**CORE JAVA**

**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 field 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 Disruptor 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 it's guaranteed that any thread accessing that variable will see the value you wrote and anything you did before doing that right 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](http://java67.blogspot.sg/2012/08/what-is-volatile-variable-in-java-when.html))  
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 ensures 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](http://javarevisited.blogspot.sg/2015/07/how-to-use-wait-notify-and-notifyall-in.html))  
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 it's 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) {

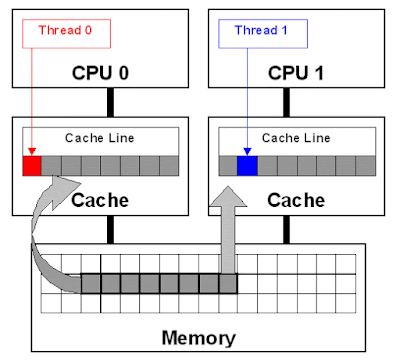
while (condition does not hold)

obj.wait(); // (Releases lock, and reacquires on wakeup)

... // Perform action appropriate to condition

}

See [Effective Java Item 69](https://javarevisited.blogspot.com/2017/10/effective-java-3rd-edition-coming-soon.html#axzz5Nf1J69kw) 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:

[](https://javarevisited.blogspot.com/2020/04/top-10-advanced-core-java-courses-for-experienced-developers.html)

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 microseconds or nanoseconds. [LMAX Disrupter](http://lmax-exchange.github.io/disruptor/) 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](http://javarevisited.blogspot.sg/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html))  
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](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html))  
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 managed 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](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html))  
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](http://javarevisited.blogspot.in/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))  
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](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html))  
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](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))  
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

Now, let's take a look at Java Interview questions which are based upon core Java and basic Java concepts like data types int, float, long, double and conditional and logical operator including the ternary operator

**17) What is the right data type to represent a price in Java?**([answer](http://javarevisited.blogspot.sg/2012/02/java-mistake-1-using-float-and-double.html))  
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](http://javarevisited.blogspot.sg/2014/08/2-examples-to-convert-byte-array-to-String-in-Java.html))  
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](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html))  
  
  
**22) Which class contains clone method? Cloneable or Object?** ([answer](http://javarevisited.blogspot.sg/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html))  
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](http://java67.blogspot.com/2014/11/how-to-convert-double-to-long-in-java-example.html))  
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](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html))  
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](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html))  
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 detailed answer for more explanation and discussion.  
  
**30) What is constructor chaining in Java?** ([answer](http://java67.blogspot.sg/2012/12/how-constructor-chaining-works-in-java.html))  
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

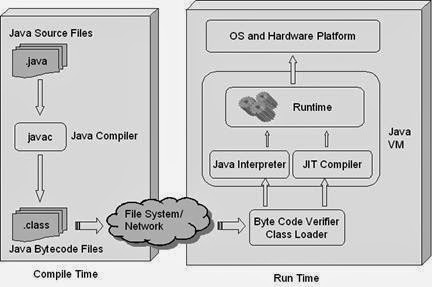
In last a couple of years I have seen increased focus on JVM internal and Garbage collection tuning, monitoring Java application, dealing with Java performance issues on various Java interviews. This is actually become the prime topic for interviewing any experienced Java developer for senior position 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 at least one book mentioned in my list of [Java Performance books](http://javarevisited.blogspot.com/2014/07/top-5-java-performance-tuning-books.html). 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](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html))  
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](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html))  
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](http://javarevisited.blogspot.com/2012/06/what-is-xxusecompressedoops-in-64-bit.html))  
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.

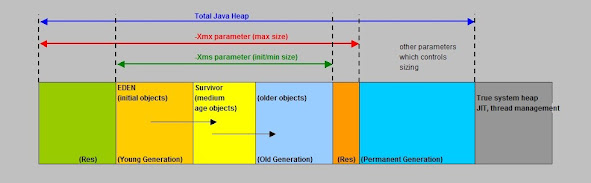
This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP.  
  
  
**37) How do you find if JVM is 32-bit or 64-bit from Java Program?**([answer](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html))  
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](http://javarevisited.blogspot.sg/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html))  
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](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html))  
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.

[](https://javarevisited.blogspot.com/2018/07/top-5-websites-to-learn-coding-in-java.html)

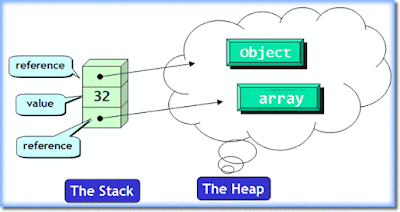
**40) Explain Java Heap space and Garbage collection?** ([answer](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html))  
When a Java process is started using java command, memory is allocated to it. Part of this memory is used to create heap space, which is used to allocate 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.

[](https://javarevisited.blogspot.com/2019/04/top-5-courses-to-learn-jvm-internals.html)

**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](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html))  
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.

[](https://medium.com/javarevisited/top-5-java-online-courses-for-beginners-best-of-lot-1e1e240a758)

## Basic Java concepts Interview Questions

In this part, we will take a look at various Java keywords, and concepts like equals and hashcode, compile time and runtime behavior and much which are very important for beginner level Java interviews.

**44) What's the difference between "a == b" and "a.equals(b)"?** ([answer](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html))  
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](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
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.  
  
  
**46) Difference between final, finalize and finally?**([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-final-finally-and-finalize-java.html))  
The final is a modifier which you can apply to variable, methods and classes. If you make a variable final it means its value cannot be changed once initialized. finalize is a method, which is called just before an object is a garbage collected, giving it last chance to resurrect itself, but the call to finalize is not guaranteed.

finally is a keyword which is used in exception handling along with try and catch. the finally block is always executed irrespective of whether an exception is thrown from try block or not.  
  
  
**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](http://java67.blogspot.sg/2013/01/difference-between-set-list-and-map-in-java.html))  
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](http://javarevisited.blogspot.sg/2013/10/what-is-priorityqueue-data-structure-java-example-tutorial.html))  
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](http://java67.blogspot.sg/2012/12/difference-between-arraylist-vs-LinkedList-java.html))  
The obvious difference between them is that ArrayList is backed by array data structure, supports random access and LinkedList is backed by linked list data structure and doesn't support 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](http://java67.blogspot.sg/2012/07/sort-list-ascending-descending-order-set-arraylist.html))  
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](http://java67.blogspot.sg/2014/03/how-to-print-array-in-java-example-tutorial.html))  
You can print an array by using the Arrays.toString() and Arrays.deepToString() method. Since array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but Arrays.toString() will print each element.

**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](http://javarevisited.blogspot.com/2010/10/eclipse-tutorial-most-useful-eclipse.html), 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](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html))  
There are many differences between these two classes, some of them are following:  
a) Hashtable is a legacy class and present from JDK 1, HashMap was added later.  
b) Hashtable is synchronized and slower but HashMap is not synchronized and faster.  
c) Hashtable 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](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html))  
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](http://javarevisited.blogspot.sg/2014/01/ow-to-remove-objects-from-collection-arraylist-java-iterator-traversing.html))  
 Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Here is the [sample code](http://java67.blogspot.com/2015/10/how-to-solve-concurrentmodificationexception-in-java-arraylist.html) 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](http://javarevisited.blogspot.sg/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html))  
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

private static final int DEFAULT\_CAPACITY = 10;

//from HashMap.java JDK 7

static final int 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](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
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](http://java67.blogspot.sg/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html))  
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](http://javarevisited.blogspot.sg/2015/01/why-override-equals-hashcode-or-tostring-java.html))  
 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 alsong 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?**

The number of threads needed to handle three sockets in a Java program depends on the specific requirements and design of your application. Generally, a common approach is to use a multi-threaded model where each socket connection is handled by a separate thread.

This allows concurrent processing of multiple connections, preventing one slow or blocking operation from affecting others. Therefore, for three sockets, you might create three threads, each responsible for managing a single socket connection.

It's also essential to implement proper synchronization mechanisms if shared resources are accessed among these threads to avoid potential race conditions and ensure thread safety.

However, the optimal threading strategy can vary based on factors such as the nature of the tasks performed during socket handling, the expected number of concurrent connections, and the overall architecture of your application.

**67) How do you create ByteBuffer in Java? (**[**example**](https://javarevisited.blogspot.com/2020/03/bytebuffer-read-write-example-in-java.html)**)**

In Java, you can create a ByteBuffer using the ByteBuffer.allocate() method or ByteBuffer.allocateDirect() method. The allocate() method creates a non-direct buffer in the Java heap, while allocateDirect() creates a direct buffer outside of the Java heap

Here is the sample code:

// Creating a non-direct ByteBuffer with a specified capacity

int capacity = 1024;

ByteBuffer buffer = ByteBuffer.allocate(capacity);

// Creating a direct ByteBuffer with a specified capacity

ByteBuffer directBuffer = ByteBuffer.allocateDirect(capacity);

After creating a ByteBuffer, you can perform various operations such as reading and writing data, flipping, and compacting, depending on your specific use case.

**68) How do you write and read from ByteBuffer in Java?**

In Java, writing to and reading from a ByteBuffer involves using the put() methods to write data and the get() methods to read data. For instance, you can use put(byte), putInt(int), or putDouble(double) to write various data types into the buffer.

After writing, **it's essential to flip the buffer using the flip() method to prepare it for reading**.

Reading data involves using corresponding get() methods to retrieve the data from the buffer. Additionally, managing the buffer's position and limit is crucial; the position indicates the current read/write position, and the limit marks the end of valid data.

Depending on your needs, you may need to compact the buffer to remove read data or clear it to start writing from the beginning. Exception handling, such as handling BufferOverflowException and BufferUnderflowException, is important when dealing with dynamic data sizes to ensure robustness in your application.  
  
69) Is Java BIG endian or LITTLE endian?  
  
70) What is the byte order of ByteBuffer?  
  
71) The difference between direct buffer and non-direct buffer in Java? ([answer](http://javarevisited.blogspot.sg/2015/08/difference-between-direct-non-direct-mapped-bytebuffer-nio-java.html))  
  
72) What is the memory mapped buffer in Java? ([answer](http://javarevisited.blogspot.sg/2012/01/memorymapped-file-and-io-in-java.html))  
  
73) What is TCP NO DELAY socket option?  
  
74) What is the difference between TCP and UDP protocol? ([answer](http://javarevisited.blogspot.com/2014/07/9-difference-between-tcp-and-udp-protocol.html))  
  
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](http://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html))  
Here are couple of best practices which I follow while writing concurrent code in Java:  
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 wait() and notify for inter thread communication e.g. BlockingQueue, CountDownLatch and Semaphore.  
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 Collection 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](http://java67.blogspot.com/2014/01/10-points-about-thread-and-javalangthread-in-java.html))  
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](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html))  
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](http://javarevisited.blogspot.sg/2013/01/java-best-practices-method-overloading-constructor.html))  
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](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html))  
No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a ThreadLocal variable.

If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **joda-time** library.  
  
  
**83) How do you format a date in Java? e.g. in the ddMMyyyy format?** ([answer](http://javarevisited.blogspot.com/2011/09/convert-date-to-string-simpledateformat.html))  
You can either use SimpleDateFormat class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. dd-MM-yyyy or ddMMyyyy.  
  
  
84) How do you show timezone in formatted date in Java? ([answer](http://java67.blogspot.sg/2013/01/how-to-format-date-in-java-simpledateformat-example.html))  
  
85) The difference between java.util.Date and java.sql.Date in Java? ([answer](http://java67.blogspot.sg/2014/02/how-to-convert-javautildate-to-javasqldate-example.html))  
  
86) How to you calculate the difference between two dates in Java? ([program](http://javarevisited.blogspot.sg/2015/07/how-to-find-number-of-days-between-two-dates-in-java.html))  
  
87) How do you convert a String(YYYYMMDD) to date in Java? ([answer](http://java67.blogspot.sg/2014/12/string-to-date-example-in-java-multithreading.html))

### 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](http://javarevisited.blogspot.sg/2013/04/JUnit-tutorial-example-test-exception-thrown-by-java-method.html))  
  
91) Which unit testing libraries you have used for testing Java programs? (answer)  
  
92) What is the difference between @Before and @BeforeClass annotation? ([answer](http://javarevisited.blogspot.sg/2013/04/JUnit-tutorial-example-test-exception-thrown-by-java-method.html))

### Programming and Coding Questions

93) How to check if a String contains only numeric digits? ([solution](http://java67.blogspot.com/2014/01/java-regular-expression-to-check-numbers-in-String.html))  
  
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](http://java67.blogspot.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.htm))  
  
97) How to find the word with the highest frequency from a file in Java? ([solution](http://java67.blogspot.com/2015/10/java-program-to-find-repeated-words-and-count.html))  
  
98) How do you check if two given String are anagrams? ([solution](http://javarevisited.blogspot.sg/2013/03/Anagram-how-to-check-if-two-string-are-anagrams-example-tutorial.html))  
  
99) How to print all permutation of a String in Java? ([solution](http://javarevisited.blogspot.com/2015/08/how-to-find-all-permutations-of-string-java-example.html))  
  
100) How do you print duplicate elements from an array in Java? ([solution](http://javarevisited.blogspot.com/2015/06/3-ways-to-find-duplicate-elements-in-array-java.html))  
  
101) How to convert String to int in Java? ([solution](http://java67.blogspot.com/2015/08/2-ways-to-parse-string-to-int-in-java.html))  
  
102) How to swap two integers without using temp variable? ([solution](http://java67.blogspot.com/2015/08/how-to-swap-two-integers-without-using.html))

### 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](http://javarevisited.blogspot.com/2015/07/java-arraylist-tutorial.html) 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](http://javarevisited.blogspot.sg/2013/05/difference-between-abstract-class-vs-interface-java-when-prefer-over-design-oops.html))  
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.  
  
  
**106) Can you explain Liskov Substitution principle?** ([answer](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html))  
This is one of the toughest questions I have asked in Java interviews. Out of 50 candidates, I have almost asked only 5 have managed to answer it. I am not posting an answer to this question as I like you to do some research, practice and spend some time to understand this confusing principle well.  
  
  
**107) What is Law of Demeter violation? Why it matters?** ([answer](http://javarevisited.blogspot.com/2014/05/law-of-demeter-example-in-java.html))  
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](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html))  
  
**110) What is an abstract class? How is it different from an interface? Why would you use it?**([answer](http://java67.blogspot.sg/2014/06/why-abstract-class-is-important-in-java.html))  
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](http://javarevisited.blogspot.sg/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html))  
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](http://javarevisited.blogspot.sg/2015/06/difference-between-dependency-injection.html))  
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](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
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](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
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](https://javarevisited.blogspot.com/2022/10/template-method-pattern-in-java-example.html))  
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](https://www.java67.com/2022/12/visitor-design-patterns-in-java.html))  
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](https://javarevisited.blogspot.com/2018/02/composite-design-pattern-in-java-real.html))  
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](http://javarevisited.blogspot.sg/2015/06/difference-between-inheritance-and-Composition-in-Java-OOP.html))  
Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies.  
  
  
**119) Describe overloading and overriding in Java?** ([answer](http://java67.blogspot.sg/2012/09/difference-between-overloading-vs-overriding-in-java.html))  
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](http://javarevisited.blogspot.sg/2012/12/inner-class-and-nested-static-class-in-java-difference.html))  
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](http://javarevisited.blogspot.sg/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))  
If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation.

In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.  
  
  
**122) Give me an example of design pattern which is based upon open closed principle?** ([answer](http://javarevisited.blogspot.sg/2011/11/great-example-of-open-closed-design.html))  
This is one of the practical questions I ask experienced Java programmer. I expect them to know about OOP design principles as well as patterns. Open closed design principle asserts that your code should be open for extension but closed for modification.

Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code.  There are several design patterns which are based upon open closed design principle e.g. [Strategy pattern](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) if you need a new strategy, just implement the interface and configure, no need to modify core logic.

One working example is Collections.sort() method which is based on Strategy pattern and follows the open-closed principle, you don't modify sort() method to sort a new object, what you do is just implement Comparator in your own way.  
  
  
**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.

### Miscellaneous Java Interview Questions

It contains XML Processing in Java Interview question, JDBC Interview question, Regular expressions Interview questions, Java Error and Exception Interview Questions, Serialization,  
  
**125) The difference between nested static class and top level class?**([answer](http://java67.blogspot.sg/2012/10/nested-class-java-static-vs-non-static-inner.html))  
One of the fundamental questions from Java basics. I ask this question only to junior Java developers of 1 to 2 years of experience as it's too easy for an experience Java programmers. The answer is simple, a public top level class must have the same name as the name of the source file, there is no such requirement for nested static class.

A nested class is always inside a top level class and you need to use the name of the top-level class to refer nested static class e.g. HashMap.Entry is a nested static class, where HashMap is a top level class and Entry is nested static class.  
  
  
**126) Can you write a regular expression to check if String is a number?**([solution](http://javarevisited.blogspot.sg/2012/10/regular-expression-example-in-java-to-check-String-number.html))  
If you are taking Java interviews then you should ask at least one question on the regular expression. This clearly [differentiates an average programmer with a good programmer](http://javarevisited.blogspot.com/2015/05/how-to-differentiate-between-average.html). Since one of the traits of a good developer is to know tools, regex is the best tool for searching something in the log file, preparing reports etc.

Anyway, answer to this question is, a numeric String can only contain digits i.e. 0 to 9 and + and - sign that too at start of the String, by using this information you can write following regular expression to check if given String is number or not  
  
  
**127) The difference between checked and unchecked Exception in Java?**([answer](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html))  
checked exception is checked by the compiler at compile time. It's mandatory for a method to either handle the checked exception or declare them in their throws clause. These are the ones which are a sub class of Exception but doesn't descend from RuntimeException.

The unchecked exception is the descendant of RuntimeException and not checked by the compiler at compile time. This question is now becoming less popular and you would only find this with interviews with small companies, both investment banks and startups are moved on from this question.  
  
  
**128) The difference between throw and throws in Java?** ([answer](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html))  
the throw is used to actually throw an instance of java.lang.Throwable class, which means you can throw both Error and Exception using throw keyword e.g.

throw new IllegalArgumentException("size must be multiple of 2")

On the other hand, throws is used as part of method declaration and signals which kind of exceptions are thrown by this method so that its caller can handle them. It's mandatory to declare any unhandled checked exception in **throws** clause in Java. Like the previous question, this is another frequently asked Java interview question from errors and exception topic but too easy to answer.  
  
  
**129) The difference between Serializable and Externalizable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html))  
This is one of the frequently asked questions from Java Serialization. The interviewer has been asking this question since the day Serialization was introduced in Java, but yet only a few good candidate can answer this question with some confidence and practical knowledge.

Serializable interface is used to make Java classes serializable so that they can be transferred over network or their state can be saved on disk, but it leverages default serialization built-in JVM, which is expensive, fragile and not secure.

Externalizable allows you to fully control the Serialization process, specify a custom binary format and add more security measure.  
  
  
**130) The difference between DOM and SAX parser in Java?**([answer](http://javarevisited.blogspot.sg/2011/12/difference-between-dom-and-sax-parsers.html))  
Another common Java question but from XML parsing topic. It's rather simple to answer and that's why many interviewers prefers to ask this question on the telephonic round. DOM parser loads the whole XML into memory to create a tree based DOM model which helps it quickly locate nodes and make a change in the structure of XML while SAX parser is an event based parser and doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and not suitable to parse large XML files.  
  
  
**131) Tell me 3 features introduced on JDK 1.7?**([answer](http://javarevisited.blogspot.sg/2014/04/10-jdk-7-features-to-revisit-before-you.html))  
This is one of the good questions I ask to check whether the candidate is aware of recent development in Java technology space or not. Even though JDK 7 was not a big bang release like JDK 5 or JDK 8, it still has a lot of good feature to count on e.g. try-with-resource statements, which free you from closing streams and resources when you are done with that, Java automatically closes that.

Fork-Join pool to implement something like the Map-reduce pattern in Java. Allowing String variable and literal into switch statements. Diamond operator for improved type inference, no need to declare generic type on the right-hand side of variable declaration anymore, results in more readable and succinct code.

Another worth noting feature introduced was improved exception handling e.g. allowing you to catch multiple exceptions in the same catch block.  
  
  
**132) Tell me 5 features introduced in JDK 1.8?**([answer](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html))  
This is the follow-up question of the previous one. Java 8 is path breaking release in Java's history, here are the top 5 features from JDK 8 release

* **Lambda expression**, which allows you pass an anonymous function as object.
* **Stream API**, take advantage of multiple cores of modern CPU and allows you to write succinct code.
* **Date and Time API**, finally you have a solid and easy to use date and time library right into JDK
* **Extension methods**, now you can have static and default method into your interface
* **Repeated annotation**, allows you apply the same annotation multiple times on a type

**133) What is the difference between Maven and ANT in Java?** ([answer](http://javarevisited.blogspot.sg/2015/01/difference-between-maven-ant-jenkins-and-hudson.html))  
Another great question to check the all round knowledge of Java developers. It's easy to answer questions from core Java but when you ask about setting things up, building Java artifacts, many Java software engineer struggles.

Coming back to the answer of this question, Though both are build tools and used to create Java application build, Maven is much more than that. It provides a standard structure for Java project based upon the "convention over configuration" concept and automatically manages dependencies (JAR files on which your application is dependent) for Java applications. Please see the answer for more differences between the Maven and ANT tools.

**SPRING BOOT**

1.

What is Spring Boot, and how does it differ from Spring Framework?

Hide Answer

Spring Boot is a framework designed to simplify the development of Spring-based applications. It builds upon the Spring Framework, providing a convention-over-configuration approach and auto-configuration capabilities.

Unlike the Spring Framework, which requires explicit configuration, Spring Boot aims to minimize boilerplate code and provides defaults for various components. This makes it easier to get started with Spring-based applications.

2.

Explain the benefits of using Spring Boot for application development.

Hide Answer

Some benefits of using Spring Boot for application development include:

• Simplified setup and configuration through auto-configuration and starter dependencies.

• Reduced boilerplate code, enabling developers to focus more on application logic.

• Embedded server support, allowing applications to be run as standalone JAR files.

• Enhanced testability through the provision of t est utilities and annotations.

3.

What are the key features of Spring Boot?

Hide Answer

Key features of Spring Boot include:

Auto-configuration: Automatically configures Spring-based applications based on dependencies and defaults.

Starter dependencies: Pre-packaged dependencies that simplify the setup of specific application features or technologies.

Developer tools: Tools that enhance developer productivity such as automatic application restarts and live reload.

Actuator: Provides endpoints for monitoring and managing applications at runtime.

4.

Explain the concept of Spring Boot starters and provide an example.

Hide Answer

In the context of Spring Boot, Starters are a set of convenient dependency management providers that one can include in a Spring Boot application. Starters are a collection of dependency descriptors, which can help simplify your dependency management.

For instance, if you want to get started with Spring JPA, you just have to include the spring-boot-starter-data-jpa dependency and everything required for it (like Hibernate, Spring Data, etc.) will be added to your application.

Here's an example of what the Spring Boot Starter for JPA might look like in a pom.xml file:

By including this dependency, Spring Boot provides all the required dependencies for creating a JPA application.

5.

What is the purpose of the @SpringBootApplication annotation?

Hide Answer

The @SpringBootApplication annotation is a convenience annotation provided by Spring Boot. It serves as the entry point for the Spring Boot application. It combines three commonly used annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan.

With @SpringBootApplication, developers can enable auto-configuration, component scanning, and configuration properties in a single step.

6.

What is the default port number for a Spring Boot application?

Hide Answer

The default port number for a Spring Boot application is 8080. However, you can change it by specifying the desired port number in the application's configuration file (e.g., application.properties or application.yml) using the property server.port.

7.

How can you enable the auto-configuration feature in Spring Boot?

Hide Answer

Auto-configuration is enabled by default in Spring Boot. It leverages the classpath and the defined dependencies to automatically configure the application. Spring Boot analyzes the dependencies and uses their presence to configure various components such as data sources, web servers, and messaging systems.

If needed, you can disable specific auto-configuration classes or customize the configuration by providing your own beans.

8.

Explain the concept of starters in Spring Boot.

Hide Answer

Starters in Spring Boot are a set of dependencies that make it easier to configure and use specific features or technologies in an application. They encapsulate the required dependencies and configurations, allowing developers to add them to their projects with minimal effort.

For example, the spring-boot-starter-web starter includes all the necessary dependencies for building web applications including the Spring MVC framework, embedded web server, and JSON support.

9.

How does Spring Boot handle external configuration?

Hide Answer

Spring Boot provides multiple ways to handle external configurations. It supports property files (application.properties or application.yml) that can be placed in various locations including the classpath, file system, or external directories.

Spring Boot also supports environment variables, command-line arguments, and the use of profiles for different deployment environments. The configuration values can be accessed using the @Value annotation or by binding them to Java objects using the @ConfigurationProperties annotation

10.

What is the purpose of the application.properties (or application.yml) file?

Hide Answer

The application.properties or application.yml file is used for external configuration in a Spring Boot application. It allows developers to specify various properties and their values to configure the application.

These properties can control various aspects of the application such as server port, database connection details, logging configuration, and much more. The properties file can be placed in the classpath or other predefined locations, and Spring Boot will automatically load and apply the configuration during application startup.

11.

Describe the Spring Boot auto-configuration mechanism.

Hide Answer

The Spring Boot auto-configuration mechanism automatically configures the Spring application based on the dependencies present in the classpath. It uses the concept of conditionals to determine which beans and configurations should be enabled or disabled.

By analyzing the classpath and the available configuration, Spring Boot can provide sensible defaults and reduce the need for explicit configuration. This makes it easier to start and configure a Spring application.

12.

What is the purpose of the @Component annotation in Spring Boot?

Hide Answer

The @Component annotation is a core annotation from the Spring Framework and is also used in Spring Boot. It is a generic stereotype annotation used to mark a class as a Spring-managed component.

Components are auto-detected by Spring and can be used for dependency injection and component scanning. The @Component annotation serves as a base annotation for more specific annotations like @Repository, @Service, and @Controller.

13.

Explain the difference between @Component, @Repository, @Service, and @Controller annotations in Spring Boot.

Hide Answer

@Component: It is a generic stereotype annotation used to mark a class as a Spring-managed component. It is a broad and generic term that can be used for any type of Spring-managed component.

@Repository: It is a specialized form of @Component used to indicate that a class is a repository or data access component. It typically encapsulates database operations and exception translation.

@Service: It is a specialized form of @Component used to indicate that a class is a service component. It encapsulates business logic and is often used as an intermediate layer between controllers and repositories.

@Controller: It is a specialized form of @Component used to indicate that a class is a web controller component. It handles incoming requests, performs business logic, and prepares the response to be sent back to the client.

14.

What is the role of the @Autowired annotation in Spring Boot?

Hide Answer

The @Autowired annotation is used for dependency injection in Spring Boot. When applied to a field, setter method, or constructor, it allows Spring to automatically resolve and inject the required dependencies.

By using @Autowired, developers don't need to manually instantiate and wire dependencies. Spring Boot scans the application context for beans matching the required type and injects them automatically.

15.

How can you implement logging in a Spring Boot application?

Hide Answer

In a Spring Boot application, logging is typically implemented using a logging framework such as Logback or Log4j2. Spring Boot provides a default logging configuration out of the box.

You can configure logging levels, appenders, and log formats using the application.properties or application.yml file. Additionally, you can include the desired logging framework dependencies in your project's build configuration and use the framework's APIs to perform logging within your application code.

16.

What is the purpose of the SpringApplication.run() method?

Hide Answer

The SpringApplication.run() method is used to bootstrap and launch a Spring Boot application. It is typically invoked from the main method of the application's entry point class.

The run() method initializes the Spring application context, performs auto-configuration, starts the embedded server, and starts the application lifecycle. It returns an instance of the ApplicationContext, allowing access to the application context and its beans.

17.

What is Spring Boot CLI?

Hide Answer

Spring Boot Command Line Interface (CLI) is a command line tool that you can use to run and test Spring Boot applications from a command prompt. It provides a fast way to get Spring applications up and running. The CLI incorporates spring scripts into the unix-based shell to launch the boot applications.

Some of the advantages of using Spring Boot CLI are:

• It allows you to write your application using Groovy, which is a more succinct and expressive alternative to Java.

• It automatically includes useful external libraries whenever possible. For example, if you're writing a web application and importing classes such as @RestController, the CLI will automatically provide a dependency for Spring MVC.

• You can use various commands for different operations like run (to run the application), test (to test the application), jar (to create a jar file), init (to create a basic Java or Groovy project), etc.

18.

How does Spring Boot handle data validation?

Hide Answer

In Spring Boot, data validation can be performed using various mechanisms. One common approach is to use the validation annotations provided by the Bean Validation API, such as @NotNull, @Size, and @Pattern, on the fields of model objects.

By including the necessary validation annotations, Spring Boot automatically validates the input data and generates validation errors. These errors can be handled using BindingResult or Errors objects. Additionally, custom validation logic can be implemented by creating custom validation classes and methods.

19.

What is the purpose of the @RequestMapping annotation in Spring Boot?

Hide Answer

The @RequestMapping annotation is used to map HTTP requests to specific handler methods in a Spring Boot application. It is applied at the method or class level to define the URL patterns that should trigger the execution of the annotated method.

@RequestMapping allows developers to specify various attributes, such as the HTTP method (GET, POST, etc.), request parameters, headers, and more to further refine the mapping.

20.

How does Spring Boot integrate with containerization platforms like Docker and Kubernetes?

Hide Answer

Spring Boot integrates seamlessly with containerization platforms like Docker and Kubernetes. You can package a Spring Boot application as a Docker image by creating a Dockerfile that includes the necessary dependencies and configurations.

The image can be built and deployed to a containerization platform like Docker Swarm or Kubernetes. Spring Boot also provides features like externalized configuration and health indicators which can be leveraged by container orchestration platforms for efficient management and scaling of the application.

21.

Explain the concept of message-driven microservices using Spring Boot and Apache Pulsar.

Hide Answer

Message-driven microservices using Spring Boot and Apache Pulsar leverage the publish-subscribe messaging pattern to enable loosely coupled and scalable communication between microservices. Apache Pulsar acts as the messaging system, and Spring Boot provides the necessary abstractions for consuming and producing messages.

With Pulsar's messaging features and Spring Boot's integration, you can implement event-driven architectures where microservices communicate asynchronously through messages. This ensures decoupling and fault tolerance.

22.

What is the purpose of the @Value annotation in Spring Boot?

Hide Answer

The @Value annotation is used to inject values from properties files, environment variables, or other sources into Spring-managed beans. It can be applied to fields, methods, or constructor parameters.

With @Value, developers can easily access and use configuration properties or other values within their application code. The values can be specified directly or referenced using SpEL (Spring Expression Language) expressions.

23.

Describe the role of the CommandLineRunner and ApplicationRunner interfaces in Spring Boot.

Hide Answer

In Spring Boot, the CommandLineRunner and ApplicationRunner interfaces are used for performing specific tasks during the application startup process. When implemented, these interfaces provide a callback method (run()) that gets executed once the application context is initialized.

They are particularly useful for performing tasks like data initialization, cache population, or other one-time setup operations. The main difference between them is that CommandLineRunner receives the application's command-line arguments as a parameter, while ApplicationRunner receives an ApplicationArguments object.

24.

How can you implement pagination in a Spring Boot application?

Hide Answer

To implement pagination in a Spring Boot application, you can utilize features provided by libraries like Spring Data JPA or Spring Data MongoDB. They offer built-in support for pagination through the use of Pageable objects and repository methods.

You can retrieve a subset of data from a larger dataset by specifying the page number, page size, and sort criteria. The result is typically returned as a Page object that contains the requested data along with metadata such as total elements, total pages, and more.

25.

Explain the concept of bean scopes in Spring Boot.

Hide Answer

Bean scopes define the lifecycle and visibility of Spring-managed beans in a Spring Boot application. The following are the commonly used bean scopes:

Singleton (default): Only one instance of the bean is created and shared across the entire application context.

Prototype: A new instance of the bean is created each time it is requested.

Request: A new instance of the bean is created for each HTTP request. It is only applicable in a web application context.

Session: A new instance of the bean is created for each user session. It is only applicable in a web application context.

Custom scopes: Spring Boot allows defining custom bean scopes by implementing the Scope interface and registering them in the application context.

26.

What is the purpose of the @Qualifier annotation in Spring Boot?

Hide Answer

The @Qualifier annotation in Spring is used to disambiguate bean references when we have multiple beans of the same type defined in the Spring container. It is used in scenarios where a given type has more than one implementation and we need to inject a specific implementation.

By default, Spring uses the by-type autowiring mechanism. This means that if we have more than one bean of the same type, Spring will throw a NoUniqueBeanDefinitionException because it won't know which one to autowire.

The @Qualifier annotation can be used in conjunction with @Autowired to specify which exact bean should be wired, by providing the name of the bean as the qualifier value.

27.

How does Spring Boot handle exception logging and error handling?

Hide Answer

In Spring Boot, exception logging and error handling can be configured using various mechanisms. Spring Boot automatically provides a default error page that displays a standardized error message for unhandled exceptions.

However, you can customize the error-handling behavior by implementing exception handlers using the @ControllerAdvice annotation and handling specific exceptions in dedicated methods.

Additionally, you can configure logging frameworks to capture and log exceptions with desired levels of detail and appenders.

28.

Describe the purpose and usage of the @RestControllerAdvice annotation.

Hide Answer

The @RestControllerAdvice annotation is a specialized form of the @ControllerAdvice annotation in Spring Boot. It combines the functionality of @ControllerAdvice and @ResponseBody, making it convenient for implementing global exception handling in RESTful APIs.

By using @RestControllerAdvice, you can define exception handlers that handle exceptions thrown by any @RequestMapping or @RestController method within the application. The exception handlers can return error responses in JSON or other supported formats.

29.

What is the purpose of the @ConfigurationProperties annotation in Spring Boot?

Hide Answer

The @ConfigurationProperties annotation is used to bind external configuration properties to Spring-managed beans. By annotating a bean class with @ConfigurationProperties and specifying a prefix, you can map properties with matching names to the fields or setter methods of the bean.

Spring Boot will automatically bind the values from the configuration sources to the corresponding bean properties. The annotation simplifies the retrieval and usage of configuration properties within your application.

30.

Describe the purpose and usage of the @DynamicPropertySource annotation in Spring Boot testing.

Hide Answer

The @DynamicPropertySource annotation in Spring Boot testing allows you to dynamically define and modify configuration properties during the test execution. You can use this annotation in conjunction with the TestPropertyValues class to set or override properties based on dynamic values or test conditions.

This provides flexibility in configuring the environment for testing and allows you to simulate different scenarios or configurations during testing.

31.

What is the purpose of the @TransactionalEventListener annotation in Spring Boot?

Hide Answer

The @TransactionalEventListener annotation in Spring Boot lets you listen to transactional events and perform actions based on those events. You can use this annotation on methods that should be invoked when a specific transactional event occurs such as before or after a transaction is committed or rolled back.

The @TransactionalEventListener annotation provides a convenient way to handle domain-specific logic or side effects based on transactional events in a Spring Boot application.

32.

What is the purpose of the @Scheduled annotation in Spring Boot?

Hide Answer

The @Scheduled annotation is used to configure scheduled tasks in a Spring Boot application. Applying this annotation to a method enables you to specify the schedule at which the method should be executed.

The schedule can be defined using various options such as fixed-rate, fixed-delay, or cron expressions. Spring Boot automatically detects and executes the scheduled methods based on the specified schedule.

33.

Describe the role of the @Profile annotation in Spring Boot.

Hide Answer

The @Profile annotation is used to activate or deactivate specific configuration components or beans based on the current environment or profile in a Spring Boot application. Annotating a class or method with @Profile and specifying the desired profile name lets you control when that component or bean should be active. This allows you to have different configurations for different deployment environments such as development, testing, or production.

34.

What is the purpose of Spring Boot's dynamic reloading and how does it work?

Hide Answer

Spring Boot's dynamic reloading feature allows you to make changes to the application code or resources without the need to restart the entire application. It improves development productivity by automatically reloading the modified classes or resources on the fly.

The dynamic reloading feature uses class reloading mechanisms provided by the underlying JVM, such as Java Instrumentation API or custom class loaders, to reload the changed classes while preserving the application's state.

35.

Explain the concept of externalized logging in Spring Boot using Logback or Log4j2.

Hide Answer

Externalized logging in Spring Boot allows you to configure and customize logging behavior without modifying the application code. Logback or Log4j2 can be used as the underlying logging framework.

The configuration is typically done in an external configuration file, such as logback.xml or log4j2.xml, which can be placed in the classpath or specified using the logging.config property. The externalized logging configuration file provides flexibility in defining log levels, appenders, formatters, and other logging-related properties.

36.

What is the purpose of the @ModelAttribute annotation in Spring Boot?

Hide Answer

The @ModelAttribute annotation is used in Spring Boot to bind request parameters or form data to method parameters or model attributes. It can be applied to method parameters or method return values.

When applied to method parameters, the @ModelAttribute annotation binds the incoming request parameters or form data to the corresponding method parameters. When applied to method return values, it binds the method's return value to a model attribute, making it available in the view for rendering.

37.

Explain the concept of reactive messaging with Spring Boot and Apache Kafka Streams.

Hide Answer

Reactive messaging with Spring Boot and Apache Kafka Streams enables the building of real-time streaming applications that react to events and reactively process data streams. Spring Cloud Stream provides abstractions to integrate Spring Boot applications with Kafka Streams.

With @StreamListener annotations, you can consume Kafka topics as reactive streams and perform processing operations using the reactive programming model. This approach facilitates the development of scalable and resilient streaming applications.

38.

Describe the purpose and usage of the @Transactional(propagation = Propagation.NESTED) annotation.

Hide Answer

The @Transactional(propagation = Propagation.NESTED) annotation is used to define a nested transactional scope in a Spring Boot application. When a method is annotated with this annotation, a nested transaction is created within the current transaction.

The nested transaction behaves as an independent transaction and can be rolled back separately from the outer transaction. If the nested transaction fails, only the changes made within the nested transaction are rolled back, while the outer transaction remains unaffected.

39.

What is the purpose of the @DataJpaTest annotation in Spring Boot testing?

Hide Answer

The @DataJpaTest annotation is used to configure and customize the testing environment for JPA repositories in a Spring Boot application. When applied to a test class, it sets up an in-memory database, configures Spring Data JPA, and loads only the necessary components for testing JPA repositories.

@DataJpaTest provides a lightweight and isolated environment for testing JPA-related functionality without requiring a full application context or a real database connection.

40.

Describe the purpose and usage of the Spring Boot Admin Server for monitoring and managing applications.

Hide Answer

The Spring Boot Admin Server is a tool that provides a web-based interface for monitoring and managing multiple Spring Boot applications in a centralized manner. It collects and displays various metrics, health statuses, and other information about the registered Spring Boot applications.

The Admin Server allows you to view and manage application details, monitor JVM metrics, and receive alerts on specific conditions. It simplifies the monitoring and management of Spring Boot applications in a production environment.

Looking for remote developer job at US companies?

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

Apply Now

INTERMEDIATE SPRING BOOT INTERVIEW QUESTIONS AND ANSWERS

1.

Differentiate between Spring MVC and Spring Boot.

Hide Answer

Spring MVC is a framework for building web applications using the Model-View-Controller (MVC) architectural pattern. It provides features for handling requests, managing controllers, rendering views, and managing data flow.

Spring Boot, on the other hand, is an opinionated framework built on top of Spring that aims to simplify the setup and configuration of Spring applications. It provides out-of-the-box defaults and auto-configuration, reducing the need for manual configuration and boilerplate code.

2.

What is the role of the @RestController annotation in Spring Boot?

Hide Answer

The @RestController annotation is used to define a RESTful controller in a Spring Boot application. It combines the functionality of the @Controller and @ResponseBody annotations, simplifying the process of building RESTful APIs by automatically serializing the return values of methods into JSON or XML responses.

3.

How can you implement exception handling in a Spring Boot application?

Hide Answer

Exception handling in Spring Boot can be implemented using the @ControllerAdvice annotation. By creating a class annotated with @ControllerAdvice and defining methods annotated with @ExceptionHandler, you can handle specific exceptions and return appropriate responses.

You can also use the @ResponseStatus annotation to specify the HTTP status code for the response.

4.

Explain the concept of dependency injection in Spring Boot.

Hide Answer

Dependency injection is a core concept in Spring Boot. It allows objects to be loosely coupled by providing their dependencies from external sources. Spring Boot uses inversion of control (IoC) and the dependency injection pattern to manage dependencies.

Spring Boot automatically resolves and injects the required dependencies at runtime by annotating classes with appropriate annotations such as @Autowired.

5.

How does Spring Boot support database operations?

Hide Answer

Spring Boot provides excellent support for database operations through its integration with Spring Data JPA. By defining entities and repositories, you can perform CRUD (Create, Read, Update, Delete) operations on databases with minimal boilerplate code.

Spring Boot automatically configures the database connection and transaction management, and provides powerful querying capabilities.

6.

Describe the role of the Spring Boot Actuator.

Hide Answer

Spring Boot Actuator is a feature that provides insight into the runtime of a Spring Boot application. It offers a set of production-ready endpoints that expose information about application health, metrics, environment, logging, and more.

The Actuator enables monitoring and management of the application, making it easier to understand and troubleshoot in production environments.

7.

How can you implement caching in a Spring Boot application?

Hide Answer

Caching in a Spring Boot application can be implemented using the @Cacheable, @CacheEvict, and other cache-related annotations provided by the Spring Framework. Adding these annotations to methods lets you cache the results and improve performance. Spring Boot integrates with popular caching providers like Ehcache, Hazelcast, and Redis.

8.

What is the purpose of the @Scheduled annotation in Spring Boot?

Hide Answer

The @Scheduled annotation is used to schedule the execution of a method at fixed intervals or specific times. It allows you to define cron expressions or fixed delay/initial delay values. Spring Boot automatically triggers the annotated method based on the specified schedule, making it suitable for performing recurring tasks such as data synchronization or sending periodic notifications.

9.

How can you enable cross-origin resource sharing (CORS) in a Spring Boot application?

Hide Answer

To enable CORS in a Spring Boot application, you can use the @CrossOrigin annotation at the controller level or globally configure CORS using a WebMvcConfigurer bean. The annotation allows you to specify the allowed origins, HTTP methods, headers, and other CORS-related settings. Enabling CORS ensures that web browsers can make requests to your application from different domains.

10.

Explain the concept of profiles in Spring Boot.

Hide Answer

Profiles in Spring Boot allow you to define different configurations for different environments or scenarios. By using the @Profile annotation on classes or methods, you can specify which profiles should be active for the corresponding beans or configurations.

Profiles enable you to have different property values, component configurations, or dependencies based on the active profile. This facilitates easy deployment and testing across different environments.

11.

Explain the concept of the Spring Boot Actuator and its major features.

Hide Answer

Spring Boot Actuator provides a set of production-ready features and endpoints that help monitor and manage a Spring Boot application.

Its major features include health checks, which provide information about the application's health; metrics, which gather and expose various runtime metrics; info, which displays custom application information; logging, which allows changing log levels at runtime, and many more endpoints for managing and understanding the application in a production environment.

12.

How can you integrate Spring Security in a Spring Boot application?

Hide Answer

To integrate Spring Security in a Spring Boot application, you need to add the appropriate dependencies and configure security settings. Spring Security provides comprehensive authentication and authorization mechanisms.

You can configure security rules using Java configuration or annotations, define user roles and permissions, and customize authentication providers, such as in-memory authentication, database-backed authentication, or integration with external identity providers.

13.

Describe the role of the @Transactional annotation in Spring Boot.

Hide Answer

The @Transactional annotation is used to mark a method or class for transaction management in Spring Boot. It ensures that the annotated method or all methods within the annotated class are executed within a transactional context.

The @Transactional annotation manages the transaction boundaries, rollback behavior, and other transactional aspects to ensure data consistency and integrity.

14.

How does Spring Boot handle internationalization (i18n) and localization (l10n)?

Hide Answer

Spring Boot automatically resolves the appropriate message based on the user's locale, making it convenient to build multi-language applications. It provides support for internationalization and localization through properties files and the use of the MessageSource interface.

By defining message bundles for different locales and configuring the message source, you can easily retrieve and display localized messages in your application.

15.

What is the purpose of the @RestControllerAdvice annotation?

Hide Answer

The @RestControllerAdvice annotation combines the functionalities of @ControllerAdvice and @ResponseBody annotations. It is used to define a global exception handler for RESTful controllers in a Spring Boot application.

Annotated classes can contain exception-handling methods annotated with @ExceptionHandler which handles exceptions thrown within any @RestController in the application. These methods can return custom error responses or perform other actions based on the exception type.

16.

Explain the concept of Spring Data REST and its advantages.

Hide Answer

Spring Data REST is a project built on top of Spring Data, it takes the features of Spring HATEOAS and Spring Data to build Spring MVC-based RESTful services with less code. With Spring Data REST, you can leverage your Spring Data repositories and convert them into full-featured RESTful services with ease.

Some of its advantages are:

Rapid Development: With Spring Data REST, a great deal of your HTTP resource implementation time can be saved. It's quick and easy to build a RESTful service with full CRUD functionality.

Data Access: It leverages Spring Data's repositories and provides seamless, RESTful access to your data model.

HAL Browser: Spring Data REST includes support for the HAL Browser, allowing users to navigate, create, update, and delete resources directly from their web browsers.

Search Support: It has built-in support for searches. Custom repository methods are automatically exposed as HTTP resources.

17.

How can you implement file upload and download functionality in a Spring Boot application?

Hide Answer

File upload and download functionality can be implemented in a Spring Boot application by configuring multipart file handling. By using the MultipartFile object as a method parameter, Spring Boot automatically binds uploaded files to it.

For file download, you can return the file as a response with appropriate headers. Additionally, you can leverage storage services like Amazon S3 or Azure Blob Storage for file storage and retrieval.

18.

Describe the purpose and usage of the @Async annotation in Spring Boot.

Hide Answer

The @Async annotation is used to indicate that a method should be executed asynchronously. When a method is annotated with @Async, Spring Boot runs it in a separate thread from a task executor, allowing the caller to continue execution without waiting for the asynchronous method to complete.

This annotation is useful for offloading time-consuming tasks, improving performance, and providing a more responsive user experience.

19.

What is the role of the embedded servlet container in Spring Boot?

Hide Answer

The embedded servlet container in Spring Boot allows you to run web applications without the need for a separate web server. It provides a lightweight servlet container, such as Tomcat, Jetty, or Undertow, that is embedded within the application.

Spring Boot automatically configures and starts the embedded servlet container, simplifying the deployment and execution of web applications.

20.

How can you implement request and response logging in a Spring Boot application?

Hide Answer

Request and response logging in a Spring Boot application can be implemented using filters or interceptors. Creating a custom filter or interceptor lets you intercept incoming requests and outgoing responses and log their details, such as headers, payloads, and other relevant information.

Spring Boot allows you to register these filters or interceptors in the application's configuration, enabling centralized logging across the application.

21.

Explain the concept of reactive data access in Spring Boot using Spring Data R2DBC.

Hide Answer

Reactive data access in Spring Boot allows you to build non-blocking and efficient applications that handle a large number of concurrent requests. Spring Data R2DBC provides reactive database access by integrating with R2DBC (Reactive Relational Database Connectivity).

It enables you to perform asynchronous database operations using reactive programming paradigms, such as Flux and Mono, providing better scalability and responsiveness compared to traditional blocking database access.

22.

How can you perform asynchronous messaging using Spring Boot and RabbitMQ?

Hide Answer

Asynchronous messaging using Spring Boot and RabbitMQ can be achieved by integrating the Spring AMQP (Advanced Message Queuing Protocol) library. By configuring the RabbitMQ connection details and using the appropriate annotations and components, you can send and receive messages asynchronously.

Spring Boot provides abstractions like @RabbitListener for message consumption and RabbitTemplate for message production. They make it easy to implement asynchronous messaging patterns like publish-subscribe and request-reply.

23.

Describe the purpose and usage of the @Conditional annotation in Spring Boot.

Hide Answer

The @Conditional annotation in Spring Boot allows you to conditionally activate or deactivate beans or configurations based on specific conditions. By annotating a bean or configuration class with @Conditional and providing a condition class implementing the Condition interface, you can control whether the bean or configuration should be created and registered based on runtime conditions. This enables flexible configuration based on environment, properties, or other factors.

24.

What is the purpose of the @SpringBootTest annotation in Spring Boot testing?

Hide Answer

The @SpringBootTest annotation is used to bootstrap a Spring Boot application context for testing purposes. It allows you to load the entire application context, including all configurations and beans, during integration tests.

@SpringBootTest provides features like auto-configuration, dependency injection, and easy access to application-specific components, enabling comprehensive testing of Spring Boot applications.

25.

Explain the concept of Spring Boot's actuator health checks and customizing health indicators.

Hide Answer

Spring Boot's actuator health checks are endpoints provided by the Actuator that give insights into the application's health. By default, health indicators check the overall system health. You can customize them by implementing the HealthIndicator interface and registering them with the application context.

Custom health indicators allow you to monitor specific aspects of the application's health such as database connectivity, external service availability, or custom health checks.

26.

How can you secure REST APIs in a Spring Boot application using JSON Web Tokens (JWT)?

Hide Answer

You can secure REST APIs in a Spring Boot application using JSON Web Tokens (JWT) by integrating Spring Security and JWT libraries. Spring Security provides mechanisms for authentication and authorization, while JWT facilitates token-based authentication.

By configuring Spring Security filters, implementing authentication and authorization providers, and validating JWT tokens, you can protect your REST APIs and control access based on user roles and permissions.

27.

Describe the purpose and usage of the @EntityScan annotation in Spring Boot.

Hide Answer

The @EntityScan annotation is used to specify the base packages to scan for entity classes in a Spring Boot application. When using JPA (Java Persistence API) with Spring Boot, @EntityScan helps the JPA provider locate and manage entity classes.

By default, Spring Boot scans the package of the application's main class and its sub-packages. However, if entity classes are located in different packages, you need to use @EntityScan to include those packages.

28.

What is the purpose of the @Retryable annotation in Spring Boot?

Hide Answer

The @Retryable annotation is used to specify that a method should be retried if it fails due to specified exceptions. Adding @Retryable and configuring the desired retry behavior enables Spring Boot to automatically retry the method when exceptions occur.

This can be useful for handling transient errors, such as network timeouts or temporary resource unavailability, and ensuring the successful execution of critical operations.

29.

Explain the concept of auto-reconfiguration in Spring Boot and its limitations.

Hide Answer

Auto-reconfiguration in Spring Boot is a feature that automatically configures certain components and dependencies based on the classpath and available resources. It simplifies the configuration process by detecting and configuring components like data sources, messaging brokers, and caches.

Auto-reconfiguration has limitations when it comes to complex or custom configurations. It may not always provide the desired configuration out of the box. In such cases, manual configuration may be required.

30.

How can you implement distributed caching in a Spring Boot application using Hazelcast or Redis?

Hide Answer

To implement distributed caching in a Spring Boot application with Hazelcast or Redis, you can leverage the respective cache providers' integration libraries. Configuring the cache manager and cache-related settings lets you enable distributed caching.

Spring Boot simplifies the setup and configuration process by providing auto-configuration support for both Hazelcast and Redis. Additionally, you can annotate methods with cache-related annotations like @Cacheable or @CacheEvict to cache and retrieve data efficiently.

Looking for remote developer job at US companies?

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

Apply Now

ADVANCED SPRING BOOT INTERVIEW QUESTIONS

1.

How can you create a Spring Boot application using Gradle?

Hide Answer

To create a Spring Boot application using Gradle, follow these steps:

• Set up a new Gradle project or add Spring Boot dependencies to an existing Gradle project.

• Make sure you have the required plugins configured in the build.gradle file, such as the org.springframework.boot and io.spring.dependency-management plugins.

• Define the necessary dependencies in the dependencies section of the build.gradle file. Specify the desired Spring Boot starter dependencies.

• Create the main application class and annotate it with @SpringBootApplication.

• Implement the application logic within the main application class or other components.

• Use the Gradle command line or an IDE plugin to build and run the application.

2.

How can you customize the default error pages in a Spring Boot application?

Hide Answer

To customize the default error pages in a Spring Boot application, you can create an error page template or controller method that handles the error. Defining an error template with the appropriate name and placing it in the src/main/resources/templates/error directory enables Spring Boot to automatically render that template for the corresponding error status code.

Alternatively, you can create a controller method with @ExceptionHandler annotation to handle specific exceptions and return a custom error response.

3.

How can you create a Spring Boot application using Maven?

Hide Answer

To create a Spring Boot application using Maven, follow these steps:

• Set up a new Maven project or add Spring Boot dependencies to an existing Maven project.

• Ensure that the project's dependencies include the spring-boot-starter-parent as the parent project.

• Define the necessary dependencies in the project's pom.xml file, such as spring-boot-starter-web for web applications.

• Create the main application class and annotate it with @SpringBootApplication.

• Implement the application logic within the main application class or other components.

• Use the Maven command line or an IDE plugin to build and run the application.

4.

How can you implement security in a Spring Boot application?

Hide Answer

Security can be implemented in a Spring Boot application by adding the appropriate dependencies, such as spring-boot-starter-security, and configuring the security settings. This can be done by creating a security configuration class that extends WebSecurityConfigurerAdapter and overriding its methods to define authentication and authorization rules.

Additionally, you can customize the login page, handle logout, and secure specific endpoints using annotations like @EnableWebSecurity and @EnableGlobalMethodSecurity.

5.

What are the different deployment options for a Spring Boot application?

Hide Answer

Spring Boot applications can be deployed in various ways including:

Standalone JAR: Packaging the application as a self-contained executable JAR file with an embedded servlet container like Tomcat or Jetty.

WAR deployment: Packaging the application as a traditional WAR file and deploying it to a servlet container.

Docker: Containerizing the application using Docker and running it on Docker containers.

Cloud platforms: Deploy the application to cloud platforms like AWS, Azure, or Google Cloud using platform-specific deployment options such as AWS Elastic Beanstalk or Azure App Service.

6.

Describe the process of creating a RESTful API using Spring Boot.

Hide Answer

To create a RESTful API using Spring Boot, you can follow these steps:

• Define your domain model and business logic.

• Create a Spring MVC controller class and define handler methods annotated with @RequestMapping or other mapping annotations.

• Implement the required CRUD operations within the handler methods using appropriate annotations like @GetMapping, @PostMapping, etc.

• Customize the request and response handling with annotations such as @RequestBody to map request payloads and @ResponseBody to define the response body.

• Configure additional features like exception handling, input validation, and security, if required.

• Run the Spring Boot application and the API endpoints will be accessible based on the mappings defined in the controller.

7.

How does Spring Boot integrate with messaging systems such as RabbitMQ?

Hide Answer

Spring Boot provides integration with messaging systems like RabbitMQ through the Spring AMQP project. To integrate RabbitMQ with a Spring Boot application, you can include the spring-boot-starter-amqp dependency and configure the necessary properties in the application's configuration file (application.properties or application.yml).

You can use the RabbitTemplate class to send messages to RabbitMQ and consume messages using @RabbitListener annotations on appropriate methods.

8.

Explain the concept of Spring Boot Data JPA and provide an example.

Hide Answer

Spring Boot Data JPA is a sub-project of Spring Data that provides enhanced support for JPA (Java Persistence API)-based repositories in Spring Boot applications. It simplifies the implementation of the data access layer by automatically generating the boilerplate code for common database operations.

For example, by defining a JPA entity class and extending JpaRepository, you can get CRUD operations for that entity without writing any additional code. You can also define custom queries using method name conventions or @Query annotations

9.

How can you handle large file uploads in a Spring Boot application?

Hide Answer

To handle large file uploads in a Spring Boot application, you can configure the maximum file size limit in the application's properties file by setting the spring.servlet.multipart.max-file-size and spring.servlet.multipart.max-request-size properties to appropriate values. Additionally, you can use the MultipartFile parameter in the controller method to receive the uploaded file and process it as needed.

10.

What is the purpose of Spring Boot Actuator endpoints?

Hide Answer

Spring Boot Actuator endpoints provide insights into the internals of a Spring Boot application such as health status, metrics, environment information, and more. These endpoints expose management and monitoring capabilities over HTTP or other protocols, allowing you to monitor and manage the application in production. Actuator endpoints can be customized and secured based on the specific requirements of the application.

11.

Explain the concept of Spring Boot Actuator metrics and monitoring.

Hide Answer

Spring Boot Actuator metrics allow you to collect and monitor various application metrics such as HTTP request counts, response times, JVM memory usage, and database connection pool metrics.

Actuator metrics are collected by integrating with metrics libraries like Micrometer and can be exposed through various endpoints such as /actuator/metrics. These metrics can be visualized using monitoring tools like Prometheus, Grafana, or the built-in Actuator endpoints.

12.

How can you implement microservices architecture using Spring Boot?

Hide Answer

To implement a microservices architecture using Spring Boot, you can follow these steps:

• Identify the different business capabilities and boundaries of your application.

• Design and develop each microservice as a separate Spring Boot application, encapsulating a specific business capability.

• Use lightweight communication mechanisms like REST or messaging for inter-service communication.

• Implement service discovery and registration using tools like Netflix Eureka or HashiCorp Consul.

• Apply fault tolerance and resilience patterns like circuit breakers (Hystrix) and distributed tracing (Sleuth) for better reliability.

• Deploy and manage microservices using containerization platforms like Docker and orchestration tools like Kubernetes.

13.

Describe the role of the Spring Cloud Netflix stack in a Spring Boot application.

Hide Answer

The Spring Cloud Netflix stack provides integration with various Netflix OSS components to simplify the development of distributed systems in a Spring Boot application. It includes modules like Eureka for service discovery, Ribbon for client-side load balancing, Hystrix for fault tolerance, and Zuul for API gateway functionality.

These components enable developers to build scalable and resilient microservices architectures by providing out-of-the-box solutions for common distributed system challenges.

14.

What is the purpose of Spring Boot DevTools and how does it enhance development productivity?

Hide Answer

Spring Boot DevTools is a set of developer tools that enhance the development experience for Spring Boot applications. It provides features like automatic application restart on code changes, live reloading of static resources, and enhanced logging during development.

DevTools helps in reducing the development turnaround time by eliminating the need for manual restarts and providing quick feedback on code changes.

15.

How can you implement distributed tracing in a Spring Boot application using Spring Cloud Sleuth?

Hide Answer

To implement distributed tracing in a Spring Boot application with Spring Cloud Sleuth, you can include the necessary dependencies like spring-cloud-starter-sleuth. Sleuth integrates with popular distributed tracing systems like Zipkin or Jaeger.

Once configured, Sleuth automatically adds trace and span identifiers to the application's logs and propagates them across different services. This allows you to trace the flow of requests across multiple services and analyze performance bottlenecks.

16.

Explain the concept of reactive programming in Spring Boot with Spring WebFlux.

Hide Answer

Reactive programming in Spring Boot with Spring WebFlux is based on the Reactive Streams specification and enables non-blocking, event-driven programming for building scalable and resilient applications.

Spring WebFlux provides an alternative to the traditional Servlet-based programming model and allows developers to handle requests asynchronously using reactive types like Mono and Flux. This approach is well-suited for handling high concurrency and building reactive systems that can handle a large number of concurrent connections with limited resources.

17.

How does Spring Boot integrate with Apache Kafka for event-driven architectures?

Hide Answer

Spring Boot provides integration with Apache Kafka through the Spring Kafka project. You can include the spring-boot-starter-kafka dependency to get started.

Spring Kafka provides abstractions to produce and consume messages from Kafka topics using the KafkaTemplate and @KafkaListener annotations, respectively. Additionally, Spring Kafka integrates with Spring Boot's auto-configuration to simplify the configuration of Kafka-related properties.

18.

Describe the purpose and usage of Spring Boot's caching abstraction.

Hide Answer

Spring Boot's caching abstraction provides a convenient way to cache the results of expensive operations, reducing the response time and improving application performance. By using annotations like @Cacheable, @CachePut, and @CacheEvict, you can easily cache method results based on specified cache names or keys.

The caching abstraction supports various cache providers, such as Redis or Ehcache, and can be easily configured using the application's properties file.

19.

How can you perform database migrations in a Spring Boot application using Flyway or Liquibase?

Hide Answer

Include corresponding dependencies (flyway-core or liquibase-core) to perform database migrations in a Spring Boot application using Flyway or Liquibase. By placing the migration scripts in the classpath (src/main/resources/db/migration), Flyway or Liquibase will automatically execute the scripts during application startup.

These migration scripts allow you to manage database schema changes, versioning, and data initialization in a controlled manner.

20.

Explain the concept of externalized configuration in Spring Boot and its benefits.

Hide Answer

Externalized configuration in Spring Boot allows you to configure the application using external properties files, environment variables, or command-line arguments. This approach decouples the application configuration from the code, making it more flexible and easier to manage.

The externalized configuration enables the application to be deployed in different environments without modifying the code. Spring Boot provides a standardized and flexible way to read and use these external configurations.

21.

How can you implement distributed session management in a Spring Boot application using Spring Session?

Hide Answer

To implement distributed session management in a Spring Boot application using Spring Session, follow these steps:

• Include the necessary dependencies for Spring Session and a session store implementation like Redis or Hazelcast.

• Configure the session store details, such as the connection properties, in the application's configuration file (e.g., application.properties or application.yml).

• Enable Spring Session support by annotating your configuration class with @EnableRedisHttpSession (for Redis) or @EnableHazelcastHttpSession (for Hazelcast).

• Spring Session will automatically handle session creation, serialization, and synchronization with the session store, allowing session data to be shared across multiple instances of your application.

22.

How can you implement serverless functions using Spring Boot and AWS Lambda?

Hide Answer

You can use the spring-cloud-function-adapter-aws dependency to implement serverless functions using Spring Boot and AWS Lambda. You can deploy Spring Boot applications as serverless functions on AWS Lambda by creating a function bean and configuring the AWS Lambda handler.

The adapter takes care of the integration between Spring Cloud Function and AWS Lambda, allowing you to develop serverless functions using the familiar Spring Boot programming model.

23.

How can you implement method-level security in a Spring Boot application?

Hide Answer

To implement method-level security in a Spring Boot application, you can use the @PreAuthorize or @PostAuthorize annotations provided by Spring Security. Here's an example:

• Configure Spring Security in your application by including the necessary dependencies and configuration.

• Annotate the methods that require security checks with @PreAuthorize or @PostAuthorize.

• Specify the desired security expressions in the annotations to define the required conditions for method invocation.

• Spring Security will evaluate the expressions and allow or deny access to the methods based on the configured security rules.

24.

How can you implement server-sent events (SSE) in a Spring Boot application?

Hide Answer

To implement server-sent events (SSE) in a Spring Boot application, you can use the SseEmitter class provided by Spring Framework. Here's an example of how to implement SSE:

• Create a controller method that returns an SseEmitter object.

• In this method, use the SseEmitter to send events to the client.

• Use the send() method of SseEmitter to send events periodically or based on specific triggers.

• Set appropriate headers, such as Content-Type and Cache-Control, for SSE support.

• Register the SseEmitter as a handler method in your controller.

25.

How can you integrate Spring Boot with OAuth 2.0 for secure authentication and authorization?

Hide Answer

You can use the Spring Security OAuth2 module to integrate Spring Boot with OAuth 2.0. By configuring the appropriate OAuth 2.0 provider details and defining the client registration properties, Spring Boot can handle the authentication and authorization flow.

You can secure your endpoints by applying Spring Security annotations like @PreAuthorize or by using declarative configuration. This integration enables secure authentication and authorization using OAuth 2.0 standards.

26.

How can you implement data caching in a Spring Boot application using the Spring Cache Abstraction?

Hide Answer

To implement data caching in a Spring Boot application, you can leverage the Spring Cache Abstraction. Follow these steps:

• Enable caching support by annotating your configuration class with @EnableCaching.

• Add the desired cache implementation library, such as Ehcache or Caffeine, as a dependency.

• Annotate the methods that should be cached with @Cacheable and specify the cache name or key.

• Configure the cache properties, such as eviction policies and time-to-live, in the cache implementation's configuration file or using Spring Boot's properties.

27.

How can you configure a custom error page in a Spring Boot application?

Hide Answer

To configure a custom error page in a Spring Boot application, you can create a custom error controller and map it to a specific URL or error status code. Here's an example:

• Create a class implementing the ErrorController interface.

• Annotate the class with @Controller and, optionally, with @RequestMapping to specify the mapping URL or error status code.

• Implement a method that handles the error and returns the desired error page view or response.

• Register the custom error controller as a bean in the application context.

• Optionally, configure the error page mapping in the application.properties or application.yml file using properties like server.error.path or server.error.whitelabel.enabled.

28.

How can you implement distributed tracing in a Spring Boot application using OpenTelemetry?

Hide Answer

To implement distributed tracing in a Spring Boot application using OpenTelemetry, you can include the necessary OpenTelemetry dependencies such as open telemetry-API and an OpenTelemetry exporter. Configuring the exporter lets you send trace data to a distributed tracing system like Jaeger or Zipkin.

OpenTelemetry automatically instructs the application to capture and propagate trace information across different services, allowing you to trace the execution path of requests in a distributed system.

29.

How can you implement asynchronous processing in a Spring Boot application?

Hide Answer

Spring Boot provides support for asynchronous processing through the use of the @Async annotation and the TaskExecutor interface. To implement asynchronous processing, follow these steps:

• Configure a TaskExecutor bean in your application's configuration. This bean defines the thread pool or executor service used for executing asynchronous tasks.

• Annotate the methods that need to be executed asynchronously with the @Async annotation.

• Invoke the annotated methods from other parts of your application. The invocation will return a Future object which can be used to obtain the result of the asynchronous task or monitor its progress.

30.

How can you enable HTTPS in a Spring Boot application?

Hide Answer

To enable HTTPS in a Spring Boot application, you need to configure the appropriate SSL certificate and modify the application's configuration. Here are the general steps:

• Obtain an SSL certificate and private key.

• Configure the certificate and private key in the application's configuration, such as application.properties or application.yml, using properties like server.ssl.key-store and server.ssl.key-password.

• Set the server.ssl.enabled property to true to enable HTTPS.

• Optionally, configure other SSL-related properties like the SSL protocol and cipher suites.

• Restart the application for the changes to take effect.

Looking for remote developer job at US companies?

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

**HIBERNATE**

### 1. What is an ORM tool?

An Object Relational Mapping (ORM) tool helps to simplify data creation, manipulation, and access by internally using Java API to interact with the databases. It’s a technique that maps objects stored in a database.

### 2. What does “lightweight” mean?

In the context of computers, “lightweight” describes an app, computer program, or device that doesn’t use many system resources due to its small memory footprint (RAM) and low CPU usage.

### 3. What are the advantages of Hibernate?

Here’s a list of Hibernate’s many advantages:

* It’s fast
* It’s lightweight and open source
* It reduces code length, removing boilerplate code, freeing up developers for other tasks
* It strengthens the object-level relationship
* It facilitates the generation of independent database queries
* It provides resources for creating tables automatically
* It’s easy to integrate with other Java Enterprise Edition (EE) frameworks.

### 4. Why is Hibernate better than Java Database Connectivity (JDBC)?

Hibernate outclasses JDBC because:

* Hibernate code is cleaner and more readable thanks to the elimination of boiler-plate code, something found in JDBC
* Unlike JDBC API, Hibernate supports associations, collections, and inheritances
* HQL (Hibernate Query Language) is closer to Java and is more object-oriented
* Developers don’t need to write code to store and load data into the database
* Hibernate enables faster application development

### 5. What is “persistence”?

In the context of Java, persistence describes data and objects that last beyond the process used to create them.

### 6. Name some databases that Hibernate supports.

Hibernate supports databases like:

* DB2/NT
* FrontBase
* HSQL Database Engine
* Informix Dynamic Server
* Microsoft SQL Server Database
* MySQL
* Oracle
* PostgreSQL
* SQL Server
* Sybase

### 7. What is HQL?

HQL stands for Hibernate Query Language, a powerful object-oriented language independent of the database. It’s like [SQL](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql), except that it uses objects instead of table names. HQL is a very simple, efficient, and flexible query language used to do various operations on a relational database without the need for complex database queries.

### 8. Name the four ORM levels in Hibernate.

Hibernate’s four ORM levels are:

* Full Object Mapping
* Light Object Mapping
* Medium Object Mapping
* Pure Relational

## **Become a Software Development Professional**

#### Full Stack Java Developer

* + Kickstart Full Stack Java Developer career with industry-aligned curriculum by experts
  + Hands-on practice through 20+ projects, assessments, and tests

6 months

[**View Program**](javascript:void(0))

#### Full Stack Web Developer - MEAN Stack

* + Comprehensive Blended Learning program
  + 8X higher interaction in live online classes conducted by industry experts

11 months

[**View Program**](javascript:void(0))

Not sure what you’re looking for?[View all Related Programs](https://www.simplilearn.com/mobile-and-software-development?source=InpageBannerCategory)

### 9. What is a Session in Hibernate?

A Session in Hibernate is a lightweight, non-thread-safe object representing a single unit of work with the database. It is used to perform CRUD operations on persistent objects.

### 10. What is a SessionFactory?

A SessionFactory in Hibernate is a heavyweight and thread-safe object used to create and manage multiple sessions across an application. It is also configuring and managing the underlying connection pooling and caching mechanisms.

### 11. What do you think about the statement - "session being a thread-safe object"?

The statement is incorrect. A Session in Hibernate is a lightweight and non-thread-safe object and should not be shared across multiple threads.

### 12. What is the difference between first-level cache and second-level cache?

The first level cache is associated with a Session and is used to store the currently loaded objects in memory. The second level cache is associated with a SessionFactory and is used to hold objects across multiple sessions, thereby reducing the number of database queries needed.

### 13. What can you tell about the Hibernate Configuration File?

The Hibernate Configuration File (hibernate.cfg.xml) is an [XML file](https://www.simplilearn.com/tutorials/programming-tutorial/what-is-xml) used to configure the basic settings of Hibernate, such as the database URL, username, password, and dialect. It also contains the mapping information of the persistent classes and the resources required for connection pooling.

### 14. How do you create an immutable class in hibernate?

To create an immutable class in Hibernate, you should mark all class properties as 'final' and provide only getter methods for them without any setters. Additionally, you can use the 'mutable' attribute of the 'property' or 'component' element in the Hibernate mapping file to make a property or component immutable.

### 15. Can you explain the concept behind Hibernate Inheritance Mapping?

Hibernate Inheritance Mapping is used to represent the inheritance relationships between classes in a relational database. It allows the developer to map a single table to multiple classes using techniques like table per class hierarchy, table per subclass, and table per concrete class.

### 16. Is hibernate prone to SQL injection attacks?

Hibernate, by itself, is not prone to [SQL injection attacks](https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-sql-injection). However, the application may be vulnerable to SQL injection attacks if user input is concatenated with the HQL or Criteria queries.

## **Intermediate Level Hibernate Interview Questions**

Let’s push the difficulty level up a few notches with this set of eight moderately challenging Hibernate interview questions and answers.

### 17. Name Hibernate’s five collection types used in one-to-many relationship mappings.

The five collection types are:

* Array
* Bag
* List
* Map
* Set

### 18. What is “dirty checking”?

The dirty checking feature helps developers and users avoid time-consuming write actions, thereby reducing database write times. Dirty checking changes or updates only the fields that require action, while keeping the rest of the fields untouched and unchanged.

### 19. What is Hibernate’s default cache service?

Hibernate’s default cache service is EHCache, though the framework additionally supports OSCache, SWARMCache, and TreeCache.

### 20. What is Light Object Mapping?

Light Object Mapping is one of the more valuable levels of ORM quality. This approach uses specific design patterns to hide the syntax from business logic. All entities are represented as classes and mapped manually. The Light Object Mapping approach works well with applications that have fewer entities and applications that use metadata-driven data models.

### 21. List and describe the Hibernate framework’s essential interfaces.

Hibernate’s important interfaces are:

* SessionFactory (org.hibernate.SessionFactory). SessionFactory is an immutable thread-safe cache of compiled mappings meant for a single database. After users initialize SessionFactory once, they can cache and reuse it. SessionFactory is designed to return the session objects for database operations.
* Session (org.hibernate.Session). A session is a single-threaded, short-lived object that represents a dialogue between the persistent store and the application. It is the interface that exists between the Hibernate framework and the Java application code, providing methods for CRUD operations. A session should be opened only when required, then closed as soon as the user is finished.
* Transaction (org.hibernate.transaction). The transaction is a single-threaded, short-lived object that the application uses to specify atomic units of work.

### 22. What is lazy loading?

Lazy loading is a technique where objects are loaded as needed, instead of an entire page, for example. This technique became default since Hibernate version 3.

### 23. What are the concurrency strategies?

Concurrency strategies are mediators responsible for storing and retrieving cached items. When enabling a second-level cache, the developer must decide which cache concurrency to implement for each persistent class and collection.

The concurrency strategies are:

* Non-strict-Read-Write: This strategy works with data that can be altered and can tolerate a small chance of stale data. This strategy offers no guarantee of consistency between the database and the cache.
* Read-Only: This strategy works best with data that can’t be changed, and consequently, is only used to reference data.
* Transactional: This strategy is used primarily for read-mostly data in cases where it’s essential to prevent stale data in concurrent transactions, in those rare instances of an update.
* Read-Write: This strategy is like the transactional strategy.

### 24. Define Hibernate’s validator framework.

Data validation is an integral part of any application and is used in the presentation layer when using JavaScript and server-side code before processing. Validation is a cross-cutting task that occurs before making it persistent so that it adheres to the correct format.

### 25. Explain hibernate mapping file.

A hibernate mapping file is an XML file that defines the relationship between a Java class and a database table. It specifies the mapping of the class properties to the table columns and describes any additional connections between the class and other classes or tables.

### 26. What are the most commonly used annotations available to support hibernate mapping?

The most commonly used annotations for hibernate mapping are @Entity, @Table, @Column, @Id, @GeneratedValue, @ManyToOne, and @OneToMany.

### 27. Explain Hibernate architecture

Hibernate architecture consists of several components, such as the Session Factory, Session, Transaction, and Query. For example, the Session Factory is responsible for creating and managing sessions, interacting with the database, and executing queries.

### 28. Can you tell the difference between the getCurrentSession and openSession methods?

The getCurrentSession method creates or retrieves the current Session from the current session context. In contrast, the open Session process begins a new session that is not bound to the current context.

## **Become a Software Development Professional**

#### Full Stack Java Developer

* + Kickstart Full Stack Java Developer career with industry-aligned curriculum by experts
  + Hands-on practice through 20+ projects, assessments, and tests

6 months

[**View Program**](javascript:void(0))

#### Full Stack Web Developer - MEAN Stack

* + Comprehensive Blended Learning program
  + 8X higher interaction in live online classes conducted by industry experts

11 months

[**View Program**](javascript:void(0))

Not sure what you’re looking for?[View all Related Programs](https://www.simplilearn.com/mobile-and-software-development?source=InpageBannerCategory)

### 29. Differentiate between save() and saveOrUpdate() methods in hibernate Session.

The save() method is used to persist a new object in the database. In contrast, the saveOrUpdate() method is used to continue a new object or update an existing object in the database.

### 30. Differentiate between get() and load() in Hibernate session

The get() method retrieves an object from the database by its [primary key](https://www.simplilearn.com/tutorials/sql-tutorial/primary-key-in-sql) and throws an exception if the object is not found. The load() method is also used to retrieve an object from the database by its primary key, but it returns a proxy object if it is not found.

### 31. What are the criteria for API in hibernate?

The criteria API in hibernate is a programmatic way of creating and executing queries. It allows developers to build complex queries using a fluent interface rather than writing raw SQL.

### 32. Can you tell me something about one too many associations and how we can use them in Hibernate?

A one-to-many association occurs when one entity is associated with multiple other entities. In Hibernate, one-to-many associations can be implemented using the @OneToMany annotation, and the @JoinColumn annotation is used to specify the column that will be used to join the two entities.

### 33. What are Many to Many associations?

A many-to-many association occurs when multiple entities are associated with various other entities. In Hibernate, many-to-many associations can be implemented using the @ManyToMany annotation, and a join table is used to store the relationship between the two entities.

### 34. What does Session.lock() method in hibernate do?

The SessionSession.lock() method is used to acquire a lock on an object in the current Session. This is useful for preventing concurrent updates to the same thing.

### 35. What is hibernate caching?

Hibernate caching refers to storing data in memory to retrieve it quickly without hitting the database again. This improves performance and reduces the load on the database.

### 36. Types of Hibernate Caching

There are two types of hibernate caching: first-level caching and second-level caching. First-level caching is enabled by default and is associated with the Session object. Second-level caching is optional and is associated with the SessionFactory object.

### 37. When is the merge() method of the hibernate Session useful?

The merge() method of the hibernate Session is useful when you want to update an existing object in the database without reattaching it to the Session.

### 38. Collection mapping can be done using One-to-One and Many-to-One Associations. What do you think?

Yes, collection mapping can be done using both One-to-One and Many-to-One associations. One-to-One association is used when one object is associated with one other object. At the same time, the Many-to-One association is used when one object is associated with multiple other objects.

### 39. Can you tell the difference between setMaxResults() and setFetchSize() of Query?

The setMaxResults() method limits the number of results returned by a query, while the setFetchSize() method controls the number of rows retrieved from the database at a time. setMaxResults() is used to limit the total number of results returned, while setFetchSize() controls the number of rows retrieved at a time to avoid memory issues.

### 40. Does Hibernate support Native SQL Queries?

Yes, Hibernate supports Native SQL Queries, which allow you to use SQL statements directly to interact with the database. This can be useful when you want to perform complex queries that are impossible with HQL or Criteria API.

Now that we have learned some of the intermediate level Hibernate interview questions, let us next increase the difficulty level and look into some of the advanced level Hibernate interview questions and answers.

## **Advanced Level Hibernate Interview Questions**

We round out the Hibernate interview questions with eight expert questions.

### 41. What design patterns does the Hibernate framework use?

Some design patterns include:

* Data Mapper, which moves data between objects and a database, keeping them independent of each other and the mapper
* Domain Model Pattern, which is a domain object model that incorporates both behavior and data
* Proxy Pattern, for lazy loading
* Factory pattern in SessionFactory

### 42. What is Hibernate tuning?

The process of Hibernate tuning is designed to optimize Hibernate applications’ performance. The three strategies are:

* SQL Optimization
* Session Management
* Data Caching

### 43. Name the states that a persistent entity exists in.

Persistent entities exist in only three states:

* Transient
* Persistent
* Detached

### 44. How can you view the Hibernate-generated SQL on a console?

To enable viewing SQL on a console for debugging purposes, you must add the following in the Hibernate configuration file:

1 <property name="show\_sql">true</property>

### 45. What’s the difference between Session and SessionFactory?

A Session provides the first-level cache and is a single-threaded, short-lived object. A SessionFactory provides the second-level cache and is immutable and shared by all Sessions. It lives until Hibernate is running.

### 46. How many ways can an object be fetched from Hibernate’s database?

There are four ways to fetch objects from Hibernate’s database:

* Criteria API
* HQL
* The identifier
* Standard SQL

### 47. How many ways can you disable Hibernate’s second-level cache?

There are three ways to disable the cache:

* By setting hibernate. cache. use\_second\_level\_cache property to false
* By using CACHEMODE.IGNORE
* Using a cache provider such as org.hibernate.cache.NoCacheProvider

### 48. Describe the differences between Hibernate’s transient, persistent, and detached states.

Here is how the states differ:

* Transient. This state describes new objects that are created in Java but not associated with a Hibernate session.
* Persistent. This state describes objects associated with a Hibernate session.
* Detached. This state describes an object that was formerly Persistent and associated with a Hibernate session. Developers can reattach the object to a Hibernate session by using either update() or saveOrUpdate().

### 49. What happens when the no-args constructor is absent in the Entity bean?

If the no-args constructor is absent in the Entity bean, Hibernate cannot instantiate the object and throw an exception. Therefore, it is always recommended to have a no-args constructor in the Entity bean for Hibernate to work properly.

### 50. Can we declare the Entity class final?

No, the Entity class cannot be declared final, as Hibernate uses runtime proxies to enhance the class for persistence. Therefore, a final class cannot be subclassed and thus cannot be proxied.

### 51. Explain Query Cache

The Query Cache is a second-level cache that stores the results of a query so that they can be reused later. It improves performance by avoiding multiple times hitting the database for the same Query.

### 52. Can you say something about the N+1 SELECT problem in Hibernate?

The N+1 SELECT problem in Hibernate refers to a single SELECT statement being executed to retrieve the parent object. Then N SELECT statements are executed to retrieve the child objects, resulting in poor performance.

### 53. How to solve the N+1 SELECT problem in Hibernate?

The N+1 SELECT problem can be solved using fetching strategies such as eager loading, lazy loading, or batch fetching. Another solution is to use the JOIN FETCH clause in your HQL or JPQL Query.

### 54. What is a Single Table Strategy?

The Single Table Strategy maps multiple classes that inherit from a single root class to a single table in the database. Each row in the table represents an instance of one of the classes in the hierarchy.

## **Become a Software Development Professional**

#### Full Stack Java Developer

* + Kickstart Full Stack Java Developer career with industry-aligned curriculum by experts
  + Hands-on practice through 20+ projects, assessments, and tests

6 months

[**View Program**](javascript:void(0))

#### Full Stack Web Developer - MEAN Stack

* + Comprehensive Blended Learning program
  + 8X higher interaction in live online classes conducted by industry experts

11 months

[**View Program**](javascript:void(0))

Not sure what you’re looking for?[View all Related Programs](https://www.simplilearn.com/mobile-and-software-development?source=InpageBannerCategory)

### 55. Can you say something about the Table Per Class Strategy?

The Table Per Class Strategy maps classes in an inheritance hierarchy to separate tables in the database. Each class in the hierarchy has its table, and the data for that class is stored in the corresponding table.

### 56. Can you say something about Named SQL Query?

Named SQL Query is a feature in Hibernate that allows you to define a named query and reuse it throughout your application. It is defined in the mapping file or by using the @NamedQuery annotation.

### 57. What are the benefits of NamedQuery?

The benefits of NamedQuery include the following:

* Improved code maintainability and readability.
* Reduced code duplication.
* Ability to easily change the Query without affecting its code.

It also improves performance when the same Query is used multiple times by caching the query results.

**SPRING MVC**

1) What is MVC?

The MVC (Model-View-Controller) is a software architectural design pattern. It separates the functionality of an application into three interconnected parts - Model, View, and Controller. This approach facilitates the reusability of the code and parallel development.

2) What is Spring MVC?

A Spring MVC is a Java Framework which is used to develop dynamic web applications. It implements all the basic features of a core spring framework like Inversion of Control and Dependency Injection. It follows the Model-View-Controller design pattern.



Here,

* **Model** - A model contains the data of the application. Data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. So, we can create a view page by using view technologies like JSP+JSTL, Apache Velocity, Thymeleaf, and FreeMarker.

3) What is the front controller of Spring MVC?

The front controller is a **DispatcherServlet** class present in **org.springframework.web.servlet** package. It dispatches the request to the appropriate controller and manages the flow of the application. It is required to specify the **DispatcherServlet** class in the web.xml file.

4) Explain the flow of Spring MVC?



* Once the request has been generated, it intercepted by the DispatcherServlet that works as the front controller.
* The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
* The controller returns an object of ModelAndView.
* The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

5) What are the advantages of Spring MVC Framework?

The following are the advantages of Spring MVC Framework : -

* **Separate roles** - The Spring MVC separates the application into three interconnected layers where each layer has its role.
* **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
* **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
* **Rapid development** - The Spring MVC facilitates fast and parallel development.
* **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
* **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

6) What does an additional configuration file contain in Spring MVC application?

The Spring MVC application contains an additional configuration file that contains the properties information. This file can be created either in the form of **an xml** file or **properties** file. In this file, we generally define the base-package and view resolver where **DispatcherServlet** searches for the controller classes and view components path. However, it can also contain various other configuration properties.

7) What is an InternalResourceViewResolver in Spring MVC?

The **InternalResourceViewResolver** is a class which is used to resolve internal view in Spring MVC. Here, you can define the properties like prefix and suffix where prefix contains the location of view page and suffix contains the extension of view page. For example:-

1. **<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
2. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
3. **<property** name="suffix" value=".jsp"**></property>**
4. **</bean>**

8) How to declare a class as a controller class in Spring MVC?

The @Controller annotation is used to declare a class as a controller class. It is required to specify this annotation on the class name. For example:-

1. @Controller
2. class Demo
3. {
5. }

9) How to map controller class and its methods with URL?

The **@RequestMapping** annotation is used to map the controller class and its methods. You can specify this annotation on the class name as well as method name with a particular URL that represents the path of the requested page. For example:-

1. @Controller
2. @RequestMapping("/ form")
3. class Demo
4. {
5. @RequestMapping("/show")
6. public String display()
7. {
9. }
11. }

10) Name the annotations used to handle different types of incoming HTTP request methods?

The following annotations are used to handle different types of incoming HTTP request methods: -

* @GetMapping
* @PostMapping
* @PutMapping
* @PatchMapping
* @DeleteMapping

11) What is the purpose of @PathVariable annotation in Spring MVC?

The @PathVariable annotation is used to extract the value of the URI template. It is passed within the parameters of the handler method.

For example :-

1. @RequestMapping("/show/{id}")
2. public String handler(@PathVariable("id") String s, Model map)
3. {
4. }

12) What is the role of @ResponseBody annotation in Spring MVC?

The @ResponseBody annotation is used to serialize the returned object automatically in JSON and bind it with the Http response body. Here, it not required to invoke the model.

For example :-

1. @RequestMapping("/show")
2. @ResponseBody
3. public ResponseHandler display(
4. @RequestBody ShowForm form) {
5. return new ResponseHandler("display form");
6. }
7. }

13) What is the role of the Model interface in Spring MVC?

The **Model** interface works as a container that contains the data of the application. Here, data can be in any form such as objects, strings, information from the database, etc.

[click here for more details](https://www.javatpoint.com/spring-mvc-model-interface)

14) What do you mean by ModelAndView in Spring MVC?

The **ModelAndView** is a class that holds both Model and View where the model represents the data, and view represents the representation of that data. This class returns the model and view in the single return value.

15) What is ModelMap in Spring MVC?

The **ModelMap** is a class that provides the implementation of Map. It extends the LinkedHashMap class. It facilitates to pass a collection of values as if they were within a Map.

16) What are the ways of reading data from the form in Spring MVC?

The following ways to read the data from the form are: -

ADVERTISEMENT

* **HttpServletRequest interface** - The **HttpServletRequest** is a java interface present in javax.servlet.http package. Like Servlets, you can use HttpServletRequest in Spring to read the HTML form data provided by the user.
* **@RequestParam annotation** - The **@RequestParam** annotation reads the form data and binds it automatically to the parameter present in the provided method.
* **@ModelAttribute annotation** - The **@ModelAttribute** annotation binds a method parameter or its return value to a named model attribute.

17) What is Spring MVC form tag library?

The Spring MVC form tags can be seen as data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it. These tags are the configurable and reusable building blocks for a web page. It provides view technologies, an easy way to develop, read, and maintain the data.

[click here for more details](https://www.javatpoint.com/spring-mvc-form-tag-library)

18) What do you understand by validations in Spring MVC?

The validation is one of the most important features of Spring MVC, that is used to restrict the input provided by the user. To validate the user's input, it is required to use the Spring 4 or higher version and Bean Validation API. Spring validations can validate both server-side as well as client-side applications.

19) What is Bean Validation API?

The **Bean Validation API** is a Java specification which is used to apply constraints on object model via annotations. Here, we can validate a length, number, regular expression, etc. Apart from that, we can also provide custom validations.

As Bean Validation API is just a specification, it requires an implementation. So, for that, it uses Hibernate Validator. The Hibernate Validator is a fully compliant JSR-303/309 implementation that allows to express and validate application constraints.

20) What is the use of @Valid annotation in Spring MVC?

The **@Valid** annotation is used to apply validation rules on the provided object.

21) What is the purpose of BindingResult in Spring MVC validations?

The **BindingResult** is an interface that contains the information of validations. For example :-

1. @RequestMapping("/helloagain")
2. public String submitForm( @Valid @ModelAttribute("emp") Employee e, BindingResult br)
3. {
4. if(br.hasErrors())
5. {
6. return "viewpage";
7. }
8. else
9. {
10. return "final";
11. }
12. }

22) How to validate user's input within a number range in Spring MVC?

In Spring MVC Validation, we can validate the user's input within a number range by using the following annotations: -

* **@Min annotation** - It is required to pass an integer value with @Min annotation. The user input must be equal to or greater than this value.
* **@Max annotation** - It is required to pass an integer value with @Max annotation. The user input must be equal to or smaller than this value.

[click here for more details](https://www.javatpoint.com/spring-mvc-number-validation)

23) How to validate the user input in a particular sequence in Spring MVC?

The Spring MVC Validation allows us to validate the user input in a particular sequence by using @Pattern annotation. Here, we can provide the required regular expression to **regexp** attribute and pass it with the annotation.

[click here for more details](https://www.javatpoint.com/spring-mvc-regular-expression-validation)

24) What is the purpose of custom validations in Spring MVC?

The Spring MVC framework allows us to perform custom validations. In such a case, we declare our own annotations. We can perform validation based on own business logic.

[click here for more details](https://www.javatpoint.com/spring-mvc-custom-validation)

25) What do you understand by Spring MVC Tiles?

The Spring provides integration support with apache tiles framework. So we can manage the layout of the Spring MVC application with the help of spring tiles support. The following are the advantages of Tiles support in Spring MVC: -

* **Reusability:** We can reuse a single component in multiple pages like header and footer components.
* **Centralized control:** We can control the layout of the page by a single template page only.
* **Easy to change the layout:** By the help of a single template page, we can change the layout of the page anytime. So, your website can easily adopt new technologies such as bootstrap and jQuery.

**SPRING**

### 1) What is Spring?

It is a lightweight, loosely coupled, and integrated framework for developing enterprise applications in java.

### 2) What are the advantages of spring framework?

1. Predefined Templates
2. Loose Coupling
3. Easy to test
4. Lightweight
5. Fast Development
6. Powerful Abstraction
7. Declarative support

[More details...](https://www.javatpoint.com/spring-tutorial)

### 3) What are the modules of spring framework?

1. Test
2. Spring Core Container
3. AOP, Aspects and Instrumentation
4. Data Access/Integration
5. Web

[More details...](https://www.javatpoint.com/spring-modules)

### 4) What is IOC and DI?

IOC (Inversion of Control) and DI (Dependency Injection) is a design pattern to provide loose coupling. It removes the dependency from the program.

Let's write a code without following IOC and DI.

1. **public** **class** Employee{
2. Address address;
3. Employee(){
4. address=**new** Address();//creating instance
5. }
6. }

Now, there is dependency between Employee and Address because Employee is forced to use the same address instance.

Let's write the IOC or DI code.

1. **public** **class** Employee{
2. Address address;
3. Employee(Address address){
4. **this**.address=address;//not creating instance
5. }
6. }

Now, there is no dependency between Employee and Address because Employee is not forced to use the same address instance. It can use any address instance.

### 5) What is the role of IOC container in spring?

IOC container is responsible to:

ADVERTISEMENT

ADVERTISEMENT

* create the instance
* configure the instance, and
* assemble the dependencies

[More details...](https://www.javatpoint.com/ioc-container)

### 6) What are the types of IOC container in spring?

There are two types of IOC containers in spring framework.

1. BeanFactory
2. ApplicationContext

[More details...](https://www.javatpoint.com/ioc-container)

### 7) What is the difference between BeanFactory and ApplicationContext?

BeanFactory is the **basic container** whereas ApplicationContext is the **advanced container**. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc.

ADVERTISEMENT

ADVERTISEMENT

### 8) What is the difference between constructor injection and setter injection?

|  |  |  |
| --- | --- | --- |
| **No.** | **Constructor Injection** | **Setter Injection** |
| 1) | No Partial Injection | Partial Injection |
| 2) | Desn't override the setter property | Overrides the constructor property if both are defined. |
| 3) | Creates new instance if any modification occurs | Doesn't create new instance if you change the property value |
| 4) | Better for too many properties | Better for few properties. |

[More details...](https://www.javatpoint.com/difference-between-constructor-and-setter-injection)

### 9) What is autowiring in spring? What are the autowiring modes?

Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic.

Let's see the code to inject bean using dependency injection.

1. <bean id="emp" **class**="com.javatpoint.Employee" autowire="byName" />

The autowiring modes are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | this is the default mode, it means autowiring is not enabled. |
| 2) | byName | injects the bean based on the property name. It uses setter method. |
| 3) | byType | injects the bean based on the property type. It uses setter method. |
| 4) | constructor | It injects the bean using constructor |

The "autodetect" mode is deprecated since spring 3.

### 10) What are the different bean scopes in spring?

There are 5 bean scopes in spring framework.

|  |  |  |
| --- | --- | --- |
| **No.** | **Scope** | **Description** |
| 1) | singleton | The bean instance will be only once and same instance will be returned by the IOC container. It is the default scope. |
| 2) | prototype | The bean instance will be created each time when requested. |
| 3) | request | The bean instance will be created per HTTP request. |
| 4) | session | The bean instance will be created per HTTP session. |
| 5) | globalsession | The bean instance will be created per HTTP global session. It can be used in portlet context only. |

### 11) In which scenario, you will use singleton and prototype scope?

Singleton scope should be used with EJB **stateless session bean** and prototype scope with EJB **stateful session bean**.

### 12) What are the transaction management supports provided by spring?

Spring framework provides two type of transaction management supports:

ADVERTISEMENT

ADVERTISEMENT

1. **Programmatic Transaction Management**: should be used for few transaction operations.
2. **Declarative Transaction Management**: should be used for many transaction operations.

## **» Spring JDBC Interview Questions**

### 13) What are the advantages of JdbcTemplate in spring?

**Less code**: By using the JdbcTemplate class, you don't need to create connection,statement,start transaction,commit transaction and close connection to execute different queries. You can execute the query directly.

[More details...](https://www.javatpoint.com/spring-JdbcTemplate-tutorial)

### 14) What are classes for spring JDBC API?

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

[More details...](https://www.javatpoint.com/spring-JdbcTemplate-tutorial)

### 15) How can you fetch records by spring JdbcTemplate?

ADVERTISEMENT

ADVERTISEMENT

You can fetch records from the database by the **query method of JdbcTemplate**. There are two interfaces to do this:

1. [ResultSetExtractor](https://www.javatpoint.com/ResultSetExtractor-example)
2. [RowMapper](https://www.javatpoint.com/RowMapper-example)

### 16) What is the advantage of NamedParameterJdbcTemplate?

NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than ? (question mark of PreparedStatement).

ADVERTISEMENT

ADVERTISEMENT

It is **better to remember**.

[More details...](https://www.javatpoint.com/spring-NamedParameterJdbcTemplate-example)

### 17) What is the advantage of SimpleJdbcTemplate?

The **SimpleJdbcTemplate** supports the feature of var-args and autoboxing.

[More details...](https://www.javatpoint.com/spring-SimpleJdbcTemplate-example)

## **» Spring AOP Interview Questions**

### 18) What is AOP?

ADVERTISEMENT

ADVERTISEMENT

AOP is an acronym for Aspect Oriented Programming. It is a methodology that divides the program logic into pieces or parts or concerns.

It increases the modularity and the key unit is Aspect.

[More details...](https://www.javatpoint.com/spring-aop-tutorial)

### 19) What are the advantages of spring AOP?

AOP enables you to dynamically add or remove concern before or after the business logic. It is **pluggable** and **easy to maintain**.

[More details...](https://www.javatpoint.com/spring-aop-tutorial)

### 20) What are the AOP terminology?

AOP terminologies or concepts are as follows:

* JoinPoint
* Advice
* Pointcut
* Aspect
* Introduction
* Target Object
* Interceptor
* AOP Proxy
* Weaving

[More details...](https://www.javatpoint.com/spring-aop-tutorial)

### 21) What is JoinPoint?

JoinPoint is any point in your program such as field access, method execution, exception handling etc.

### 22) Does spring framework support all JoinPoints?

No, spring framework supports method execution joinpoint only.

### 23) What is Advice?

ADVERTISEMENT

ADVERTISEMENT

Advice represents action taken by aspect.

ADVERTISEMENT

ADVERTISEMENT

### 24) What are the types of advice in AOP?

There are 5 types of advices in spring AOP.

1. Before Advice
2. After Advice
3. After Returning Advice
4. Throws Advice
5. Around Advice

### 25) What is Pointcut?

Pointcut is expression language of Spring AOP.

### 26) What is Aspect?

Aspect is a class in spring AOP that contains advices and joinpoints.

### 27) What is Introduction?

Introduction represents introduction of new fields and methods for a type.

### 28) What is target object?

Target Object is a proxy object that is advised by one or more aspects.

### 29) What is interceptor?

Interceptor is a class like aspect that contains one advice only.

ADVERTISEMENT

ADVERTISEMENT

### 30) What is weaving?

Weaving is a process of linking aspect with other application.

### 31) Does spring perform weaving at compile time?

No, spring framework performs weaving at runtime.

### 32) What are the AOP implementation?

There are 3 AOP implementation.

1. Spring AOP
2. Apache AspectJ
3. JBoss AOP

## **» Spring MVC Interview Questions**

### 33) What is the front controller class of Spring MVC?

The **DispatcherServlet** class works as the front controller in Spring MVC.

[More details...](https://www.javatpoint.com/spring-3-mvc-tutorial)

### 34) What does @Controller annotation?

The **@Controller** annotation marks the class as controller class. It is applied on the class.

### 35) What does @RequestMapping annotation?

The **@RequestMapping** annotation maps the request with the method. It is applied on the method.

ADVERTISEMENT

ADVERTISEMENT

### 36) What does the ViewResolver class?

The **View Resolver** class resolves the view component to be invoked for the request. It defines prefix and suffix properties to resolve the view component.

### 37) Which ViewResolver class is widely used?

The **org.springframework.web.servlet.view.InternalResourceViewResolver** class is widely used.

### 38) Does spring MVC provide validation support?

Yes.

**SPRING SECURITY**

## **Explain Spring Security Architecture using Spring Boot?**

Let us understand how Spring Security Works.  
A diagram of a security system

Description automatically generated  
[Understand Spring Security Architecture and implement Spring Boot Security](https://www.javainuse.com/boot3/sec/3)

## **How is Security mechanism implemented using Spring**

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.  
**Spring makes use of the DelegatingFilterProxy for implementing security mechanisms.** It is a Proxy for standard Servlet Filter, delegating to a Spring-managed bean that implements the Filter interface. Its the starting point in the springSecurityFilterChain which instantiates the Spring Security filters according to the Spring configuration  
Some of the features of Spring Security are

* Comprehensive and extensible support for both Authentication and Authorization
* Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
* Servlet API integration Optional integration with Spring Web MVC

## **What is OAuth2 Authorization code grant type? How to implement it using Spring Boot Security?**

OAuth (Open Authorization) is a simple way to publish and interact with protected data.  
It is an open standard for token-based authentication and authorization on the Internet. It allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password.  
The OAuth specification describes five grants for acquiring an access token:

* Authorization code grant
* Implicit grant
* Resource owner credentials grant
* Client credentials grant
* Refresh token grant

Consider the use case of Quora. Go to Quora.com.  
If you are a new user you need to signup. You can signup using google or facebook account. When doing so you are authorizing Google or Facebook to allow quora to access you profile info with Quora. **This authorizing is done using OAuth**. Here you have in no way shared your credentials with Quora.  
[Understanding What Is OAuth2](https://www.javainuse.com/spring/spring-boot-oauth-introduction)  
[Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)  
[Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to fetch data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)

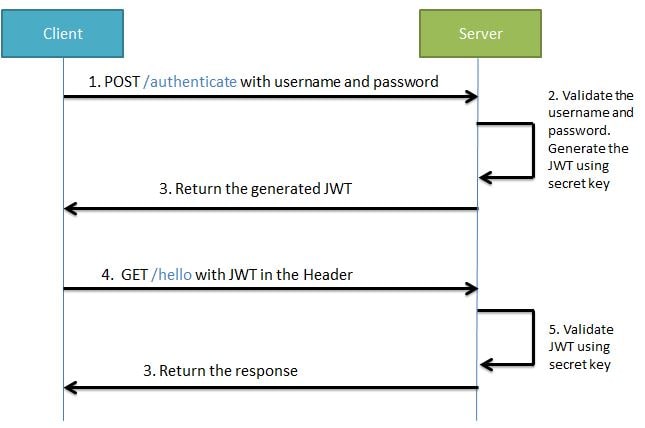
## **Using Spring Boot Security how to refresh expired JSON Web Token?**

In [previous tutorial we had implemented Spring Boot + JWT Example](https://www.javainuse.com/webseries/spring-security-jwt/chap4). We had also covered the topic of JWT Expiration. We had implemented the solution such that if the JWT has expired then the user gets JWTExpiredException.  
Suppose our requirement is such that if the token has expired, still the user should be allowed to access the system if the token is valid. That is the token should be refreshed or a new valid token should be provided.  
We will be working on a solution where if the user he receives JWT expired exception, then he can call another API with the expired token. A new token will then provided to the user which he can use for future interactions. Previously we had implemented an example for [programmatically consuming the JWT secure API using Spring RestTemplate](https://www.javainuse.com/webseries/spring-security-jwt/chap6). We will be testing this refresh Token generation API both using Postman as well as the Spring RestTemplate.  
A diagram of a server

Description automatically generated

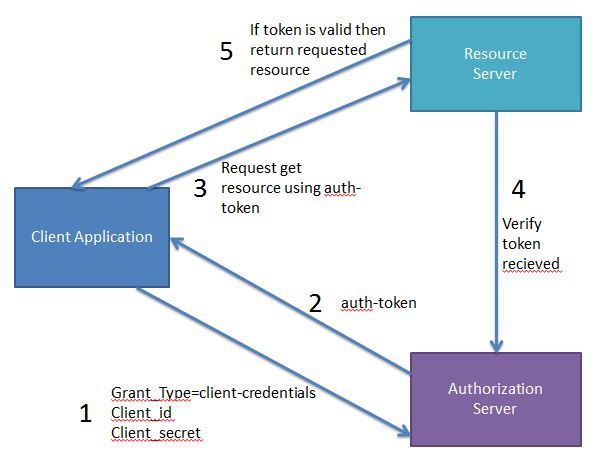
## **What is JWT ? How to implement it using Spring Boot Security?**

For better understanding we will be developing the project in stages

* Develop a Spring Boot Application to expose a Simple REST GET API with mapping /hello.
* Configure Spring Security for JWT. Expose REST POST API with mapping /authenticate using which User will get a valid JSON Web Token. And then allow the user access to the api /hello only if it has a valid token  
  

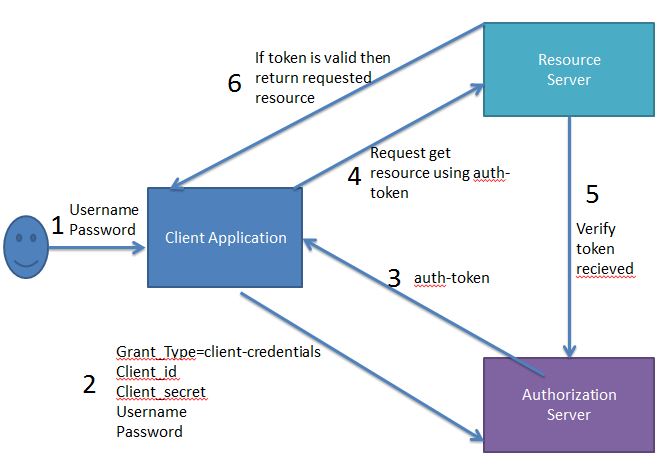
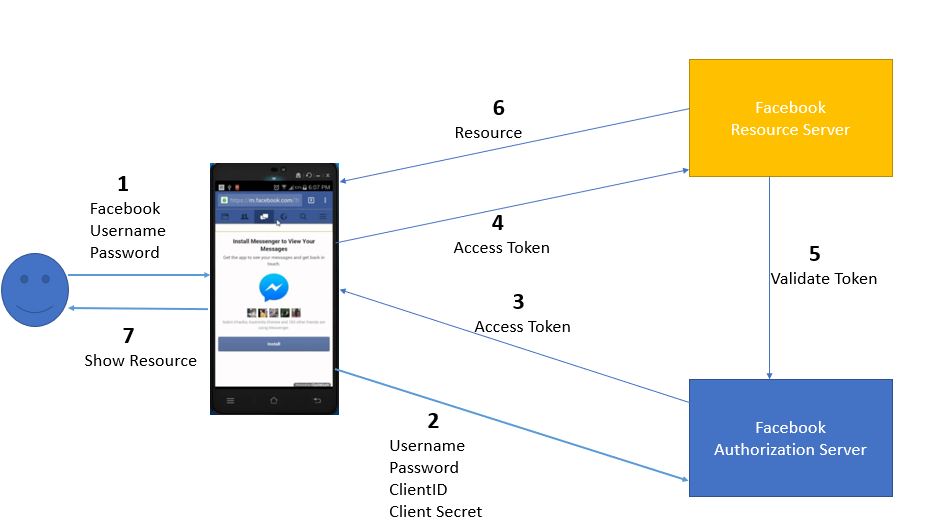
[What is JWT(JSON Web Token)](https://www.javainuse.com/spring/jwt)  
[Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)

## **What is OAuth2 Client Credentials Grant? How to implement it using Spring Boot Security?**

The Client Credentials Grant involves machine to machine authentication. In case of Client credentials grant type the user has no role to play. As previously stated it is machine to machine communication. This is typically used by clients to access resources about themselves rather than to access a user's resources.  


This type of Authentication does not involve any end-user. Unlike Authorization Grant where the end user had to authenticate himself using Authorization Server like Gmail, here the machine it self authenticates itself to access a protected resource.  
[Spring Boot + OAuth 2 Client Credentials Grant - Hello World Example.](https://www.javainuse.com/spring/springboot-oauth2-client-grant)

## **What is OAuth2 Password Grant? How to implement it using Spring Boot Security**

In case of Password grant type the user triggers the client to get some resource. While doing so it passes the username and password to the client. The client then communicates with the authorization server using the provided username, password and also its own clientId and clientSecret to get the access token. Using this access token it then gets the required resource from the resource server.  
  
The real life example of Password grant will be you doing a login to you facebook account using its mobile application. Here the user will have to specify the facebook credentials to the app. Also the app will be having its own client id and client secret.  
  
[Spring Boot + OAuth 2 Password Grant - Hello World Example.](https://www.javainuse.com/spring/springboot-oauth2-password-grant)

## **How to configure Spring Security using Spring Boot?**

[Spring Boot + Simple Security Configuration](https://www.javainuse.com/spring/sprboot_sec)

## **How to use Form Login Authentication using Spring Boot**

We make use of Spring Boot Security to get default login page and authentication users.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome")

.hasAnyRole("USER", "ADMIN").antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN")

.antMatchers("/addNewEmployee").hasAnyRole("ADMIN").anyRequest().authenticated().and().formLogin()

.permitAll().and().logout().permitAll();

http.csrf().disable();

}

[Spring Boot Form Security Login Hello World Example](https://www.javainuse.com/spring/boot_form_security)

## **How to create Custom Login Page using Spring Boot Security?**

We can create our own custom login page and use it for authentication.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome").hasAnyRole("USER", "ADMIN")

.antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN").antMatchers("/addNewEmployee")

.hasAnyRole("ADMIN").anyRequest().authenticated()

.and().formLogin().**loginPage("/login")**.permitAll()

.and().logout().permitAll();

http.csrf().disable();

}

[Spring Boot Security - Custom Login Page Example](https://www.javainuse.com/spring/boot_form_security_custom_login)

## **How to do authentication against database tables using Spring Boot Security?**

Spring Authentication using username, password and authorization using roles can be done using either

* In Memory Configuration -
* @Autowired
* public void configureGlobal(AuthenticationManagerBuilder authenticationMgr) throws Exception {
* authenticationMgr.inMemoryAuthentication().withUser("employee").password("employee")
* .authorities("ROLE\_USER").and().withUser("javainuse").password("javainuse")
* .authorities("ROLE\_USER", "ROLE\_ADMIN");
* }

[Spring Boot Security In Memory Authentication Example](https://www.javainuse.com/spring/boot_form_security_custom_login)

* Database Authentication-
* @Autowired
* public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {
* auth.jdbcAuthentication().dataSource(dataSource);
* }

[Spring Boot Security - JDBC Authentication Example](https://www.javainuse.com/spring/boot_security_jdbc_authentication)

## **How to configure Spring Security with in-memory configuration?**

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth)

throws Exception {

auth.inMemoryAuthentication()

.withUser("user").password("password").roles("USER")

.and()

.withUser("admin").password("password").roles("USER", "ADMIN");

## **What is the use of Spring Boot Security AuthenticationHandler class?**

In some scenarios we might want to redirect different users to different pages depending on the roles assigned to the users.  
For example we might want users with role USER to be redirected to the welcome page, while users with role ADMIN to be redirected to the add employee page.  
We will be making use of the AuthenticationSuccessHandler.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome").hasAnyRole("USER", "ADMIN")

.antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN").antMatchers("/addNewEmployee")

.hasAnyRole("ADMIN").anyRequest().authenticated()

.and().formLogin().**successHandler(successHandler)**

.loginPage("/login").permitAll().and().logout().permitAll();

http.csrf().disable();

}

[Spring Boot Form Security Login Hello World Example](https://www.javainuse.com/spring/boot_form_security)

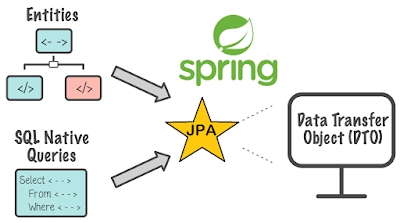
## **What is the difference between ROLE\_USER and ROLE\_ANONYMOUS in a Spring intercept url configuration?**

* **ROLE\_ANONYMOUS** is the default role assigned to an unauthenticated (anonymous) user when a configuration uses Spring Security's "anonymous authentication" filter . This is enabled by default. However, it is probably clearer if you use the expression isAnonymous() instead, which has the same meaning.
* **ROLE\_USER** has no meaning unless you assign this role to your users when they are authenticated (you are in charge of loading the roles (authorities) for an authenticated user). It isn't a name that is built in to Spring Security's infrastructure. In the given example, presumably that role is assigned to an authenticated user.

**SPRING JPA**

**1. What is JPA?**  
Answer: JPA stands for Java Persistence API. It is a Java specification used to persist data between the relational database and Java objects. It acts as a bridge between object-oriented domain models and relational databases.  Since interaction with database from Java application is very common, JPA was created to standardize this interaction.

There are many popular JPA implementations available in the Java world like Hibernate. You can further see these [**Spring Data JPA using Hibernate course**](https://medium.com/javarevisited/5-best-spring-data-jpa-courses-for-java-developers-45e6438be3c9)to learn more about how to use Hibernate with Spring Data JPA in Java application.

[](https://javarevisited.blogspot.com/2020/06/10-advanced-spring-framework-courses.html)

**2. What are some advantages of using JPA?**

Answer: Here are some advantages of Java Persistence API or JPA:

* JPA reduces the burden of interacting with databases.
* Annotation in JPA reduces the cost of creating a definition file.
* It is user-friendly.
* JPA providers help merge applications.

**3. What is the Spring data repository? (**[**answer**](https://javarevisited.blogspot.com/2021/10/what-is-spring-data-repository.html)**)**  
Answer: Spring data repository is a very important feature of JPA. It helps in reducing a lot of boilerplate code. Moreover, it decreases the chance of errors significantly. This is also the key abstraction that is provided using the Repository interface. It takes the domain class to manage as well as the id type of the domain class as Type Arguments.   
  
  
**4. What is the naming convention for finder methods in the Spring data repository interface?**  
Answer:  This is another key feature of Spring Data JPA API which makes writing query method really easy. The finder method should use a special keyword, i.e. "find", followed by the name of the variable. For example, findByLastName().  
  
  
**5. Why is an interface not a class?**  
Answer: Interface is not a class because it does not contain concrete methods. It can contain only abstract methods.  
  
  
**6. Can we perform actual tasks like access, persist, and manage data with JPA?**  
Answer: No, we can't because JPA is only a Java specification.

**7. How can we create a custom repository in Spring data JPA?**  
Answer: To create a custom repository, we have to extend it to any of the following interfaces:  
a) Repository  
b) PagingAndSortingRepository  
c) CrudRepository  
d) JpaRepository  
e) QueryByExampleRepository  
  
  
**8. What is PagingAndSortingRepository? (**[**answer**](https://www.java67.com/2023/08/difference-between-jparepository.html)**)**  
Answer: The PagingAndSortingRepository provides methods that are used to retrieve entities using pagination and sorting. It extends the CrudRepository interface.  
  
  
**9. What is @Query used for? (**[**example**](https://javarevisited.blogspot.com/2021/09/spring-data-jpa-query-example-tutorial.html)**)**  
Answer:  Spring Data API provides many ways to define SQL query which can be executed and Query annotations one of them. The @Query is an annotation that is used to execute both JPQL and native SQL queries.  
  
**10. Give an example of using @Query annotation with JPQL.**  
Answer: Here is an example of @Query annotation from Spring Data Application which returns all active orders from the database:

@Query("**SELECT** **order** **FROM** Orders o **WHERE** o.Disabled= 0")

Collection<User> findAllActiveOrders();

and, here is another example, which returns matching employees from the database

@Query("select e from Employee e where se.name = ?1")

List<Employee> getEmployees(String name);

You can further see [**Spring Framework: Spring Data JPA** course](https://javarevisited.blogspot.com/2020/05/top-5-cloud-courses-for-java-and-spring-boot-developers.html#axzz6UV11QHE1) on Pluralsight to learn more about @Query annotation of Spring Data JPA.

[](https://medium.com/javarevisited/12-advanced-spring-framework-courses-for-java-programmers-a273f6e4448c)

**11. Can you name the different types of entity mapping.**  
Answer: one-to-one mapping, one-to-many mapping, many-to-one mapping, and many-to-many mapping.  
  
**12. Define entity and name the different properties of an entity.**  
Answer: An entity is a group of states bundled (or associated) together in a single unit. It behaves like an object. It also becomes a major constituent of the object-oriented paradigm.  
  
  
**13. What is PlatformTransactionMangaer?**  
Answer: PlatformTransactionMangaer is an interface that extends TransactionManager. It is the central interface in Spring's transaction infrastructure.  
  
  
**14. How can we enable Spring Data JPA features?**  
Answer: To enable Spring data JPA features, first we have to define a configuration class and then, we can use @EnableJpaRepositoties annotation with it. This annotation will enable the features.

**15. Differentiate between findById() and getOne().**  
Answer: The findById() is available in CrudRepository while getOne() is available in JpaRepository. The findById() returns null if record does not exist while the getOne() will throw an exception called EntityNotFoundException.