<https://www.quora.com/What-are-the-scenario-based-Java-interview-questions-you-have-come-across>

<https://www.quora.com/What-would-be-the-trickiest-question-about-coding-Java-in-technical-interviews/answer/Renuka-Peshwani-1?ch=10&share=dce9e60f&srid=2ne1z>

<https://www.geeksforgeeks.org/oops-generalization-as-extension-and-restriction-using-java/>

**What Methods Does the Object Class Have?**

The [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) class, in the java.lang package, sits at the top of the class hierarchy tree. Every class is a descendant, direct or indirect, of the Object class. Every class you use or write inherits the instance methods of Object. You need not use any of these methods, but, if you choose to do so, you may need to override them with code that is specific to your class. The methods inherited from Object that are discussed in this section are:

* protected Object clone() throws CloneNotSupportedException  
  Creates and returns a copy of this object.
* public boolean equals(Object obj)  
  Indicates whether some other object is “equal to” this one.
* protected void finalize() throws Throwable  
  Called by the garbage collector on an object when garbagecollection determines that there are no more references to the object.
* public final Class getClass()  
  Returns the runtime class of an object.
* public int hashCode()  
  Returns a hash code value for the object.
* public String toString()  
  Returns a string representation of the object.

The notify, notifyAll, and wait methods of Object all play a part in synchronizing the activities of independently running threads in a program, which is discussed in a later lesson and won’t be covered here. There are five of these methods:

* public final void notify()
* public final void notifyAll()
* public final void wait()
* public final void wait(long timeout)
* public final void wait(long timeout, int nanos)

String is immutable and final in Java. Java runtime maintains a [String pool](https://www.journaldev.com/797/what-is-java-string-pool) that makes it a special class.

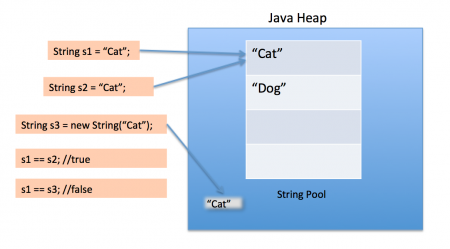
1. [String pool](https://www.journaldev.com/797/what-is-java-string-pool) is possible only because String is immutable in Java. This way Java Runtime saves a lot of [heap space](https://www.journaldev.com/4098/java-heap-space-vs-stack-memory) because different String variables can refer to the same String variable in the pool. If String would not have been immutable, then String interning would not have been possible because if any variable would have changed the value, it would have been reflected in the other variables too.
2. If String is not immutable then it would cause a severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](https://www.journaldev.com/741/java-socket-programming-server-client) host and port details passed as String. Since String is immutable, its value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.
3. Since String is immutable, it is safe for [multithreading](https://www.journaldev.com/1079/multithreading-in-java). A single String instance can be shared across different threads. This avoids the use of synchronization for thread safety. Strings are implicitly thread-safe.
4. Strings are used in [java classloader](https://www.journaldev.com/349/java-classloader) and immutability provides security that correct class is getting loaded by Classloader. For example, think of an instance where you are trying to load java.sql.Connection class but the referenced value is changed to myhacked.Connection class that can do unwanted things to your database.
5. Since String is immutable, its **hashcode** is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for the key in a Map and its processing is faster than other HashMap key objects. This is why String is the most widely used as HashMap keys.

String data type is not a good option to hold password information in the application. Since String is immutable in nature and there is no any way to change its value ( apart from Java reflection ). So you can not erase or override its contents and it will be saved in “String pool” and if anyone gets the memory dump then password might get exposed. It could lead to a security breach.

Instead of String, We can use the char array as you can easily erase or overwrite its contents after

Java 7 onwards string pool has been moved to heap area and will be available for GC

<https://www.journaldev.com/797/what-is-java-string-pool>



String pool helps in saving a lot of space for Java Runtime although it takes more time to create the String.

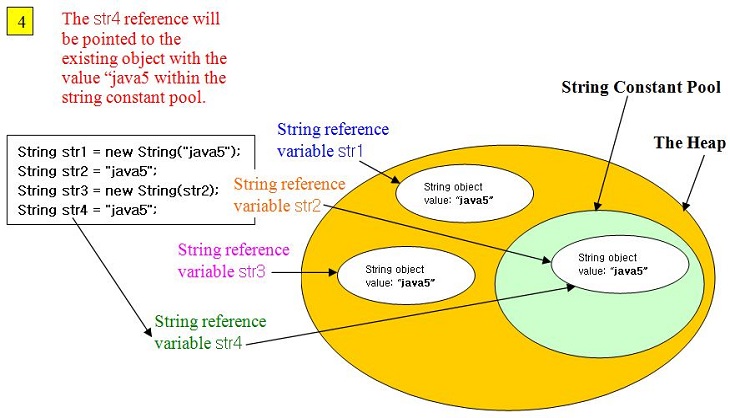
When we use double quotes to create a String, it first looks for String with the same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference.

String is immutable. When you call toUpperCase(), a new string is created and assigned. The original string remains unchanged.

## How many Strings are getting Created in the String Pool?

String [literals](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html) will go into **String Constant Pool**.

The below snapshot might help you to understand it visually to remember it for longer time.



Object creation line by line:

String str1 = new String("java5");

Using string literal "java5" in the constructor, a new string value is stored in string constant pool. Using new operator, a new string object is created in the heap with "java5" as value.

String str2 = "java5"

Reference "str2" is pointed to already stored value in string constant pool

String str3 = new String(str2);

A new string object is created in the heap with the same value as reference by "str2"

String str4 = "java5";

Reference "str4" is pointed to already stored value in string constant pool

Total objects : Heap - 2, Pool - 1

String abc = "a"+"b"+"c";

//Here how many objects created in string pool memory?

//it’s FIve

//a, b, c, ab, abc

package testPackage;

class Test {

public static void main(String[] args) {

String hello = "Hello", lo = "lo";

System.out.print((hello == "Hello") + " ");

System.out.print((Other.hello == hello) + " ");

System.out.print((other.Other.hello == hello) + " ");

System.out.print((hello == ("Hel"+"lo")) + " ");

System.out.print((hello == ("Hel"+lo)) + " ");

System.out.println(hello == ("Hel"+lo).intern());

}

}

class Other { static String hello = "Hello"; }

and the compilation unit:

package other;

public class Other { public static String hello = "Hello"; }

produces the output:

true true true true false true

final String a = "a";

final String b = "b";

String ab = "ab";

System.out.println(a+b == ab);  // returns true: Constant expression.

String a2 = "a";

String b2 = "b"

System.out.println(a2 + b2 = ab); // returns false: Not a constant expression

<https://www.journaldev.com/1562/flyweight-design-pattern-java>

Flyweight design pattern is used when we need to create a lot of Objects of a class. Since every object consumes memory space that can be crucial for low memory devices, such as mobile devices or embedded systems, flyweight design pattern can be applied to reduce the load on memory by sharing objects.

Before we apply flyweight design pattern, we need to consider following factors:

* The number of Objects to be created by application should be huge.
* The object creation is heavy on memory and it can be time consuming too.
* The object properties can be divided into intrinsic and extrinsic properties, extrinsic properties of an Object should be defined by the client program.

All the [wrapper classes](https://www.journaldev.com/1002/wrapper-class-in-java) valueOf() method uses cached objects showing use of Flyweight design pattern. The best example is [Java String](https://www.journaldev.com/1321/java-string-interview-questions-and-answers) class [String Pool](https://www.journaldev.com/797/what-is-java-string-pool) implementation.

**Flyweight Design Pattern Important Points**

1. In our example, the client code is not forced to create object using Flyweight factory but we can force that to make sure client code uses flyweight pattern implementation but its a complete design decision for particular application.
2. Flyweight pattern introduces complexity and if number of shared objects are huge then there is a trade of between memory and time, so we need to use it judiciously based on our requirements.
3. Flyweight pattern implementation is not useful when the number of intrinsic properties of Object (instance variables used in class) is huge, making implementation of Factory class complex.

**final** **int** a;

**final** **int** b;

StringSample(**int** a,**int** b){

**this**.a=a;

**this**.b=b; }

//if final variables are not initialized in constructor it throws compile time error. It must initialize at declaration or in constructor

Immutable objects are always thread-safe

<https://dzone.com/articles/do-immutability-really-means>

"An immutable object is an object that will not change its state after its construction"

"Use an immutable object, it is thread-safe! But take care, you have to make your immutable object thread-safe!"

You have to make sure that the object fields of your object don't change their state, which is sometimes impossible.

The immutable class in itself is thread safe but reference of immutable class is not thread safe.

public class ThreadString extends Thread {

String str = "ABC";

public void run() {

str = "abc";

}

}

Above code is not thread safe as str is not private

<https://www.journaldev.com/129/how-to-create-immutable-class-in-java>

**Immutable objects** are instances whose state doesn’t change after it has been initialized. For example, [String](https://www.journaldev.com/16928/java-string) is an immutable class and once instantiated its value never changes.

An immutable class is good for caching purpose because you don’t need to worry about the value changes.

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.

public HashMap<String, String> getTestMap() {

//return testMap; return

(HashMap<String, String>) testMap.clone(); }

<https://www.journaldev.com/1425/builder-design-pattern-in-java>

## Builder Design Pattern

Builder pattern was introduced to solve some of the problems with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.

There are three major issues with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.

1. Too Many arguments to pass from client program to the Factory class that can be error prone because most of the time, the type of arguments are same and from client side its hard to maintain the order of the argument.
2. Some of the parameters might be optional but in [Factory pattern](https://www.journaldev.com/1392/factory-design-pattern-in-java), we are forced to send all the parameters and optional parameters need to send as NULL.
3. If the object is heavy and its creation is complex, then all that complexity will be part of Factory classes that is confusing.

## Builder Design Pattern in Java

Let’s see how we can implement builder design pattern in java.

1. First of all you need to create a [static nested class](https://www.journaldev.com/996/java-inner-class) and then copy all the arguments from the outer class to the Builder class. We should follow the naming convention and if the class name is Computer then builder class should be named as ComputerBuilder.
2. Java Builder class should have a public constructor with all the required attributes as parameters.
3. Java Builder class should have methods to set the optional parameters and it should return the same Builder object after setting the optional attribute.
4. The final step is to provide a build() method in the builder class that will return the Object needed by client program. For this we need to have a private constructor in the Class with Builder class as argument.

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

Well, nothing is wrong, depending on how you use it. For example, if you initialize the HashMap just by one thread and then all threads are only reading from it, then it’s perfectly fine. One example of this is a Map which contains configuration properties.

The real problem starts when at-least one of that thread is updating HashMap i.e. adding, changing or removing any key value pair. Since put() operation can cause re-sizing and which can further lead to infinite loop, that’s why either you should use [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) or [ConcurrentHashMap](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html), later is better.

**Question 2. Does overriding the hashCode() method have any performance implication?** ([answer](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html))

This is a good question and open to all, as per my knowledge a poor hash code function will result in the frequent collision in HashMap which eventually increases the time for adding an object into Hash Map.

From Java 8 onwards though, collision will not impact performance as much as it does in earlier versions, because after a threshold the linked list will be replaced by the binary tree, which will give you O(logN) performance in the worst case, as compared to O(n) of linked list.

**Question 3: Do all properties of an Immutable Object need to be final?** ([answer](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))

Not necessarily, as stated above you can achieve same functionality by making the member non-final but private and not modifying them except in a constructor. Don’t provide setter methods for them and if it is a mutable object, then don’t ever leak any reference for that member.

Remember making a reference variable final, only ensures that it will not be reassigned to a different value, but you can still change individual properties of object, pointed by that reference variable.

**Question 4: How does the substring() method inside String works?**

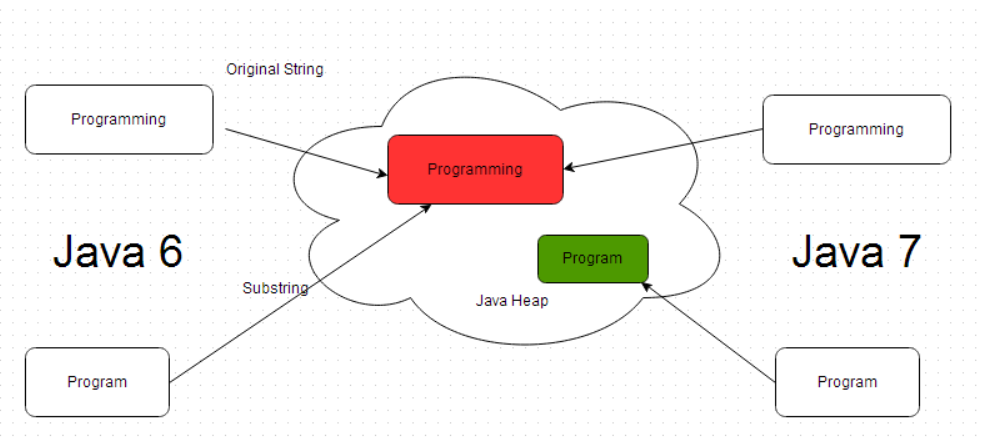
“Substring creates a new object out of source string by taking a portion of original string”.

**Can you explain what does substring do?** My friend got an opportunity to show off his talent, and how much he knows about Java API;  He said substring method is used to get parts of String in Java. It’s defined in java.lang.String class, and it's an [overloaded method](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html). One version of substring method takes just beginIndex, and returns part of String started from beginIndex till end, while other takes two parameters, beginIndex and endIndex, and returns part  of String starting from beginIndex to endIndex-1. He also stressed that every time you call  substring() method in Java,  it will return a new String because [String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html).

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

If you look substring method inside String class, you will figure out that it calls String (int offset, int count, char value []) [constructor](http://javarevisited.blogspot.com/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) to create new String object. What is interesting here is, value[], which is the same character array used to represent original string. So **what's wrong with this**?

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

[](https://2.bp.blogspot.com/-bpVcO5iKthk/VAhfFQLeciI/AAAAAAAAB1Y/hzrzYa2ZmlI/s1600/How+SubString+works+in+Java.png)

## How SubString in Java works

in case you are creating substring of significant longer String. Simple solution is to trim the string, and keep size of character array according to length of substring. Luckily java.lang.String has constructor to do this, as shown in below example.

*// comma separated stock symbols from NYSE*

String listOfStockSymbolsOnNYSE = getStockSymbolsForNYSE();

*//calling String(string) constructor*

String apple = **new** String(

listOfStockSymbolsOnNYSE.substring(appleStartIndex, appleEndIndex)

);

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

**public** String(String original) {

...

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

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

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

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

**int** off = original.offset;

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

} **else** {

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

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

v = originalValue;

}

...

}

Another way to solve this problem is to call intern() method on substring, which will than fetch an [existing string](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html) from pool or add it if necessary. Since the String in the pool is a real string it only take space as much it requires. It’s also worth noting that sub-strings are not internalized, when you call intern() method on original String

### Update:  This issue was actually a bug http://bugs.sun.com/view\_bug.do?bug\_id=6294060,  which is fixed in substring implementation of Java 7. Now, Instead of sharing original character array, substring method creates a copy of it. In short, substring method only retains as much data, as it needed.  [java.lang.String class](http://javarevisited.blogspot.com/2013/07/java-string-tutorial-and-examples-beginners-programming.html) has also grown into some change in Java 1.7 version and offset and count variable which is used to track positions are removed from String. This may save some bytes with each String instance, but not sharing original array makes substring perform linearly, as compared to constant time previously What is java.lang.OutOfMemoryError in Java

### [java.lang.OutOfMemoryError in Java, PermGen space or heap space](http://javarevisited.blogspot.com/2011/07/javalangunsupportedclassversionerror.html)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. 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**  
  
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**](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html), as shown in below example.  
  
**export JVM\_ARGS="-Xmx1024m -XX:MaxPermSize=256m"**

Another reason of "**java.lang.OutOfMemoryError: PermGen**" is memory leak through [Classloaders](http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html) 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*](http://javarevisited.blogspot.com/2011/08/increase-heap-size-maven-ant.html) 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**](http://www.eclipse.org/mat/) 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**.

## 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.

**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.

Use Visualvm

VisualVM is a powerful tool that provides a visual interface to see **deep** and **detailed** information about local and remote Java applications while they are running on a Java Virtual Machine (JVM). It utilizes and integrates some of the [command-line tools](https://stackify.com/top-command-line-tools/) that JDK provides and bundles them up to see the application within JVM; this bundle includes command-line tools jmap, jstack, jConsolem, jstat, and jinfo. All these tools are available in standard JDK distribution.

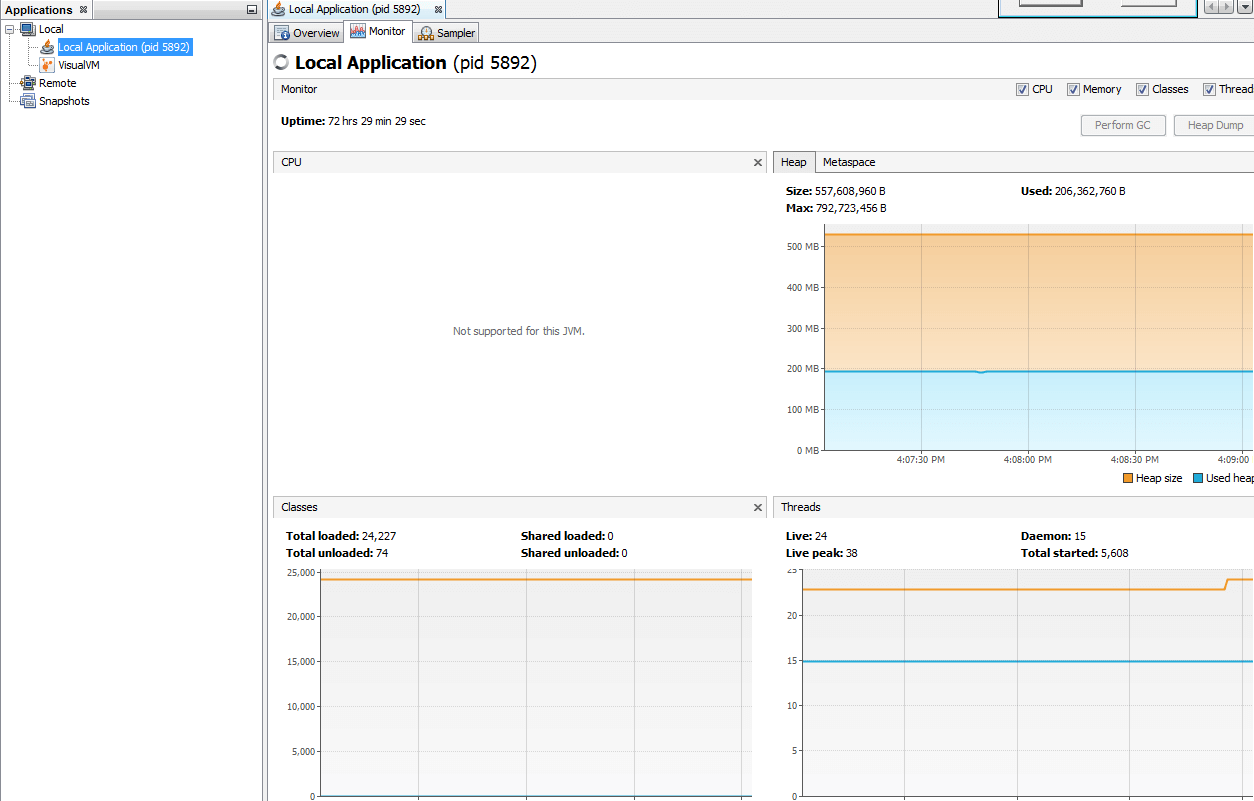
It helps the programmers and architects to track memory leaks, analyze the heap data, monitor the [garbage collector](https://stackify.com/what-is-java-garbage-collection/) and CPU profiling. It also helps to improve the application performance and ensure that memory usage is optimized. With features like thread analysis and head dump analysis, it is very handy in solving run-time problems.

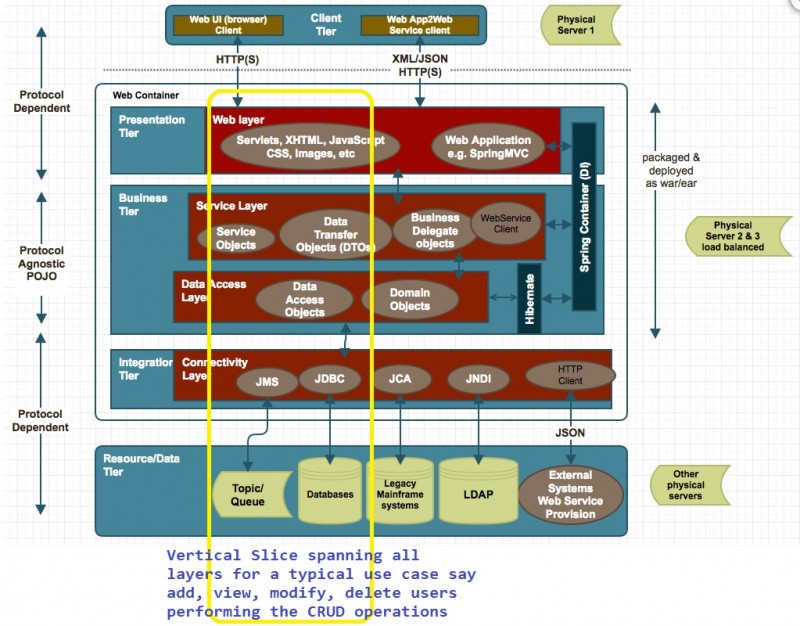
## Benefits

There are many important features that VisualVM supports, such as:

1. Visual interface for local and remote Java applications running on JVM.
2. Monitoring of application’s memory usage and application’s runtime behavior.
3. Monitoring of application threads.
4. Analyzing the memory allocations to different applications.
5. Thread dumps: very handy in case of deadlocks and race conditions.
6. Heap dumps: very handy in analyzing the heap memory allocation.

The important tab here is the “Monitor” tab. It gives you a detailed analysis of heap space and utilization, classes and threads. It can read and interpret binary heap dump files. You can either generate a heap dump, or you can read one you have generated outside this application (e.g. a dump created using kill-3 command on Unix/Linux server where the application is hosted).





Singleton Object creation

*public* *static* *Singleton* getInstanceDC() {

if (\_instance == **null**) { *// Single Checked*

*synchronized* (*Singleton*.class) {

 if (\_instance == **null**) { *// Double checked*

\_instance = new *Singleton*();

} } } return \_instance; }  
  
In the above code You need to not bear cost of synchronization all the time you call this method, while synchronization is only needed on first class, when Singleton instance is created.

Without volatile modifier it's possible for another thread in Java to see half initialized state of \_instance variable, but with volatile variable guaranteeing happens-before relationship, all the write will happen on volatile \_instance before any read of \_instance variable.

By the way this is not the best way to create thread-safe Singleton, you can [use Enum as Singleton](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html), which provides inbuilt thread-safety during instance creation. Another way is to use static holder pattern.

For example see the below link

Read more: <https://javarevisited.blogspot.com/2014/05/double-checked-locking-on-singleton-in-java.html#ixzz5iFRQdlOG>

1) Two unequal objects may return the same hashcode.

2) When two objects are equal by equals(), they must have the same hashcode.

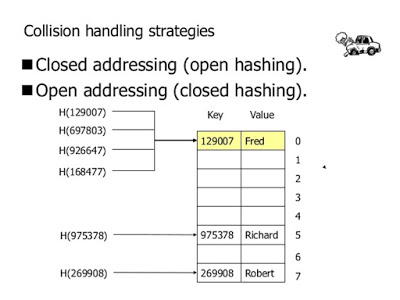
**1) How does get(Key key) method works internally in HashMap, and Hashtable in Java?**

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

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

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

Things get little tricky, when Interviewer ask the second question, **What happens if two keys have the same hashCode?** If multiple keys have the same hashCode, then during put() operation collision had occurred, which means multiple Entry objects stored in a bucket location. Each Entry keeps track of another Entry, forming a [linked list data structure](http://javarevisited.blogspot.com/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html) there.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https://www.pluralsight.com/courses/java-fundamentals-collections)

This has also changed from Java 8, where after a threshold is crossed then a binary tree is used instead of linked list to lift the worst case performance from O(n) to O(logN). You can see [how HashMap and LinkedHashMap handle collision in Java](http://javarevisited.blogspot.com/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html) to learn more about this change.   
  
Now, if we need to retrieve value object in this situation, following steps will be followed :

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

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

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

1. *AUTO*: Hibernate selects the generation strategy based on the used dialect,
2. *IDENTITY*: Hibernate relies on an auto-incremented database column to generate the primary key,
3. *SEQUENCE*: Hibernate requests the primary key value from a database sequence,
4. *TABLE*: Hibernate uses a database table to simulate a sequence.

**Should we make our exceptions checked or unchecked?**  
  
If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception

<https://www.geeksforgeeks.org/reverse-an-array-in-java/>

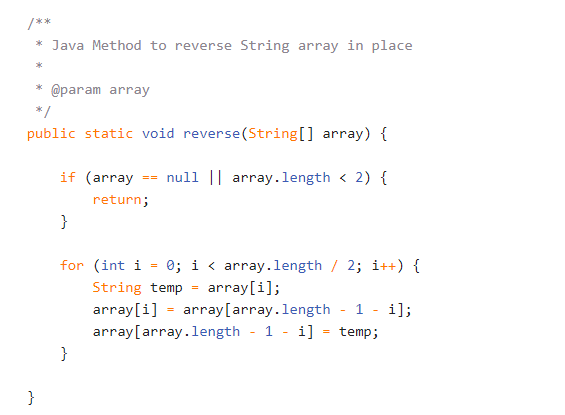
## Array vs linked list in Java

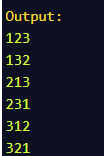
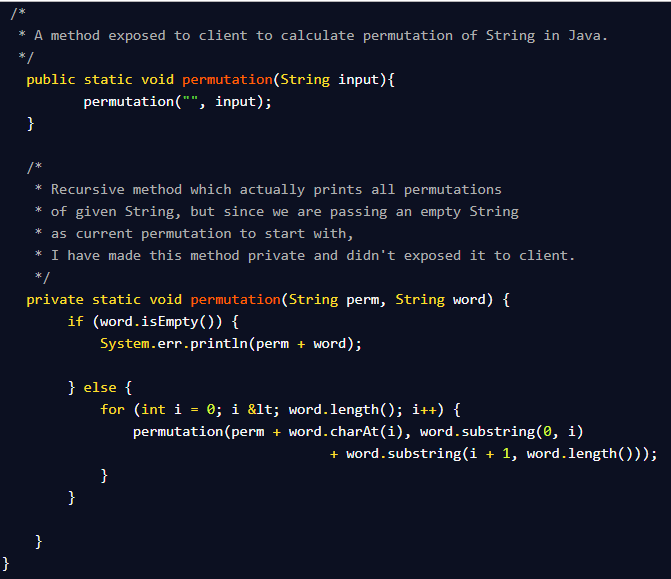
1. First and major difference between linked list and array data structure is that former doesn't support random access, while later support random access. linked list is sequential, in order to retrieve an element, you need to traverse till that, while if you know index, you can retrieve an element from array very quickly, because it doesn't involved traversal.  
  
2. Second major difference between array and linked-list data structure is that, array needs contiguous memory allocation, which may result in [java.lang.OutOfMemoryError: Java Heap Space](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html) if there is not enough contiguous ( a big chunk) of memory in Java Heap. On the other hand, linked list is distributed data structure, it's element are scattered over heap and doesn't need a contiguous memory allocation. This makes linked list ideal, if you have scattered memory.

3. Third major difference is fixed length, array is a fixed length data structure, you provide length or size of array at the time of creation, later you can not modify that size. On the other hand, linked list is dynamic data structure, it can grow and doesn't required size to be specified at the time of creation, because each node keep tracks of other.  
  
4. It's easy to insert and delete elements from linked list than array, especially inserting element at beginning of linked list, and deleting element from end of linked list is O(1) operation. On the other hand array is fixed length data structure, so memory is allocated during initialization, and doesn't really change due to addition and removal of elements.  Though you can set a particular index null, to cut the reference count of that object.

5. Array is ideal for implementing fast caches e.g. [HashMap or Hashtable](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), which requires constant time retrieval e.g. Map data structure provides O(1) performance for get(Key key) operation, while linked list based structure provides liner performance i.e. O(n) for retrieval operation, where n is the number of elements in linked list.

6. Array can be one or multi-dimensional, while linked list can be singly, doubly or circular linked list. Two dimensional array are most common in multi-dimensional and used to represent matrix in Java. You can use two dimensional array to represent a plain of x,y coordinates, frequently used in Game programming. Java programming language provides support for creating array at syntax level, it supports both single and multidimensional array. Java API also provides a class called java.util.LinkedList, which is an implementation of doubly linked list data structure.





<https://beyondcorner.com/create-own-marker-interface-in-java/>

