Table of Contents

1. How Hashmap works in Java

HashMap works on the principle of Hashing. To understand Hashing, we should understand the three terms first i.e., Hash Function, Hash Value and Bucket.

What is Hash Function, Hash Value and Bucket?
hashCode() function which returns an integer value is the Hash function. The important point to note that, this method is present in Object class (Mother of all class).

```
public native int hashCode();
```

The most important point to note from the above line: hashCode method return int value .

What is bucket?

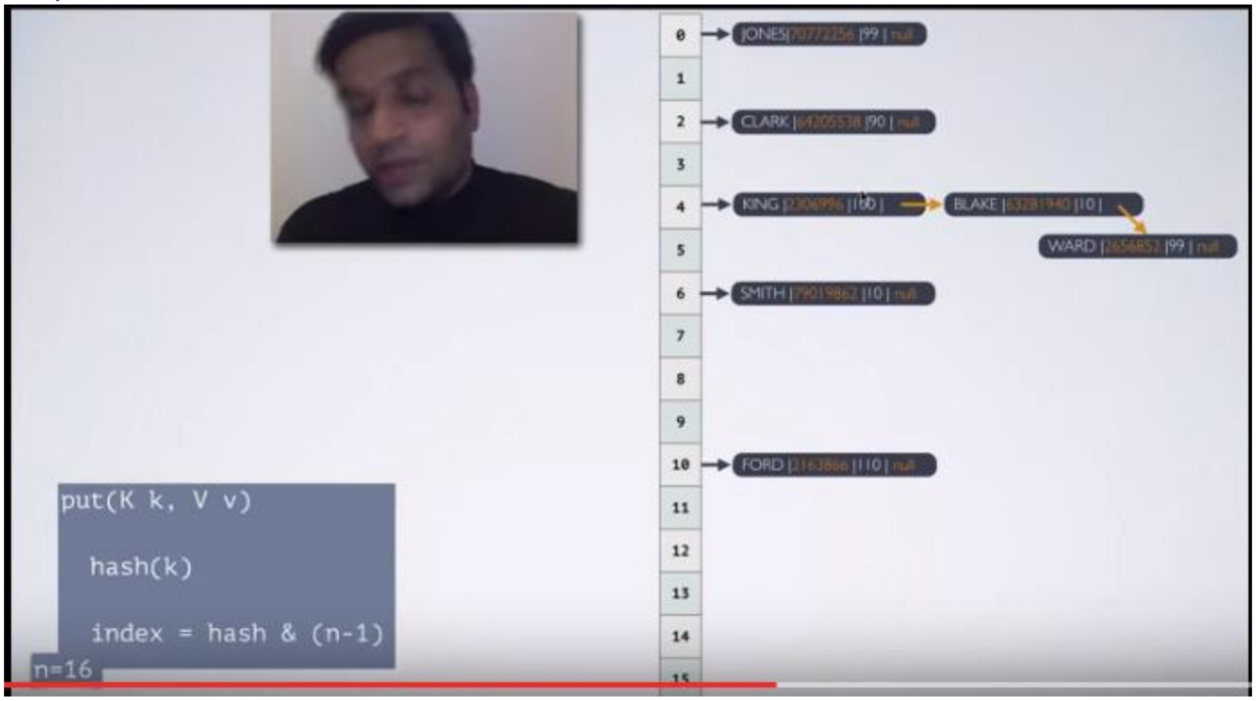
A bucket is used to store key value pairs. A bucket can have multiple key-value pairs. In hash map, bucket used simple linked list to store objects.

Code inside Java Api (HashMap class internal implementation) for HashMap get(Obejct key) method

```
 Public  V get(Object key)
  {
    if (key ==null)
    //Some code

    int hash = hash(key.hashCode());

    // if key found in hash table then return value
    //    else return null
  }
```


0
JONES|70772256 |99 | null
1
2
CLARK |64205538 |90 | null
3
4
KING |2306996 |100 |
BLAKE |63281940 |10 |
WARD |2656852 |99 | null
5
6
SMITH |79019862 |10 | null
7
8
9
10
FORD |2163866 |110 | null
11
12
13
14
15

```
put(K k, V v)

  hash(k)

  index = hash & (n-1)
n=16
```

Hash map works on the principle of hashing

HashMap get(Key k) method calls hashCode method on the key object and applies returned hashValue to its own static hash function to find a bucket location(backing array)where keys and values are stored in form of a nested class called Entry (Map.Entry) . So you have concluded that from the previous line that Both key and value is stored in the bucket as a form of Entry object. So thinking that only value is stored in the bucket is not correct and will not give a good impression on the interviewer.

* Whenever we call get(Key k) method on the HashMap object . First it checks that whether key is null or not . Note that there can only be one null key in HashMap .

If key is null , then Null keys always map to hash 0, thus index 0.

If key is not null then, it will call hashfunction on the key object, see line 4 in above method i.e. key.hashCode() ,so after key.hashCode() returns hashValue , line 4 looks like

```
4.                    int hash = hash(hashValue)
```

, and now, it applies returned hashValue into its own hashing function.

We might wonder why we are calculating the hashvalue again using hash (hashValue). Answer is, It defends against poor quality hash functions.

Now step 4 final hash value is used to find the bucket location at which the Entry object is stored . Entry object stores in the bucket like this (hash, key, value, bucketindex) .

What if when two different keys have the same hashcode ?

Solution, equals () method comes to rescue. Here candidate gets puzzled. Since bucket is one and we have two objects with the same hashcode .Candidate usually forgets that bucket is a simple linked list.

The bucket is the linked list effectively. It’s not a Linked List as in a java.util.LinkedList - It's a separate (simpler) implementation just for the map .

So we traverse through linked list , comparing keys in each entries using keys.equals() until it return true. Then the corresponding entry object Value is returned .

When the functions 'equals' traverses through the linked list does it traverses from start to end one by one...in other words brute method. Or the linked list is sorted based on key and then it traverses?

Answer is when an element is added/retrieved, same procedure follows:

a. Using key.hashCode() [see above step 4],determine initial hash value for the key

b. Pass intial hashvalue as hashValue in hash(hashValue) function, to calculate the final hashvalue.

c. Final hash value is then passed as a first parameter in the indexFor(int ,int)method .
The second parameter is length which is a constant in HashMap Java Api , represented by DEFAULT_INITIAL_CAPACITY

The default value of DEFAULT_INITIAL_CAPACITY is 16 in HashMap Java Api .

indexFor(int,int) method returns the first entry in the appropriate bucket. The linked list in the bucket is then iterated over - (the end is found and the element is added or the key is matched and the value is returned)

Explanation about indexFor(int,int) is below :

```
/**
* Returns index for hash code h.
*/
static int indexFor(int h, int length) {
return h & (length-1);
}
```

The above function indexFor() works because Java HashMaps always have a capacity, i.e. number of buckets, as a power of 2.
Let's work with a capacity of 256, which is 0x100, but it could work with any power of 2. Subtracting 1 from a power of 2 yields the exact bit mask needed to bitwise-and with the hash to get the proper bucket index, of range 0 to length - 1.
256 - 1 = 255
0x100 - 0x1 = 0xFF
E.g. a hash of 257 (0x101) gets bitwise-anded with 0xFF to yield a bucket number of 1.

What if when two keys are same and have the same hashcode ?

If key needs to be inserted and already inserted hashkey's hashcodes are same, and keys are also same (via reference or using equals() method) then override the previous key value pair with the current key value pair.

The other important point to note is that in Map ,Any class(String etc.) can serve as a key if and only if it overrides the equals() and hashCode() method .

What algorithm does Java use to avoid HashMap collision?

It is using Separate Chaining [1].

How will you measure the performance of HashMap?

According to Oracle Java docs,

An instance of HashMap has two parameters that affect its performance: initial capacity and load factor.

The capacity is the number of buckets in the hash table(HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.), and the initial capacity is simply the capacity at the time the hash table is created.

The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

In HashMap class, the default value of load factor is (.75) .

What is the time complexity of Hashmap get() and put() method ?

The hashmap implementation provides constant time performance for (get and put) basic operations i.e the complexity of get() and put() is O(1) , assuming the hash function disperses the elements properly among the buckets.

When should we not use HashMap in Java?

If you are not bothered about the ordering, but just want to check if the element is present or not perhaps you can use HashSet.

If you are not bothered about the ordering, but have a key-value usecase, then using HashMap/HashTable make sense.

If you are interested in ordering, resorting to ArrayList/LinkedList would make sense.

Your claim that HashMaps should be avoided is not valid.

HashMap in Java8

Instead of Linked List, Hash map uses balanced tree data structure in bucket.

2. Which data structure is better to implement a phone book: Trie or Hash?

You Should use TRIE data Structure for Implementing Phonebook. TRIE is an ordered tree data structure that uses strings as keys. Trie is an ordered tree data structure that uses strings as keys. Unlike Binary Trees, Tries do not store keys associated with the node. The key is actually determined based on the position of the node on the tree. Any descendants of a node shares a common prefix of the key string

associated with that node. Hence, trie is also called as Prefix Tree. The word "trie" comes from Retrieval, and it is pronounced as "try". To read more aboutTrie click here.
Since this data structure is a prefix tree, trie is commonly used in Dictionaries, Phone Directories and matching algorithms. Trie is best-suited for phone directory (any matching application for that matter) because it is very efficient in matching strings.
So I have decided to implement Trie myself in C#. I have created three classes:

- Node: Represents a single tree node;
- NodeCollection: Represents the children of a node;
- Trie: Trie implementation to insert and search nodes.

Implementation
Node: Node represents a basic tree node. Node implements both Depth First and Breadth First algorithms to search its children. It also contains its Parent node and Children node. Node has a key and a Value. Key contains the single character and the value has the actual value of the node. The actual key of the node will be determined by suffixing the single character to its parent's key. Node has a special property called IsTerminal. This property is set to true if the key or a value represents a complete string. See the picture below:

http://www.codeproject.com/Articles/18033/Phone-Directory-Implementation-Using-TRIE

NodeCollection: This class is a simple collection class which implements the IEnumerable interface for iteration operations. This class contains an internal List

3. How HashSet Works Internally In Java?

Each and every element in the set is unique . So that there is no duplicate element in set .

```
public class JavaHungry {

public static void main(String[] args)
    {
        HashSet<Object> hashset = new HashSet<Object>();
hashset.add(3);
hashset.add("Java Hungry");
hashset.add("Blogspot");
System.out.println("Set is "+hashset);
    }
}
```

It will print the result :Set is [3, Java Hungry, Blogspot]

Now let add duplicate element in the above code

```
public class JavaHungry {

public static void main(String[] args)
    {
        HashSet<Object> hashset = new HashSet<Object>();
hashset.add(3);
hashset.add("Java Hungry");
hashset.add("Blogspot");
hashset.add(3);                     // duplicate elements
hashset.add("Java Hungry");              // duplicate elements
System.out.println("Set is "+hashset);
    }
}
```

It will print the result :   Set is [3, Java Hungry, Blogspot]

Now , what happens internally when you pass duplicate elements in the add() method of the Set object , It will return false and do not add to the HashSet , as the element is already present .So far so good .

But the main problem arises that how it returns false . So here is the answer

When you open the HashSet implementation of the add() method in Java Apis that is rt.jar , you will find the following code in it

```
public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, java.io.Serializable

{
private transient HashMap<E,Object> map;

    // Dummy value to associate with an Object in the backing Map

private static final Object PRESENT = new Object();


public HashSet() {
map = new HashMap<>();
    }

    // SOME CODE ,i.e Other methods in Hash Set


public boolean add(E e) {
return map.put(e, PRESENT)==null;
    }

    // SOME CODE ,i.e Other methods in Hash Set
}
```

So, we are achieving uniqueness in Set,internally in java through HashMap . Whenever you create an object of HashSet it will create an object of HashMap as you can see in the italic lines in the above code . We already discussed How HashMap works internally in java .

As we know in HashMap each key is unique . So what we do in the set is that we pass the argument in the add (Elemene E) that is E as a key in the HashMap . Now we need to associate some value to the key , so what Java apis developer did is to pass the Dummy value that is (new Object ()) which is referred by Object reference PRESENT .

So , actually when you are adding a line in HashSet like hashset.add(3)   what java does internally is that it will put that element E here 3 as a key in the HashMap(created during HashSet object creation) and some dummy value that is Object's object is passed as a value to the key .

Now if you see the code of the HashMap put(Key k,Value V) method , you will find something like this

```
public V put(K key, V value) {
//Some code
}
```

The main point to notice in above code is that put (key,value) will return

1. null , if key is unique and added to the map
2. Old Value of the key , if key is duplicate

So , in HashSet add() method , we check the return value of map.put(key,value) method with null value i.e.

```
public boolean add(E e) {
return map.put(e, PRESENT)==null;
       }
```

So , if map.put(key,value) returns null ,then
map.put(e, PRESENT)==null   will return true and element is added to the HashSet.

So , if map.put(key,value) returns old value of the key ,then
map.put(e, PRESENT)==null   will return false and element is not added to the HashSet .

How Add Method works Internally in ArrayList

Before going into the details , first look at the code example of the ArrayList add(Object) method :

```
publicclassJavaHungry {

publicstaticvoidmain(String[] args)
    {
        // TODO Auto-generated method stub

        ArrayList<Object> arrobj = new ArrayList<Object>();
arrobj.add(3);
arrobj.add("Java Hungry");
arrobj.add("Blogspot");
System.out.println(" is "+ arrobj);
    }
}
```

Output : `[3, Java Hungry, Blogspot]`

So in the above example , we have created an ArrayList object arrobj . To add elements into the arrobj we called the add method on arrobj. After printing the arrobj , we get the desired result ,i.e , values are added to the arrobj.
But the question is how add(Object) method adds the value in ArrayList. So lets find out :

There are two overloaded add() methods in ArrayList class:

1. add(Object) : adds object to the end of the list.
2. add(int index , Object) : inserts the specified object at the specified position in the list.

As internal working of both the add methods are almost similar.

Internal working of ArrayList or How add(Object) method works internally in ArrayList in Java.
*ArrayList **internally uses array object** to add(or store) the elements. In other words, ArrayList is backed by Array data -structure.The array of ArrayList is **resizable (or dynamic).***

If you look into the ArrayList Api in jdk rt.jar , you will find the following code snippets in it.

```
private transient Object[] elementData;
```

When you create the ArrayList object i.e new ArrayList() , the following code is executed :

```
this.elementData = new Object[initialCapacity];
```

There are two ways to create an ArrayList object .

a. Creates the empty list with initial capacity
1. Listarrlstobj = new ArrayList();

When we create ArrayList this way , the default constructor of the ArrayList class is invoked. It will create internally an array of Object with default size set to 10.

2. List arrlstobj = new ArrayList(20);

When we create ArrayList this way , the ArrayList will invoke the constructor with the integer argument. It will create internally an array of Object . The size of the Object[] will be equal to the argument passed in the constructor . Thus when above line of code is executed ,it creates an Object[] of capacity 20.

b. Creates the non empty list containing the elements of the specified collection.

Listarrlstobj = new ArrayList(Collection c)

The above ArrayList constructor will create an non empty list containing the elements of the collection passed in the constructor.

How the size of ArrayList grows dynamically?

Inside the add(Object) , you will find the following code

```
public boolean add(E e)
     {
         ensureCapacity(size+1);
       elementData[size++] = e;
       return true;
       }
```

Important point to note from above code is that we are checking the capacity of the ArrayList , before adding the element. ensureCapacity() determines what is the current size of occupied elements and what is the maximum size of the array. If size of the filled elements (including the new element to be added to the ArrayList class) is greater than the maximum size of the array then increase the size of array. But the size of the array cannot be increased dynamically. So what happens internally is new Array is created with capacity

Till Java 6

```
   int newCapacity = (oldCapacity * 3)/2 + 1;
```

(Update) From Java 7

```
      int newCapacity = oldCapacity + (oldCapacity >> 1);
```

also, data from the old array is copied into the new array.

Interviewer : Which copy technique internally used by the ArrayList class clone() method?

There are two copy techniques present in the object oriented programming language, deep copy and shallow copy.

Just like HashSet , ArrayList also returns the shallow copy of the HashSet object. It means elements themselves are not cloned. In other words, shallow copy is made by copying the reference of the object.

Interviewer : How to create ArrayList

One liner answer :   List

Interviewer : What happens if ArrayList is concurrently modified while iterating the elements ?

According to ArrayList Oracle Java docs , The iterators returned by the ArrayList class's iterator and listiterator method are fail-fast. See the difference between fail fast and fail safe iterator .

Interviewer : What is the runtime performance of the get() method in ArrayList , where n represents the number of elements ?

get() ,set() , size() operations run in constant time i.e O(1)

add() operation runs in amortized constant time , i.e adding n elements require O(n) time.

What happens when element is removed
When elements are removed from an ArrayList using either remove(int i) (i.e using index) or remove(Object o), in the underlying array the gap created by the removal of an element has to be filled. That is done by Shifting any subsequent elements to the left (subtracts one from their indices).

`System.arrayCopy` method is used for that.

```
System.arraycopy(elementData, index+1, elementData, index, numMoved);
```

Here index+1 is the source position and index is the destination position. Since element at the position index is removed so elements starting from index+1 are copied to destination starting from index.

4. HashMap vs ConcurrentHashMap

1. Thread -Safe:

ConcurrentHashMap is thread-safe that is the code can be accessed by single thread at a time . while HashMap is not thread-safe .

2. Synchronization Method :

HashMap can be synchronized by using synchronizedMap(HashMap) method . By using this

Method we get a HashMap object which is equivalent to the HashTable object . So every modification is performed on Map is locked on Map object.

```
import java.util.*;

public class HashMapSynchronization {
public static void main(String[] args) {
        // create map
        Map<String,String> map = new HashMap<String,String>();

        // populate the map
map.put("1","ALIVE ");
map.put("2","IS");
map.put("3","AWESOME");

        // create a synchronized map
        Map<String,String> syncMap = Collections.synchronizedMap(map);
System.out.println("Synchronized map :"+syncMap);
    }
}
```

ConcurrentHashMap synchronizes or locks on the certain portion of the Map . To optimize the performance of ConcurrentHashMap , Map is divided into different partitions depending upon the Concurrency level . So that we do not need to synchronize the whole Map Object.

3. Null Key

ConcurrentHashMap does not allow NULL values. So the key cannot be null in ConcurrentHashMap .While In HashMap there can only be one null key .

4. Performance

In multiple threaded environment HashMap is usually faster than ConcurrentHashMap . As only single thread can access the certain portion of the Map and thus reducing the performance . While in HashMap any number of threads can access the code at the same time.

Why we need ConcurrentHashMap when we already had Hashtable ?

Hashtable provides concurrent access to the Map.Entries objects by locking the entire map to perform any sort of operation (update,delete,read,create). Suppose we have a web application, the overhead created by Hashtable(locking the entire map) can be ignored under normal load. But under heavy load, the overhead of locking the entire map may prove fatal and may lead to delay response time and overtaxing of the server.

This is where ConcurrentHashMap comes to rescue. According toConcurrentHashMap Oracle docs, ConcurrentHashMap class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details. So the main purpose of this class is to provide the same functionality as of Hashtable but with a performance comparable to HashMap.

ConcurrentHashMap achieves this by a simple tweak. So this leads to our main question

5. How ConcurrentHashMap works in Java

Discuss internals of a ConcurrentHashmap (CHM) in Java

Carvia Tech | May 25, 2019 | 4 min read | 1,262 views | **Multithreading and Concurrency**

In Java 1.7, A ConcurrentHashMap is a hashmap supporting full concurrency of retrieval via volatile reads of segments and tables without locking, and adjustable expected concurrency for updates. All the operations in this class are thread-safe, although the retrieval operations does not depend on locking mechanism (non-blocking). And there is not any support for locking the entire table, in a way that prevents all access. The allowed concurrency among update operations is guided by the optional concurrencyLevel constructor argument (default is16), which is used as a hint for internal sizing.

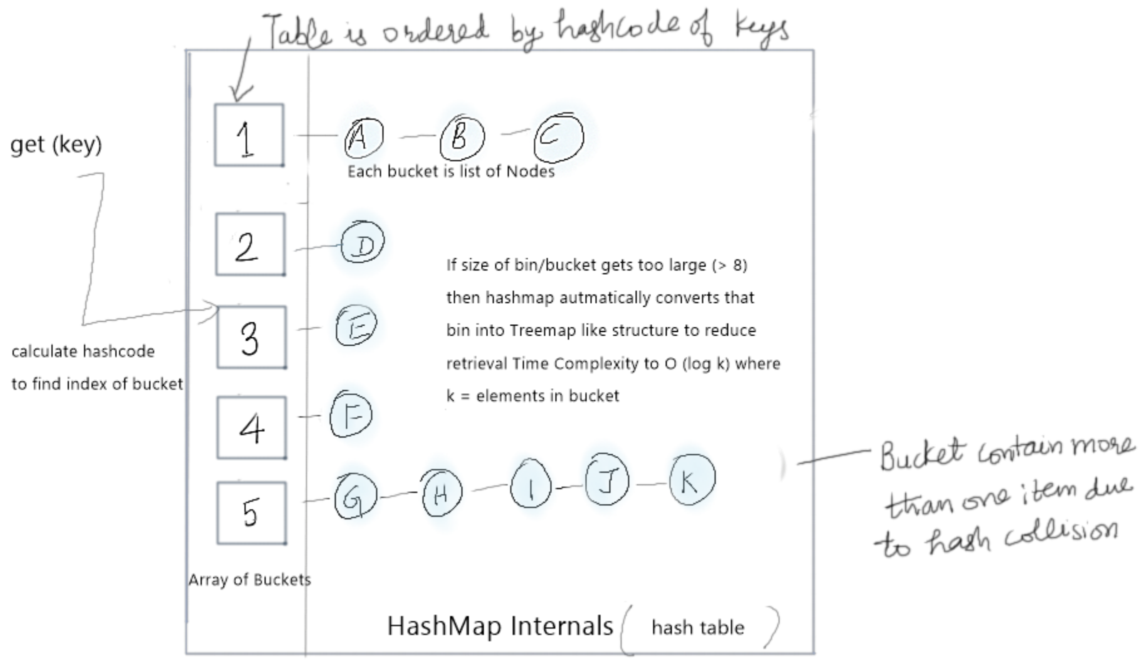


Table is ordered by hashcode of keys
get (key)
calculate hashcode to find index of bucket
1
2
3
4
5
A
B
C
Each bucket is list of Nodes
D
E
F
G
H
I
J
K
If size of bin/bucket gets too large (> 8) then hashmap autmatically converts that bin into Treemap like structure to reduce retrieval Time Complexity to O (log k) where k = elements in bucket
Bucket contain more than one item due to hash collision
Array of Buckets
HashMap Internals
hash table

ConcurrentHashMap is similar in implementation to that of HashMap, with resizable array of hash buckets, each consisting of List of HashEntry elements. Instead of a single collection lock, ConcurrentHashMap uses a fixed pool of locks that form a partition over the collection of buckets.

HashEntry class takes advantage of final and volatile variables to reflect the changes to other threads without acquiring the expensive lock for read operations.

The table inside ConcurrentHashMap is divided among Segments (which extends Reentrant Lock), each of which itself is a concurrently readable hash table. Each segment requires uses single lock to consistently update its elements flushing all the changes to main memory.

put() method holds the bucket lock for the duration of its execution and doesn't necessarily block other threads from calling get() operations on the map. It firstly searches the appropriate hash chain for the given key and if found, then it simply updates the volatile value field. Otherwise it creates a new HashEntry object and inserts it at the head of the list.

Iterator returned by the ConcurrentHashMap is fail-safe but weakly consistent. keySet().iterator() returns the iterator for the set of hash keys backed by the original map. The iterator is a “weakly consistent” iterator that will never throw ConcurrentModificationException, and guarantees to traverse elements as they existed upon construction of the iterator, and may (but is not guaranteed to) reflect any modifications subsequent to construction.

Re-sizing happens dynamically inside the map whenever required in order to maintain an upper bound on hash collision. Increase in number of buckets leads to rehashing the existing values. This is achieved by recursively acquiring lock over each bucket and then rehashing the elements from each bucket to new larger hash table.

Can two threads update the ConcurrentHashMap simultaneously

Yes it is possible that two threads can simultaneously write on the ConcurrentHashMap. ConcurrentHashMap default implementation allows 16 threads to read and write in parallel.
But in the worst case scenario , when two objects lie in the same segment or same partition, then parallel write would not be possible.

Why ConcurrentHashMap does not allow null keys and null values?

According to the author of the ConcurrentHashMap (Doug lea himself)

The main reason that nulls aren't allowed in ConcurrentMaps (ConcurrentHashMaps, ConcurrentSkipListMaps) is that ambiguities that may be just barely tolerable in non-concurrent maps can't be accommodated. The main one is that if map.get(key) returns null, you can't detect whether the key explicitly maps to null vs the key isn't mapped. In a non-concurrent map, you can check this via map.contains(key), but in a concurrent one, the map might have changed between
calls.

In simple words,

The code is like this :

```
if (map.containsKey(k)) {
return map.get(k);
} else {
throw new KeyNotPresentException();
}
```

It might be possible that key k might be deleted in between the get(k) and containsKey(k) calls. As a result , the code will return null as opposed to KeyNotPresentException (Expected Result if key is not present).

What is the difference between HashMap and ConcurrentHashMap?

The HashMap was not thread safe and therefore could not be utilized in multi-threaded applications. The ConcurrentHashMap was introduced to overcome this shortcoming and also as an alternative to using

HashTable and synchronized Maps for greater performance and uses the standard Hashing algorithms to generate hash code for storing the key value pairs.For more difference between HashMap and ConcurrentHashMap check this popular interview question HashMap vs ConcurrentHashMap in java.

Can multiple threads read from the Hashtable concurrently ?

No multiple threads cannot read simultaneously from Hashtable. Reason, the get() method of Hashtable is synchronized. As a result , at a time only one thread can access the get() method .
It is possible to achieve full concurrency for reads (all the threads read at the same time) in ConcurrentHashMap by using volatile keyword.

Does ConcurrentHashMap Iterator behaves like fail fast iterator or fail safe Iterator?

ConcurrentHashMap iterator behaves like fail safe iterator. It will not throw ConcurrentModificationException . We have already discussed Fail Fast Iterator vs Fail Safe Iterator.

Why does Java provide default value of partition count as 16 instead of very high value ?

According to Java docs ,
Ideally, you should choose a value to accommodate as many threads as will ever concurrently modify the table. Using a significantly higher value than you need can waste space and time, and a significantly lower value can lead to thread contention.

Can you write the simple example which proves ConcurrentHashMap class behaves like fail safe iterator?

```
ConcurrentHashMap Example :

import java.util.concurrent.ConcurrentHashMap;
import java.util.Iterator;


public class ConcurrentHashMapExample
{

public static void main(String[] args)
    {
        ConcurrentHashMap<String,String>        premiumPhone        =        new
ConcurrentHashMap<String,String>();
premiumPhone.put("Apple", "iPhone6");
premiumPhone.put("HTC", "HTC one");
premiumPhone.put("Samsung","S6");
```

```
        Iterator iterator = premiumPhone.keySet().iterator();

while (iterator.hasNext())
        {
System.out.println(premiumPhone.get(iterator.next()));
premiumPhone.put("Sony", "Xperia Z");
        }

    }

}
```

Output :

```
S6
HTC one
iPhone6
```

LinkedList vs ArrayList in Java

All the differences between LinkedList and ArrayList has there root on difference between Array and LinkedList data-structure. If you are familiar with Array and LinkedList data structure you will most likely derive following differences between them:

1) Since Array is an indexbased data-structure searching or getting element from Array with index is pretty fast. Array provides O(1) performance for get(index) method but remove is costly in ArrayList as you need to rearrange all elements. On the Other hand LinkedList doesn't provide Random or index based access and you need to iterate over linked list to retrieve any element which is of order O(n).

2) Insertions are easy and fast in LinkedList as compared to ArrayList because there is no risk of resizing array

and copying content to new array if array gets full which makes adding into ArrayList of O(n) in worst case, while adding is O(1) operation in LinkedList in Java. ArrayList also needs to update its index if you insert something anywhere except at the end of array.

3) Removal is like insertions better in LinkedList than ArrayList.

4) LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node.

When to use LinkedList and ArrayList in Java

As I said LinkedList is not as popular as ArrayList but still there are situation where a LinkedList is better choice than ArrayList in Java. Use LinkedList in Java if:

1) Your application can live without Random access. Because if you need nth element in LinkedList you need to first traverse up to nth element O(n) and than you get data from that node.

2) Your application is more insertion and deletion driver and you insert or remove more than retrieval. Since insertion or

removal doesn't involve resizing its much faster than ArrayList.

Read more: http://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html#ixzz4QDRVFDXj

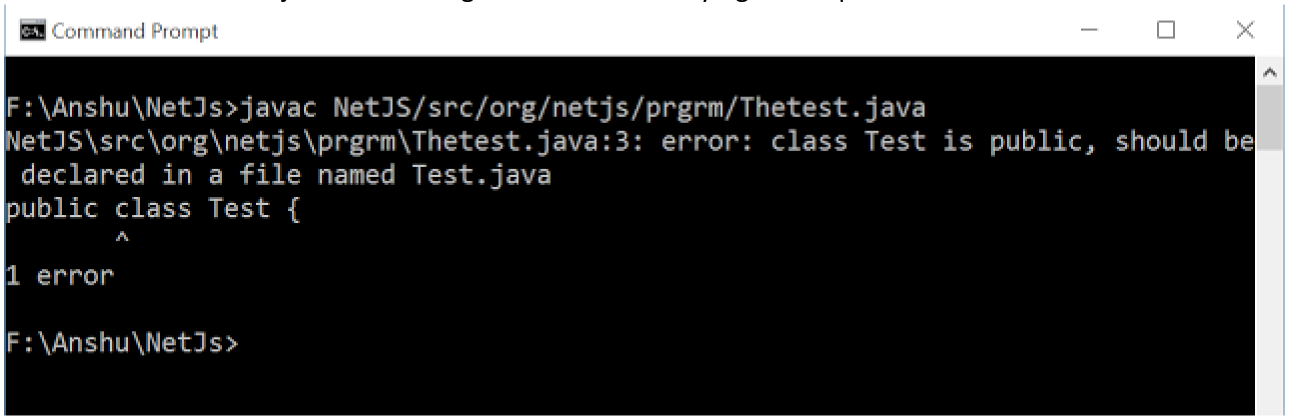
Why class name and file name should be same in Java

When we start learning java and make our first "Hello World" program, there are two things which stand out.
File name and class name should be same.
Main method signature - Public static void main(String[] args)
Here we'll talk about why this requirement of File name and class name should be same. First of all it is not true unless until there is one public class in the file. If there is public class in the file then it has to be saved with the same file name.
Let's see the case when we have a public class.

```
public class Test {
public static void main(String[] args) {
System.out.println("This is a test class");
    }

}
```

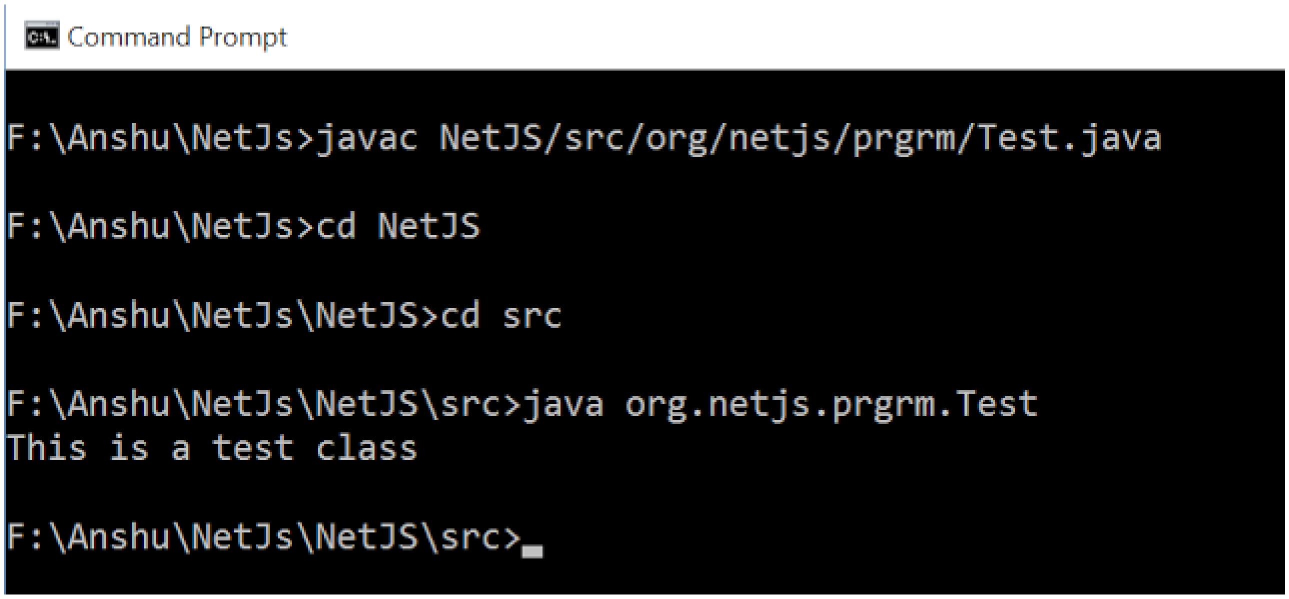
If we save it as Thetest.java then we'll get an error while trying to compile it.



```
F:\Anshu\NetJs>javac NetJS/src/org/netjs/prgrm/Thetest.java
NetJS\src\org\netjs\prgrm\Thetest.java:3: error: class Test is public, should be
 declared in a file named Test.java
public class Test {
       ^
1 error

F:\Anshu\NetJs>
```

It can be seen how compiler complains about having the public class Test and how it should be declared as Test.java.
Now if we save the class as Test.java then it compiles and runs just fine.



```
F:\Anshu\NetJs>javac NetJS/src/org/netjs/prgrm/Test.java

F:\Anshu\NetJs>cd NetJS

F:\Anshu\NetJs\NetJS>cd src

F:\Anshu\NetJs\NetJS\src>java org.netjs.prgrm.Test
This is a test class

F:\Anshu\NetJs\NetJS\src>
```

Now let's take an example when there is no public class -

```
class FinalClass{
 String a;
final void finalMethod(){

  }

}
class FinalClassDemo {
public static void main(String[] args) {

  }

}
```

I have this file with 2 classes and none of them is public, now I can save it as any name, let's say I saved it as ABC.java. Yes, it is possible if there is no public class. But that is only half of the truth! When this java file is compiled it will create 2 classes
FinalClassDemo.class
FinalClass.class
It can be seen that even if the file name is different compiled classes have the same name as the class names.
Now if we have to run it then we have to use -
java FinalClassDemo
This shows that at compile time file name may be different from the class name (provided there is no public class in the file) but at the run time it has to be same.
So the next question is why even that restriction to have the same name as class name at run time. Answer is that is how java interpreter will know which class to load and where is the entry point (main() method) otherwise interpreter may have to scan a lot of class files to determine where to start.
Points to note -
If there is no public class then file name may be different from the class name.
In case there is a public class then it is enforced that the file name is same as the public class.
Even, in the case, where the file name is different, after compilation .class files have the same name as the class names.

Having the same name as class name is how the JVM will know which class to load and where to look for entry point (main method).
That's all for this topic Why class name and file name should be same in Java.

Immutable

An immutable class is good for caching purpose because you don't need to worry about the value changes. Other benefit of immutable class is that it is inherently **thread-safe**, so you don't need to worry about thread safety in case of multi-threaded environment.

Here I am providing a way to create an immutable class in Java via an example for better understanding.

To create an immutable class in java, you have to do following steps.

1. Declare the class as final so it can't be extended.
2. Make all fields private so that direct access is not allowed.
3. Don't provide setter methods for variables
4. Make all **mutable fields final** so that it's value can be assigned only once.
5. Initialize all the fields via a constructor performing deep copy.
6. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

To understand points 4 and 5, let's run the sample Final class that works well and values don't get altered after instantiation.

FinalClassExample.java

```
package com.journaldev.java;

import java.util.HashMap;
import java.util.Iterator;

public final class FinalClassExample {

	private final int id;

	private final String name;

	private final HashMap<String,String> testMap;
	public int getId() {
		return id;
	}


	public String getName() {
		return name;
	}

	/**
	 * Accessor function for mutable objects
	 */
	public HashMap<String, String> getTestMap() {
		//return testMap;
		return (HashMap<String, String>) testMap.clone();
	}

	/**
	 * Constructor performing Deep Copy
	 * @param i
	 * @param n
	 * @param hm
	 */

	public FinalClassExample(int i, String n, HashMap<String,String> hm){
		System.out.println("Performing Deep Copy for Object initialization");
		this.id=i;
		this.name=n;
		HashMap<String,String> tempMap=new HashMap<String,String>();
		String key;
		Iterator<String> it = hm.keySet().iterator();
		while(it.hasNext()){
			key=it.next();
			tempMap.put(key, hm.get(key));
		}
		this.testMap=tempMap;
	}


	/**
	 * Constructor performing Shallow Copy
	 * @param i
	 * @param n
	 * @param hm
	 */
	/**
	public FinalClassExample(int i, String n, HashMap<String,String> hm){
		System.out.println("Performing Shallow Copy for Object initialization");
		this.id=i;
		this.name=n;
		this.testMap=hm;
	}
	*/

	/**
	 * To test the consequences of Shallow Copy and how to avoid it with Deep Copy for
creating immutable classes
	 * @param args
	 */
	public static void main(String[] args) {
		HashMap<String, String> h1 = new HashMap<String,String>();
		h1.put("1", "first");
		h1.put("2", "second");

		String s = "original";

		int i=10;

		FinalClassExample ce = new FinalClassExample(i,s,h1);

		//Lets see whether its copy by field or reference
		System.out.println(s==ce.getName());
		System.out.println(h1 == ce.getTestMap());
		//print the ce values
		System.out.println("ce id:"+ce.getId());
		System.out.println("ce name:"+ce.getName());
		System.out.println("ce testMap:"+ce.getTestMap());
		//change the local variable values
		i=20;
		s="modified";
		h1.put("3", "third");
		//print the values again
		System.out.println("ce id after local variable change:"+ce.getId());
		System.out.println("ce name after local variable change:"+ce.getName());
		System.out.println("ce testMap after local variable
change:"+ce.getTestMap());
```

```
		HashMap<String, String> hmTest = ce.getTestMap();
		hmTest.put("4", "new");
		
		System.out.println("ce testMap after changing variable from accessor
methods:"+ce.getTestMap());

	}

}
```

Output of the above immutable class in java example program is:

```
Performing Deep Copy for Object initialization
true
false
ce id:10
ce name:original
ce testMap:{2=second, 1=first}
ce id after local variable change:10
ce name after local variable change:original
ce testMap after local variable change:{2=second, 1=first}
ce testMap after changing variable from accessor methods:{2=second, 1=first}
```

Now let's comment the constructor providing deep copy and uncomment the constructor providing a shallow copy. Also uncomment the return statement in `getTestMap()` method that returns the actual object reference and then execute the program once again.

```
Performing Shallow Copy for Object initialization
true
true
ce id:10
ce name:original
ce testMap:{2=second, 1=first}
ce id after local variable change:10
ce name after local variable change:original
ce testMap after local variable change:{3=third, 2=second, 1=first}
ce testMap after changing variable from accessor methods:{3=third, 2=second, 1=first,
4=new}
```

As you can see from the output, HashMap values got changed because of shallow copy in the constructor and providing a direct reference to the original object in the getter function.

That’s all for how to create an immutable class in java. If I have missed something here, feel free to comment.

Singleton Pattern

Following are some reasons which make sense to me for using Enum to implement Singleton pattern in Java. By the way If you like articles on design pattern than you can also check my post on Builder design pattern and Decorator design pattern .

This is by far biggest advantage, if you have been writing Singletons prior to Java 5 than you know that even with double checked locking you can have more than one instances. Though that issue is fixed with Java memory model improvement and guarantee provided by volatile variables from Java 5 onwards but it still tricky to write for many beginners.Compared to double checked locking with synchronization Enum singletons are cake walk. If you don't believe than just compare below code for conventional singleton with double checked locking and Enum

Singleton using Enum in Java

This is the way we generally declare Enum Singleton , it may contain instace variable and instance method but for sake of simplicity I haven’t used any, just beware that if you are using any instance method than you need to ensure thread-safety of that method if at all it affect the state of object. By default creation of Enum instance is thread safe but any other method on Enum is programmers’ responsibility.

```
/**
* Singleton pattern example using Java Enumj
*/

public enum EasySingleton{
    INSTANCE;
}
```

You can acess it by EasySingleton.INSTANCE, much easier than calling getInstance() method on Singleton.

Singleton example with double checked locking

Below code is an example of double checked locking in Singleton pattern, here getInstance() method checks two times to see whether INSTANCE is null or not and that’s why it’s called double checked locking pattern, remember that double checked locking is broker before Java 5 but with the guranteed of volatile variable in Java 5 memory model, it should work perfectly.

```java
/**
* Singleton pattern example with Double checked Locking
*/

public class DoubleCheckedLockingSingleton{
     private volatile DoubleCheckedLockingSingleton INSTANCE;

     private DoubleCheckedLockingSingleton(){}

     public DoubleCheckedLockingSingleton getInstance(){
          if(INSTANCE == null){
             synchronized(DoubleCheckedLockingSingleton.class){
                 //double checking Singleton instance
                 if(INSTANCE == null){
                      INSTANCE = new DoubleCheckedLockingSingleton();
                 }
             }
          }
          return INSTANCE;
     }
}
```

You can call DoubleCheckedLockingSingleton.getInstance() to get access of this Singleton class.

Now Just look at amount of code needed to create a lazy loaded thread-safe Singleton. With Enum Singleton pattern you can have that in one line because creation of Enum instance is thread-safe and guaranteed by JVM.

People may argue that there are better way to write Singleton instead of Double checked locking approach but every approach has there own advantages and disadvantages like I mostly prefer static field Singleton initialized during class loading as shown in below example, but keep in mind that is not a lazy loaded Singleton:

Singleton pattern with static factory method

This is one of my favorite methods to implement Singleton pattern in Java, Since Singleton instance is static and final variable it initialized when class is first loaded into memory so creation of instance is inherently thread-safe.

```java
/**
* Singleton pattern example with static factory method
*/

public class Singleton{
    //initailzed during class loading
    private static final Singleton INSTANCE = new Singleton();
    //to prevent creating another instance of Singleton
    private Singleton(){}

    public static Singleton getSingleton(){
        return INSTANCE;
    }
}
```

You can call Singleton.getSingleton() to get access of this class.

2) Enum Singletons handled Serialization by themselves

Another problem with conventional Singletons are that once you implement serializable interface they are no longer remain Singleton because readObject() method always return a new instance just like constructor in Java. you can avoid that by using readResolve() method and discarding newly created instance by replacing with Singeton as shwon in below example :

```
    //readResolve to prevent another instance of Singleton
    private Object readResolve(){
        return INSTANCE;
    }
```

This can become even more complex if your Singleton Class maintain state, as you need to make them transient, but witn Enum Singleton, Serialization is guarnateed by JVM.

3) Creation of Enum instance is thread-safe

As stated in point 1 since creation of Enum instance is thread-safe by default you don't need to worry about double checked locking.

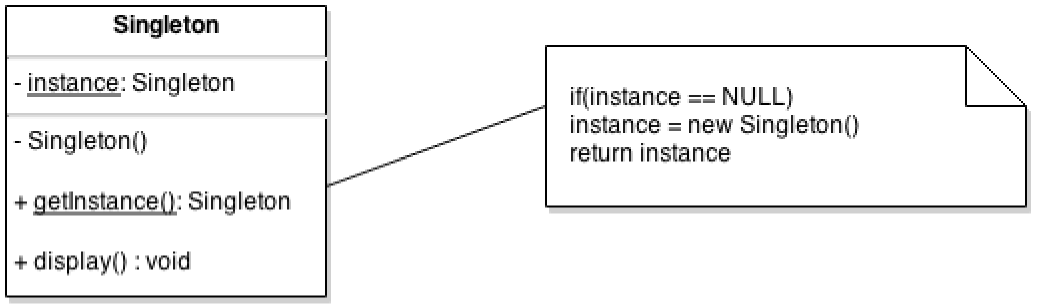
Read more: http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html#ixzz4NTrXnx7A

Singleton design pattern belongs to the creational family of patterns that governs the instantiation process. This pattern ensures at most one instance of a particular class is ever created in your application. Following are some of the real time examples listed below.

1. **Project Configuration**: Class that reads your project configuration can be made Singleton. By making this singleton, you are allowing global access for all classes in your application. If the project configs are stored in a file, it just reads once and holds on application cache. You don't have to read the file multiple times.
2. **Application Log:** Logger will be used everywhere in your application. It must be initialized once and used everywhere.
3. **Analytics and Reporting:** If you are using some kind of data analysis tool like Google Analytics, you will notice that they are designed to be singleton. It initializes once and being used everywhere for each user action.

1. Class Diagram

In the above class diagram, the Singleton class has a private static instance variable named “instance”. Default constructor of “Singleton” class is made private, to prevent other class to instantiate it. Static getInstance() method, will be accessible globally which returns the singleton class object.



2. How to Implement

1. Make the default constructor private. Private constructor prevents the direct instantiation of the object from other classes.
2. Create a public static getInstance() method. A member declared as static can be accessed without creating object. This method returns the instance of Singleton class.
3. Lazy initialization is preferable, so create object on first use.

Example: Creating a singleton class using lazy initialization

Singleton.java

```java
package com.javatechig.creational.singleton;

class Singleton {
		private static Singleton instance;

		/* Private Constructor prevents any other class from instantiating */
		private Singleton() {
		}

		public static Singleton getInstance() {

				/* Lazy initialization, creating object on first use */
				if (instance == null) {
						instance = new Singleton();
				}
				return instance;
		}

		public void display() {
				System.out.println("Hurray! I am display from Singleton!");
		}
}
```

SingletonTest.java

```java
package com.javatechig.creational.singleton;

public class SingletonTest {
		public static void main(String args[]) {

				/* Compilation error not allowed */
				// Singleton object = new Singleton();

				Singleton object = Singleton.getInstance();
				object.display();

				/* Your Program Logic */
				System.out.println("Singleton object obtained");
		}
}
```

In the above example, we have created Singleton instance using lazy initialization. The getInstance() method is instantiating the class for the first use.

3. Singleton and Thread Safety

Thread-safe code is particularly important in Singleton. Two instances of Singleton class will be created if the getInstance() called simultaneously by two threads. To avoid such issues, we'll make the getInstance() method synchronized. This way we force every thread to wait until its turn before it executes. I.e. no two threads can be entered into getInstance() method at the same time.

```
public static synchronized Singleton getInstance() {

                        /* Lazy initialization, creating object on first use */
                        if (instance == null) {
                                    instance = new Singleton();
                        }
                        return instance;
}
```

The above implementation will answer to thread safety problem, however synchronized methods are expensive and will have serious performance hit. We will change the getInstance() method not to check for synchronization, if instance is already created.

```
public static synchronized Singleton getInstance() {

                        /* Lazy initialization, creating object on first use */
                        if (instance == null) {
                                    synchronized (Singleton.class) {
                                                if (instance == null) {
                                                            instance = new Singleton();
                                                }
                                    }
                        }

            return instance;

}
```

4. Singleton and Early Initialization

Using early initialization we will initialize upfront before your class is being loaded. This way you don't need to check for synchronization as it is initialized before being used ever.

```
package com.javatechig.creational.singleton;

class Singleton implements Cloneable {
            private static Singleton instance;

            /* Private Constructor prevents any other class from instantiating */
            private Singleton() {
            }

            public static synchronized Singleton getInstance() {

                        /* Lazy initialization, creating object on first use */
                        if (instance == null) {
                                    synchronized (Singleton.class) {
                                                if (instance == null) {
                                                            instance = new Singleton();
                                                }
                                    }
                        }

                        return instance;
            }

            /* Prevent cloning */
            @Override
            public Object clone() throws CloneNotSupportedException {
                        throw new CloneNotSupportedException();
            }

            public void display() {
                        System.out.println("Hurray! I am display from Singleton!");
            }
}
```

5. Singleton and Object Cloning

Java has the ability to create a copy of object with similar attributes and state form original object. This concept in java is called cloning. To implement cloning, we have to implement `java.lang.Cloneable` interface and override `clone()` method from Object class. It is a good idea to prevent cloning in a singleton class. To prevent cloning on singleton object, let us explicitly throw `CloneNotSupportedException` exception in clone() method.

```
package com.javatechig.creational.singleton;

import java.io.Serializable;

class Singleton implements Cloneable, Serializable {

            private static Singleton instance;

            private int value;

            /* Private Constructor prevents any other class from instantiating */
            private Singleton() {
            }

            public static synchronized Singleton getInstance() {

                        /* Lazy initialization, creating object on first use */
                        if (instance == null) {
                                    synchronized (Singleton.class) {
                                                if (instance == null) {
                                                            instance = new Singleton();
                                                }
                                    }
                        }

                        return instance;
            }

            /* Restrict cloning of object */
            @Override
            public Object clone() throws CloneNotSupportedException {
                        throw new CloneNotSupportedException();
            }

            public void display() {
                        System.out.println("Hurray! I am display from Singleton!");
            }

            public int getValue() {
                        return value;
            }

            public void setValue(int value) {
                        this.value = value;
            }
}
```

6. Singleton and Serialization

Java Serialization allows to convert the state of an object into stream of bytes so that it can easily stored or transferred. Once object is serialized, you can deserialize it, back to object from byte stream. If a singleton class is meant to be serialized, it will end up creating duplicate objects. Let us have a look into the example below, explaining the problem,

Singleton.java

```
package com.javatechig.creational.singleton;

import java.io.Serializable;

class Singleton implements Cloneable, Serializable {

            private static Singleton instance;

            private int value;

            /* Private Constructor prevents any other class from instantiating */
            private Singleton() {
            }

            public static synchronized Singleton getInstance() {
```

```
                        /* Lazy initialization, creating object on first use */
                        if (instance == null) {
                                    synchronized (Singleton.class) {
                                                if (instance == null) {
                                                            instance = new Singleton();
                                                }
                                    }
                        }

                        return instance;
            }

            /* Restrict cloning of object */
            @Override
            public Object clone() throws CloneNotSupportedException {
                        throw new CloneNotSupportedException();
            }

            public void display() {
                        System.out.println("Hurray! I am display from Singleton!");
            }

            public int getValue() {
                        return value;
            }

            public void setValue(int value) {
                        this.value = value;
            }
}
```

SerializationTest.java

```
package com.javatechig.creational.singleton;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;

public class SerializationTest {

            public static void main(String[] args) {

                        //getting singleton instance
                        Singleton instanceOne = Singleton.getInstance();
                        instanceOne.setValue(10);

                        try {
            // Serialize to a file
                                    ObjectOutput out = new ObjectOutputStream(new FileOutputStream("filename.txt"));
            out.writeObject(instanceOne);
            out.close();

            instanceOne.setValue(20);

            // Serialize to a file
            ObjectInput in = new ObjectInputStream(new FileInputStream("filename.txt"));
            Singleton instanceTwo = (Singleton) in.readObject();
            in.close();

            System.out.println("Instance One Value= " + instanceOne.getValue());
            System.out.println("Instance Two Value= " + instanceTwo.getValue());

        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
  }
}
```

In the above example, the Singleton class is implementing Serializable interface, which means the state of its object can be persisted. SerializableTest is the test class, used to test Singleton class. Inside main() method we are persisting the state of Singleton instance into a file and retrieve it later. Now compile and run the program, you will notice that the state of both instances (instanceOne and instanceTwo) are different, which means that they are two different objects. Here we are violating the rules of singleton by allowing it to create two different objects of same class.

To solve this issue, we need to include `readResolve()` method in our DemoSingleton class. This method will be invoked before the object is deserialized. Inside this method, we will call `getInstance()` method to ensure single instance of Singleton class is exist application wide.

Singleton.java

```
package com.javatechig.creational.singleton;

import java.io.Serializable;

class Singleton implements Cloneable, Serializable{

            private static final long serialVersionUID = 1L;
            private static Singleton instance;
            private int value;

            /* Private Constructor prevents any other class from instantiating */
            private Singleton() {
            }

            public static synchronized Singleton getInstance() {

                        /* Lazy initialization, creating object on first use */
                        if (instance == null) {
                                    synchronized (Singleton.class) {
                                                if (instance == null) {
                                                            instance = new Singleton();
                                                }
                                    }
                        }

                        return instance;
            }

            protected Object readResolve() {
        return getInstance();
    }

            /* Restrict cloning of object */
            @Override
            public Object clone() throws CloneNotSupportedException {
                        throw new CloneNotSupportedException();
            }

            public void display() {
                        System.out.println("Hurray! I am display from Singleton!");
            }

            public int getValue() {
                        return value;
            }

            public void setValue(int value) {
                        this.value = value;
            }
}
```

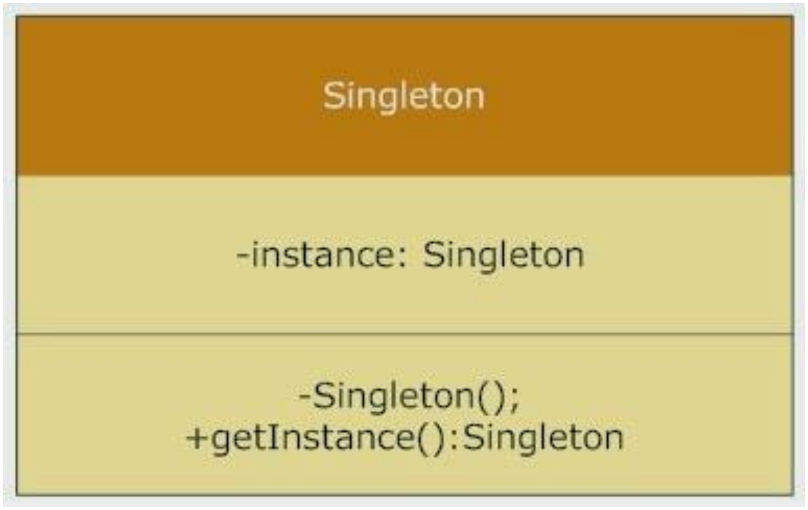
Behind the scene

1. Serializable is a marker interface. A marker interface contains no fields or method declaration. Serializable interface is used as a marker to specify that the class can be serialized.
2. `writeObject()` and `readObject()` methods are called while the object is serialized or deserialized. `writeObject()` method is used to write the state of the object for its particular class so that its corresponding `readObject()`method can read it.
3. `readObject()` method is responsible for reading the object from stream and to restore the class fields.

What is Singleton class? Have you used Singleton before?
Singleton is a class which has only one instance in whole application and provides a `getInstance()` method to access the singleton instance. There are many classes in JDK which is implemented using Singleton pattern like `java.lang.Runtime` which provides `getRuntime()` method to get access of it and used to get free memory and total memory in Java.

Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?
Answer : Any class which you want to be available to whole application and whole only one instance is viable is candidate of becoming Singleton. One example of this is `Runtime` class , since on whole java application only one runtime environment can be possible making `Runtime` Singleton is right decision. Another example is a utility classes like Popup in GUI application, if you want to show popup with message you can have one PopUp class on whole GUI application and anytime just get its instance, and call `show()` with message.

Can you write code for getInstance() method of a Singleton class in Java?
Answer : Until asked don't write code using double checked locking as it is more complex and chances of errors are more but if you have deep knowledge of double checked locking, volatile variable and lazy loading than this is your chance to shine. I have shared code examples of writing singleton classes using enum, using static factory and with double checked locking in my recent post Why Enum Singletons are better in Java, please see there.



Is it better to make whole getInstance() method synchronized or just critical section is enough? Which one you will prefer?
This is really nice question and I mostly asked to just quickly check whether candidate is aware of performance trade off of unnecessary locking or not. Since locking only make sense when we need to create instance and rest of the time its just read only access so locking of critical section is always better option. read more about synchronization on How Synchronization works in Java

Answer : This is again related to double checked locking pattern, well synchronization is costly and when you apply this on whole method than call to `getInstance()` will be synchronized and contented. Since synchronization is only needed during initialization on singleton instance, to prevent creating another instance of Singleton, It's better to only synchronize critical section and not whole method. Singleton pattern is also closely related to factory design pattern where `getInstance()` serves as static factory method.

What is lazy and early loading of Singleton and how will you implement it?
Answer : As there are many ways to implement Singleton like using double checked locking or Singleton class with static final instance initialized during class loading. Former is called lazy loading because Singleton instance is created only when client calls `getInstance()` method while later is called early loading because Singleton instance is created when class is loaded into memory.

Give me some examples of Singleton pattern from Java Development Kit?
This is open question to all, please share which classes are Singleton in JDK. Answer to this question is `java.lang.Runtime`

Answer : There are many classes in Java Development Kit which is written using singleton pattern, here are few of them:

1. Java.lang.Runtime with getRuntime() method
2. Java.awt.Toolkit with getDefaultToolkit()
3. Java.awt.Desktop with getDesktop()

What is double checked locking in Singleton?
One of the most hyped question on Singleton pattern and really demands complete understanding to get it right because of Java Memory model caveat prior to Java 5. If a guy comes up with a solution of using volatile keyword with Singleton instance and explains it then it really shows it has in depth knowledge of Java memory model and he is constantly updating his Java knowledge.

Answer : Double checked locking is a technique to prevent creating another instance of Singleton when call to `getInstance()` method is made in multi-threading environment. In Double checked locking pattern as shown in below example, singleton instance is checked two times before initialization. See here to learn more about double-checked-locking in Java.

```
public static Singleton getInstance(){
if(_INSTANCE == null){
synchronized(Singleton.class){
//double checked locking - because second check of Singleton instance with lock
if(_INSTANCE == null){
_INSTANCE=newSingleton();
            }
          }
       }
return_INSTANCE;
}
```

Double checked locking should only be used when you have requirement for lazy initialization otherwise use Enum to implement singleton or simple static final variable.

How do you prevent for creating another instance of Singleton using clone() method?
Answer : Preferred way is not to implement Cloneable interface as why should one wants to create `clone()` of Singleton and if you do just throw Exception from `clone()` method as "Can not create clone of Singleton class".

How do you prevent for creating another instance of Singleton using reflection?
Open to all. In my opinion throwing exception from constructor is an option.
Answer: This is similar to previous interview question. Since constructor of Singleton class is supposed to be private it prevents creating instance of Singleton from outside but Reflection can access private fields and methods, which opens a threat of another instance. This can be avoided by throwing Exception from constructor as "Singleton already initialized"

How do you prevent for creating another instance of Singleton during serialization?
Another great question which requires knowledge of Serialization in Java and how to use it for persisting Singleton classes. This is open to you all but in my opinion use of readResolve()

method can sort this out for you.
Answer: You can prevent this by using `readResolve()` method, since during serialization `readObject()` is used to create instance and it return new instance every time but by using readResolve you can replace it with original Singleton instance. I have shared code on how to do it in my post Enum as Singleton in Java. This is also one of the reason I have said that use Enum to create Singleton because serialization of enum is taken care by JVM and it provides guaranteed of that.

When is Singleton not a Singleton in Java?
There is a very good article present in Sun's Java site which discusses various scenarios when a Singleton is not really remains Singleton and multiple instance of Singleton is possible. Here is the link of that article http://java.sun.com/developer/technicalArticles/Programming/singletons/

Apart from these questions on Singleton pattern, some of my reader contribute few more questions, which I included here. Thank you guys for your contribution.

Why you should avoid the singleton anti-pattern at all and replace it with DI?
Answer : Singleton Dependency Injection: every class that needs access to a singleton gets the object through its constructors or with a DI-container.

Why Singleton is Anti pattern
With more and more classes calling `getInstance()` the code gets more and more tightly coupled, monolithic, not testable and hard to change and hard to reuse because of not configurable, hidden dependencies. Also, there would be no need for this clumsy double checked locking if you call `getInstance` less often (i.e. once).

How many ways you can write Singleton Class in Java?
Answer : I know at least four ways to implement Singleton pattern in Java

1. Singleton by synchronizing `getInstance()` method
2. Singleton with public static final field initialized during class loading.
3. Singleton generated by static nested class, also referred as Singleton holder pattern.
4. From Java 5 on-wards using Enums

How to write thread-safe Singleton in Java?
Answer : Thread safe Singleton usually refers to write thread safe code which creates one and only one instance of Singleton if called by multiple thread at same time. There are many ways

to achieve this like by using double checked locking technique as shown above and by using Enum or Singleton initialized by class loader.

At last few more questions for your practice, contributed by Mansi, Thank you Mansi

14) Singleton vs Static Class?
15) When to choose Singleton over Static Class?
16) Can you replace Singleton with Static Class in Java?
17) Difference between Singleton and Static Class in java?
18) Advantage of Singleton over Static Class?

Read more: http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html#ixzz4NVAxz7rl

How to make Thread-Safe Code in Java

Example of Non Thread Safe Code in Java

Here is an example of **non thread-safe code**, look at the code and find out *why this code is not thread safe* ?

```
/*
 * Non Thread-Safe Class in Java
 */
public class Counter {

   private int count;

   /*
    * This method is not thread-safe because ++ is not an atomic operation
    */
   public int getCount(){
       return count++;
   }
}
```

Above example is not thread-safe because ++ (increment operator) is not an atomic operation and can be broken down into read, update and write operation. if multiple thread call `getCount()` approximately same time each of these three operation may coincide or overlap with each other for example while thread 1 is updating value , thread 2 reads and still gets old value, which eventually let thread 2 override thread 1 increment and one count is lost because multiple thread called it concurrently.

How to make code Thread-Safe in Java

There are multiple ways to make this code thread safe in Java:

1) Use synchronized keyword in Java and lock the getCount() method so that only one thread can execute it at a time which removes possibility of coinciding or interleaving.

2) use Atomic Integer, which makes this ++ operation atomic and since atomic operations are thread-safe and saves cost of external synchronization.

```
/*
 * Thread-Safe Example in Java
 */
public class Counter {

    private int count;
    AtomicInteger atomicCount = new AtomicInteger( 0 );


    /*
     * This method thread-safe now because of locking and synchornization
     */
    public synchronized int getCount(){
        return count++;
    }

    /*
     * This method is thread-safe because count is incremented atomically
     */
    public int getCountAtomically(){
        return atomicCount.incrementAndGet();
    }
}
```

Important points about Thread-Safety in Java

Here is some points worth remembering to write thread safe code in Java, these knowledge also helps you to avoid some serious concurrency issues in Java like race condition or deadlock in Java:

1) Immutable objects are by default thread-safe because there state can not be modified once created. Since String is immutable in Java, its inherently thread-safe.

2) Read only or final variables in Java are also thread-safe in Java.

3) Locking is one way of achieving thread-safety in Java.

4) Static variables if not synchronized properly becomes major cause of thread-safety issues.

5) Example of thread-safe class in Java: Vector, Hashtable, ConcurrentHashMap, String etc.

6) Atomic operations in Java are thread-safe e.g. reading a 32 bit int from memory because its an atomic operation it can't interleave with other thread.

7) local variables are also thread-safe because each thread has there own copy and using local variables is good way to writing thread-safe code in Java.

8) In order to avoid thread-safety issue minimize sharing of objects between multiple thread.

9) Volatile keyword in Java can also be used to instruct thread not to cache variables and read from main memory and can also instruct JVM not to reorder or optimize code from threading perspective.

Read more: http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html#ixzz4UmsH8WFK

Difference Between ClassNotFoundException Vs NoClassDefFoundError In Java

In Java, both ClassNotFoundException and NoClassDefFoundError occur when a particular class is not found at run time. But, they occur at different scenarios. ClassNotFoundException is an exception which occurs when you try to load a class at run time using Class.forName() or loadClass() methods and mentioned classes are not found in the classpath. On the other hand, NoClassDefFoundError is an error which occurs when a particular class is present at compile time but it was missing at run time. In this tutorial, we will see the differences between ClassNotFoundException Vs NoClassDefFoundError in java and when they occur.

ClassNotFoundException In Java :

ClassNotFoundException is a run time exception which is thrown when an application tries to load a class at run time using Class.forName() or loadClass() or findSystemClass() methods and the class with specified name are not found in the classpath. For example, you may have come across this exception when you try to connect to MySQL or Oracle databases and you have not updated the classpath with required JAR files. In most of time, this exception occurs when you try to run an application without updating the classpath with required JAR files.

For example, below program will throw ClassNotFoundException if the mentioned class “oracle.jdbc.driver.OracleDriver” is not found in the classpath.

```
        public class MainClass

{

    public static void main(String[] args)

    {

        try

        {

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

        }

        catch (ClassNotFoundException e)

        {

            e.printStackTrace();

        }

    }

}
```

If you run the above program without updating the classpath with required JAR files, you will get the exception like below,

```
        java.lang.ClassNotFoundException: oracle.jdbc.driver.OracleDriver

    at java.net.URLClassLoader.findClass(Unknown Source)

    at java.lang.ClassLoader.loadClass(Unknown Source)

    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)

    at java.lang.ClassLoader.loadClass(Unknown Source)

    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Unknown Source)
    at pack1.MainClass.main(MainClass.java:17)
```

NoClassDefFoundError In Java :

NoClassDefFoundError is an error which is thrown when Java Runtime System tries to load the definition of a class and class definition is no longer available. The required class definition was present at compile time but it was missing at run time. For example, compile the below program.

```
class A
{

}

public class B
{
    public static void main(String[] args)
    {
        A a = new A();
    }
}
```

When you compile the above program, two .class files will be generated. One is A.class and another one is B.class. If you remove the A.class file and run the B.class file, Java Runtime System will throw NoClassDefFoundError like below,

```
Exception in thread "main" java.lang.NoClassDefFoundError: A
        at MainClass.main(MainClass.java:10)
Caused by: java.lang.ClassNotFoundException: A
        at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
```

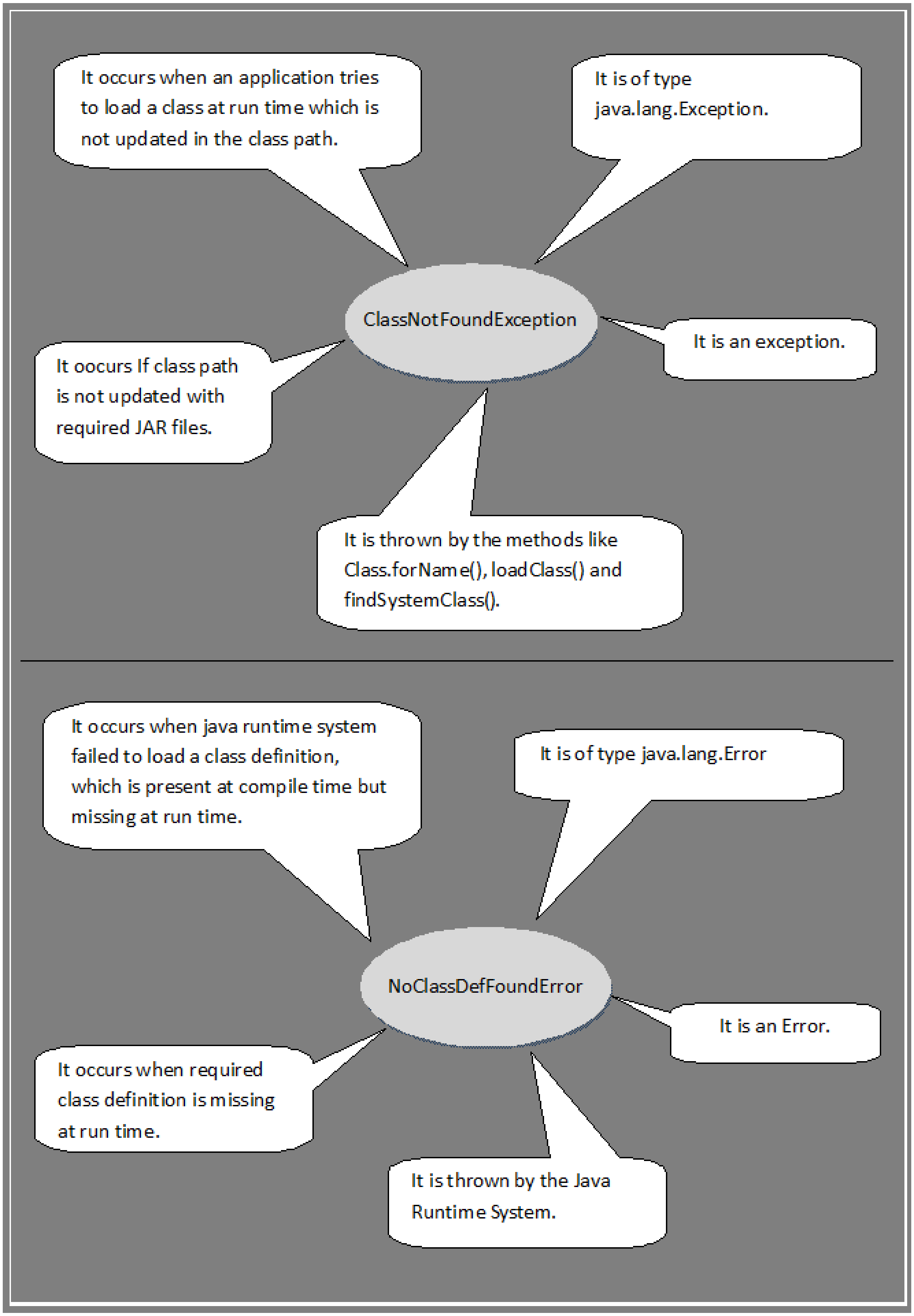
Below is the quick recap of above findings.

Difference Between ClassNotFoundException Vs NoClassDefFoundError In Java :

?

ClassNotFoundException	**NoClassDefFoundError**
It is an exception. It is of type java.lang.Exception.	It is an error. It is of type java.lang.Error.
It occurs when an application tries to load a class at run time which is not updated in the classpath.	It occurs when java runtime system doesn’t find a class definition, which is present at compile time, but missing at

	run time.
It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass().	It is thrown by the Java Runtime System.
It occurs when classpath is not updated with required JAR files.	It occurs when required class definition is missing at run time.


It occurs when an application tries to load a class at run time which is not updated in the class path.
It is of type java.lang.Exception.
ClassNotFoundException
It is an exception.
It oocurs If class path is not updated with required JAR files.
It is thrown by the methods like Class.forName(), loadClass() and findSystemClass().
It occurs when java runtime system failed to load a class definition, which is present at compile time but missing at run time.
It is of type java.lang.Error
NoClassDefFoundError
It is an Error.
It occurs when required class definition is missing at run time.
It is thrown by the Java Runtime System.

Difference between Comparable and Comparator in Java

Carvia Tech | May 05, 2019 | 3 min read | 172 views

Both comprabale and comparator are interfaces from Java Collections Framework that allow sorting of collections. But both of these interfaces are meant for different purpose.

Comparable	**Comparator**
Imposes a total ordering on the objects of each class that implements it. This ordering is referred to as class’s *natural ordering*, and the class’s `compareTo` method is referred to as its natural comparison method.	It is a comparison function, which imposes a *total ordering* on some collection of objects.
Comparable provides only single sorting sequence (based on single or multiple fields)	**N** number of comparators can be created for different sorting sequences
Comparable affects the original class	Unlike `Comparable`, `Comparator` is external to original class that we are comparing.
Comparable provides `compareTo(T other)` method for defining natural ordering of class	Comparator provides `compare(T o1, T o2)` to sort elements
It is part of `java.lang` package	It is part of `java.util` package
`Collection.sort(List)`, `SortedMap` and `SortedSet` can use this	This function can be passed to `Collections.sort(List, Comparator)`, `SortedSet` and `SortedMap` to allow precise control of sorting order.
It is strongly recommended that natural ordering be consistent with equals, else sorted set and sorted map will behave strangely. Virtually all Java classes that implement `comparable` have natural orderings that are consistent with equals	No such requirements

Example Code

Lets take a concrete example of Comparable and Comparator

Person class with Natural ordering based on name

```
class Person implements Comparable<Person>{
   int age;
   String name;
   String email;

   @Override
   public int compareTo(Person o) {
      return name.compareTo(o.getName());
   }
}
```

Now we can sort a collection of persons based on its natural ordering using `Collections.sort` method.

Sorting in Natural Order

```
List<Person> students = Arrays.asList(
      new Person(20,"Bob", "bob@mail.com"),
      new Person(19, "Jane", "Jane@mail.com"),
      new Person(21,"Foo", "foo@mail.com")
);
Collections.sort(students);
```

This program will always sort persons based on their name i.e. Bob → Foo and thenJane.

Lets define two comparators now, one for name based comparison and another on basis of age.

```
class NameComparator implements Comparator<Person> {

   @Override
   public int compare(Person o1, Person o2) {
      return o1.name.compareTo(o2.name);
   }
}

class AgeComparator implements Comparator<Person> {

   @Override
   public int compare(Person o1, Person o2) {
      return ((Integer)o1.age).compareTo(o2.age);
   }
}
```

Now we can specify one of the above defined sorting function to choose sorting on demand:

Soritng based on Age

```
List<Person> students = Arrays.asList(
      new Person(20,"Bob"),
      new Person(19, "Jane"),
      new Person(21,"Foo")
);
Collections.sort(students, new AgeComparator());
```

Eay enough?

Chaining of comparison operations in Java 8

Java 8’s lambda expressions brings syntax improvements to sorting capabilities of Java Collections Framework. We can even chain multiple sorting operations using chaining of operations, as shown in below code:

Comparing by name and then age (if name is same)

```
List<Person> students = Arrays.asList(
        new Person(20,"Foo"),
        new Person(19, "Jane"),
        new Person(21,"Foo")
);
Collections.sort(students);
students.stream()
        .sorted(Comparator
                .comparing(Person::getName)
                .thenComparing(Person::getAge)
        ).forEach(System.out::println);
```

Even reverse sort is easy enough:

Soritng by Name and then Age in reverse order

```
students.stream()
        .sorted(Comparator
                .comparing(Person::getName)
                .thenComparing(Person::getAge)
                .reversed()
        ).forEach(System.out::println);
```

Sorting Age in reverse direction. `reversed()` Returns a comparator that imposes the reverse ordering of this comparator.

Case insensitive ordering

For certain fields like email, case does not matter. Its easy to achieve this using Java 8:

Case insensitive ordering for email

```
students.stream()
        .sorted(Comparator
                .comparing(Person::getName)
                .thenComparing(Person::getAge)
                .thenComparing(Person::getEmail, Comparator.nullsLast(String.CASE_INSENSITIVE_ORDER))
```

```
    ).forEach(System.out::println);
```

null values are pushed to last

WeakReference vs SoftReference in Java

1. Strong reference
2. Weak Reference
3. Soft Reference
4. Phantom Reference

Strong Reference is most simple as we use it in our day to day programming life e.g. in the code, `String s = "abc"` , reference variable **s** has strong reference to String object `"abc"`. Any object which has Strong reference attached to it is *not eligible for garbage collection*. Obviously these are objects which is needed by Java program. Weak Reference are represented using `java.lang.ref.WeakReference` class and you can create Weak Reference by using following code :

```
Counter counter =newCounter(); // strong reference - line 1
WeakReference<Counter> weakCounter =newWeakReference<Counter>(counter); //weak reference
counter=null; // now Counter object is eligible for garbage collection
```

Now as soon as you make strong reference `counter = null`, counter object created on line 1 becomes eligible for garbage collection; because it doesn't have any more Strong reference and Weak reference by reference variable `weakCounter` can not prevent `Counter` object from being garbage collected. On the other hand, had this been Soft Reference, Counter object is not garbage collected until JVM absolutely needs memory. Soft reference in Java is represented using `java.lang.ref.SoftReference` class. You can use following code to create a `SoftReference` in Java

```
Counter prime =newCounter(); // prime holds a strong reference - line 2
SoftReference<Counter> soft =newSoftReference<Counter>(prime) ; //soft reference variable has SoftReference
to Counter Object created at line 2

prime=null; // now Counter object is eligible for garbage collection but only be collected when JVM absolutely
needs memory
```

After making strong reference `null`, `Counter` object created on line 2 only has one soft reference which can not prevent it from being garbage collected but it can delay collection, which is eager in case of `WeakReference`. Due to this major *difference between SoftReference and WeakReference*, SoftReference are more suitable for **caches** and WeakReference are more suitable for storing**meta data**. One convenient example of `WeakReference` is `WeakHashMap`, which is another implementation of Map interface like HashMap or TreeMap but with one unique feature. `WeakHashMap` wraps keys as `WeakReference` which means once strong reference to actual object removed, `WeakReference` present internally on `WeakHashMap` doesn't prevent them from being Garbage collected.

Phantom reference is third kind of reference type available in `java.lang.ref` package. Phantom reference is represented by `java.lang.ref.PhantomReference` class. Object which only has Phantom reference pointing them can be collected whenever Garbage Collector likes it. Similar to `WeakReference` and `SoftReference` you can create `PhantomReference` by using following code :

```
DigitalCounter digit =newDigitalCounter(); // digit reference variable has strong reference - line 3
PhantomReference<DigitalCounter> phantom =newPhantomReference<DigitalCounter>(digit); // phantom
reference to object created at line 3

digit=null;
```

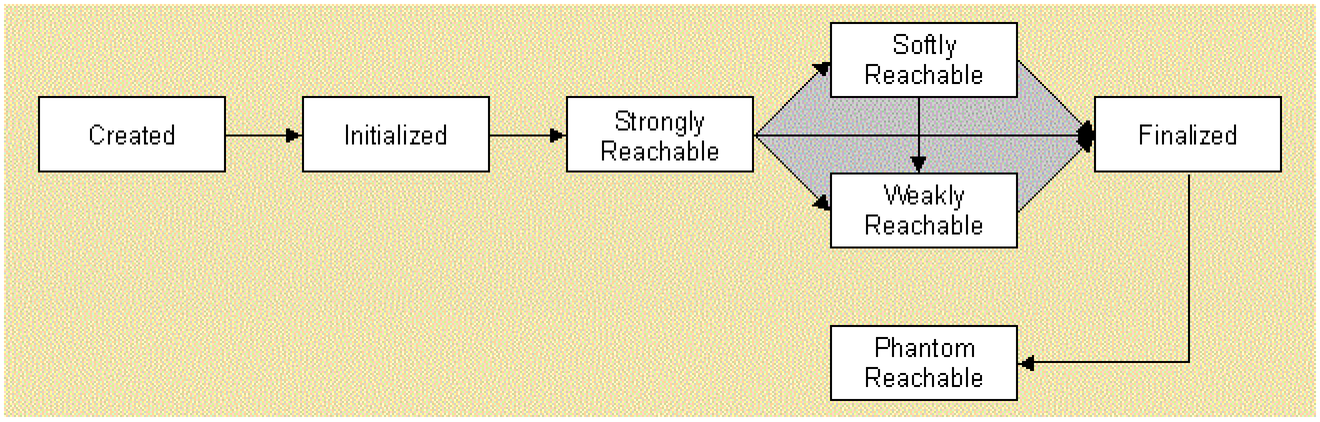
As soon as you remove Strong reference, `DigitalCounter` object created at line 3 can be garbage collected at any time as it only has one more `PhantomReference` pointing towards it, which can not prevent it from GC'd.

Apart from knowing about `WeakReference`, `SoftReference`, `PhantomReference` and `WeakHashMap` there is one more class called `ReferenceQueue` which is worth knowing. You can supply a `ReferenceQueue` instance while creating any `WeakReference`, `SoftReference` or `PhantomReference` as shown in following code :

```
ReferenceQueue refQueue =newReferenceQueue(); //reference will be stored in this queue for cleanup

DigitalCounter digit =newDigitalCounter();
PhantomReference<DigitalCounter> phantom =newPhantomReference<DigitalCounter>(digit, refQueue);
```

Reference of instance will be appended to `ReferenceQueue` and you can use it to perform any clean-up by polling `ReferenceQueue`. An Object's life-cycle is nicely summed up by this diagram.



That's all on **Difference between WeakReference and SoftReference in Java**. We also learned basics of reference classes e.g. Weak, soft and phantom reference in Java and `WeakHashMap` and `ReferenceQueue`. Careful use of reference can assist Garbage Collection and result in better memory management in Java.

Read more: http://javarevisited.blogspot.com/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html#ixzz4Q2dqCLyX

Stack vs Heap in Java

As I told, both Stack and Heap space are part of JVM but they are used for different purpose, let's see some more points to understand the difference between stack and heap memory better.

1) Size
One of the significant difference between Stack and heap comes from their size. Heap space in Java is much bigger than the Stack memory. This is partly due to the fact that whenever a new thread is created in JVM, a separate stack memory is allocated to them.

2) Resizing
JVM allows you to resize both heap and stack in Java, though you need to use different JVM flags for that. You can use -Xms and -Xmx to specify the starting and maximum heap memory in Java. Similarly, you can use the -Xss to specify stack size of individual threads in JVM. See Java Performance Companion by Charlie Hunt to learn more about JVM tuning in Java.

3) Usage
Another significant difference between heap and stack memory comes from their usage perspective. Heap memory is used to store objects in Java. No matter where you create object e.g. inside a method, a class or a code block, they are always created in heap space and memory is allocated from the heap.

One little exception of that is String literals which live in String pool, which was not part of heap until Java 7. Earlier String pool was created on meta space, which was separate memory are in JVM used to store class metadata, but from JDK 7 onwards String pool is merged into heap space.

On the other hand, Stack memory is used to store local variables e.g. primitive `int` and `boolean` variables, method frames and call stack.

4) Visibility
One more difference between heap and stack memory comes from visibility and sharing perspective. Heap memory is shared by all threads hence it is also known as the main memory but stack memory is local to threads and local variable created there was not visible to others. Threads can also cache values into Stack memory. See Java Performance by Binu John to learn more about heap and stack memory in Java.

5)Order Heap is a large memory area where objects can be created and stored in any order but Stack memory is structured as Stack data structure i.e. LIFO where method calls are stored as last in first out order. This is why you can use recursion in Java.

6)Error You get different errors when heap or stack memory gets filled. For example, a faulty recursive algorithm can quickly make Stack memory filled up with recursive method calls in that case you will see `java.lang.StackOverFlowError`, but when there is no more space left in heap to allocate a new object than you will see the `OutOfMemoryError` in java e.g. java.lang.OutOfMemoryError: Java Heap Space. This is another useful difference to know between Stack and Heap in Java from debugging and troubleshooting perspective.

That's all about the **difference between Stack and Heap memory in Java application**. It's extremely important for any Java developer, fresher or experienced to know about these fundamentals. If you don't know about the heap, it would be very difficult to survive or clear any Java programming interviews. Remember, stack memory is used to store local variables and methods calls while heap memory is used to store objects, also heap memory is much larger than stack memory but access to Stack is faster than the heap. See Java Performance The Definitive Guide By Scott Oaks to learn more about heap and stack memory in Java.

Read more: http://www.java67.com/2016/10/difference-between-heap-and-stack-memory-in-java-JVM.html#ixzz4QgU9F8uD

Accessing private fields in Java using reflection

In order to access `private` field using reflection, you need to know the name of field than by calling `getDeclaredFields(String name)` you will get a `java.lang.reflect.Field` instance representing that field. remember using **`getDclaredFields()` method and not `getFields()`** method which returns all non `private` fields both from sub class and super class. while `getDeclaredFields()` returns both `private` and non`private` fields declared in the class. Once you get the field reference you need to make it accessible by calling `Field.setAccessible(true)` because you are going to access `private` field. Now you can get value or private field by calling Field.get(String field_name).if you don't call setAccessible(true) and try to access private field using reflection you will get Exception as shown in below example

```
java.lang.IllegalAccessException: Class test.ReflectionTest can not access a member
of class test.Person with modifiers "private"

        at sun.reflect.Reflection.ensureMemberAccess(Reflection.java:65)
        at java.lang.reflect.Field.doSecurityCheck(Field.java:960)
        at java.lang.reflect.Field.getFieldAccessor(Field.java:896)
        at java.lang.reflect.Field.get(Field.java:358)
        at test.ReflectionTest.main(ReflectionTest.java:31)
```

Calling private methods in Java using reflection

In our last Java tutorial on Reflection we have seen how to call a method by its String name and we will use that information here for invoking private method. Calling private method using reflection is similar to accessing private fields reflectively. Use getDeclaredMethods(String name, Class.. parameter) to get declared private method. pass all the argument type needed by method or nothing if method doesn't accept any argument. This will give you instance of java.lang.reflect.Method which can than be used to call private method using reflection, as shown in code example.

Code example of accessing private field and method using reflection

```
package test;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ReflectionTest {

    public static void main(String args[]) throws ClassNotFoundException {

        Class<Person> person = (Class<Person>) Class.forName("test.Person");

        //getFields() does not return private field
        System.out.println("Fields : " + Arrays.toString(person.getFields()));

        //getDeclaredFields() return both private and non private fields using
reflection
        System.out.println("Declared Fields :
" + Arrays.toString(person.getDeclaredFields()));

        //getDeclaredMethods() return both private and non private methods using
reflection
        System.out.println("Declared methods :
" + Arrays.toString(person.getDeclaredMethods()));

        try {
            //accessing value of private field using reflection in Java
            Person privateRyan = new Person("John" , "8989736353");
            Field privateField = person.getDeclaredField("phone");

            //this call allows private fields to be accessed via reflection
            privateField.setAccessible(true);

            //getting value of private field using reflection
            String value = (String) privateField.get(privateRyan);

            //print value of private field using reflection
            System.out.println("private field: " + privateField + " value: " + value);


            //accessing private method using reflection
            Method privateMethod = person.getDeclaredMethod("call");

            //making private method accessible using reflection
            privateMethod.setAccessible(true);

            //calling private method using reflection in java
            privateMethod.invoke(privateRyan);


        } catch (InvocationTargetException ex) {
            Logger.getLogger(ReflectionTest.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (NoSuchMethodException ex) {
            Logger.getLogger(ReflectionTest.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (IllegalArgumentException ex) {
            Logger.getLogger(ReflectionTest.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (IllegalAccessException ex) {
            Logger.getLogger(ReflectionTest.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (NoSuchFieldException ex) {
            Logger.getLogger(ReflectionTest.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (SecurityException ex) {
            Logger.getLogger(ReflectionTest.class.getName()).log(Level.SEVERE, null,
ex);
        }
      }
}

class Person{
    public String name;
    private String phone;

    public Person(String name, String phone){
        this.name = name;
        this.phone = phone;
    }

    private void call(){
        System.out.println("Calling " + this.name +" at " + this.phone);
    }

    public String getName(){
        return name;
    }
```

```
}

Output:
Fields : [public java.lang.String test.Person.name]
Declared Fields
: [public java.lang.String test.Person.name, private java.lang.String test.Person.phon
e]
Declared methods
: [public java.lang.String test.Person.getName(), private void test.Person.call()]
private field: private java.lang.String test.Person.phone value: 8989736353
Calling John at 8989736353
```

Read more: http://javarevisited.blogspot.com/2012/05/how-to-access-private-field-and-method.html#ixzz4NVSPiRf6

Why should you make your members private by default:

As I said earlier keeping member variable or method `private` as default provides Encapsulation. if you need it access from other class keep increasing accessibility step by step i.e. from `private` to default which is package-private i.e only accessible inside same package to protected and finally `public`. Don't make any method by default `public` which most of Java programmers do, that's against concept of Encapsulation in Java. Here are some **benefits of making variables or methods `private` in Java**:

1) `private` methods are well encapsulated in class and developer knows that they are not used anywhere else in code which gives them confident to change, modify or enhance `private` method without any side-effect.

2) By looking to the `private` method you know by sure that no one is using it which is great help during debugging java program.

3) IDE like Netbeans and Eclipse use this information and automatically check whether `private` methods are used inside Class or not and can display relevant warning to improve code quality or delete unused statements, variables or methods.

4) private methods use static binding in Java and they are bonded during compile time which is fast compared to dynamic binding which occurs during runtime and also gives chance to JVM to either inline the method or optimize it.

5) Making variables private in java and providing getter and setter for them makes your class compatible Java bean naming convention and other reflection based frameworks like displaytag .

Can we override Private method in Java

No you cannot override private methods in Java, private methods are non virtual and accessed differently than non-private methods. Since method overriding can only be done on Sub Class and since private method is not accessible in sub class or derived class, You just can not override them. Another possibility of **overriding private methods in inner class** by extending outer class, since private methods are accessible inside inner class. This will also not work because private methods are bonded during compile time and only Type (or Class) which is used to locate a private method. For example in below code where it looks like Inner SubClass is overriding private method , if you call privateMethod() with a type of Super Class but object of Sub Class, it will only execute privateMethod() declared in super Class.

Should I always make private method as final in Java

Using final and private in Java with method is a good choice but I don't think it offer any benefit in terms of performace or not. Rather this decision should be taken on the basis of design, functionality and ensuring readability of code. Since making a method final just limit there ability to be overridden, it doesn't make much sense to mark a private method as final in java, because ***private method can not overridden in Java*** by any means.

Important Points about private keyword in Java

Few important points to remember about private keyword in java before seeing example of private keyword and **overriding private methods** in Java which is not possible.

1.private keyword can be applied to fields, methods and inner class in Java.

2.Top level classes can not be private in Java.

3.Though private methods or variables are not accessible outside of Class, they can be accessed via reflection by using setAccessible(true) and changing there private visibility.

4.Private method can not be overridden in Java, not even inside inner classes.

5.private members allows Compiler and JVM to optimize it for better performance.

Example of private field and method in Java

Here is **a code example or private keyword in Java**. This examples shows that private methods uses static binding at compile time and can not be overridden. Don’t confuse with the output though regarding private variable because its accessible inside Inner classes. It will be compiler error if private variables used outside the class.

```
public class PrivateMemberExample {

    private String i_m_private = "I am private member, not accessible outside this
Class";

    private void privateMethod() {
        System.out.println("Private method of Outer Class");
    }

    public static void main(String args[]) {
        PrivateMemberExample outerClass = new PrivateMemberExample();
        NestedClass nc = outerClass.new NestedClass();
        nc.showPrivate(); //shows that private method are accessible in inner class.

        outerClass = nc;
        nc.privateMethod(); //does not call private method from inner class because
                            // you can not override private method inside inner class.

    }

    class NestedClass extends PrivateMemberExample {

        public void showPrivate() {
            System.out.println("Accessing Private members of Outer class: " +
i_m_private);
            privateMethod();
        }

        private void privateMethod() {
              System.out.println("Private method of Nested Class");
        }
    }
}


Output:
Accessing Private members of Outer class: I am private member, not accessible
outside this Class
Private method of Outer Class
Private method of Outer Class
```

In Summary ***`private`*** ***keyword in java*** allows most restrictive access to variables and methods and offer strongest form of Encapsulation. `private` members are not accessible outside the class and **private method can not be overridden**.

Read more: http://javarevisited.blogspot.com/2012/03/private-in-java-why-should-you-always.html#ixzz4NVXTejYA

Multithreading, Concurrency and Thread basics Questions

1) Can we make array volatile in Java?
This is one of the tricky Java multi-threading questions you will see in senior Java developer Interview. Yes, you can make an array volatile in Java but only the reference which is pointing to an array, not the whole array. What I mean, if one thread changes the reference variable to points to another array, that will provide a volatile guarantee, but if multiple threads are changing individual array elements they won't be having happens before guarantee provided by the volatile modifier.

2) Can volatile make a non-atomic operation to atomic?
This another good question I love to ask on volatile, mostly as a follow-up of the previous question. This question is also not easy to answer because volatile is not about atomicity, but there are cases where you can use a volatile variable to make the operation atomic.

One example I have seen is having a long field in your class. If you know that a long field is accessed by more than one thread e.g. a counter, a price filed or anything, you better make it volatile. Why? because reading to a long variable is not atomic in Java and done in two steps, If one thread is writing or updating long value, it's possible for another thread to see half value (fist 32-bit). While reading/writing a volatile long or double (64 bit) is atomic.

3) What are practical uses of volatile modifier?
One of the practical use of the volatile variable is to make reading double and long atomic. Both double and long are 64-bit wide and they are read in two parts, first 32-bit first time and next 32-bit second time, which is non-atomic but volatile double and long read is atomic in Java. Another use of the volatile variable is to provide a memory barrier, just like it is used in Disrupter framework. Basically, Java Memory model inserts a write barrier after you write to a volatile variable and a read barrier before you read it. Which means, if you write to volatile field then its guaranteed that any thread accessing that variable will see the value you wrote and anything you did before doing that write into the thread is guaranteed to have happened and any updated data values will also be visible to all threads, because the memory barrier flushed all other writes to the cache.

4) What guarantee volatile variable provides? (answer)
volatile variables provide the guarantee about ordering and visibility e.g. volatile assignment cannot be re-ordered with other statements but in the absence of any synchronization instruction compiler, JVM or JIT are free to reorder statements for better performance. volatile also provides the happens-before guarantee which ensure changes made in one thread is visible to others. In some cases volatile also provide atomicity e.g. reading 64-bit data types like long and double are not atomic but read of volatile double or long is atomic.

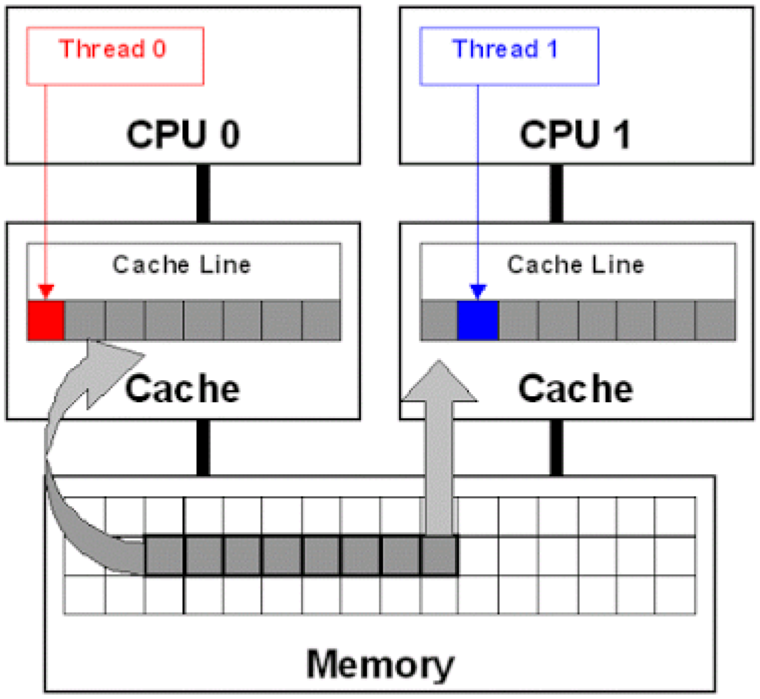
5) Which one would be easy to write? synchronization code for 10 threads or 2 threads?
In terms of writing code, both will be of same complexity because synchronization code

is independent of a number of threads. Choice of synchronization though depends upon a number of threads because the number of thread present more contention, so you go for advanced synchronization technique e.g. lock stripping, which requires more complex code and expertise.

6) How do you call wait() method? using if block or loop? Why? (answer)
`wait()` method should always be called in loop because it's possible that until thread gets CPU to start running again the condition might not hold, so its always better to check condition in loop before proceeding. Here is the standard idiom of using wait and notify method in Java:

```
// The standard idiom for using the wait method
synchronized (obj) {
while (condition does not hold)
obj.wait(); // (Releases lock, and reacquires on wakeup)
      ... // Perform action appropriate to condition
}
```
See Effective Java Item 69 to learn more about why wait method should call in the loop.

7) What is false sharing in the context of multi-threading?
false sharing is one of the well-known performance issues on multi-core systems, where each process has its local cache. false sharing occurs when threads on different processor modify variables that reside on same cache line as shown in the following image:



False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory. Like many concurrency issues, the primary way to avoid false sharing is careful code review and aligning your data structure with the size of a cache line.

8) What is busy spin? Why should you use it?
Busy spin is one of the technique to wait for events without releasing CPU. It's often done to avoid losing data in CPU cached which is lost if the thread is paused and resumed in some other core. So, if you are working on low latency system where your order processing thread currently doesn't have any order, instead of sleeping or calling `wait()`, you can just loop and then again check the queue for new messages. It's only beneficial if you need to wait for a very small amount of time e.g. in micro seconds or nano seconds. LMAX Disrupter framework, a high-performance inter-thread messaging library has a `BusySpinWaitStrategy` which is based on this concept and uses a busy spin loop for `EventProcessors` waiting on the barrier.

9) How do you take thread dump in Java?
You can take a thread dump of Java application in Linux by using **`kill -3 PID`**, where PID is the process id of Java process. In Windows, you can press **`Ctrl + Break`**. This will instruct JVM to print thread dump in standard out or err and it could go to console or log file depending upon your application configuration. If you have used Tomcat then when

10) is Swing thread-safe? (answer)
No, Swing is not thread-safe. You cannot update Swing components e.g. JTable, JList or JPanel from any thread, in fact, they must be updated from GUI or AWT thread. That's why swings provide `invokeAndWait()` and `invokeLater()` method to request GUI update from any other threads. This methods put update request in AWT threads queue and can wait till update or return immediately for an asynchronous update. You can also check the detailed answer to learn more.

11) What is a thread local variable in Java? (answer)
Thread-local variables are variables confined to a thread, its like thread's own copy which is not shared between multiple threads. Java provides a `ThreadLocal` class to support thread-local variables. It's one of the many ways to achieve thread-safety. Though be careful while using thread local variable in manged environment e.g. with web servers where worker thread out lives any application variable. Any thread local variable which is not removed once its work is done can potentially cause a memory leak in Java application.

12) Write wait-notify code for producer-consumer problem? (answer)
Please see the answer for a code example. Just remember to call wait() and notify() method from synchronized block and test waiting for condition on the loop instead of if block.

13) Write code for thread-safe Singleton in Java? (answer)
Please see the answer for a code example and step by step guide to creating thread-safe singleton class in Java. When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads. Using Java enum as Singleton class is one of the easiest ways to create a thread-safe singleton in Java.

14) The difference between sleep and wait in Java? (answer)
Though both are used to pause currently running thread, sleep() is actually meant for short pause because it doesn't release lock, while `wait()` is meant for conditional wait and that's why it release lock which can then be acquired by another thread to change the condition on which it is waiting.

15) What is an immutable object? How do you create an Immutable object in Java? (answer)
Immutable objects are those whose state cannot be changed once created. Any modification will result in a new object e.g. `String`, `Integer`, and other wrapper class. Please see the answer for step by step guide to creating Immutable class in Java.

16) Can we create an Immutable object, which contains a mutable object?
Yes, its possible to create an Immutable object which may contain a mutable object, you just need to be a little bit careful not to share the reference of the mutable component, instead, you should return a copy of it if you have to. Most common example is an Object which contain the reference of `java.util.Date` object.

Date types and Basic Java Interview Questions

17) What is the right data type to represent a price in Java? (answer)
BigDecimal if memory is not a concern and Performance is not critical, otherwise double with predefined precision.

18) How do you convert bytes to String? (answer)
you can convert bytes to the string using string constructor which accepts `byte[]`, just make sure that right character encoding otherwise platform's default character encoding will be used which may or may not be same.

19) How do you convert bytes to long in Java? (answer)
This questions if for you to answer :-)

20) Can we cast an int value into byte variable? what will happen if the value of int is larger than byte?
Yes, we can cast but int is 32 bit long in java while byte is 8 bit long in java so when you cast an int to byte higher 24 bits are lost and a byte can only hold a value from -128 to 128.

21) There are two classes B extends A and C extends B, Can we cast B into C e.g. C = (C) B; (answer)

22) Which class contains clone method? Cloneable or Object? (answer)
java.lang.Cloneable is marker interface and doesn't contain any method clone method is defined in the object class. It is also knowing that clone() is a native method means it's implemented in C or C++ or any other native language.

23) Is ++ operator is thread-safe in Java? (answer)
No it's not a thread safe operator because its involve multiple instructions like reading a value, incriminating it and storing it back into memory which can be overlapped between multiple threads.

24) Difference between a = a + b and a += b ? (answer)
The += operator implicitly cast the result of addition into the type of variable used to hold

the result. When you add two integral variable e.g. variable of type byte, short, or int then they are first promoted to int and them addition happens. If result of addition is more than maximum value of a then a + b will give compile time error but a += b will be ok as shown below

```
byte a = 127;
byte b = 127;
b = a + b; //error : cannot convert from int to byte
b += a; // ok
```

25) Can I store a double value in a long variable without casting? (answer)
No, you cannot store a double value into a long variable without casting because the range of double is more that long and you we need to type cast. It's not dificult to answer this question but many develoepr get it wrong due to confusion on which one is bigger between double and long in Java.

26) What will this return 3*0.1 == 0.3? true or false? (answer)
This is one of the really tricky questions. Out of 100, only 5 developers answered this question and only of them have explained the concept correctly. The short answer is false because some floating point numbers can not be represented exactly.

27) Which one will take more memory, an int or Integer? (answer)
An Integer object will take more memory an Integer is the an object and it store meta data overhead about the object and int is primitive type so its takes less space.

28) Why is String Immutable in Java? (answer)
One of my favorite Java interview question. The String is Immutable in java because java designer thought that string will be heavily used and making it immutable allow some optimization easy sharing same String object between multiple clients. See the link for the more detailed answer. This is a great question for Java programmers with less experience as it gives them food for thought, to think about how things works in Java, what Jave designers might have thought when they created String class etc.

29) Can we use String in the switch case? (answer)
Yes from Java 7 onward we can use String in switch case but it is just syntactic sugar. Internally string hash code is used for the switch. See the detaiedl answer for more explanation and discussion.

30) What is constructor chaining in Java? (answer)

When you call one constructor from other than it's known as constructor chaining in Java. This happens when you have multiple, overloaded constructor in the class.

JVM Internals and Garbage Collection Interview Questions

In the year 2015 I have seen increased focus on JVM internal and Garbage collection tuning, monitoring Java appliation, dealing with Java performance issues on various Java interviews. This is actually become the prime topic for interviewing any experienced Java developer for senior position e.g. technical lead, VP or team lead. If you feel you are short of experience and knowledge in this area then you should read atleast one book mentioned in my list of Java Performance books. I vote goes to Java Performance, The Definitive guide by Scott.

31) What is the size of int in 64-bit JVM?
The size of an int variable is constant in Java, it's always 32-bit irrespective of platform. Which means the size of primitive `int` is same in both 32-bit and 64-bit Java virtual machine.

32) The difference between Serial and Parallel Garbage Collector? (answer)
Even though both the serial and parallel collectors cause a stop-the-world pause during Garbage collection. The main difference between them is that a serial collector is a default copying collector which uses only one GC thread for garbage collection while a parallel collector uses multiple GC threads for garbage collection.

33) What is the size of an int variable in 32-bit and 64-bit JVM? (answer)
The size of int is same in both 32-bit and 64-bit JVM, it's always 32 bits or 4 bytes.

34) A difference between WeakReference and SoftReference in Java? (answer)
Though both WeakReference and SoftReference helps garbage collector and memory efficient, WeakReference becomes eligible for garbage collection as soon as last strong reference is lost but SoftReference even thought it can not prevent GC, it can delay it until JVM absolutely need memory.

35) How do WeakHashMap works? (answer)
WeakHashMap works like a normal HashMap but uses WeakReference for keys, which means if the key object doesn't have any reference then both key/value mapping will become eligible for garbage collection.

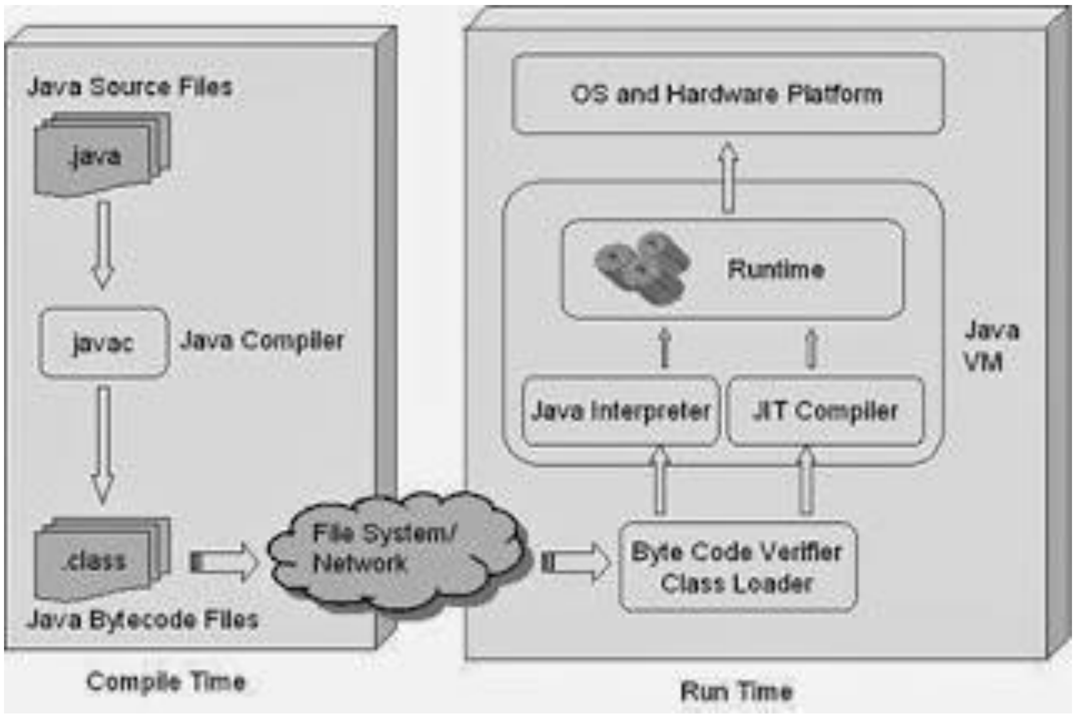
36) What is -XX:+UseCompressedOops JVM option? Why use it? (answer)
When you go migrate your Java application from 32-bit to 64-bit JVM, the heap requirement suddenly increases, almost double, due to increasing size of ordinary object pointer from 32 bit to 64 bit. This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -

XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP.

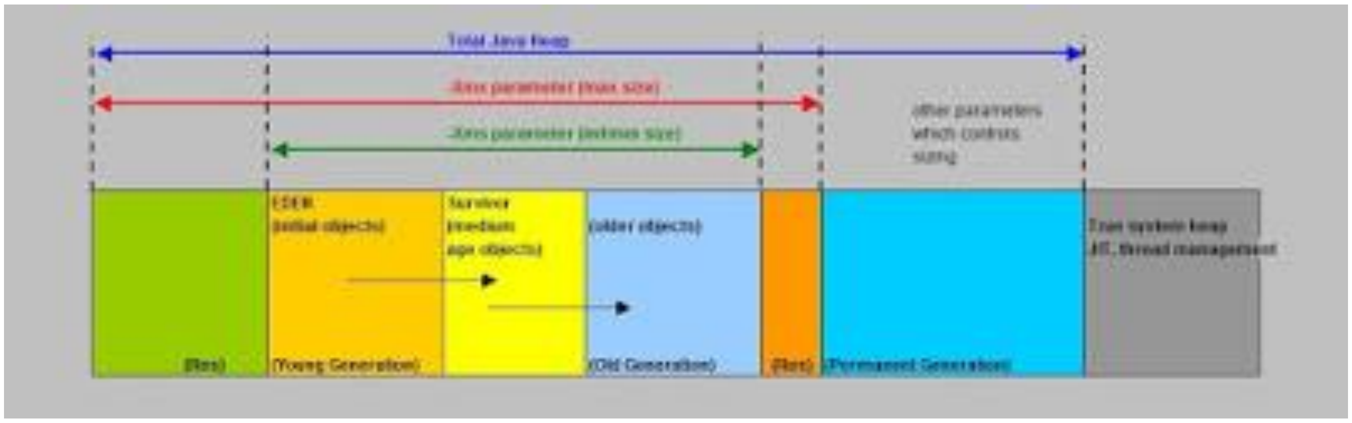
37) How do you find if JVM is 32-bit or 64-bit from Java Program? (answer)
You can find that by checking some system properties like sun.arch.data.model or os.arch

38) What is the maximum heap size of 32-bit and 64-bit JVM? (answer)
Theoretically, the maximum heap memory you can assign to a 32-bit JVM is 2^32 which is 4GB but practically the limit is much smaller. It also varies between operating systems e.g. form 1.5GB in Windows to almost 3GB in Solaris. 64-bit JVM allows you to specify larger heap size, theoretically 2^64 which is quite large but practically you can specify heap space up to 100GBs. There are even JVM e.g. Azul where heap space of 1000 gigs is also possible.

39) What is the difference between JRE, JDK, JVM and JIT? (answer)
JRE stands for Java run-time and it's required to run Java application. JDK stands for Java development kit and provides tools to develop Java program e.g. Java compiler. It also contains JRE. The JVM stands for Java virtual machine and it's the process responsible for running Java application. The JIT stands for Just In Time compilation and helps to boost the performance of Java application by converting Java byte code into native code when the crossed certain threshold i.e. mainly hot code is converted into native code.



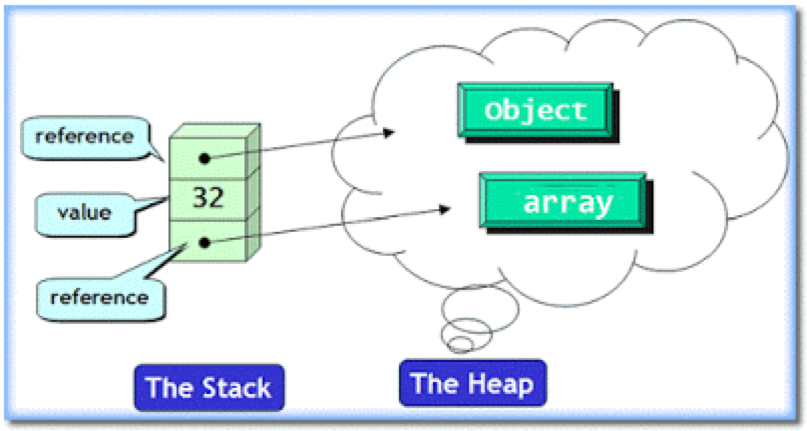
40) Explain Java Heap space and Garbage collection? (answer)
When a Java process is started using java command, memory is allocated to it. Part of this memory is used to create heap space, which is used to allocate memory to objects whenever they are created in the program. Garbage collection is the process inside JVM which reclaims memory from dead objects for future allocation.


Total Java Heap
-Xmx parameter (max size)
-Xms parameter (initial size)
other parameters which controls sizing
EDEN (initial objects)
Survivor (medium age objects)
(older objects)
Free system heap JIT, thread management
(Res)
(Young Generation)
(Old Generation)
(Res)
(Permanent Generation)

41) Can you guarantee the garbage collection process? (answer)
No, you cannot guarantee the garbage collection, though you can make a request using System.gc() or Runtime.gc() method.

42) How do you find memory usage from Java program? How much percent of the heap is used?
You can use memory related methods from java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java. By using these methods, you can find out how many percents of the heap is used and how much heap space is remaining. `Runtime.freeMemory()` return amount of free memory in bytes, `Runtime.totalMemory()` returns total memory in bytes and`Runtime.maxMemory()` returns maximum memory in bytes.

43) What is the difference between stack and heap in Java? (answer)
Stack and heap are different memory areas in the JVM and they are used for different purposes. The stack is used to hold method frames and local variables while objects are always allocated memory from the heap. The stack is usually much smaller than heap memory and also didn't shared between multiple threads, but heap is shared among all threads in JVM.



Basic Java concepts Interview Questions

44) What's the difference between "a == b" and "a.equals(b)"? (answer)
The a = b does object reference matching if both a and b are an object and only return true if both are pointing to the same object in the heap space, on the other hand, a.equals(b) is used for logical mapping and its expected from an object to override this method to provide logical equality. For example, String class overrides this equals() method so that you can compare two Strings, which are the different object but contains same letters.

45) What is a.hashCode() used for? How is it related to a.equals(b)? (answer)
hashCode() method returns an int hash value corresponding to an object. It's used in hash based collection classes e.g Hashtable, HashMap, LinkedHashMap and so on. It's very tightly related to equals() method. According to Java specification, two objects which are equal to each other using equals() method must have same hash code.

47) What is a compile time constant in Java? What is the risk of using it?
public static final variables are also known as a compile time constant, the public is optional there. They are replaced with actual values at compile time because compiler know their value up-front and also knows that it cannot be changed during run-time. One of the problem with this is that if you happened to use a public static final variable from some in-house or third party library and their value changed later than your client will still be using old value even after you deploy a new version of JARs. To avoid that, make sure you compile your

program when you upgrade dependency JAR files.

Java Collections Framework Interview Questions

It also contains Data structure and algorithm Interview question in Java, questions on array, linked list, HashMap, ArrayList, Hashtable, Stack, Queue, PriorityQueue, LinkedHashMap and ConcurrentHashMap.

48) The difference between List, Set, Map, and Queue in Java? (answer)
The list is an ordered collection which allows duplicate. It also has an implementation which provides constant time index based access, but that is not guaranteed by List interface. Set is unordered collection which

49) Difference between poll() and remove() method?
Both poll() and remove() take out the object from the Queue but if poll() fails then it returns null but if remove fails it throws Exception.

50) The difference between LinkedHashMap and PriorityQueue in Java? (answer)
PriorityQueue guarantees that lowest or highest priority element always remain at the head of the queue, but LinkedHashMap maintains the order on which elements are inserted. When you iterate over a PriorityQueue, iterator doesn't guarantee any order but iterator of LinkedHashMap does guarantee the order on which elements are inserted.

51) Difference between ArrayList and LinkedList in Java? (answer)
The obvious difference between them is that ArrrayList is backed by array data structure, supprots random access and LinkedList is backed by linked list data structure and doesn't supprot random access. Accessing an element with the index is O(1) in ArrayList but its O(n) in LinkedList. See the answer for more detailed discussion.

52) What is a couple of ways that you could sort a collection? (answer)
You can either use the Sorted collection like TreeSet or TreeMap or you can sort using the ordered collection like a list and using Collections.sort() method.

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

54) LinkedList in Java is doubly or singly linked list? (answer)
It's a doubly linked list, you can check the code in JDK. In Eclipse, you can use the shortcut, Ctrl + T to directly open this class in Editor.

55) Which kind of tree is used to implement TreeMap in Java? (answer)
A Red Black tree is used to implement TreeMap in Java.

56) What is the difference between Hashtable and HashMap? (answer)
There are many differences between these two classes, some of them are following:
a) Hashtable is a legacy class and present from JDK 1, HashMap was added later.
b) Hashtable is synchronized and slower but HashMap is not synchronized and faster.
c) Hashtable doesn't allow null keys but HashMap allows one null key.
See the answer for more differences between HashMap and Hashtable in Java.

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

58) Write code to remove elements from ArrayList while iterating? (answer)
Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Here is the sample code which uses right way o remove elements from ArrayList while looping over and avoids ConcurrentModificationException.

59) Can I write my own container class and use it in the for-each loop?
Yes, you can write your own container class. You need to implement the Iterable interface if you want to loop over advanced for loop in Java, though. If you implement Collection then you by default get that property.

60) What is default size of ArrayList and HashMap in Java? (answer)
As of Java 7 now, default size of ArrayList is 10 and default capacity of HashMap is 16, it must be power of 2. Here is code snippet from ArrayList and HashMap class :

```
// from ArrayList.java JDK 1.7
privatestaticfinalint DEFAULT_CAPACITY = 10;

//from HashMap.java JDK 7
staticfinalint DEFAULT_INITIAL_CAPACITY = 1<<4; // aka 16
```

61) Is it possible for two unequal objects to have the same hashcode?
Yes, two unequal objects can have same hashcode that's why collision happen in a hashmap. the equal hashcode contract only says that two equal objects must have the same hashcode it doesn't say anything about the unequal object.

62) Can two equal object have the different hash code?
No, thats not possible according to hash code contract.

63) Can we use random numbers in the hashcode() method? (answer)
No, because hashcode of an object should be always same. See the answer to learning more about things to remember while overriding hashCode() method in Java.

64) What is the difference between Comparator and Comparable in Java? (answer)
The Comparable interface is used to define the natural order of object while Comparator is used to define custom order. Comparable can be always one, but we can have multiple comparators to define customized order for objects.

65) Why you need to override hashcode, when you override equals in Java? (answer)
Because equals have code contract mandates to override equals and hashcode together .since many container class like HashMap or HashSet depends on hashcode and equals contract.

Java IO and NIO Interview questions

IO is very important from Java interview point of view. You should have a good knowledge of old Java IO, NIO, and NIO2 along with some operating system and disk IO fundamentals. Here are some frequently asked questions form Java IO.

66) In my Java program, I have three sockets? How many threads I will need to handle that?

67) How do you create ByteBuffer in Java?

This Java tip demonstrates a method of creating a ByteBuffer. Fixed capacity bufferes that hold byte values are known as ByteBuffer. There may be various ways of creating a ByteBuffer.

?

```
// Create a ByteBuffer using a byte array
  byte[] bytes = new byte[10];
  ByteBuffer buffer = ByteBuffer.wrap(bytes);

  // Create a non-direct ByteBuffer with a 10 byte capacity
  // The underlying storage is a byte array.
  buffer = ByteBuffer.allocate(10);

  // Create a memory-mapped ByteBuffer with a 10 byte
capacity.
  buffer = ByteBuffer.allocateDirect(10);
```

69) Is Java BIG endian or LITTLE endian?

You must have heard these terms Little-Endian and Big-Endian many times in your engineering course. Let’s quickly recap the concept behind these words.

These two terms are related to the direction of bytes in a word within CPU architecture. Computer memory is referenced by addresses that are positive integers. It is “natural” to store numbers with the least significant byte coming before the most significant byte in the computer memory. Sometimes computer designers prefer to use a reversed order version of this representation.

The “natural” order, where less significant byte comes before more significant byte in memory, is called little-endian. Many vendors like IBM, CRAY, and Sun preferred the reverse order that, of course, is called big-endian.

For example, the 32-bit hex value 0x45679812 would be stored in memory as follows:

```
Address       00  01  02  03
-------------------------------
Little-endian   12  98  67  45
Big-endian      45  67  98  12
```

Difference in endian-ness can be a problem when transferring data between two machines.

Everything in Java binary format files is stored in big-endian order. This is sometimes called network order. This means that if you use only Java, all files are done the same way on all platforms: Mac, PC, UNIX, etc. You can freely exchange binary data electronically without any concerns about endian-ness.

The problem comes when you must exchange data files with some program not written in Java that uses little-endian order, most commonly a program written in C. Some platforms use big-endian order internally (Mac, IBM 390); some uses little-endian order (Intel).

70) What is the byte order of ByteBuffer?

71) The difference between direct buffer and non-direct buffer in Java? (answer)

72) What is the memory mapped buffer in Java? (answer)

73) What is TCP NO DELAY socket option?

74) What is the difference between TCP and UDP protocol? (answer)

75) The difference between ByteBuffer and StringBuffer in Java? (answer)

Java Best Practices Interview question

Contains best practices from different parts of Java programming e.g. Collections, String, IO, Multi-threading, Error and Exception handling, design patterns etc. This section is mostly for experience Java developer, technical lead, AVP, team lead and coders who are responsible for products. If you want to create quality products you must know and follow the best practices.

76) What best practices you follow while writing multi-threaded code in Java? (answer)
Here are couple of best practices which I follow while writing concurrent code in Java:
a) Always name your thread, this will help in debugging.
b) minimize the scope of synchronization, instead of making whole method synchronized, only critical section should be synchronized.
c) prefer volatile over synchronized if you can can.
e) use higher level concurrency utilities instead of waitn() and notify for inter thread communication e.g. BlockingQueue, CountDownLatch and Semeaphore.
e) Prefer concurrent collection over synchronized collection in Java. They provide better scalability.

77) Tell me few best practices you apply while using Collections in Java? (answer)
Here are couple of best practices I follow while using Collectionc classes from Java:
a) Always use the right collection e.g. if you need non-synchronized list then use ArrayList and

not Vector.
b) Prefer concurrent collection over synchronized collection because they are more scalable.
c) Always use interface to a represent and access a collection e.g. use List to store ArrayList, Map to store HashMap and so on.
d) Use iterator to loop over collection.
e) Always use generics with collection.

78) Can you tell us at least 5 best practice you use while using threads in Java? (answer)
This is similar to the previous question and you can use the answer given there. Particularly with thread, you should:
a) name your thread
b) keep your task and thread separate, use Runnable or Callable with thread pool executor.
c) use thread pool
d) use volatile to indicate compiler about ordering, visibility, and atomicity.
e) avoid thread local variable because incorrect use of ThreadLocal class in Java can create a memory leak.
Look there are many best practices and I give extra points to the developer which bring something new, something even I don't know. I make sure to ask this question to Java developers of 8 to 10 years of experience just to gauge his hands on experience and knowledge.

79) Name 5 IO best practices? (answer)
IO is very important for performance of your Java application. Ideally you should avoid IO in critical path of your application. Here are couple of Java IO best practices you can follow:

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

b) Use classes from NIO and NIO2

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

d) use memory mapped file for faster IO.

If a Java candidate doesn't know about IO and NIO, especially if he has at least 2 to 4 years of experience, he needs some reading.

80) Name 5 JDBC best practices your follow? (answer)
Another good Java best practices for experienced Java developer of 7 to 8 years experience. Why it's important? because they are the ones which set the trend in the code and educate junior developers. There are many best practices and you can name as per your comfort and convenience. Here are some of the more common ones:
a) use batch statement for inserting and updating data.
b) use PreparedStatement to avoid SQL exception and better performance.

c) use database connection pool
d) access resultset using column name instead of column indexes.
e) Don't generate dynamic SQL by concatenating String with user input.

81) Name couple of method overloading best practices in Java? (answer)
Here are some best practices you can follow while overloading a method in Java to avoid confusion with auto-boxing:
a) Don't overload method where one accepts int and other accepts Integer.
b) Don't overload method where number of argument is same and only order of argument is different.
c) Use varargs after overloaded methods has more than 5 arguments.

Date, Time and Calendar Interview questions in Java

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

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

84) How do you show timezone in formatted date in Java? (answer)

85) The difference between java.util.Date and java.sql.Date in Java? (answer)

86) How to you calculate the difference between two dates in Java? (program)

87) How do you convert a String(YYYYMMDD) to date in Java? (answer)

Unit testing JUnit Interview questions

89) How do you test static method? (answer)
You can use PowerMock library to test static methods in Java.

90) How to do you test a method for an exception using JUnit? (answer)

91) Which unit testing libraries you have used for testing Java programs? (answer)

92) What is the difference between @Before and @BeforeClass annotation? (answer)
he code marked `@Before` is executed before each test, while `@BeforeClass` runs once before the entire test fixture. If your test class has ten tests, `@Before` code will be executed ten times, but `@BeforeClass` will be executed only once.

In general, you use `@BeforeClass` when multiple tests need to share the same computationally expensive setup code. Establishing a database connection falls into this category. You can move code from `@BeforeClass` into `@Before`, but your test run may take longer. Note that the code marked `@BeforeClass` is run as static initializer, therefore it will run before the class instance of your test fixture is created.

In JUnit 5, the tags `@BeforeEach` and `@BeforeAll` are the equivalents of `@Before` and `@BeforeClass` in JUnit 4. Their names are a bit more indicative of when they run, loosely interpreted: 'before each tests' and 'once before all tests'.

Programming and Coding Questions

93) How to check if a String contains only numeric digits? (solution)

94) How to write LRU cache in Java using Generics? (answer)

95) Write a Java program to convert bytes to long? (answer)

96) How to reverse a String in Java without using StringBuffer? (solution)

97) How to find the word with the highest frequency from a file in Java? (solution)

98) How do you check if two given String are anagrams? (solution)

99) How to print all permutation of a String in Java? (solution)

100) How do you print duplicate elements from an array in Java? (solution)

101) How to convert String to int in Java? (solution)

102) How to swap two integers without using temp variable? (solution)

Java Interview questions from OOP and Design Patterns

It contains Java Interview questions from SOLID design principles, OOP fundamentals e.g. class, object, interface, Inheritance, Polymorphism, Encapsulation, and Abstraction as well as more advanced concepts like Composition, Aggregation, and Association. It also contains questions from GOF design patterns.

103) What is the interface? Why you use it if you cannot write anything concrete on it?
The interface is used to define API. It tells about the contract your classes will follow. It also supports abstraction because a client can use interface method to leverage multiple implementations e.g. by using List interface you can take advantage of random access of ArrayList as well as flexible insertion and deletion of LinkedList. The interface doesn't allow you to write code to keep things abstract but from Java 8 you can declare static and default methods inside interface which are concrete.

104) The difference between abstract class and interface in Java? (answer)
There are multiple differences between abstract class and interface in Java, but the most important one is Java's restriction on allowing a class to extend just one class but allows it to implement multiple interfaces. An abstract class is good to define default behavior for a family of class, but the interface is good to define Type which is later used to leverage Polymorphism. Please check the answer for a more thorough discussion of this question.

105) Which design pattern have you used in your production code? apart from Singleton?
This is something you can answer from your experience. You can generally say about dependency injection, factory pattern, decorator pattern or observer pattern, whichever you have used. Though be prepared to answer follow-up question based upon the pattern you choose.

107) What is Law of Demeter violation? Why it matters? (answer)
Believe it or not, Java is all about application programming and structuring code. If you have good knowledge of common coding best practices, patterns and what not to do than only you can write quality code. Law of Demeter suggests you "talk to friends and not stranger", hence used to reduce coupling between classes.

108) What is Adapter pattern? When to use it?
Another frequently asked Java design pattern questions. It provides interface conversion. If your client is using some interface but you have something else, you can write an Adapter to bridge them together. This is good for Java software engineer having 2 to 3 years experience

because the question is neither difficult nor tricky but requires knowledge of OOP design patterns.

109) What is "dependency injection" and "inversion of control"? Why would someone use it? (answer)

110) What is an abstract class? How is it different from an interface? Why would you use it? (answer)
One more classic question from Programming Job interviews, it is as old as chuck Norris. An abstract class is a class which can have state, code and implementation, but an interface is a contract which is totally abstract. Since I have answered it many times, I am only giving you the gist here but you should read the article linked to answer to learn this useful concept in much more detail.

111) Which one is better constructor injection or setter dependency injection? (answer)
Each has their own advantage and disadvantage. Constructor injection guaranteed that class will be initialized with all its dependency, but setter injection provides flexibility to set an optional dependency. Setter injection is also more readable if you are using an XML file to describe dependency. Rule of thumb is to use constructor injection for mandatory dependency and use setter injection for optional dependency.

112) What is difference between dependency injection and factory design pattern? (answer)
Though both patterns help to take out object creation part from application logic, use of dependency injection results in cleaner code than factory pattern. By using dependency injection, your classes are nothing but POJO which only knows about dependency but doesn't care how they are acquired. In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern. Please see the answer for a more detailed discussion on this topic.

113) Difference between Adapter and Decorator pattern? (answer)
Though the structure of Adapter and Decorator pattern is similar, the difference comes on the intent of each pattern. The adapter pattern is used to bridge the gap between two interfaces, but Decorator pattern is used to add new functionality into the class without the modifying existing code.

114) Difference between Adapter and Proxy Pattern? (answer)
Similar to the previous question, the difference between Adapter and Proxy patterns is in

their intent. Since both Adapter and Proxy pattern encapsulate the class which actually does the job, hence result in the same structure, but Adapter pattern is used for interface conversion while the Proxy pattern is used to add an extra level of indirection to support distribute, controlled or intelligent access.

115) What is Template method pattern? (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.

116) 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.

117) 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.

118) The difference between Inheritance and Composition? (answer)
Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies.

119) Describe overloading and overriding in Java? (answer)
Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity. Though you can overload a method in the same class, but you can only override a method in child classes. Inheritance is necessary for overriding.

120) The difference between nested public static class and a top level class in Java? (answer)
You can have more than one nested public static class inside one class, but you can only have one top-level public class in a Java source file and its name must be same as the name of Java source file.

121) Difference between Composition, Aggregation and Association in OOP? (answer)
If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation. In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.

122) Give me an example of design pattern which is based upon open closed principle? (answer)
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 if you need a new strategy, just implement the interface and configure, no need to modify core logic. One working example is `Collections.sort()` method which is based on Strategy pattern and follows the open-closed principle, you don't modify `sort()` method to sort a new object, what you do is just implement Comparator in your own way.

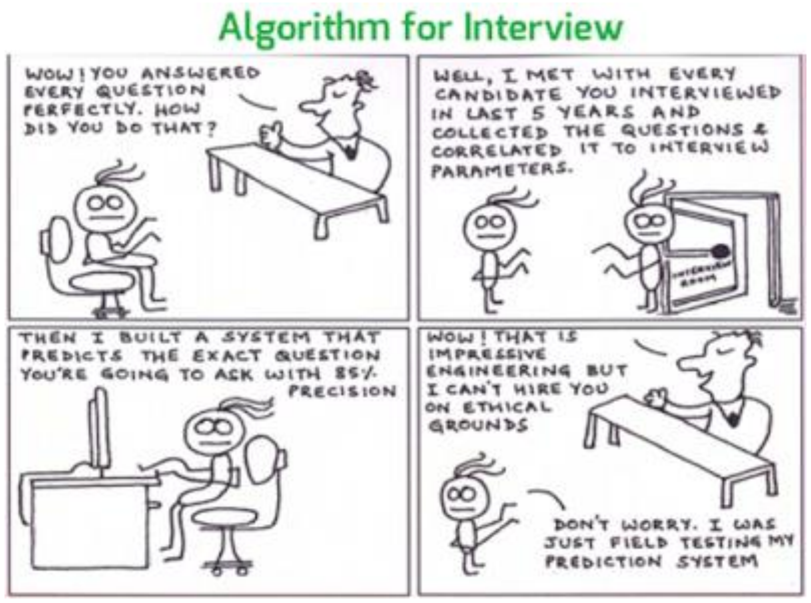
123) Difference between Abstract factory and Prototype design pattern? (answer)
This is the practice question for you, If you are feeling bored just reading and itching to write something, why not write the answer to this question. I would love to see an example the, which should answer where you should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

124) 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.

Read more: http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#ixzz4NVsWX1uw

Top 50 Programming Questions for Telephonic Interviews

Here is a list of almost 50 questions from phone round of programming job interviews. These questions are good for any programmers, developers, software engineer, QA and support engineer because they are based upon fundamentals of programming, but most suited for programmers and developers. BTW, If you are Java developer and looking for Java Questions for Phone interviews, check out that list. This list is more general and applicable for all programmers including Python, Ruby, Perl and C# developers.


Algorithm for Interview
WOW! YOU ANSWERED EVERY QUESTION PERFECTLY. HOW DID YOU DO THAT?
WELL, I MET WITH EVERY CANDIDATE YOU INTERVIEWED IN LAST 5 YEARS AND COLLECTED THE QUESTIONS & CORRELATED IT TO INTERVIEW PARAMETERS.
INTERVIEW ROOM
THEN I BUILT A SYSTEM THAT PREDICTS THE EXACT QUESTION YOU'RE GOING TO ASK WITH 85% PRECISION
WOW! THAT IS IMPRESSIVE ENGINEERING BUT I CAN'T HIRE YOU ON ETHICAL GROUNDS
DON'T WORRY. I WAS JUST FIELD TESTING MY PREDICTION SYSTEM

1. How much time does it take to retrieve an element if stored in HashMap, Binary tree, and a Linked list? how it change if you have millions of records?
In HashMap it takes `O(1)` time, in the binary tree it takes `O(logN)` where N is a number of nodes in the tree and in linked list it takes `O(n)` time where n is a number of element in the list. Millions of records don't affect the performance if the data structure is working as expected e.g. HashMap has no or relatively less number of collision or binary tree is balanced. If that's not the case then their performance degrades as a number of records grows.

3. What is the difference between forking a process and spawning a thread?
When you fork a process, the new process will run the same code as parent process but in different memory space, but when you spawn a new thread in existing process, it just creates another independent path of execution but share same memory space.

4. What is a critical section? (answer)
A critical section is the part of a code, which is very important and in multi-threading must be exclusively modified by any thread. Semaphore or mutex is used to protect critical section. In Java, you can use synchronized keyword or `ReentrantLock` to protect a critical section.

5. What is the difference between a value type and a reference type? (answer)
A value type is a more optimized type and always immutable e.g. primitive int, long, double and float in Java while a reference type points to an object, which can be mutable or

Immutable. You can also say that value type points to a value while reference type points to an object.

6. What is heap and stack in a process? (detailed answer)
They are two separate areas of memory in the same process. Talking about Java, the stack is used to store primitive values and reference type to object but actual object is always created in heap. One critical difference between heap and stack is that heap memory is shared by all threads but each thread has their own stack.

7. What is revision/version control? (answer)
Version control is software which is used to store code and manage versions of codebase e.g. SVN, CVS, Git, Perforce, and ClearCase. They are very effective while comparing code, reviewing code and creating a build from previous stable version. All professional development use some sort of revision or version control tool, without them, you cannot manage code effectively, especially if 20 developers are working on same code base at the same time. Version control tool plays very important role to keep code base consistent and resolving code conflicts.

8. What is a strongly typed programming language? (answer)
In a strongly typed language compiler ensure type correctness, for example, you can not store the number in String or vice-versa. Java is a strongly typed language, that's why you have different data types e.g. `int`, `float`, `String`, `char`, `boolean` etc. You can only store compatible values in respective types. On the other hand, weakly typed language don't enforce type checking at compile time and they tree values based upon context. Python and Perl are two popular example of weakly typed programming language, where you can store a numeric string in number type.

9. Can you describe the difference between valid and well-formed XML?
A well-formed XML is the one which has root element and all tags are closed properly, attributes are defined properly, their value is also quoted properly. On another hand, a valid XML is the one which can be validated against an XSD file or schema. So it's possible for an XML to be well-formed but not valid because they contain tags which may not be allowed by their schema.

10. What is the difference between DOM and SAX parser? (detailed answer)
DOM parser is an in-memory parser so it loads whole XML file in memory and create a DOM tree to parse. SAX parser is an event based parser, so it parses XML document based on the event received e.g. opening tag, closing tag, the start of attribute or end of the attribute. Because of their working methodology, DOM parser is not suitable for large XML file as they

will take a lot of space in memory and your process may run out of memory, SAX is the one which should be used to parse large files. For small files, DOM is usually much faster than SAX.

11. What is the relationship between threads and processes? (detailed answer)
A process can have multiple threads but a thread always belongs to a single process. Two processes cannot share memory space until they are purposefully doing inter-process communication via shared memory but two threads from the same process always share the same memory.

12. What is Immutable class mean? (detailed answer)
A class is said to be Immutable if its state cannot be changed once created, for example, String in Java is immutable. Once you create a String say "Java", you cannot change its content. Any modification in this string e.g. converting into upper case, concatenating with another String will result in the new object. An immutable object is very useful for concurrent programming because they can be shared between multiple threads without worrying about synchronization. In fact, the whole model of functional programming is built on top of Immutable objects.

13. Why would you ever want to create a mock object? (answer)
A mock object is very useful to test an individual unit in your Software, in fact, stub and mocks a are a powerful tool for creating automated unit tests. Suppose you write a program to display currency conversion rates but you don't have a URL to connect to, now if you want to test your code, you can use mock objects. In Java world, there are a lot of frameworks which can create powerful mock objects for you e.g. `Mockito` and `PowerMock`.

14. What is SQL injection?
SQL injection is a security vulnerability which allows an intruder to steal data from the system. Any system which takes input from the user and creates SQL query without validating or sanitizing that input is vulnerable to SQL injection. In such system, an intruder can inject SQL code instead of data to retrieve more than expected data. There are many instances on which sensitive information e.g. user id, password, and personal details are stolen by exploiting this vulnerability. In Java, you can avoid SQL injection by using Prepared statement.

15. What is the difference between an inner join and a left join in SQL? (answer)
In SQL, there are mainly two types of joins, inner join, and outer join. Again outer joins can be two types right and left outer join. The main difference between inner join and left join is that in the case of former only matching records from both tables are selected while in the

case of left join, all records from left table are selected in addition to matching records from both tables. Always watch out for queries which have "all" in it, they usually require left join e.g. to write SQL query to find all departments and a number of employees on it. If you use inner join to solve this query, you will miss empty departments where no one works.

16. What does the V in MVC stand for, and what does it signify? (answer)
V stands for View in MVC pattern. The view is what user sees e.g. web pages. This is a very important design pattern of web development which is based upon segregation of concern so that each area can be modified without impacting other areas. In Java world, there are lots of open source framework which provides an implementation of MVC pattern e.g. Struts 2 and Spring MVC. By the way, M stands the for model and C stands the for controller. Modes are actual business objects e.g. User, Employee, Order; while the controller is used for the routing request to correct processor.

17. What is the difference between a class and an object? (detailed answer)
A class is a blueprint on which objects are created. A class has code and behavior but an object has both the state and behavior. You cannot create an object without creating a class to represent its structure. The class is also used to map an object in memory, in Java, JVM does that for you.

18. What is loose-coupling?
Loose coupling is a desirable quality of software, which allows one part of the software to modify without affecting another part of the software. For example, in a loosely coupled software, a change in UI layout should not affect the back-end class structure.

19. What is the difference between composition, aggregation, and association? (detailed answer)
Association means two objects are related to each other but can exist without each other, Composition is a form of association where one object is composed of multiple objects, but they only exists together e.g. human body is the composition of organs, individual organs cannot live they only useful in the body. Aggregation is a collection of object e.g. city is an aggregation of citizens.

21. What is unit testing? (answer)
Unit testing is a way to test individual unit for their functionality instead of testing whole application. There are a lot of tools to do the unit testing in different programming language e.g. in Java, you can use `JUnit` or `TestNG` to write unit tests. It is often run automatically during the build process or in a continuous environment like Jenkins.

22. Can you describe three different kinds of testing that might be performed on an application before it goes live?
unit testing, integration testing and smoke testing. Unit testing is used to test individual units to verify whether they are working as expected, integration testing is done to verify whether individually tested module can work together or not and smoke testing is a way to test whether most common functionality of software is working properly or not e.g. in a flight booking website, you should be able to book, cancel or change flights.

23. What is the difference between iteration and recursion? (detailed answer)
Iteration uses a loop to perform the same step again and again while recursion calls the function itself to do the repetitive task. Many times recursion result in a clear and concise solution to complex problem e.g. tower of Hanoi, reversing a linked list or reversing a String itself. One drawback of recursion is depth since recursion stores intermediate result in the stack you can only go up to a certain depth, after that your program will die with `StackOverFlowError`, this is why iteration is preferred over recursion in production code.

24. What is difference between & and && operator? (detailed answer)
& is a bitwise operator while && is a logical operator. One difference between `&` and `&&` is that bitwise operator (`&`) can be applied to both integer and boolean but logical operator (`&&`) can only be applied to boolean variables. When you do a & b then AND operator is applied to each bit of both integer number, while in the case of a && b, the second argument may or may not be evaluated, that's why it is also known as short circuit operator, at least in Java. I like this question and often asked it to junior or developer and college graduates.

25. What is the result of 1 XOR 1?
The answer is zero because XOR returns 1 if two operands are distinct and zero if two operands are same, for example, 0 XOR 0 is also zero, but 0 XOR 1 or 1 XOR 0 is always 1.

26. How do you get the last digit of an integer? (answer)
By using modulus operator, number % 10 returns the last digit of the number, for example, `2345%10` will return 5 and `567%10` will return 7. Similarly, division operator can be used to get rid of the last digit of a number e.g. 2345/10 will give 234 and 567/10 will return 56. This is an important technique to know and useful to solve problems like number palindrome or reversing numbers.

27. What is test-driven development?
Test driven is one of the popular development methodologies in which tests are written

before writing any function code. In fact, test drives the structure of your program. Purists never wrote a single line of application code without writing a test for that. It greatly improve code quality and often attributed as a quality of rockstar developers.

28. What is the Liskov substitution principle? (answer)
Liskov substitution principle is one of the five principle introduced by Uncle Bob as SOLID design principles. It's the 'L' in SOLID. Liskov substitution principle asserts that every subtype should be able to work as the proxy for parent type. For example, if a method except object of Parent class then it should work as expected if you pass an object of the Child class. Any class which cannot stand in place of its parent violate LSP or Liskov substitution principle. This is actually a tough question to answer and if you do that you end up with creating a good impression on interviewers mind.

29. What is the Open closed design principle? (answer)
Open closed is another principle from SOLID, which asserts that a system should be open for extension but closes for modification. Which means if a new functionality is required in a stable system then your tried and tested code should not be touched and new functionality should be provided by adding new classes only.

30. What is the difference between a binary tree and a binary search tree?
Binary search tree is an ordered binary tree, where the value of all nodes in the left tree are less than or equal to node and values of all nodes in right subtree is greater than or equal to the node (e.g. root). It's an important data structure and can be used to represent a sorted structure.

31. Can you give a practical example of a recursive algorithm? (example)
There are lots of places where recursive algorithm fits e.g. algorithm related to binary and linked list. A couple of examples of a recursive algorithm is reversing String and calculating Fibonacci series. Other examples include reversing linked list, tree traversal, and quick sort algorithm.

31. What is time complexity of an algorithm?
Time complexity specifies the ratio of time to the input. It shows how much time an algorithm will take to complete for a given number of input. It's approximated valued but enough to give you an indication that how your algorithm will perform if the number of input is increased from 10 to 10 million?

32. What are some important differences between a linked list and an array? (detailed answer)
linked list and array are two of the most important data structure in the programming world. The most significant difference between them is that array stores its element at the contiguous location while linked list stores its data anywhere in memory. This gives linked list enormous flexibility to expand itself because memory is always scattered. It's always possible that you wouldn't be able to create an array to store 1M integers but can do by using linked list because space is available but not as contiguous chunk. All other differences are the result of this fact. For example, you can search an element in array with O(1) time if you know the index but searching will take O(n) time in linked list. For more differences see the detailed answer.

33. What is a couple of ways to resolve collision in the hash table?
linear probing, double hashing, and chaining. In linear probing, if the bucket is already occupied then function check next bucket linearly until it finds an empty one, while in chaining, multiple elements are stored in same bucket location.

34. What is a regular expression? (answer)
A regular expression is a way to perform pattern matching on text data. It's very powerful tool to find something e.g. some character in a long string e.g. finding if a book contains some word or not. Almost all major programming language supports regular expression but Perl has been renowned for its enormous capability. Java also supports Perl-like regular expression using `java.util.regex` package. You can use the regular expression to check if an email is valid or not, if a phone number is valid, or if a zip code is valid, or even an SSN number is valid or not. One of the simplest examples of the regular expression is to check if a String is a number or not.

35. What is a stateless system?
A stateless system is a system which doesn't maintain any internal state. Such system will produce the same output for same input at any point of time. It's always easier to code and optimize a stateless system, so you should always strive for one if possible.

36. Write SQL query to find second highest salary in employee table? (solution)
This is one of the classic questions from SQL interviews, the event it's quite old it is still interesting and has lots of follow-ups you can use to check the depth of candidate's knowledge. You can find second highest salary by using the correlated and non-correlated subquery. You can also use keyword's like TOP or LIMIT if you are using SQL Server or MySQL, given Interviewer allows you. The simplest way to find 2nd highest salary is following:

```
SELECTMAX(Salary) FROM Employee WHERE Salary NOT IN (SELECTMAX(Salary) FROM
Employee)
```

This query first finds maximum salary and then exclude that from the list and again finds maximum salary. Obviously second time, it would be second highest salary.

37. Can you describe the difference between correlated and non-correlated subquery? (answer)
In correlated sub-query, inner query depends upon the outer query and executes for each row in the outer query. While non-correlated subquery doesn't depend upon the outer query and can be executed independently. Due to this reason former is slow and later is fast. BTW, correlated subquery has some nice application, one of them is finding Nth highest salary in Employee table, as seen on previous SQL question as well.

39. How do you find if a number is a power of two, without using arithmetic operator? (solution)
Assume it's a question of using the bitwise operator as soon as you hear restriction about not allowed to use arithmetic operator. If that restriction is not in place then you can easily check if a number is a power of two by using modulus and division operator. By the using bitwise operator, there is a nice trick to do this. You can use following code to check if a number if power of two or not

```
public static boolean powerOfTwo(int x) {
return (x & (x -1)) ==0;
}
```

`x & (x-1)` is a nice trick to convert right most bit to zero if it's on, I learned from hackers delight book.

40. How do you find a running Java process on UNIX? (command)
You can use the combination of `'ps'` and `'grep'` command to find any process running on UNIX machine. Suppose your Java process has a name or any text which you can use to match against just use following command.

```
ps-ef | grep "myJavaApp"
```

`ps -e` will list every process i.e. process from all user not just you and `ps -f` will give you full details including `PID`, which will be required if you want to investigate more or would like to kill this process using `kill` command.

41. How do you find large files in UNIX e.g. more than 1GB? (command)
You can easily find big files by using find command because it provides an option to search files based upon their size. Use this if your file system is full and your Java process is crashing with no more space. This command will list all files which are more than 1GB. You can tweak the size easily e.g. to find all files with more than 100 MB just use +100M.

```
find .-type f -size +1G -print
```

42. What is the shell script?
A shell script is set of shell commands with some programming constructs e.g. if and for loop, which allow you to automate some repetitive task. For example, you can write a shell script to the daily cleanup of logs files, for backing up data for historical use and for other housekeeping jobs, releases, and monitoring.

Read more: http://javarevisited.blogspot.com/2015/02/50-programmer-phone-interview-questions-answers.html#ixzz4NVtHsDLW

20 String Algorithm based Coding Interview Questions

Here is my collection of some of the most frequently asked String based coding questions from programming interview. Remember, there are many algorithms to solve the same problem, and you should know that, mainly to handle follow-up question better. Also remember to solve same question using both recursion and iteration, as interviewer really like to ask iterative version if you come up with recursive algorithm and vice-versa. Nevertheless, if you see your favorite question is not included in list, feel free to suggest, I will include it. You can also post question asked to you on your interview and you have not find its solution yet. Some questions are still unsolved or solution is not yet posted on my blog. Also difficult level increases as you move questions.

1) How to Print duplicate characters from String? (solution)
To start with, we have a simple String related coding question frequently asked in programming interviews. You need to write a program in C, C++, Java or Python to print duplicate characters from a given String, for example if String is `"Java"` then program should print `"a"`. Bonus points if your program is robust and handle different kinds of input e.g. String without duplicate, null or empty String etc. Bonus points if you also write unit tests for normal and edge cases.

2) How to check if two Strings are anagrams of each other? (solution)
A simple coding problem based upon String, but could also be asked with numbers. You need to write a Java program to check if two given strings are anagrams of Each other. Two strings are anagrams if they are written using the same exact letters, ignoring space, punctuation and capitalization. Each letter should have the same count in both strings. For example, `Army` and `Mary` are anagram of each other.

3) How to program to print first non repeated character from String? (solution)
One of the most common string interview questions: Find the first non-repeated (unique) character in a given string. for Example if given String is `"Morning"` then it should print "M". This question demonstrates efficient use of Hashtable. We scan the string from left to right counting the number occurrences of each character in a Hashtable. Then we perform a second pass and check the counts of every character. Whenever we hit a count of 1 we return that character, that's the first unique letter. Be prepared for follow-up question for improving memory efficiency, solving it without hash table as well.

4) How to reverse String in Java using Iteration and Recursion? ***(Solution)***
Your task is to write a program to reverse String in Java without using StringBuffer class. You also need to provide both iterative and recursive algorithm for String reversal. You can use other String utility methods
e.g. `charAt()`, `toCharArray()` or `substring()` from `java.lang.String` class.

5) How to check if a String contains only digits? ***(solution)***
You need to write a program to check a String contains only numbers by using Regular expression in Java. You can use Java API but a solution without using Java API will be better

because that is what interviewer can always ask.

6) How to find duplicate characters in a String? ***(solution)***

You need to write a program to print all duplicate character and their count in Java. For example if given String is "Programming" then your program should print

g : 2

r : 2

m : 2

7) How to count number of vowels and consonants in a String? ***(solution)***

One of easiest String question you will ever see. You have to write a Java program which will take a String input and print out number of vowels and consonants on that String. For example the if input is "Java" then your program should print `"2 vowels and 2 consonants"`. If you get this question on Interview, you should clarify that whether String can contain numbers, special characters or not e.g. anything other than vowels and consonants.

8) How to count occurrence of a given character in String? ***(solution)***

If interviewer ask you to count occurrence of more than one character than you can either use an array, hash table or any additional data structure. In order to solve this problem, you are not allowed to do so. Your method must return count of given character, for example if input String is "Java" and given character is 'a' then it should return 2. Bonus point if you handle case, null and empty String and come up with unit tests.

9) How to convert numeric String to an int? (solution)

A classical coding interview question based upon String. You need to write a method like atoi() from C/C++, which takes a numeric String and return its int equivalent. For example, if you pass "67263" to the program then it should return 67263. Make sure your solution is robust i.e. it should be able to handle + and - character, null and empty String, integer overflow and other corner cases. Bonus points if you come up with good unit test cases. By the way, if your interviewer doesn't mention to you about `atoi()` then you can also use Java API's `parseInt()` or `valueOf()` method to solve this problem.

10) How to replace each given character to other e.g. blank with %20? ***(solution)***
Write a Java program to replace a given character in a String to other provided character, for example if you are asked to replace each blank in a String with `%20`, similar to URL encoding done by the browser, so that Server can read all request parameters. For example if the input is "Java is Great" and asked to replace space with %20 then it should be `"Java%20is%20Great"`.

11) How to find all permutations of String? ***(solution)***
I have seen this String interview question on many interviews. It has a easy recursive solution but thinks get really tricky when Interviewer ask you to solve this question without using recursion. You can use Stack though. Write a program to print all permutations of a String in Java, for example, the if input is `"xyz"` then it should print `"xyz"`, `"yzx"`, `"zxy"`, `"xzy"`, `"yxz"`, `"zyx"`.

12) How to reverse words in a sentence without using library method? ***(solution)***
Write a function, which takes a String word and returns sentence on which words are reversed in order e.g. if an input is "Java is best programming language", the output should be "language programming best is Java".

*13) How to check if String is Palindrome?****(solution)***
Another easy coding question based upon String, I am sure you must have done this numerous time. Your program should return true if String is Palindrome, otherwise false. For example, if the input is `"radar"`, the output should be true, if input is `"madam"` output will be true, and if input is `"Java"` output should be false.

14) How to remove duplicate characters from String? ***(solution)***
This is one of the interesting String question, which also has lots of variants. You need to remove duplicate characters from a given string keeping only the first occurrences. For example, if the input is `'bananas'` the output will be `'bans'`. Pay attention to what output could be, because if you look closely original order of characters are retained the in output. This is where many developer make the a mistake of shorting character array of String and removing duplicates, similar to how you remove duplicates from array. That destroys original

order of characters and will not be correct solution in this case.

15) How to check if a String is valid shuffle of two String? ***(solution)***
One more difficult String algorithm based coding question for senior developers. You are given 3 strings: `first`, `second`, and `third`. third String is said to be a shuffle of first and second if it can be formed by interleaving the characters of first and second String in a way that maintains the left to right ordering of the characters from each string. For example, given first = `"abc"` and second = `"def"`, third = `"dabecf"` is a valid shuffle since it preserves the character ordering of the two strings. So, given these 3 strings write a function that detects whether third String is a valid shuffle of first and second String.

16) Write a program to check if a String contains another String e.g. indexOf()? ***(solution)***
You need to write a function to search for the existence of a string (target) in another string (source). The function takes two strings as the input and returns the index where the second string is found. If the target string cannot be found, then return -1. If you are a Java developer, then you can related its behavior to `indexOf()` method from java.lang.String class. This question is also asked as Code and algorithm to check if a given short string is a substring of a main string. Can you get a linear solution (O(n)) if possible?

17) How to return highest occurred character in a String? ***(solution)***
You need to write a function to implement an algorithm which will accept a string of characters and should find the highest occurrence of the character and display it. For example if input is "aaaaaaaaaaaaaaaaabbbbcddddeeeeee" it should return "a".

18) Write a program to remove a given characters from String? ***(solution)***
One of my favorite coding question, when I interview Java developers. You need to write a Java method which will accept a String and a character to be removed and return a String, which doesn't has that character e.g `remove(String word, char ch)`. You need to provide both iterative and recursive solution of this method and also has to write JUnit tests to cover cases like null and empty String, input which only contains letter to be removed, String which doesn't contain given character etc.

19) Write a program to find longest palindrome in a string? ***(solution)***
This is one of the tough coding question based upon String. It's hard to think about an algorithm to solve this problem until you have practiced good. What makes it more difficult is the constraint that your solution has O(n) time complexity and O(1) space complexity.

20) How to sort String on their length in Java? ***(solution)***
Write a Program to sort String on their length in Java? Your method should accept an array of String and return a sorted array based upon the length of String. Don't forget to write unit tests for your solution.

That's all on this list of **15 String Algorithm based coding questions**. These are really good question to prepare for programming job interviews, not only you can expect same question on a real interview but also it will prepare you how to tackle algorithmic coding interview questions. Even if you don't find same question, you would be able to apply the knowledge you gain by solving these question by yourself.

Always remember, you are judged by the code you write, so always write production quality code, which would pass general test, corner cases, invalid inputs, robustness test and also pass performance test. Whenever asked to solve a coding problem, always think about all possible input and write test for that.

Read more: http://javarevisited.blogspot.com/2015/01/top-20-string-coding-interview-question-programming-interview.html#ixzz4NVu3Z4cJ

30 Array Interview Questions for Programmers

1. How to find the missing number in integer array of 1 to 100? (solution)
This is one of the most simple array problems you will see, mostly asked in a telephonic round of Interview. You have given an integer array which contains numbers from 1 to 100 but one number is missing, you need to write a Java program to find that missing number in an array. You cannot use any open source library or Java API method which solves this problem. One trick to solve this problem is to calculate sum of all numbers in the array and compare with expected sum, the difference would be the missing number.

2. How to find duplicate number on Integer array in Java? (solution)
An array contains n numbers ranging from 0 to n-2. There is exactly one number is repeated in the array. You need to write a program to find that duplicate number. For example, if an array with length 6 contains numbers {0, 3, 1, 2, 3}, then duplicated number is 3. Actually this problem is very similar to previous one and you can apply the same trick of comparing actual sum of array to expected sum of series to find out that duplicate. This is generally asked as follow-up question of previous problem.

3. How to check if array contains a number in Java? (solution)
Another interesting array problem, because array doesn't provide any builtin method to check if any number exists or not. This problem is essentially how to search an element in array. There are two options sequential search or binary search. You should ask interviewer about whether array is sorted or not, if array is sorted then you can use binary search to check if given number is present in array or not. Complexity of binary search is O(logN). BTW, if interviewer say that array is not sorted then you can still sort and perform binary search otherwise you can use sequential search. Time complexity of sequential search in array is O(n).

4. How to find largest and smallest number in unsorted array? (solution)
This is a rather simple array interview question. You have given an unsorted integer array and you need to find the largest and smallest element in the array. Of course you can sort the array and then pick the top and bottom element but that would cost you O(NLogN) because of sorting, getting element in array with index is O(1) operation.

5. How to find all pairs on integer array whose sum is equal to given number? (solution)
This is an intermediate level of array coding question, its neither too easy nor too difficult. You have given an integer array and a number, you need to write a program to find all elements in array whose sum is equal to the given number. Remember, array may contain both positive and negative numbers, so your solution should consider that. Don't forget to write unit test though, even if interviewer is not asked for it, that would separate you from lot of developers. Unit testing is always expected from a professional developer.

6. How to find repeated numbers in an array if it contains multiple duplicate? (solution)
This is actually the follow-up question of problem 2, how to find duplicate number on integer array. In that case, array contains only one duplicate but what if it contains multiple duplicates? Suppose, an array contains n numbers ranging from 0 to n-1 and there are 5 duplicates on it, how do you find it? You can use the approach, we have learned in similar

String based problem of finding repeated characters in given String.

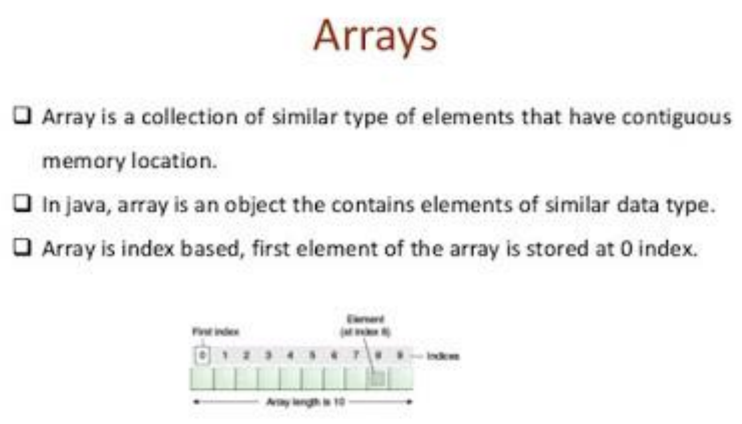
7. Write a program to remove duplicates from array in Java? (solution)
This is another follow-up question from problem 2 and 6. You have given an array which contains duplicates, could be one or more. You need to write a program to remove all duplicates from array in Java. For example if given array is {1, 2, 1, 2, 3, 4, 5} then your program should return an array which contains just {1, 2, 3, 4, 5}. This array question is also comes at intermediate category because there is no way to delete an element from array. If substituting with another value is not an option then you need to create another array to mimic deletion.

8. How to sort an array in place using QuickSort algorithm? (solution)
You will often see sorting problems on array related questions, because sorting mostly happen on array data structure. You need to write a program to implement in place quick sort algorithm in Java. You can implement either recursive or iterative quick sort, its your choice but you cannot use additional buffer, array or list, you must sort array in place.

9. Write a program to find intersection of two sorted array in Java? (solution)

Another interesting array interview question, where you need to treat array as Set. Your task is to write a function in your favorite language e.g. Java, Python, C or C++ to return intersection of two sorted array. For example, if the two sorted arrays as input are {21, 34, 41, 22, 35} and {61, 34, 45, 21, 11}, it should return an intersection array with numbers {34, 21}, For the sake of this problem you can assume that numbers in each integer array are unique.

10. There is an array with every element repeated twice except one. Find that element? (solution)
This is an interesting array coding problem, just opposite of question related to finding duplicates in array. Here you need to find the unique number which is not repeated twice. For example if given array is {1, 1, 2, 2, 3, 4, 4, 5, 5} then your program should return 3. Also, don't forget to write couple of unit test for your solution.



Arrays

- ❑ Array is a collection of similar type of elements that have contiguous memory location.
- ❑ In java, array is an object the contains elements of similar data type.
- ❑ Array is index based, first element of the array is stored at 0 index.

First index
Element (at index 8)
0 1 2 3 4 5 6 7 8 9
Indices
Array length is 10

11. How to find kth smallest element in unsorted array? (solution)

You are given an unsorted array of numbers and k, you need to find the kth smallest number in the array. For example if given array is {1, 2, 3, 9, 4} and k=2 then you need to find the 2nd smallest number in the array, which is 2. One way to solve this problem is to sort the array in ascending order then pick the k-1th element, that would be your kth smallest number in array because array index starts at zero, but can you do better? Once you are able to solve this array coding question, you can solve many similar questions easily e.g. our next question.

12. How to find kth largest element in unsorted array? (solution)
This problem is exactly same as previous question with only difference being finding kth largest element instead of kth smallest number. For example if given array is {10, 20, 30, 50, 40} and k = 3 then your program should return 30 because 30 is the 3rd largest number in array. You can also solve this problem by sorting the array in decreasing order and picking k-1th element. I often seen this array question on Java interviews with 2 to 3 years experienced guys.

13 How to find common elements in three sorted array? (solution)

Now we are coming on territory of tough array questions. Given three arrays sorted in non-decreasing order, print all common elements in these arrays.

Examples:

input1 = {1, 5, 10, 20, 40, 80}
input2 = {6, 7, 20, 80, 100}

input3 = {3, 4, 15, 20, 30, 70, 80, 120}

Output: 20, 80

14. How find the first repeating element in an array of integers? (solution)

Given an array of integers, find the first repeating element in it. We need to find the element that occurs more than once and whose index of the first occurrence is smallest.

Examples:

Input: input [] = {10, 5, 3, 4, 3, 5, 6}

Output: 5 [5 is the first element that repeats]

15. How to find first non-repeating element in array of integers? (solution)

This array interview question is exactly opposite of previous problem, In that you need to find first repeating element while in this you need to find first non-repeating element. I am sure you can use similar approach to solve this problem, just need to consider non repeating element though.

16. How to find top two numbers from an integer array? (solution)
This is another one of the easy array questions you will find on telephonic round of Interviews, but its also little bit tricky. You are asked to find top two numbers not just the top or highest numbers? Can you think of how you would do it without sorting? before looking at solution.

17. How to find the smallest positive integer value that cannot be represented as sum of any subset of a given array? (solution)
This is another tough array question you will see on Amazon, Microsoft or Google. You have given a sorted array (sorted in non-decreasing order) of positive numbers, find the smallest positive integer value that cannot be represented as sum of elements of any subset of given set. What makes it more challenging is expected time complexity of O(n).

Examples:

Input: {1, 3, 6, 10, 11, 15};

Output: 2

18. How to rearrange array in alternating positive and negative number? (solution)
Given an array of positive and negative numbers, arrange them in an alternate fashion such that every positive number is followed by negative and vice-versa maintaining the order of appearance.

Number of positive and negative numbers need not be equal. If there are more positive numbers they appear at the end of the array. If there are more negative numbers, they too appear in the end of the array. This is also a difficult array problem to solve and you need lot of practice to solve this kind of problems in real interviews, especially when you see it first time. If you have time constraint then always attempt these kind of questions once you are done with easier ones.

Example:

Input: {1, 2, 3, -4, -1, 4}

Output: {-4, 1, -1, 2, 3, 4}

Input: {-5, -2, 5, 2, 4, 7, 1, 8, 0, -8}

output: {-5, 5, -2, 2, -8, 4, 7, 1, 8, 0}

19. How to find if there is a sub array with sum equal to zero? (solution)

There is whole set of array related questions which are based upon sub-array or only selective elements of array e.g. from some range, this is one of such problem. Here you are given an array of positive and negative numbers, find if there is a sub-array with 0 sum.

Examples:

Input: {4, 2, -3, 1, 6}

Output: true

There is a sub-array with zero sum from index 1 to 3.

20. How to remove duplicates from array in place? (solution)

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,

Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].

When you see a questions which asked you do to sorting or task in place, it means you cannot use additional array or buffer, but using couple of variables is fine.

21. How to remove a given element from array in Java? (solution)
This is another array coding questions similar to previous one. Here you don't have to find and remove duplicates but a given number. In this problem you are given an array and a value, remove all instances of that value in place and return the new length. The order of elements can be changed. It doesn't matter what you leave beyond the new length.

22. How to merge sorted array? (solution)

Given two sorted integer arrays A and B, merge B into A as one sorted array. You may assume that A has enough space (size that is greater or equal to m + n) to hold additional elements from B. The number of elements initialized in A and B are m and n respectively. This is another intermediate array coding question, its not as simple as previous one but neither very difficult.

23. How to find sub array with maximum sum in an array of positive and negative number? (solution)

Another array coding question based upon sub-array. Here you have to find the contiguous sub-array within an array (containing at least one number) which has the largest sum.

For example, given the array [−2,1,−3,4,−1,2,1,−5,4],

the contiguous subarray [4,−1,2,1] has the largest sum = 6.

24. How to find sub array with largest product in array of both positive and negative number? (solution)
In this problem, your task is to write a program in Java or C++ to find the contiguous sub-array within an array (containing at least one number) which has the largest product.

For example, given the array [2,3,-2,4],

the contiguous subarray [2,3] has the largest product = 6.

25. Write a program to find length of longest consecutive sequence in array of integers? (solution)

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,

Given [100, 4, 200, 1, 3, 2],

The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Challenging part of this question is that your algorithm should run in O(n) complexity.

26. How to find minimum value in a rotated sorted array? (solution)

This is another advanced level array coding question and you should only attempt this one, once you have solved the easier ones. Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

You may assume no duplicate exists in the array. One follow-up question of this question is What if duplicates are allowed? Would this affect the run-time complexity? How and why?

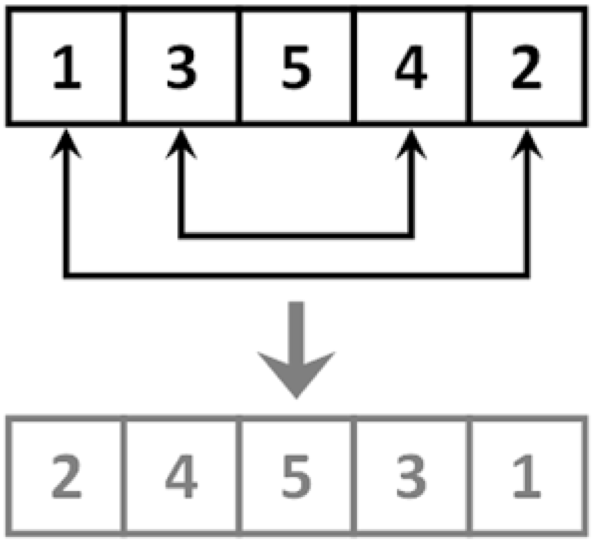
27. Given an array of of size n and a number k, find all elements that appear more than n/k times? (solution)
Another tough array based coding questions from Interviews. You are given an array of size n, find all elements in array that appear more than n/k times. For example, if the input arrays is {3, 1, 2, 2, 1, 2, 3, 3} and k is 4, then the output should be [2, 3]. Note that size of array is 8 (or n = 8), so we need to find all elements that appear more than 2 (or 8/4) times. There are two elements that appear more than two times, 2 and 3.

1. Returns the largest sum of contiguous integers in the array
Example: if the input is (-10, 2, 3, -2, 0, 5, -15), the largest sum is 8

2. Return the sum two largest integers in an array

3. Given an array of integers write a program that will determine if any two numbers add up to a specified number N. Do this without using hash tables

28. How to reverse array in place in Java? (solution)
Now let's see one of the most frequently asked array interview question. You need to write a program which accepts an integer array and your program needs to reverse that array in place, which means you cannot use additional buffer or array, but one or two variables will be fine. Of course you cannot use any open source library or Java API method to directly solve this problem, you need to create your own logic. Here is one such logic to solve this problem :



29. Difference between array and linked list data structure? (answer)
This is a theoretical questions from phone interviews. There are several differences between array and linked list e.g. array stores element in contiguous memory location while linked list stores at random places, this means linked list better utilizes the places. Consequently, its possible to have large linked list in limited memory environment compare to array of same size. Advantage of using array is random access it provides if you know the index, while in linked list you need to search an element by traversing which is O(n) operation.

30. How to check if array contains a duplicate number? (answer)
This may look a repeated question because we have already done similar question, but in this question, most from Java interviews, you need to write a `contains()` like method from Collections, which returns true or false if you pass an element and it is repeated or not.

Read more: http://javarevisited.blogspot.com/2015/06/top-20-array-interview-questions-and-answers.html#ixzz4NVuLHzoV

Data Structures and Algorithm Interview Questions for Java Programmers

Question 1: How to find middle element of linked list in one pass?
One of the most popular question from data structures and algorithm, mostly asked on telephonic interview. Since many programmer know that, in order to find length of linked list we need to first traverse through linked list till we find last node, which is pointing to null, and then in second pass we can find middle element by traversing only half of length. They get confused when interviewer ask him to do same job in one pass. In order to find middle element of linked list in one pass, you need to maintain two-pointer, one increment at each node while other increments after two nodes at a time, by having this arrangement, when first pointer reaches end, second pointer will point to middle element of linked list. See this trick to find middle element of linked list in single pass for more details.

Question 2: How to find if linked list has a loop ?
This question has bit of similarity with earlier algorithm and data structure interview question. I mean we can use two pointer approach to solve this problem. If we maintain two pointers, and we increment one pointer after processing two nodes and other after processing every node, we are likely to find a situation where both the pointers will be pointing to same node. This will only happen if linked list has loop.

Question 3 : How to find 3rd element from end in a linked list in one pass?
This is another frequently asked linked list interview question. This question is exactly similar to finding middle element of linked list in single pass. If we apply same trick of maintaining two pointers and increment other pointer, when first has moved up to 3rd element, than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

Question 4: In an integer array, there is 1 to 100 number, out of one is duplicate, how to find?
This is a rather simple data structures question, especially for this kind of. In this case you can simply add all numbers stored in array, and total sum should be equal to `n(n+1)/2`. Now just subtract actual sum to expected sum, and that is your duplicate number. Of course there is a brute force way of checking each number against all other numbers, but that will result in performance of O(n^2) which is not good. By the way this trick will not work if array have multiple duplicates or its not numbers forming arithmetic progression. Here is example of one way to find duplicate number in array.

Question 6 : How to reverse String in Java ?
This is one of my favorite question. Since `String` is one of the most important type of programming, you expect lot of question related to String any data structure interview. There are many ways to reverse Sting in Java or any other programming language, and interviewer will force you to solve this problem by using without API i.e. without using `reverse()` method of `StringBuffer`. In follow-up he may ask to reverse String using recursion as well. See 3 ways to reverse String in Java to learn reversing String using both loops and recursion in Java.

Question 7: Write a Java program to sort an array using Bubble Sort algorithm?
I have always send couple of questions from searching and sorting in data structure interviews. Bubble sort is one of the simplest sorting algorithm but if you ask anyone to implement on the spot it gives you an opportunity to gauge programming skills of a candidate. See How to sort array using Bubble Sort in Java for complete solution of this datastrucutre interview question.

Question 8: What is the difference between Stack and Queue data structure?
One of the classical data structure interviews question. I guess every one know, No? Any way main difference is that Stack is LIFO(Last In First Out) data structure while Queue is a FIFO(First In First Out) data structure.

Question 9: How do you find duplicates in an array if there is more than one duplicate?
Sometime this is asked as follow-up question of earlier data structure interview question, related to finding duplicates in Array. One way of solving this problem is using a Hashtable or HashMap data structure. You can traverse through array, and store each number as key and number of occurrence as value. At the end of traversal you can find all duplicate numbers, for which occurrence is more than one. In Java if a number already exists in HashMap then calling `get(index)` will return number otherwise it return null. this property can be used to insert or update numbers in `HashMap`.

Question 10 : What is difference between Singly Linked List and Doubly Linked List data structure?
This is another classical interview question on data structure, mostly asked on telephonic rounds. Main difference between singly linked list and doubly linked list is ability to traverse. In a single linked list, node only points towards next node, and there is no pointer to previous node, which means you can not traverse back on a singly linked list. On the other hand doubly linked list maintains two pointers, towards next and previous node, which allows you to navigate in both direction in any linked list.

Question 11 : Write Java program to print Fibonacci series ?
This is not a data structures question, but a programming one, which many times appear during data structure interview. Fibonacci series is a mathematical series, where each number is sum of previous two numbers e.g. 1,1, 2, 3, 5, 8, 13, 21. Interviewer is often interested in two things, a function which returns nth number in Fibonacci series and solving this problem using recursion in Java. Though, its easy question, recursion part often confuses beginners. See this link to find nth Fibonacci number in Java.

Question 12: Write Java program to check if a number is a palindrome or not?
This is similar to previous question, not directly related to data structures, but quite popular along with other questions. A number is called palindrome, if reverse of number is equal to number itself. Interviewer ask to solve this problem without taking help from Java API or any open source library. Any way it’s simple question, you can use division operator (/) and remainder operator (%) to solve this question. Just remember, division operator can be used to get rid of last digit e.g. 1234/10 will give you 123, and modulus operator can give you last digit e.g. 1234%10 will return 4. By the way, here is a Java program check if number is palindrome or not.

Question 13 : What is binary search tree?
This is a data structure question from Tree data structures. Binary Search Tree has some special properties e.g. left nodes contains items whose value is less than root , right sub tree contains keys with higher node value than root, and there should not be any duplicates in the tree. Apart from definition, interview can ask you to implement binary search tree in Java and questions on tree traversal e.g. IN order, preorder, and post order traversals are quite popular data structure question.

Question 14 : How to reverse linked list using recursion and iteration?
This is another good question on data structures. There are many algorithms to reverse linked list and you can search for them using google. I am thinking of writing another blog post to explain linked list reversal and will share with you later.

Question 15: Write a Java program to implement Stack in Java?
You can implement Stack by using array or linked list. This question expect you to implement standard method provided by stack data structure e.g. `push()` and `pop()`. Both `push()` and `pop()` should be happen at top of stack, which you need to keep track. It's also good if you can implement utility methods like `contains()`, `isEmpty()` etc. By the way JDK has java.util.Stack class and you can check it's code to get an idea. You can also check Effective Java book, where Josh Bloch has explains how an incorrect implementation of stack can cause memory leak in Java.

Read more: http://javarevisited.blogspot.com/2013/03/top-15-data-structures-algorithm-interview-questions-answers-java-programming.html#ixzz4NVuS4rzj

Difference between PATH and CLASSPATH in Java

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

1)The main difference between PATH and CLASSPATH is that PATH is an environment variable which is used to locate JDK binaries like "`java`" or "`javac`" command used to run java program and compile java source file. On the other hand, CLASSPATH, an environment variable is used by System or Application ClassLoader to locate and load compile Java bytecodes stored in the .class file.

2) In order to set PATH in Java, you need to include `JDK_HOME/bin` directory in PATH environment variable while in order to set CLASSPATH in Java you need to include all those directories where you have put either your `.class` file or JAR file which is required by your Java application.

3) Another significant difference between PATH and CLASSPATH is that PATH can not be overridden by any Java settings but CLASSPATH can be overridden by providing command line option `-classpath` or `-cp` to both "`java`" and "`javac`" commands or by using Class-Path attribute in Manifest file inside JAR archive.

4) PATH environment variable is used by operating system to find any binary or command typed in the shell, this is true for both Windows and Linux environment while CLASSPATH is only used by Java ClassLoaders to load class files.

These were some notable difference between PATH vs CLASSPATH in Java and they are worth remembering to debug and troubleshoot Java-related issues. Though, I highly recommend you to read Core Java Volume 1 - Fundamentals by Cay S. Horstmann to build your fundamentals in Java.

How to set PATH and CLASSPATH in Windows and Unix

If you are familiar with DOS operating system and how to use command prompt in Windows or shell in Linux setting PATH and CLASSPATH is a trivial exercise. Both PATH and CLASSPATH are environment variable and can be set using `export command` in Linux and using `set` keyword in DOS and Windows as shown below:

Command to set PATH in Windows

```
set PATH=%PATH%;C:\Program Files\Java\JDK1.6.20\bin
```

Command to set PATH in UNIX/Linux

```
export PATH = ${PATH}:/opt/Java/JDK1.6.18/bin
```

Look at the difference between two commands, in Linux use a colon(:) as a separator and in Windows use semi-colon(;) as a separator.

Command to set CLASSPATH in windows

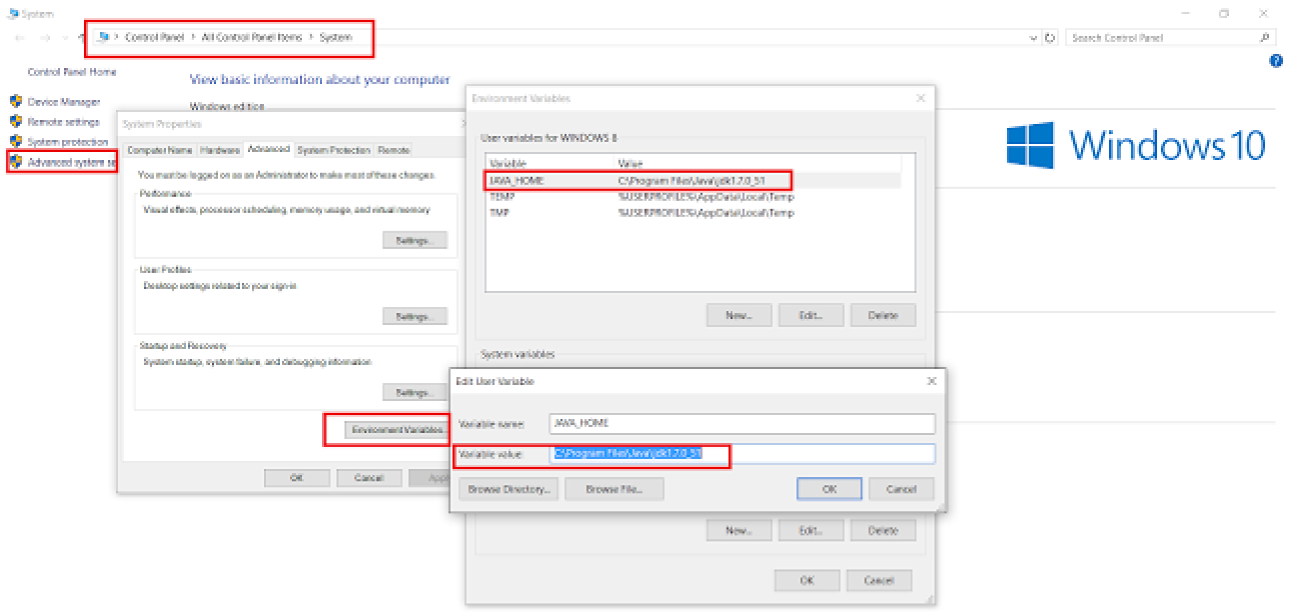
```
set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\JDK1.6.20\lib
```

Command to set CLASSPATH in Unix/Linux

```
export CLASSPATH= ${CLASSPATH}:/opt/Java/JDK1.6.18/lib
```

Also, don't forget to include current directory, denoted by a dot(.) to include in CLASSPATH, this will ensure that it will look first in the current directory and if it found the class it will use that even if that class also exists in another directory which exists in CLASSPATH. If you using Windows 8 or Windows 10 then you can also follow steps given here in this article to setup the JAVA_HOME, PATH and CLASSPATH for Java programs.

It will look something like this in Windows 10 Operating System :



You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles: Desktop settings related to your sign-in

Startup and Recovery: System startup, system failure, and debugging information

User variables for WINDOWS 8

Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk1.7.0_51
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.7.0_51

Read more: http://www.java67.com/2012/08/what-is-path-and-classpath-in-java-difference.html#ixzz4Q1001Ic

What is java.lang.OutOfMemoryError in Java

OutOfMemoryError in Java is a subclass **of** **`java.lang.VirtualMachineError`** and JVM throws java.lang.OutOfMemoryError when it ran *out of memory in the heap*. OutOfMemoryError in Java can come anytime in heap mostly while you try to create an object and there is not enough space on the heap to allocate that object. Javadoc of OutOfMemoryError is not very informative about this, though.

Types of OutOfMemoryError in Java
I have seen mainly two types of OutOfMemoryError in Java:

1) The **java.lang.OutOfMemoryError: Java heap space**
2) The **java.lang.OutOfMemoryError: PermGen space**

Though both of them occur because JVM ran out of memory they are quite different to each other and their solutions are independent of each other.

The difference between "java.lang.OutOfMemoryError: Java heap space" and "java.lang.OutOfMemoryError: PermGen space"

If you are familiar with different generations on the heap and How garbage collection works in java and aware of new, old and permanent generation of heap space then you would have easily figured out this OutOfMemoryError in Java. Permanent generation of the heap is used to store String pool and various Metadata required by JVM related to Class, method and other java primitives.

Since **in most of JVM default size of Perm Space is around "64MB"** you can easily run out of memory if you have too many classes or a huge number of Strings in your project.

An important point to remember is that it doesn't depend on **-Xmx** value so no matter how big your total heap size you can run OutOfMemory in perm space. The good thing is you can specify **the size of permanent generation** using JVM options **"-XX: PermSize"** and **"-XX: MaxPermSize"** based on your project need.

One small thing to remember is that "=" is used to separate parameter and value while specifying the **size of perm space in the heap** while "=" is not required while **setting maximum heap size in java**, as shown in below example.

```
export JVM_ARGS="-Xmx1024m -XX:MaxPermSize=256m"
```

Another reason of "**java.lang.OutOfMemoryError: PermGen**" is memory leak through Classloaders and it's very often surfaced in WebServer and application server like tomcat, WebSphere, glassfish or WebLogic.

In Application server different classloaders are used to load different web applications so that you can deploy and undeploy one application without affecting other application on the same server, but while undeploying if container somehow keeps reference of any class loaded by application class loader then that class and all other related class will not be garbage collected and can quickly fill the PermGen space if you deploy and undeploy your application many times.

"*java.lang.OutOfMemoryError: PermGen*" has been observed many times in tomcat in our last project, but the solution of this problem are really tricky because first you need to know which class is causing a memory leak and then you need to fix that. Another reason of `OutOfMemoryError` in PermGen space is if any thread started by the application doesn't exit when you undeploy your application.

These are just some example of infamous classloader leaks, anybody who is writing code for loading and unloading classes has to be very careful to avoid this. You can also use **visualgc** for monitoring PermGen space, this tool will show the graph of PermGen space and you can see how and when Permanent space getting increased. I suggest using this tool before reaching to any conclusion.

Another rather unknown but interesting cause of "java.lang.OutOfMemoryError: PermGen" we found is introduction of JVM options **"-Xnoclassgc**".

This option sometimes used to avoid loading and unloading of classes when there are no further live references of it just to avoid performance hit due to frequent loading and unloading, but using this option is J2EE environment can be very dangerous because many framework e.g. Struts, spring etc uses reflection to create classes and with frequent deployment and undeployment you can easily run out of space in **PermGen** if earlier references were not cleaned up. This instance also points out that sometimes bad JVM arguments or configuration can cause OutOfMemoryError in Java.

So the conclusion is to avoid using ***"-Xnoclassgc*"** in the J2EE environment especially with AppServer.

Tomcat to Solve OutOfMemoryError in PermGen Space
From tomcat > 6.0 onward tomcat provides memory leak detection feature which can detect many common memory leaks on web-app perspective e.g ThreadLocal memory leaks, JDBC driver registration, RMI targes, LogFactory and Thread spawned by web-apps. You can check complete details on htp://wiki.apache.org/tomcat/MemoryLeakProtection you can also detect memory leak by accessing manager application which comes with tomcat, in case you are experiencing memory leak on any java web-app its good idea to run it on tomcat.

How to solve java.lang.OutOfMemoryError: Java heap space

1) An easy way to solve OutOfMemoryError in java is to *increase the maximum heap size* by using JVM options "-Xmx512M", this will immediately solve your OutOfMemoryError. This is my preferred solution when I get OutOfMemoryError in Eclipse, Maven or ANT while building project because based upon size of project you can easily run out of Memory.here is **an example of increasing maximum heap size of JVM**, Also its better to keep **-Xmx to -Xms** ration either 1:1 or 1:1.5 if you are setting heap size in your java application

```
export JVM_ARGS="-Xms1024m -Xmx1024m"
```

2) The second way to resolve OutOfMemoryError in Java is rather hard and comes when you don't have much memory and even after increase maximum heap size you are still getting java.lang.OutOfMemoryError, in this case, you probably want to profile your application and look for any memory leak. You can use Eclipse Memory Analyzer to examine your heap dump or you can use any profiler like Netbeans or JProbe. This is tough solution and requires some time to analyze and **find memory leaks**.

How to solve java.lang.OutOfMemoryError: PermGen space

As explained in above paragraph this OutOfMemory error in java comes when Permanent generation of heap filled up. To fix this OutOfMemoryError in Java, you need to *increase heap size of Perm space* by using JVM option **"-XX: MaxPermSize".** You can also specify initial size of Perm space by using **"-XX: PermSize"** and keeping both initial **and maximum Perm Space** you can prevent some full garbage collection which may occur when Perm Space gets re-sized. Here is **how you can specify initial and maximum Perm size in Java**:

```
export JVM_ARGS="-XX:PermSize=64M -XX:MaxPermSize=256m"
```

Some time `java.lang.OutOfMemoryError` in Java gets tricky and on those cases profiling remains ultimate solution.Though you have the freedom to increase heap size in java, it’s recommended that to follow memory management practices while coding and setting null to any unused references.
That’s all from me on **OutOfMemoryError in Java** I will try to write more about finding the memory leak in java and using profiler in some other post. Please share what is your approach to solving *java.lang.OutOfMemoryError in Java*.

Important Note: From Tomcat > 6.0 onward tomcat provides memory leak detection feature which can detect many common memory leaks on Java application e.g ThreadLocal memory leaks, JDBC driver registration, RMI targes, LogFactory, and Thread spawned by web apps. You can check complete details on htp://wiki.apache.org/tomcat/MemoryLeakProtection. You can also detect memory leak by accessing manager application which comes with tomcat, in case you are experiencing memory leak on any java web app it's a good idea to run it on tomcat to find out the reason of OutOfMemoryError in PermGen space.

Tools to investigate and fix OutOfMemoryError in Java
Java.lang.OutOfMemoryError is a kind of error which needs a lot of investigation to find out the root cause of the problem, which object is taking memory, how much memory it is taking or finding dreaded memory leak and you can't do this without having knowledge of available tools in java space. Here I am listing out some free tools which can be used to analyze heap and will help you to find culprit of OutOfMemoryError

1) **Visualgc**
Visualgc stands for Visual Garbage Collection Monitoring Tool and you can attach it to your instrumented hotspot JVM. The main strength of visualgc is that it displays all key data graphically including class loader, garbage collection, and JVM compiler performance data.

The target JVM is identified by its virtual machine identifier also called as vmid. You can read more about visualgc and vmid options here.

2) Jmap

Jmap is a command line utility comes with JDK6 and allows you to take a memory dump of the heap in a file. It’s easy to use as shown below:

```
jmap -dump:format=b,file=heapdump 6054
```

Here file specifies the name of memory dump file which is "heap dump" and 6054 is PID of your Java progress. You can find the PDI by using "ps -ef” or windows task manager or by using the tool called "jps"(Java Virtual Machine Process Status Tool).

3) Jhat
Jhat was earlier known as hat (heap analyzer tool) but it is now part of JDK6. You can use jhat to analyze heap dump file created by using "**jmap**". Jhat is also a command line utility and you can run it from cmd window as shown below:

```
jhat -J-Xmx256m heapdump
```

Here it will analyze memory dump contained in file "heapdump". When you start **jhat** it will read this heap dump file and then start listening on HTTP port, just point your browser into port where jhat is listening by default 7000 and then you can start analyzing objects present in heap dump.

4) Eclipse memory analyzer
Eclipse memory analyzer (MAT) is a tool from eclipse foundation to analyze java heap dump. It helps to find classloader leaks and memory leaks and helps to minimize memory

consumption.you can use MAT to analyze heap dump carrying millions of object and it also helps you to extract suspect of memory leak. See here for more information.

5) Books to learn Profiling

As your Java experience grows, expectation also grows in terms of niche skills like analyzing performance issue and comfortable with profiling. You won't normally learn those skill unless you take extra effort. In order to effectively deal with the error like java.lang.OutOfMemoryError, you should read good books on troubleshooting and performance tuning e.g. Java Performance The Definitive Guide By Scott Oaks as shown below:

Read more: http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html#ixzz4Q192kVn6

How ClassLoader Works in Java

Java class loaders are used to load classes at runtime. ClassLoader in Java works on three principle: `delegation`, `visibility` and `uniqueness`. Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class. Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child. Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent. Correct understanding of class loader is must to resolve issues like NoClassDefFoundError in Java and java.lang.ClassNotFoundException, which are related to class loading. ClassLoader is also an important topic in advanced Java Interviews, where good knowledge of working of Java ClassLoader and How classpath works in Java is expected from Java programmer. I have always seen questions like, **Can one class be loaded by two different ClassLoader in Java** on various Java Interviews. In this Java programming tutorial, we will learn what is ClassLoader in Java, How ClassLoader works in Java and some specifics about Java ClassLoader.

What is ClassLoader in Java

ClassLoader in Java is a class which is used to load class files in Java. Java code is compiled into class file by `javac` compiler and JVM executes Java program, by executing byte codes written in class file. ClassLoader is responsible for loading class files from file system, network or any other source. There are three default class loader used in Java, **Bootstrap** , **Extension** and **System or Application class loader**.

Every class loader has a predefined location, from where they loads class files. Bootstrap ClassLoader is responsible for loading standard JDK class files from `rt.jar` and it is parent of all class loaders in Java. Bootstrap class loader don't have any parents, if you call `String.class.getClassLoader()` it will return `null` and any code based on that may throw NullPointerException in Java. Bootstrap class loader is also known as **Primordial ClassLoader** in Java.

Extension ClassLoader delegates class loading request to its parent, `Bootstrap` and if unsuccessful, loads class form `jre/lib/ext` directory or any other directory pointed by `java.ext.dirs` system property. Extension ClassLoader in JVM is implemented by `sun.misc.Launcher$ExtClassLoader`.

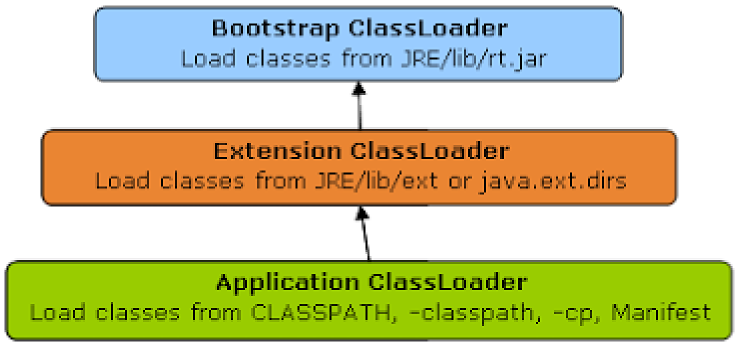
Third default class loader used by JVM to load Java classes is called System or Application class loader and it is responsible for loading application specific classes from CLASSPATH environment variable, `-classpath` or `-cp` command line option, `Class-Path` attribute of Manifest file inside JAR. Application class loader is a child of Extension ClassLoader and its implemented by `sun.misc.Launcher$AppClassLoader`class. Also, except Bootstrap class loader, which is implemented in native language mostly in C, all Java class loaders are implemented using `java.lang.ClassLoader`.

In short here is the location from which Bootstrap, Extension and Application ClassLoader load Class files.

1) Bootstrap ClassLoader - JRE/lib/rt.jar

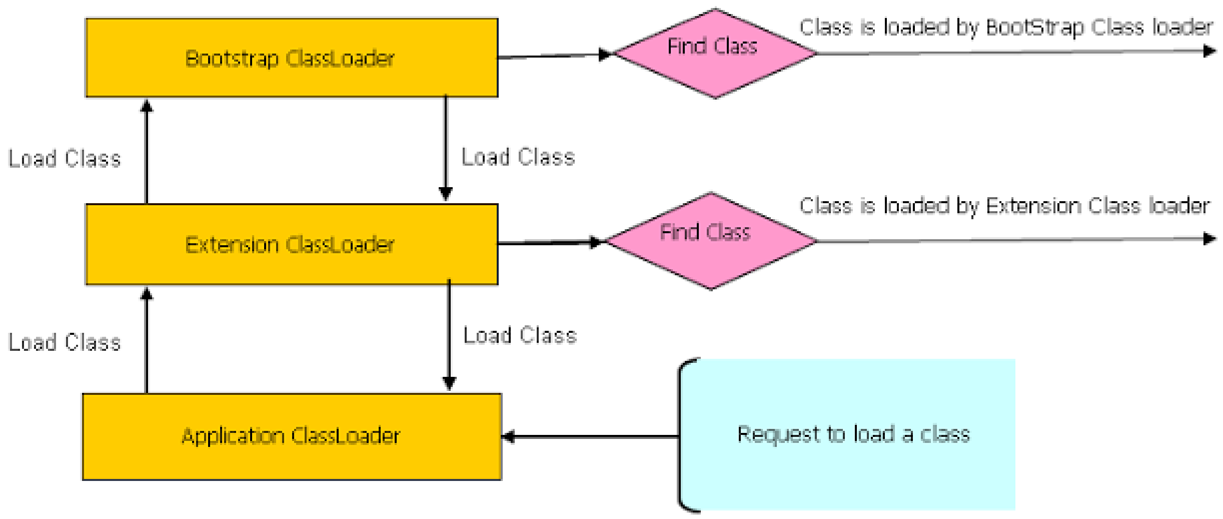
2) Extension ClassLoader - JRE/lib/ext or any directory denoted by `java.ext.dirs`

3) Application ClassLoader - `CLASSPATH` environment variable, `-classpath` or `-cp` option, Class-Path attribute of Manifest inside JAR file.


Bootstrap ClassLoader
Load classes from JRE/lib/rt.jar
Extension ClassLoader
Load classes from JRE/lib/ext or java.ext.dirs
Application ClassLoader
Load classes from CLASSPATH, -classpath, -cp, Manifest

How ClassLoader works in Java

As I explained earlier Java ClassLoader works in three principles : `delegation`, `visibility` and `uniqueness`. In this section we will see those rules in detail and understand working of Java ClassLoader with example. By the way here is a diagram which explains How ClassLoader load class in Java using delegation.



Delegation principles

As discussed on when a class is loaded and initialized in Java, a class is loaded in Java, when its needed. Suppose you have an application specific class called `Abc.class`, first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to `Primordial` or `Bootstrap` class loader. Primordial will look for that class in `rt.jar` and since that class is not there, request comes to Extension class loader which looks on `jre/lib/ext` directory and tries to locate this class there, if class is found there than Extension class loader will load that class and Application class loader will never load that class but if its not loaded by extension class-loader than Application class loader loads it from Classpath in Java. Remember `Classpath` is used to load class files while PATH is used to locate executable like javac or java command.

Visibility Principle

According to visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. Which mean if class `Abc` is loaded by Application class loader than trying to load class ABC explicitly using extension ClassLoader will throw either java.lang.ClassNotFoundException. as shown in below Example

```
package test;

import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Java program to demonstrate How ClassLoader works in Java,
 * in particular about visibility principle of ClassLoader.
 *
 * @author Javin Paul
 */
```

```
public class ClassLoaderTest {

    public static void main(String args[]) {
        try {
            //printing ClassLoader of this class
            System.out.println("ClassLoaderTest.getClass().getClassLoader() : "
                                     + ClassLoaderTest.class.getClassLoader());


            //trying to explicitly load this class again using Extension class loader
            Class.forName("test.ClassLoaderTest", true
                                ,  ClassLoaderTest.class.getClassLoader().getParent());
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(ClassLoaderTest.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

}

Output:
ClassLoaderTest.getClass().getClassLoader() : sun.misc.Launcher$AppClassLoader@601bb1
16/08/2012 2:43:48 AM test.ClassLoaderTest main
SEVERE: null
java.lang.ClassNotFoundException: test.ClassLoaderTest
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at sun.misc.Launcher$ExtClassLoader.findClass(Launcher.java:229)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:247)
        at test.ClassLoaderTest.main(ClassLoaderTest.java:29)
```

Uniqueness Principle

According to this principle a class loaded by Parent should not be loaded by Child ClassLoader again. Though its completely possible to write class loader which violates Delegation and Uniqueness principles and loads class by itself, its not something which is beneficial. You should follow all class loader principle while writing your own ClassLoader.

How to load class explicitly in Java

Java provides API to explicitly load a class by `Class.forName(classname)` and `Class.forName(classname, initialized, classloader)`, remember JDBC code which is used to load JDBC drives we have seen in Java program to Connect Oracle database. As shown in above example you can pass name of ClassLoader which should be used to load that particular class along with binary name of class. Class is loaded by calling `loadClass()` method of `java.lang.ClassLoader` class which calls `findClass()` method to locate bytecodes for corresponding class. In this example Extension ClassLoader uses `java.net.URLClassLoader` which search for class files and resources in JAR and directories. any search path which is ended using "/" is considered directory. If `findClass()` does not found the class than it throws java.lang.ClassNotFoundException and if it finds it calls `defineClass()` to convert bytecodes into a .class instance which is returned to the caller.

Where to use ClassLoader in Java

ClassLoader in Java is a powerful concept and used at many places. One of the *popular example of ClassLoader* is `AppletClassLoader` which is used to load class by `Applet`, since `Applets` are mostly loaded from internet rather than local file system, By using separate ClassLoader you can also loads same class from multiple sources and they will be treated as different class in JVM. J2EE uses multiple class loaders to load class from different location like classes from WAR file will be loaded by Web-app ClassLoader while classes bundled in EJB-JAR is loaded by another class loader. Some web server also supports hot deploy functionality which is implemented using ClassLoader. You can also use ClassLoader to load classes from database or any other persistent store.

That's all about **What is ClassLoader in Java** and **How ClassLoader works in Java**. We have seen delegation, visibility and uniqueness principles which is quite important to debug or troubleshoot any ClassLoader related issues in Java. In summary knowledge of How ClassLoader works in Java is must for any Java developer or architect to design Java application and packaging.

Read more: http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html#ixzz4Q1o05DpE