of the file as argument to File I/O classes.

In case, if String is not immutable, this would lead serious security threat, I mean someone can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access to that file.

Read more: <http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html#ixzz4774no3mX>

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

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 is no way contents of Strings can be changed because [any change will produce new String](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html), 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.

[**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) 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  
Read more: <http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html#ixzz4g5FdOLQY>

Read more: <http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html#ixzz4g5F1kPdC>  
Read more: <http://javarevisited.blogspot.com/2012/03/why-character-array-is-better-than.html#ixzz4g5EvaBmk>

## Tips to avoid NullPointerException in JAVA.

1) Call equals() and equalsIgnoreCase() method on known String literal rather unknown object. Always call equals() method on known String which is not null. Since [equals() method is symmetric](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html), calling a.equals(b) is same as calling b.equals(a), and that’s why many programmer don’t pay attention on object a and b. One side effect of this call can result in NullPointerException, if caller is null.

Object unknownObject **=** *null*;

//wrong way - may cause NullPointerException

**if**(unknownObject**.**equals("**knownObject**")){

System**.**err**.**println("This may result in NullPointerException if unknownObject is null");

}

//right way - avoid NullPointerException even if unknownObject is null

**if**("knownObject"**.**equals(unknownObject)){

System**.**err**.**println("better coding avoided NullPointerException");

}

Read more on this: <http://javarevisited.blogspot.com/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html#ixzz477BVUVEP>

Can Enum extend a class in Java? (No, because Java allows a class to only extend one class and enum by default extends java.lang.Enum, see here for more [Enum interview questions](http://java67.blogspot.sg/2013/07/15-java-enum-interview-questions-amswers-for-experienced-programmers.html))

# [Search Element in array or collection](http://javarevisited.blogspot.in/2012/11/4-ways-to-search-object-in-java-array-example.html).

Searching element in collection using contains() method.

Searching element in Array using Arrays.binarySearch()

Binary Search is another faster way of *searching elements in Java array* but it requires array to be sorted while earlier examples of finding elements on Array can be used with both [sorted and unsorted array](http://javarevisited.blogspot.sg/2012/01/how-to-sort-arraylist-in-java-example.html). java.util.Arrays class provides both sort() and binarySearch() for first sorting an array and than performing binary search on it. Arrays.binarySearch() method returns >=0 if it finds elements in Array. See code section for full code example of binarySearch in Java array.

# What Is Fail Safe and Fail Fast Iterator in Java

Java Collections supports two types of Iterator, fail safe and fail fast. The main distinction between a fail fast and fail safe Iterator is whether or not the underlying collection can be modified while its begin iterated. If you have used Collection like ArrayList then you know that when you iterate over them, no other thread should modify the collection. If Iterator detects any structural change after iteration has begun e.g adding or removing a new element then it throws ConcurrentModificationException, this is known as fail-fast behavior and these iterator are called *fail-fast iterator* because they fail as soon as they detect any modification. Though it’s not necessary that iterator will throw this exception when multiple threads modified it simultaneously. it can happen even with single thread when you try to remove elements  by using ArrayList's remove() method instead of Iterator's remove method, as discussed in my earlier post, [2 ways to remove objects from ArrayList](http://java67.blogspot.sg/2014/03/2-ways-to-remove-elementsobjects-from-ArrayList-java.html).

Most of the Collection classes from Java 1.4 e.g. Vector, ArrayList, HashMap, HashSet has fail-fast iterators. The other type of iterator was introduced in Java 1.5 when concurrent collection classes e.g. ConcurrentHashMap, CopyOnWriteArrayList and CopyOnWriteArraySet was introduced. These iterator uses a view of original collection for doing iteration and that's why they doesn't throw ConcurrentModificationException even when original collection was modified after iteration has begun.  This means you could iterate and work with stale value, but this is the cost you need to pay for fail-safe iterator and this feature is clearly documented.

## Difference between Fail Safe and Fail Fast Iterator in Java

In order to best understand difference between these two iterator you need to try out examples with both traditional collections like ArrayList and [concurrent collections](http://javarevisited.blogspot.sg/2010/10/what-is-difference-between-synchronized.html) like CopyOnWriteArrayList. Nevertheless let's first see some key differences one at a time:

1. Fail-fast Iterator throws ConcurrentModfiicationException as soon as they detect any structural change in collection during iteration, basically which changes the modCount variable hold by Iterator. While fail-fast iterator doesn't throw CME.
2. Fail-fast iterator traverse over original collection class while fail-safe iterator traverse over a copy or view of original collection. That's why they don't detect any change on original collection classes and this also means that you could operate with stale value.
3. Iterators from Java 1.4 Collection classes e.g. ArrayList, HashSet and Vector are fail-fast while Iterators returned by concurrent collection classes e.g. [CopyOnWriteArrayList](http://java67.blogspot.com/2012/09/what-is-copyonwritearraylist-in-java-example-vs-arraylist.html) or [CopyOnWriteArraySet](http://javarevisited.blogspot.sg/2014/06/how-to-use-copyonwritearrayset-in-java-example-tutorial.html) are fail-safe.
4. Iterator returned by synchronized Collection are fail-fast while iterator returned by concurrent collections are fail-safe in Java.

**Note:**

Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these method to remove an entry from Collection, while not iterating. Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

**What is the difference between Iterator and Enumeration? (**[**answer**](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html)**)**

Java 5 has added several new Concurrent Collection classes e.g. ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc., which has made Interview questions on Java Collection even trickier. Java Also provided a way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.synchronizedMap() Utility function. One Significant difference is that Concurrent Collections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization. See the difference between Synchronized Collection and Concurrent Collection in Java for more details.

**How does HashSet is implemented in Java, How does it use Hashing? (**[**Answer**](http://java67.blogspot.com/2014/01/how-hashset-is-implemented-or-works-internally-java.html)**)**  
This is a tricky question in Java because for hashing you need both key and value and there is no key for the store it in a bucket, then how exactly HashSet store element internally. Well, HashSet is built on top of HashMap. If you look at source code of java.util.HashSet class, you will find that that it uses a HashMap with same values for all keys, as shown below:

private transient HashMap map;  
// Dummy value to associate with an Object in the backing Map  
private static final Object PRESENT = new Object();  
When you call add() method of HashSet, it put entry in HashMap :  
public boolean add(E e) {  
  return map.put(e, PRESENT)==null;  
}  
Since keys are unique in a HashMap, it provides uniqueness guarantee of Set interface.

**What do you need to do to use a custom object as a key in Collection classes like Map or Set? (**[**answer**](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html)**)**

The answer is: If you are using any custom object in Map as key, you need to override equals() and hashCode() method, and make sure they follow their contract. On the other hand if you are storing a custom object in Sorted Collection e.g. SortedSet or SortedMap, you also need to make sure that your equals() method is consistent to compareTo() method, otherwise that collection will not follow there contacts e.g. Set may allow duplicates.

# [**How to find all Pairs in Array of Integers whose Sum is equal to a given Number**](http://javarevisited.blogspot.in/2014/08/how-to-find-all-pairs-in-array-of-integers-whose-sum-equal-given-number-java.html)

1. public static void printPairs(int[] array, int sum) {

for (int i = 0; i < array.length; i++) {

int first = array[i];

for (int j = i + 1; j < array.length; j++) {

int second = array[j];

if ((first + second) == sum) {

System.out.printf("(%d, %d) %n", first, second);

}

}

}

}

1. public static void printPairsUsingSet(int[] numbers, int n){

if(numbers.length < 2){

return;

}

Set set = new HashSet(numbers.length);

for(int value : numbers){

int target = n - value;

// if target number is not in set then add

if(!set.contains(target)){

set.add(value);

}else {

System.out.printf("(%d, %d) %n", value, target);

}

}

}

### **Volatile:**

A [volatile](http://java67.blogspot.com/2012/08/what-is-volatile-variable-in-java-when.html#ixzz458pThxhG) variable can be used as an alternative way of achieving [synchronization in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) in some cases, like Visibility. By making a variable volatile using volatile keyword in Java, application programmer ensures that its value should always been read from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) and thread should not use cached value of that variable from their own stack. With volatile variable, it's guaranteed that all reader thread will see updated value of the volatile variable once write operation completed, without volatile keyword different reader thread may see different values.

### **Java Virtual Machine (JVM)**

When you download JRE and install on your machine you got all the code required to create JVM. Java Virtual Machine is get created when you run a java program using java command e.g. java HelloWorld. JVM is responsible for **converting byte code into machine specific code** and that's why you have different JVM for Windows, Linux or Solaris but one JAR can run on all this operating system. Java Virtual machine is at heart of Java programming language and provide several feature to Java programmer including Memory Management and [Garbage Collection](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html), Security and other system level services. Java Virtual Machine can be customized e.g. we can specify [starting memory or maximum memory of heap size](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) located inside JVM at the time of JVM creation.

### **Just in Time Compiler (JIT)**

JIT are advanced part of Java Virtual machine which optimize byte code to machine instruction conversion part by compiling similar byte codes at same time and thus reducing overall execution time. JIT is part of Java Virtual Machine and also performs several other optimizations such as in-lining function.

**JDK 1.7 Features**

<http://javarevisited.blogspot.in/2014/04/10-jdk-7-features-to-revisit-before-you.html>

**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.

**How do you print Array in Java?** ([answer](http://java67.blogspot.sg/2014/03/how-to-print-array-in-java-example-tutorial.html))

You can print an array by using the Arrays.toString() and Arrays.deepToString() method. Since array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but Arrays.toString() will print each element.

**LinkedList in Java is doubly or singly linked list?**  
It's a doubly linked list.

**Which kind of tree is used to implement TreeMap in Java?**  
A Red Black tree is used to implement TreeMap in Java.

**What is the difference between Hashtable and HashMap?** ([answer](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html))  
There are many differences between these two classes, some of them are following:   
a) Hashtable is a legacy class and present from JDK 1, HashMap was added later.  
b) Hashtable is synchronized and slower but HashMap is not synchronized and faster.  
c) Hashtable and ConcurrentHashMap doesn't allow null keys but HashMap allows one null key.  
See the answer for more differences between HashMap and Hashtable in Java.

**How HashSet works internally in Java?** ([answer](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html))

HashSet is internally implemented using a HashMap. Since a Map needs key and value, a default value is used for all keys. Similar to HashMap, HashSet doesn't allow duplicate keys and only one null key, I mean you can only store one null object in HashSet.

**Write code to remove elements from ArrayList while iterating?** ([answer](http://javarevisited.blogspot.sg/2014/01/ow-to-remove-objects-from-collection-arraylist-java-iterator-traversing.html))  
 Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Use Iterator’s remove() to avoids ConcurrentModificationException.

**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.

**What best practices you follow while writing multi-threaded code in Java?** ([answer](http://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html))  
Here are couple of best practices which I follow while writing concurrent code in Java:

1. Always name your thread, this will help in debugging.
2. Minimize the scope of synchronization, instead of making whole method synchronized, only critical section should be synchronized.
3. Prefer volatile over synchronized if you can.
4. Use higher level concurrency utilities instead of wait() and notify for inter thread communication e.g. BlockingQueue, CountDownLatch and Semeaphore.
5. Prefer concurrent collection over synchronized collection in Java. They provide better scalability.

Threads concurrent package

<http://javarevisited.blogspot.in/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html>

<http://javarevisited.blogspot.in/2015/06/java-lock-and-condition-example-producer-consumer.html>

<http://javarevisited.blogspot.in/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html>

<http://javarevisited.blogspot.in/2015/06/how-to-use-callable-and-future-in-java.html>

<http://javarevisited.blogspot.in/2015/01/how-to-use-future-and-futuretask-in-Java.html>

<http://javarevisited.blogspot.in/2012/07/countdownlatch-example-in-java.html>

<http://javarevisited.blogspot.in/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html>

<http://java67.blogspot.in/2012/08/difference-between-countdownlatch-and-cyclicbarrier-java.html>

<http://javarevisited.blogspot.in/2012/05/counting-semaphore-example-in-java-5.html>

<http://javarevisited.blogspot.in/2015/06/how-to-use-callable-and-future-in-java.html>

<http://javarevisited.blogspot.in/2016/04/difference-between-ExecutorServie-submit-vs-Executor-execute-method-in-Java.html>

<http://javarevisited.blogspot.in/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html>

<http://javarevisited.blogspot.in/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html>

<http://javarevisited.blogspot.in/2015/05/top-10-java-multithreading-and.html>

**Race Conditions**

A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable. Then the first thread and second thread perform their operations on the value, and they race to see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.

**Deadlocks**

A deadlock occurs when two threads each lock a different variable at the same time and then try to lock the variable that the other thread already locked. As a result, each thread stops executing and waits for the other thread to release the variable. Because each thread is holding the variable that the other thread wants, nothing occurs, and the threads remain deadlocked.

**What is race condition?**

A race condition is a situation in which two or more threads or processes are reading or writing some shared data, and the final result depends on the timing of how the threads are scheduled. Race conditions can lead to unpredictable results and subtle program bugs. A thread can prevent this from happening by locking an object. When an object is locked by one thread and another thread tries to call a synchronized method on the same object, the second thread will block until the object is unlocked.

**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 cannot reuse ContdownLatch instance once count reaches to zero you can reuse CyclicBarrier by calling reset() method.

Read more: <http://javarevisited.blogspot.com/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html#ixzz4gB8IOLH8>

**Does SimpleDateFormat is safe to use in the multi-threaded program?** ([answer](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html))

No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a ThreadLocal variable. If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **joda-time** library.

**How do you format a date in Java? e.g. in the ddMMyyyy format?** ([answer](http://javarevisited.blogspot.com/2011/09/convert-date-to-string-simpledateformat.html))

You can either use SimpleDateFormat class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. dd-MM-yyyy or ddMMyyyy.

//Creating Date in java with today's date.

Date dateNow = **new** Date();

//change date into string **yyyyMMdd** format example "20110914"

SimpleDateFormat dateformatyyyyMMdd = **new** SimpleDateFormat("yyyyMMdd");

String date\_to\_string = dateformatyyyyMMdd.format(dateNow);

// format() pass date you get string date or pass string you get date.

System.out.println("date into yyyyMMdd format: " + date\_to\_string);

1. Common point of confusion between “m” and “M” , small case “m” represent minutes while “M” represent Month Also “d” represent date in month while “D” represent Day of week. This is most common cause of error while converting String to date and back date to string. In shot **ddMMyy** is not equal to **DDmmyy.**
2. It’s also worth noting that SimpleDateFormat  **are not thread-safe**. They are not [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) so its better you create separate DateFormat for each [thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) to avoid any race condition while parsing date in java.

DateTime myBirthDate = **new** DateTime(2014, 1, 30, 0, 0);

DateTime now = **new** DateTime();

System.***out***.println("No of Years "+Years.*yearsBetween*(myBirthDate, now).getYears());

System.***out***.println("No of Months "+Months.*monthsBetween*(myBirthDate, now).getMonths());

System.***out***.println("No of Days "+Days.*daysBetween*(myBirthDate, now).getDays());

<http://java67.blogspot.in/2014/12/string-to-date-example-in-java-multithreading.html>

**Deign Patterns:**

<http://javarevisited.blogspot.in/2011/12/observer-design-pattern-java-example.html>

<http://javarevisited.blogspot.in/2011/11/decorator-design-pattern-java-example.html>

<http://javarevisited.blogspot.in/2015/06/difference-between-dependency-injection.html> --> Deign Patterns are in below check above link.

<http://javarevisited.blogspot.in/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html>

<http://javarevisited.blogspot.in/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html>

**What is Template method pattern?** (answer)

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.

**When do you use Visitor design pattern?** (answer)

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.  
**When do you use Composite design pattern?** (answer)

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.

**When do you use Flyweight pattern?** (answer)

This is another popular question from the design pattern. Many Java developers with 4 to 6 years of experience know the definition but failed to give any concrete example. Since many of you might not have used this pattern, it's better to look examples from JDK. You are more likely have used them before and they are easy to remember as well. Now let's see the answer.

Flyweight pattern allows you to share object to support large numbers without actually creating too many objects. In order to use Flyweight pattern, you need to make your object Immutable so that they can be safely shared. String pool and pool of Integer and Long object in JDK are good examples of Flyweight pattern.

**Difference between Abstract factory and Prototype design pattern?** (answer)

This is the practice question for you, if you are feeling bored just reading and itching to write something, why not write the answer to this question. I would love to see an example the, which should answer where you should use the Abstract factory pattern and where is the Prototype pattern is more suitable.

**Give me an example of design pattern which is based upon open closed principle?** ([answer](http://javarevisited.blogspot.sg/2011/11/great-example-of-open-closed-design.html))

This is one of the practical questions I ask experienced Java programmer. I expect them to know about OOP design principles as well as patterns. Open closed design principle asserts that your code should be open for extension but closed for modification. Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code.  There are several design patterns which are based upon open closed design principle e.g. [Strategy pattern](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) if you need a new strategy, just implement the interface and configure, no need to modify core logic. One working example is Collections.sort() method which is based on Strategy pattern and follows the open-closed principle, you don't modify sort() method to sort a new object, what you do is just implement Comparator in your own way.

**The difference between Inheritance and Composition?** ([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-inheritance-and-Composition-in-Java-OOP.html))

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.

**Difference between Composition, Aggregation and Association in OOP?** ([answer](http://javarevisited.blogspot.sg/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))

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.

**Composition:**

final class Car {

private final Engine engine;

Car(EngineSpecs specs) {

engine = new Engine(specs);

}

void move() {

engine.work();

}

}

**Aggregation:**

final class Car {

private Engine engine;

void setEngine(Engine engine) {

this.engine = engine;

}

void move() {

if (engine != null)

engine.work();

}

}

In the case of composition, the Engine is completely encapsulated by the Car. There is no way for the outside world to get a reference to the Engine. The Engine lives and dies with the car. With aggregation, the Car also performs its functions through an Engine, but the Engine is not always an internal part of the Car. Engines may be swapped, or even completely removed. Not only that, but the outside world can still have a reference to the Engine, and tinker with it regardless of whether it's in the Car.

<http://java67.blogspot.in/2012/09/top-10-java-design-pattern-interview-question-answer.html>

**The difference between Serializable and Externalizable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html))

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.

# **WHAT IS SERIALIZATION IN JAVA**

Serialization is a mechanism of converting the state of an object into a byte stream, to transfer over the network or to store in files. Serialization persist all of Object's state except [transient variable](http://java67.blogspot.sg/2012/08/what-is-transient-variable-in-java.html) and a [static variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html). Serializable interface is also called [marker interface or tag interface](http://javarevisited.blogspot.sg/2012/01/what-is-marker-interfaces-in-java-and.html) because it does not contain any method or behavior and just used to tag a class, by seeing a class implementing Serializable JVM knows that it needs to serialize an instance of this class. superClass has implemented Serializable that means subclass is also Serializable (as subclass always inherits all features from its parent class),

|  |
| --- |
| FileOutputStream file = new FileOutputStream(filename);  ObjectOutputStream out = new ObjectOutputStream(file);  // Method for deserialization of object  out.writeObject(object); |

[**DESERIALIZATION**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object.html)

|  |
| --- |
| FileInputStream file = new FileInputStream(filename);  ObjectInputStream in = new ObjectInputStream(file);  // Method for deserialization of object  object = (Emp)in.readObject(); |

## **DIFFERENCE BETWEEN SERIALIZABLE AND EXTERNALIZABLE IN JAVA**

1. One of the obvious difference between Serializable and Externalizable is that Serializable is a [marker interface](http://javarevisited.blogspot.in/2012/01/what-is-marker-interfaces-in-java-and.html) i.e. does not contain any method but Externalizable interface contains two methods writeExternal() and readExternal().
2. Second difference between Serializable vs Externalizable is responsibility of Serialization. When a class implements Serializable interface, default Serialization process gets kicked off and that takes responsibility of serializing super class state. When any [class in Java](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) implement java.io.Externalizable than it’s your responsibility to implement Serialization process i.e. preserving all important information.
3. This difference between Serializable and Externalizable is **performance**. You cannot do much to improve performance of default serialization process except reducing number of fields to be serialized by using [transient](http://java67.blogspot.sg/2012/08/what-is-transient-variable-in-java.html) and [static keyword](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) but with Externalizable interface you have full control over Serialization process.

**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 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.

**What if Serialization is not available, is any other alternative way to transfer object over network?**

> We can convert JSON to transfer the object. JSON is helpful in stringify and de stringify object.

> Hibernate (ORM tool) helps in persisting object as it in database and later we can read persisted object.

> We can convert object into XML (as done in web services) and transfer object over network.

**Why static member variables are not part of java serialization process (Important)?**

Answer. Serialization is applicable on objects or primitive data types only, but static members are class level variables, therefore, different objects of same class have same value for static member. So, serializing static member will consume unnecessary space and time. Also, if modification is made in static member by any of the object, it won’t be in sync with other serialized object’s value.

**Is constructor of class called during DeSerialization process?**

This question which will check your in depth knowledge of Serialization and constructor chaining concepts. It depends on whether our object has implemented Serializable or Externalizable.

If Serializable has been implemented - constructor is not called during DeSerialization process.

if Externalizable has been implemented - constructor is called during DeSerialization process.

**Is constructor of super class called during DeSerialization process of subclass (Important)?**

Again your basic java concepts will be tested over here. It is depends on whether our superclass has implemented Serializable or not.

If superclass has implemented Serializable - constructor is not called during DeSerialization process.

If superclass has not implemented Serializable - constructor is called during DeSerialization process.

**What values will int and Integer will be initialized to during DeSerialization process if they were not part of Serialization?**

int will be initialized to 0 and Integer will be initialized to null during DeSerialization (if they were not part of Serialization process).

### Enum:

Group of named constants is called enum.

Every enum constant is always public static final.

Internally toString() method is implemented to return name of the constant.

### Reflection:

* You can accomplish any task, fulfil any requirement without reflection when you know the classes you are going to use.
* But as soon as, you are told to deal with classes which you don’t know at time of writing the code, and you must write code in too general way such that it can handle any class type then you will need reflection to do the job.
* The name reflection is used to describe code which is able to inspect other code in the same system (or itself).

So, to give you a code example of this in Java (imagine the object in question is foo) :

Method method = foo.getClass().getMethod("doSomething", null);

method.invoke(foo, null);

* Reflection is much slower than just calling methods by their name, because it has to inspect the metadata in the byte code instead of just using precompiled addresses and constants.
* Reflection is also more powerful: you can retrieve the definition of a protected or final member, remove the protection and manipulate it as if it had been declared mutable! Obviously this subverts many of the guarantees the language normally makes for your programs and can be very, very dangerous.

<https://howtodoinjava.com/core-java/reflection/real-usage-examples-of-reflection-in-java/>

<https://www.geeksforgeeks.org/reflection-in-java/>