Spring Boot.

Spring Security

Immutable class

**Collection Questions**

Concurrent Hashmap,

Question: HashSet and Hashmap

Question where use comparable and where use comparator.

Q: Arralist and vector?

Q: how to store object in hashmap as a key?

**Design Pattern:**

**OOPS**

Abstract Class And Interface.

Q:class and interfaces in java?

Q: String & string buffer, String Builder

**Q: Inheritance and how to achieve it and types of Inhiritance?**

**Q: All Oops concepts?**

Q: Why use inheritance in java

* For Method Overriding (so runtime polymorphism can be achieved).
* For Code Reusability.

**Q.Difference between hashcode() and equals methods**

Java.lang.object has two very important methods defined:

* public boolean equals(Object obj) and
* public int hashCode().

**Multithreading:**

Atomic variables

Ques: Race Condition?

**Q. why default methods in interface and then what the difference between interface and abstract class.**

**Q: create custom unchecked exception?**

**Question: what is Annotaion?**

**Questtion: what is StackOverflow and how to create stackOverflow?**

**Question: why hashmap contains only single null key?**

**Question: In Maven – artifact Id, version , snapshot?**

**Web:**

**Ques: Difference between http put and http post.**

**Spring**

Question: Spring modules?

**Hibernate**

**Q: Second level cahche in hibernate and how to implement?**

**Q: joins in hibernate?**

**Ques: difference persistent states of hibernate?**

Q: how to composite key create in Hibernate?

Ans: By @EmbeddedId,@Embeddable

**Q: what is composite key:**

* composite primary key in the database is the group of column whose value together make unique value.
* If unique key is contained in single column than @Id is used and when the primary key consists of multiple column, these columns need to be group into a different object.
* To group multiple column in a class, some rules have to be followed.

**Rules to create class with composite key**

* class must be a public class.
* must have a default constructor.
* Must implement serialize interface.
* Class not require primary key its own.
* Class must have equals and hashcode method.

**Spring Boot**

**Question: what is version ?**

**Question: @ModelAttribute & @path variable?**

**Spring annotations**

**Q:Spring boot questions**

**1.Spring Initializer?**

**2. Actuator and CLI?**

**4.How activate login in spring boot and LOGGING level in Spring Boot?**

**(Logging level describe in application properties . levels are {WARN,DEBUG,FATAL,MESSAGE})**

**5.what is microservices and advantages and disadvantages?**

**6.where you use Spring boot in you project?**

Ques: Scope of bean ;

Question: What is DI & IOC.

**-------------------------------------------------**

**Database:**

**Q: Normalization and denormalization?**

**Q: Singleton class  
Q:Binary search tree**

**5  
                    /      \  
                  3          7  
                /           / \  
              2.        6.      9  
  
If you have a pointer at 2 than return the next node on the same level i.e.., 6.  
Ex.       2. Will return 6  
       6 will return 9  
5. Immutable class implementation?  
7. Reverse a linklist in java implementation.  
8. Find Maximum length of a binary tree.  
9. Arraylist and linkedlist difference.  
10. Abstract and interface difference and what to use when.  
11. Hashmap implementation.  
12. Thread life cycle implementation.  
Ques. Difference between hashmap and concurrent hashmap?  
Ques. Java 8 (Functional Interface, Static and default method interface, )**

**Ques: Question: Internal working of hashset?**

**Ques: compare Integer i1=new Integer(10) and Integer i2=new Integer(10)**

**(i1 == i2) and i1.equals(i2)**

**Q: Two classes has main method how to call one main from others?**

**Q. custom exception**

**Question: how to get value from url in rest controller?**

**Question: Exception Handling super & subclass method?**

**Question: custom exception?**

**Question: how to process image in spring from jsp to controller?**

**Question: Encapsulation?**

**Question: how to run multithread to print value 111222333444555?**

**Question: difference between throw and throws?**

**Question: what is java action tags and difference between include and directive tags?**

**Question hashmap internal work?**

**Question: Comparabale and comparator diiference?**

**==============NIIT================**

**Question: difference between get and load?**

**Question: difference b/w servlet and filter?**

**Question: how to filter ips based on user and password?**

**Q: describe current project?**

**-----------------------------------------**

**Q.Where we use ArrayList and where we use a linked list.**

Both LinkedList and ArrayList require O(n) time to find if an element is present or not. However we can do Binary Search on ArrayList if it is sorted and therefore can search in O(Log n) time.

|  |  |  |
| --- | --- | --- |
| No. | ArrayList | LinkedList |
| 1) | ArrayList uses a dynamic array. | LinkedList uses a doubly linked list. |
| 2) | Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |
| 4) | ArrayList provides random access. | LinkedList does not provide random access. |
| 5) | ArrayList takes less memory overhead as it stores only object | LinkedList takes more memory overhead, as it stores the object as well as the address of that object. |
| 6) | An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |

**Q1) What is an immutable class?**

**Ans)** Immutable class is a class which once created, it’s contents can not be changed. Immutable objects are the objects whose state can not be changed once constructed. e.g. String class

**Q5) What are the advantages of immutability?**

* Immutable objects are automatically thread-safe, the overhead caused due to use of synchronisation is avoided.
* Once created the state of the immutable object can not be changed so there is no possibility of them getting into an inconsistent state.
* The references to the immutable objects can be easily shared or cached without having to copy or clone them as there state can not be changed ever after construction.
* The best use of the immutable objects is as the keys of a map.

**Q2) How to create an immutable class?**

**Ans)** To create an immutable class following steps should be followed:

1. Create a final class.
2. Set the values of properties using constructor only.
3. Make the properties of the class final and private
4. Do not provide any setters for these properties.
5. If the instance fields include references to mutable objects, don't allow those objects to be changed:
   1. Don't provide methods that modify the mutable objects.
   2. Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

<http://java-questions.com/ImmutableClass-interview-questions.html>

**Q. I have created an immutable class with a date field. How do I ensure that even the date field is immutable, since even if you make date field final you can still assign a different value to it later?**

Lets apply all above rules for immutable classes and make a concrete class implementation for **immutable class in Java**.

|  |  |
| --- | --- |
| ImmutableClass.java | |
| import java.util.Date;  /\*\*  \* Always remember that your instance variables will be either mutable or immutable.  \* Identify them and return new objects with copied content for all mutable objects.  \* Immutable variables can be returned safely without extra effort.  \* \*/  public final class ImmutableClass  {    /\*\*    \* Integer class is immutable as it does not provide any setter to change its content    \* \*/      private final Integer immutableField1;      /\*\* \* String class is immutable as it also does not provide setter to change its content  \* \*/      private final String immutableField2;   /\*\*    \* Date class is mutable as it provide setters to change various date/time parts\*\*/      private final Date mutableField;      //Default private constructor will ensure no unplanned construction of class      private ImmutableClass(Integer fld1, String fld2, Date date)      {          this.immutableField1 = fld1;          this.immutableField2 = fld2;          this.mutableField = new Date(date.getTime());      }        //Factory method to store object creation logic in single place      public static ImmutableClass createNewInstance(Integer fld1, String fld2, Date date)      {          return new ImmutableClass(fld1, fld2, date);      }        //Provide no setter methods        /\*\*      \* Integer class is immutable so we can return the instance variable as it is      \* \*/      public Integer getImmutableField1() {          return immutableField1;      }       /\*\*      \* String class is also immutable so we can return the instance variable as it is      \* \*/      public String getImmutableField2() {          return immutableField2;      }       /\*\*      \* Date class is mutable so we need a little care here.      \* We should not return the reference of original instance variable.      \* Instead a new Date object, with content copied to it, should be returned.      \* \*/      public Date getMutableField() {          return new Date(mutableField.getTime());      }       @Override      public String toString() {          return immutableField1 +" - "+ immutableField2 +" - "+ mutableField;      }  } | |

===============================================================

**Q.Difference between Hashmap and Concurrent Hashmap?**

[HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) is the Class which is under Traditional Collection and ConcurrentHashMap is a Class which is under Concurrent Collections, apart from this there are various differences between them which are:

* HashMap is non-Synchronized in nature i.e. HashMap is not Thread-safe whereas ConcurrentHashMap is Thread-safe in nature.
* HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously. But ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.
* While one thread is Iterating the HashMap object, if other thread try to add/modify the contents of Object then we will get Run-time exception saying **ConcurrentModificationException**.Whereas In ConcurrentHashMap we wont get any exception while performing any modification at the time of Iteration.

**Question: TreeSet< Student > s = new TreeSet<Student>();**

**TreeSet<StringBuffer> s = new TreeSet<StringBuffer>();**

Ans:

1. An object is said to be comparable if and only if the corresponding class implements **Comparable interface**.
2. **String** class and all **Wrapper** classes already implements Comparable interface but StringBuffer class doesn’t implements Comparable interface.Hence we got *ClassCastException*in the above example.

### Ques: Spring Bean Configuration: How many type to create Spring bean?

Spring Framework provides three ways to configure beans to be used in the application.

1. **Annotation Based Configuration** – By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.
2. **XML Based Configuration** – By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.
3. **Java Based Configuration** – Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

# Question: Which collection classes are synchronized or thread-safe ?

**ConcurrentHashMap, SynchronizedHashMap,** Stack, Properties, Vector,Hashtable

BlockingQueue(I), ConcurrentMap, ConcurrentNavigableMap

**Quesues: Type of classes in java?**

* Ans: POJO Class
* Static Class
* Concrete Class
* Abstract Class
* Final Class
* Inner Class(Nested Inner class,Method Local inner classes, Anonymous inner classes, Static nested classes)

**Question: @RequestParam**  **& @path variable?**

Even though both @RequestParam and @PathVariable is used to extract values from the HTTP request, there is a subtle difference between them, which makes them a useful question from an interview and [spring certification](http://javarevisited.blogspot.sg/2017/06/2-books-to-prepare-for-spring-certification-exam.html#axzz4pbqSY8Ua) point of view.

We'll examine the subtle difference between @RequestParam and @PathVaraible in this article. As the name suggests, @RequestParam is used to get the request parameters from URL, also known as query parameters, while @PathVariable extracts values from URI.

For example, if the incoming HTTP request to retrieve a book on topic "Java" is http://localhost:8080/shop/order/1001/receipts?date=12-05-2017, then you can use the @RequestParam annotation to retrieve the query parameter date and you can use @PathVariable to extract the orderId i.e. "1001" as shown below:  
  
@RequestMapping(value="/order/{orderId}/receipts", method = RequestMethod.GET)  
public List listUsersInvoices(@PathVariable("orderId") int order,  
 @RequestParam(value = "date", required = false) Date dateOrNull) {  
...  
}  
  
The required=false denotes that the query parameter can be optional, but the [URL](http://www.java67.com/2013/01/difference-between-url-uri-and-urn.html) must have the same URI.

**Question: difference 401 and 403?**

**Ans: 401 :**  Unauthorized response should be used for missing or bad authentication,

**and 403: The server understood the request, but is refusing to fulfill it.?**

401 is Authentication error, 403 is Authorization error.

**Question: how many way to create bean object (Spring Bean Configuration)?**

Ans: Spring Framework provides three ways to configure beans to be used in the application.

1. **Annotation Based Configuration** – By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.
2. **XML Based Configuration** – By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.
3. **Java Based Configuration** – Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

**----------------------- NIIT 2:------------------**

**SDLC, Triggers**

**-------HCL-----------------------**

**Q. @Controller vs RestController**

**Q. @EnableAutoConfiguration vs @SpringBootApplication**

# Q. Can we change the override method scope or [Changing visibility of method in inherited class](https://stackoverflow.com/questions/45896798/changing-visibility-of-method-in-inherited-class)?

Ans: It is not possible to reduce the visibility of a method in subclass. The subclass need to be a valid instance of the base class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parent Class Scope | public | protected | default | Private |
| Public | Y | N | N | N |
| Protected | Y | Y | N | N |
| default | Y | Y | Y | N |
| Private (Not access) |  |  |  |  |

**Q. mergesort [\_2\_|\_3\_|\_5\_|\_2\_|\_6\_\_7] Algo**

**Ques: difference b/w** CrudRepository and @JPARepository

**ITC Interview**

# Q. [JAVA Object/String method overload [duplicate]](https://stackoverflow.com/questions/17669804/java-object-string-method-overload)

Class1 a=new Class1();

a.method(null) // which method is called

public class Class1 {

public void method(Object obj){

System.out.println("Object");

}

public void method(String str){

System.out.println("String");

}

public static void main(String... arg){

new Class1().method(null);

}

}

Whenever more than one overloaded methods can be applied to the argument list, the most specific method is used.

In this case either of the methods can be called when passing null, since the "null type" is assignable to both Object and to String. The method that takes String is more specific so it will be picked.

**.**

**---------------------------------------------**

**WildTeck Interview**

**Q: Classloader:** The **Java ClassLoader** is a part of the [**Java Runtime Environment**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/) that dynamically loads Java classes into the [**Java Virtual Machine**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/).

The Java run time system does not need to know about files and file systems because of classloaders.

[Java classes](https://www.geeksforgeeks.org/classes-objects-java/) aren’t loaded into memory all at once, but when required by an application. At this point, the **Java ClassLoader** is called by the **JRE** and these ClassLoaders load classes into memory dynamically.

**Types of ClassLoaders in Java**

Not all classes are loaded by a single ClassLoader. Depending on the type of class and the path of class, the ClassLoader that loads that particular class is decided. To know the ClassLoader that loads a class the ***[getClassLoader()](https://www.geeksforgeeks.org/java-lang-class-class-java-set-1/)*** method is used. All classes are loaded based on their names and if any of these classes are not found then it returns a **[NoClassDefFoundError](https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/)** or **[ClassNotFoundException](https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/)**.

A Java Classloader is of **three types**:

1. **BootStrap ClassLoader:** A Bootstrap Classloader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class.
2. **Extension ClassLoader:** The Extension ClassLoader is a child of Bootstrap ClassLoader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from ***jre/lib/ext*** directory or any other directory pointed by the system property ***java.ext.dirs***.
3. **System ClassLoader:** An Application ClassLoader is also known as a System ClassLoader. It loads the Application type classes found in the environment variable ***CLASSPATH, -classpath or -cp command line option***. The Application ClassLoader is a child class of Extension ClassLoader.

## Difference between synchronization and volatile keyword

Volatile keyword is not a substitute of synchronized keyword, but it can be used as an alternative in certain cases. There are the following differences are as follows:

|  |  |
| --- | --- |
| **Volatile Keyword** | **Synchronization Keyword** |
| Volatile keyword is a field modifier. | Synchronized keyword modifies code blocks and methods. |
| The thread cannot be blocked for waiting in case of volatile. | Threads can be blocked for waiting in case of synchronized. |
| It improves thread performance. | Synchronized methods degrade the thread performance. |
| It synchronizes the value of one variable at a time between thread memory and main memory. | It synchronizes the value of all variables between thread memory and main memory. |
| Volatile fields are not subject to compiler optimization. | Synchronize is subject to compiler optimization. |

**Question: Stack and Heap memory in java?**

## Java Heap Space: Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space.

Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

### Java Stack Memory: Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method.

Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.

As soon as method ends, the block becomes unused and become available for next method.  
Stack memory size is very less compared to Heap memory.

**Difference between Java Heap Space and Stack Memory**

* Based on the above explanations, we can easily conclude following differences between Heap and Stack memory.
* Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
* Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.
* Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.
* Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at Java Garbage Collection.
* Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.
* We can use -Xms and -Xmx JVM option to define the startup size and maximum size of heap memory. We can use -Xss to define the stack memory size.
* When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.
* Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

**public class Memory {**

**public static void main(String[] args) { // Line 1**

**int i=1; // Line 2**

**Object obj = new Object(); // Line 3**

**Memory mem = new Memory(); // Line 4**

**mem.foo(obj); // Line 5**

**} // Line 9**

**private void foo(Object param) { // Line 6**

**String str = param.toString(); //// Line 7**

**System.out.println(str);**

**} // Line 8**

**}**

**Below image shows the Stack and Heap memory with reference to above program and how they are being used to store primitive, Objects and reference variables.**

**java memory management, java heap space, heap vs stack, java heap, stack vs heap**

**Let’s go through the steps of execution of the program.**

**As soon as we run the program, it loads all the Runtime classes into the Heap space. When main() method is found at line 1, Java Runtime creates stack memory to be used by main() method thread.**

**We are creating primitive local variable at line 2, so it’s created and stored in the stack memory of main() method.**

**Since we are creating an Object in line 3, it’s created in Heap memory and stack memory contains the reference for it. Similar process occurs when we create Memory object in line 4.**

**Now when we call foo() method in line 5, a block in the top of the stack is created to be used by foo() method. Since Java is pass by value, a new reference to Object is created in the foo() stack block in line 6.**

**A string is created in line 7, it goes in the String Pool in the heap space and a reference is created in the foo() stack space for it.**

**foo() method is terminated in line 8, at this time memory block allocated for foo() in stack becomes free.**

**In line 9, main() method terminates and the stack memory created for main() method is destroyed. Also the program ends at this line, hence Java Runtime frees all the memory and end the execution of the program.**

# Q: Difference Between =, == And === In JavaScript

* = is used for assigning values to a variable in JavaScript.
* == is used for comparison between two variables irrespective of the datatype of variable.
* === is used for comparision between two variables but this will check strict type, which means it will check datatype and compare two values.

script type="text/javascript">

function Comparision() {

var number = 100; // Here number variable assigned using =

debugger;

if (number == 100) // Here Comparision between two values using ==. This will not check datatype irrespective of datatype it will do comparision

$("#lblMessage").text("Both are equal");

else

$("#lblMessage").text("Both are not equal");

if(number == "100") //Here Comparision between two values using ==. This will not check datatype irrespective of datatype it will do comparision

$("#lblMessage1").text("Both are equal");

else

$("#lblMessage1").text("Both are not equal");

}

</script>

**script type="text/javascript">**

**function Comparision() {**

**var number = 100; // Here number variable assigned using =**

**debugger;**

**if (number === 100) // Here Comparision between two values using ==. This will not check datatype irrespective of datatype it**

**will do comparision**

**$("#lblMessage").text("Both are equal");**

**else**

**$("#lblMessage").text("Both are not equal");**

**if (number === "100") // Here Comparision between two values using ==. This will not check datatype irrespective of datatype it will do comparision**

**$("#lblMessage1").text("Both are equal");**

**else**

**$("#lblMessage1").text("Both are not equal");**

**}**

**</script>**

**-------------------------------------------**

**Q:Spring Boot Framework has mainly four major Components.**

* Spring Boot Starters
* Spring Boot AutoConfigurator
* Spring Boot CLI
* Spring Boot Actuator
* Spring Initilizr
* Spring Boot IDEs

**Q: ClassNotFoundException vs. NoClassDefFoundError?**

* **ClassNotFoundException** and **NoClassDefFoundError** occur when a particular class is not found at runtime. However, they occur at different scenarios.

**ClassNotFoundException** is an exception that 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.

* **NoClassDefFoundError** is an error that occurs when a particular class is present at compile time, but was missing at run time.

|  |  |
| --- | --- |
| **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 runtime. |

Q. Doc-type in html

The <!DOCTYPE> declaration must be the very first thing in your HTML document, before the <html> tag.

The <!DOCTYPE> declaration is not an HTML tag; it is an instruction to the web browser about what version of HTML the page is written in.

In HTML 4.01, the <!DOCTYPE> declaration refers to a DTD, because HTML 4.01 was based on SGML. The DTD specifies the rules for the markup language, so that the browsers render the content correctly.

HTML5 is not based on SGML, and therefore does not require a reference to a DTD.

Q: How does a spring container work? // Spring Study.docs

Q: How many type of container in spring mvc?

Application context and bean factory

Q:Can we declare the main in super and sub class both when super class extends in child?

Q: String a1=”ABC”; String a2=”ABC”; String a3=new String(”ABC”);

HashMap hm = new HashMap(String,Integer);

Hm.put(a1,2);

Hm.put(a2,1);

Hm.put(a3,3);

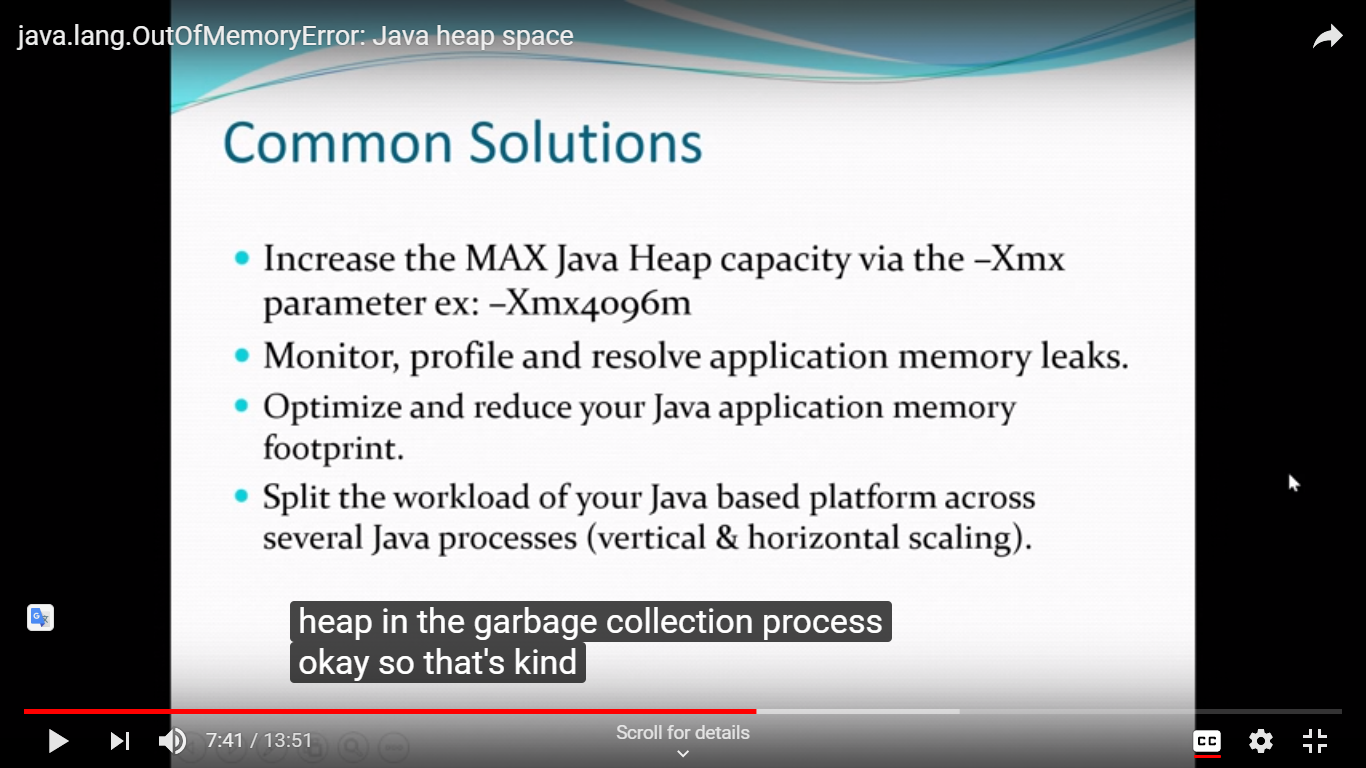
Sop(hm.size());

### Q: How to handle java.lang.OutOfMemoryError in Java on server?

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

Everyone in java development faces **java.lang.OutOfMemoryError** now and then, OutOfMemoryError in Java is one problem which is more due to system's limitation (memory) rather than due to programming mistakes in most cases though in certain cases you could have a **memory leak** which causing **OutOfMemoryError**. I have found that even though java.lang.OutOfMemoryError is quite common basic knowledge of its cause and solution is largely unknown among junior developers. Beginners books like [Head First Java](http://www.amazon.com/dp/0596009208/?tag=javamysqlanta-20) doesn't teach you much about how to deal with this kind of error. You need real experience dealing with production systems, handling a large number of user sessions to troubleshoot and fix performance issues like running out of memory.  
  
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**.



**Q: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](http://download.oracle.com/javase/6/docs/api/) is not very informative about this, though.

### Two Types of OutOfMemoryError in Java

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

Q: Shallow cloning and deep cloning?

Q: union and union All?

Q : Composite key

Question: Why use SerialVersionUID in Java? And difference between our own serializable id and jvm created?

Q: optional class?

Q:treemap internally data structure and working?

Q: Aggregation and composition are the part of Association? <https://www.youtube.com/watch?v=9nRblRcb35Y&t=585s>

Q: Why can we create constructor in abstract class and use of it?

Answer : A **constructor** in Java doesn't actually "build" the object, it **is** used to initialize fields. Imagine that your **abstract class** has fields x and y, and that **you** always want them to **be** initialized in a certain way, no matter what actual concrete subclass **is** eventually **created**.

Abstract class constructors will be called when its concrete subclass will be instantiated and this is used to initialize variables

Q: Default Array list size and how to increase?

**Q: Shutdown hookup?**

**Q:Stream.map() vs Stream.flatMap() in Java 8?**

* The function you pass to map() operation returns a single value.
* The function you pass to flatMap() opeartion returns a Stream of value.
* flatMap() is combination of map and flat operation.
* map() is used for transformation only, but flatMap() is used for both transformation and flattening.

**Ex:**   
public class Java8Demo {

public static void main(String args[]) {

// foods which helps in weight loss

List<String> loseWeight = new ArrayList<>();

loseWeight.add("avocados");

loseWeight.add("beans");

loseWeight.add("salad");

loseWeight.add("oats");

loseWeight.add("broccoli");

System.out.println("list of String : " + loseWeight);

// let's use map() method to convert list of weight

// lose food, which are String to list of ints

// which are length of each food String

List listOfInts = loseWeight.stream()

.map(s -> s.length())

.collect(Collectors.toList());

System.out.println("list of ints generate by map(): " + listOfInts);

// flatMap() example, let's first creat a list of list

List<List> listOfListOfNumber = new ArrayList<>();

listOfListOfNumber.add(Arrays.asList(2, 4));

listOfListOfNumber.add(Arrays.asList(3, 9));

listOfListOfNumber.add(Arrays.asList(4, 16));

System.out.println("list of list : " + listOfListOfNumber);

// let's use flatMap() to flatten this list into

// list of integers i.e. 2,4,3,9,4,16

List listOfIntegers = listOfListOfNumber.stream()

.flatMap( list -> list.stream())

.collect(Collectors.toList());

System.out.println("list of numbers generated by flatMap : " + listOfIntegers);

}}

Output

list of String : [avocados, beans, salad, oats, broccoli]

list of ints generate by map(): [8, 5, 5, 4, 8]

list of list : [[2, 4], [3, 9], [4, 16]]

list of numbers generated by flatMap : [2, 4, 3, 9, 4, 16]

**Q: how to filter odd no, of arraylist?**

**Q: Spring boot vs spring framework?**

**Q: How to enable security in Spring? Difference between Basic authentication and Form based ?**

**Q: DI and IOC?**

**Q: REST and Soap?**

**Q: why use Serial-version-id and difference between auto generated and manually created id?**

**Q: how to insert multiple same keys on hashmap?**

**Q: how to create custom arraylist?**

**Q: Custom Thread pool?**

**Q: marker interface and use?**

**Q: The Major Difference between User and Daemon Threads:**

|  |  |
| --- | --- |
| User Thread | Daemon Thread |
| User threads are foreground threads. | Daemon threads are background threads. |
| User threads are high priority threads. | Daemon threads are low priority threads. |
| Its life independent. | Its life depends on user threads. |
| JVM wait until user threads to finish their work. It never exit until all user threads finish their work. | The JVM will’t wait for daemon threads to finish their work. The JVM will exit as soon as all user threads finish their work. |
| JVM will not force to user threads for terminating, so JVM will wait for user threads to terminate themselves. | If all user threads have finished their work JVM will force the daemon threads to terminate |
| User threads are created by the application. | Mostly Daemon threads created by the JVM. |
| Mainly user threads are designed to do some specific task. | Daemon threads are design as to support the user threads. |

**Q: There are two classes A and b, b extends A how many table will created?**

Q: What is DLL?

 A **DLL** is a library that contains code and data that can be used by more than one program at the same time. For example, in Windows operating systems, the Comdlg32 **DLL** performs common dialog box related functions.

**Dynamic-link library** (**DLL**) is Microsoft's implementation of the shared library concept in the Microsoft Windows and OS/2 operating systems.

----------------------------

**Angular Question:**

* Observation
* Difference between Attribute Directive and Structure Directive.
* What is module?
* Lazy load.
* .remove and .map in jquery?
* Document.ready{} in jquery.
* JQuery chaning
* Rsjx
* Indexion in html 5
* Difference between Constructor and ngOnInit

=====================

IBM Question

* 1. How to communicate microservice
  2. Prime no.
  3. How to filter Strings Object from Arraylist is multiple type object are there.
  4. Autoconfiguration in Spring boot.
  5. Implement Logger in Spring(using slf4j.jar, log4j) (@Slf4j)

**NIC Interview**

* Configuration external jar class in project

Ans: (@SpringBootApplication(scanBasePackages = {"com.jkoder.shopping", "com.commons.shopping"}))

**Capgemini Interview:**

* 1. Why use setting.xml in spring boot.( https://maven.apache.org/settings.html)
  2. Student class object in hashmap.
  3. Get third even number from list using java 8.
  4. Lock on class level and method level in multi-threading
  5. How Spring Boot Application works internally?
  6. Find max occuring integer value from array .
  7. @ComponentScan

**Ques. How Spring Boot Application works internally?**

Ans. The application starts using the "main method" which calls the "run" method (static method of SpringApplication class). From the run method, the main application context kicks off which searches for the classes annotated with @Configuration and initializes all the declared beans in those configuration classes. Based on the scope of those beans, stores those beans in JVM, specifically in a space inside JVM which is known as IOC container. After the creation of all the beans, automatically configures the dispatcher servlet and registers the default handler mappings, messageConverts, etc.

Q: N+1 problem in Hibernate?

Q: blocking and non blocking request?(synchronized & not Syncronized)

Q: Hashcode and equals difference?

Q: Datastructure use in microservice or spring boot?

Q what is Autowiring?

Q: Hashmap With Employee Object and Sort by salary?

HashMap<Integer, Employee >

Q: Difference between Monolithic & Micro Service Application?

Q: Messeging Queue like Rabbit?

Q: Can we override the varags method

Void test(a1,a2,a3,……){}

Void test(a1,a2,…){}

**Solid Principle in java**

In Java, **SOLID principles** are an object-oriented approach that are applied to software structure design. These five principles have changed the world of object-oriented programming, and also changed the way of writing software. It also ensures that the software is modular, easy to understand, debug, and refactor.

The word SOLID acronym for:

* Single Responsibility Principle (SRP)
* Open-Closed Principle (OCP)
* Liskov Substitution Principle (LSP)
* Interface Segregation Principle (ISP)
* Dependency Inversion Principle (DIP)

**Memory MetaSpace in Java 8**

| PermGen | MetaSpace |
| --- | --- |
| It is removed from java 8. | It is introduced in Java 8. |
| PermGen always has a fixed maximum size. | Metaspace by default auto increases its size depending on the underlying OS. |
| Contiguous Java Heap Memory. | Native Memory(provided by underlying OS). |
| Inefficient garbage collection. | Efficient garbage collection. |

**Q. Singleton in angular**

**Q.Lazyloading in angular**

**Q. Difference Between map() And flatMap() In Java Stream**

**Q. Serializable example.**[**https://www.youtube.com/watch?v=Ki4c3hmS-6I**](https://www.youtube.com/watch?v=Ki4c3hmS-6I)

**Q. How to handle infinity in java.**

**Q. Propagation in @Transactional**

**Q. @EnableTransaction**

**Q. Stream.reduce() in Java ( https://www.geeksforgeeks.org/stream-reduce-java-examples/ )**

**Microservice Design Pattern architecture** <https://www.edureka.co/blog/microservices-design-patterns>;

### ****API Gateway Design Pattern****

### ****Circuit Breaker Pattern****

Hypothtical pattern

Saga Design pattern

**how to block multiple request from client in rest api in spring boot**

Rate Limiting a Spring API Using Bucket4j

<**dependency**> <**groupId**>com.github.vladimir-bukhtoyarov</**groupId**> <**artifactId**>bucket4j-core</**artifactId**> <**version**>4.10.0</**version**> </**dependency**>

@RestController

class AreaCalculationController {

private final Bucket bucket;

public AreaCalculationController() {

Bandwidth limit = Bandwidth.classic(20, Refill.greedy(20, Duration.ofMinutes(1)));

this.bucket = Bucket4j.builder()

.addLimit(limit)

.build();

}

//..

}