Top 140 Java Interview Questions Answers for 3 to 5 Years Experienced Programmers

Hello guys, Time is changing and so is Java interviews. Gone are the days, when knowing the difference between String and StringBuffer can help you to go through the second round of interview, questions are becoming more advanced and interviewers are asking more deep questions.

When I started my career, questions like Vector vs Array and HashMap vs Hashtable were the most popular ones and just memorizing them gives you a good chance to do well in interviews, but not anymore.

Nowadays, you will get questions from the areas where not many Java programmer looks e.g. NIO, patterns, sophisticated unit testing or those which are hard to master e.g. concurrency, algorithms, data structures and coding. Since I like to explore interview questions, I have got this huge list of questions with me, which contains lots and lots of questions on different topics. I have been preparing this MEGA list for quite some time and now It's ready to share with you guys. It contains interview questions not only from classic topics like threads, collections, equals and hashcode, sockets but also from NIO, array, string, Java 8, and much more. It has questions fo**r both entry-level Java programmers and senior developers** with years of experience. No matter whether you are a Java developer of 1, 2, 3, 4, 5, 6, 8, 9, or even 10 years of experience, you will find something interesting in this list.

It contains questions that are super easy to answer, and also, a question that is tricky enough for even seasoned Java programmers. If you need more questions then you can also check out thes**e Java Interview Courses** and **books**to better prepare yourself.

Given the list is long and we have questions from everywhere, it's imperative that answers must be short, concise, and crisp, with no fluffing at all.

So apart from this paragraph, you will only hear from me in the questions and answers, no more context, no more feedback, and no more evaluation. For that, I have already written blog posts, where you can find my views on a particular question, e.g. why I like that question, what makes them challenging, and what kind of answer you should expect from candidates. This list is a little bit different and I encourage you to share questions and answers in a similar way so that it should be easy to revise.

I hope this list can be of great use for both interviewer and candidates, the interviewer can, of course, but a little variety on questions to bring novelty and surprise element, which is important for a good interview.

While a candidate, can expand and test their knowledge about key areas of Java programming language and platform.

Nowadays and in coming years the focus will be more on advanced concurrency concepts, JVM internals, 32-bit vs 64-bit JVM, unit testing, and clean code. I am sure, once you read through this *MEGA list of Java interview questions*, you should be able to do well on both telephonic and face-to-face programming interviews.

22 Essential Topics for Java Developer Interviews

Apart from quantity, as you can see with a huge number of questions, I have worked hard to maintain quality as well. I have not only shared questions from all important topics but also ensured to include so-called advanced topics which many programmers do not prefer to prepare or just left out because they have not worked on that. Java NIO and JVM internals questions are the best examples of that. You can keep design patterns also on the same list but a growing number of experienced programmers are now well aware of GOF design patterns and when to use them.

I have also worked hard to keep this list up-to-date to include what interviewers are asking nowadays and what will be their core focus in the coming years. To give you an idea, this list of Java interview questions includes the following topics:

1. Multithreading, concurrency, and thread basics
2. Date type conversion and fundamentals
3. Garbage Collection
4. Java Collections Framework
5. Array
6. String
7. GOF Design Patterns
8. SOLID design principles
9. Abstract class and interface
10. Java basics like equals() and hashcode
11. Generics and Enum
12. Java IO and NIO
13. Common Networking protocols
14. Data structure and algorithm in Java
15. Regular expressions
16. JVM internals

17. Java Best Practices
18. JDBC
19. Date, Time, and Calendar
20. XML Processing in Java
21. JUnit
22. Programming

You guys are also lucky that nowadays there are some good books available to prepare for Java interviews, one of them which I particularly find useful and interesting to read is **Java Programming Interview Exposed**by Markham. It will take you to some of the most important topics for Java and JEE interviews, worth reading even if you are not preparing for a Java interview.

134 Java Interview Questions Answers for 3 to 5 Years Experienced Programmers

So now the time has come to introduce you to this *MEGA list of 120 Java questions* collected from various interviews of last 5 years. I am sure you have seen many of these questions personally on your interviews and many of you would have answered them correctly as well.

1. Multithreading, Concurrency and Thread basics Questions

Let's start with the toughest topic on Java interviews, multithreading and concurrency, once you master this topic, you are already ahead of many Java developers who don't understand threads in depth.

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) 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.

The volatile modifier 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.

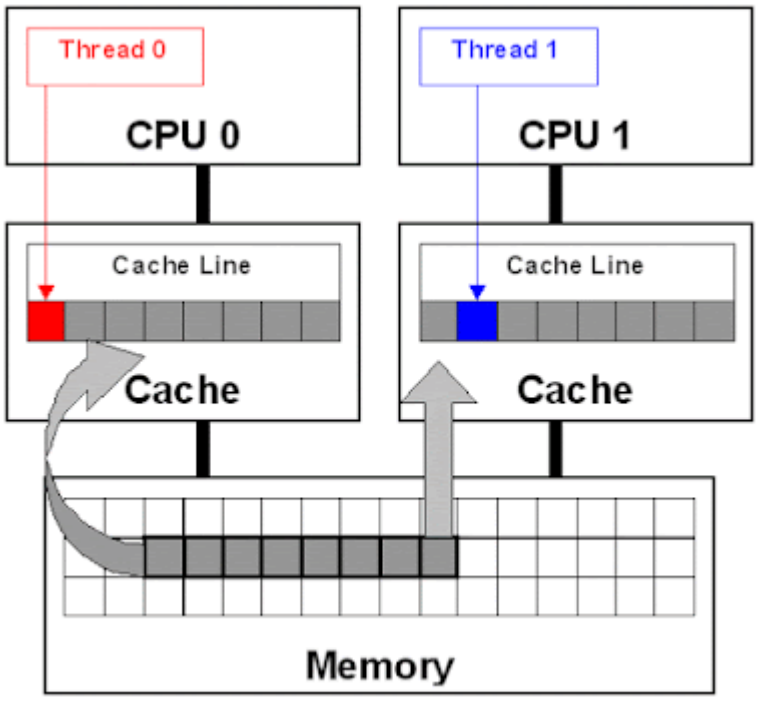
Choice of synchronization though depends upon a number of threads because the number of thread present more contention, so you go for advanced synchronization technique e.g. lock stripping, which requires more complex code and expertise. **6) How do you call wait() method? using if block or loop? Why?** (answer) The `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** to learn more about why wait method should call in the loop.

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



False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory.

Like many concurrency issues, the primary way to avoid false sharing is careful code review and **aligning your data structure with the size of a cache line.**

This question is quite popular on low latency Java developer interviews on Investment banks, hedge funds, HFT companies and Crypto exchanges. **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 framework, a high-performance inter-thread messaging library has a BusySpinWaitStrategy which is based on this concept and uses a busy spin loop for EventProcessors waiting on the barrier. **9) How do you take thread dump in Java?** You can take a thread dump of Java application in Linux by using **kill -3 PID**, where PID is the process id of Java process. In Windows, you can press **Ctrl + Break**.

This will instruct JVM to print thread dump in standard out or err and it could go to console or log file depending upon your application configuration. If you have used Tomcat then when **10) is Swing thread-safe?** (answer) No, Swing is not thread-safe. You cannot update Swing components e.g. JTable, JList or JPanel from any thread, in fact, they must be updated from GUI or AWT thread.

That's why swings provide `invokeAndWait()` and `invokeLater()` method to request GUI update from any other threads.

This methods put update request in AWT threads queue and can wait till update or return immediately for an asynchronous update. You can also check the detailed answer to learn more. **11) What is a thread local variable in Java?** (answer) Thread-local variables are variables confined to a thread, its like thread's own copy which is not shared between multiple threads. Java provides a ThreadLocal class to support thread-local variables. It's one of the many ways to achieve thread-safety.

Though be careful while using thread local variable in 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) Please see the answer for a code example. Just remember to call `wait()` and `notify()` method from synchronized block and test waiting for condition on the loop instead of if block. **13) Write code for thread-safe Singleton in Java?** (answer) Please see the answer for a code example and step by step guide to creating thread-safe singleton class in Java. When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads. Using Java enum as Singleton class is one of the easiest ways to create a thread-safe singleton in Java. **14) The difference between sleep and wait in Java?** (answer) Though both are used to pause currently running thread, sleep() is actually meant for short pause because it doesn't release lock, while wait() is meant for conditional wait and that's why it release lock which can then be acquired by another thread to change the condition on which it is waiting. **15) What is an immutable object? How do you create an Immutable object in Java?** (answer) Immutable objects are those whose state cannot be changed once created. Any modification will result in a new object e.g. String, Integer, and other wrapper class. Please see the answer for step by step guide to creating Immutable class in Java. **16) Can we create an Immutable object, which contains a mutable object?** Yes, its possible to create an Immutable object which may contain a mutable object, you just need to be a little bit careful not to share the reference of the mutable component, instead, you should return a copy of it if you have to. Most common example is an Object which contain the reference of `java.util.Date` object.

2. 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) BigDecimal if memory is not a concern and Performance is not critical, otherwise double with predefined precision. **18) How do you convert bytes to String?** (answer) you can convert bytes to the string using string constructor which accepts byte[], just make sure that right character encoding otherwise platform's default character encoding will be used which may or may not be same. **19) How do you convert bytes to long in Java?** (answer) This questions if for you to answer :-) They to answer this one in comments. **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 = © B;** (answer) **22) Which class contains clone method? Cloneable or Object?** (answer) The java.lang.Cloneable is marker interface and doesn't contain any method clone method is defined in the object class. It is also knowing that clone() is a native method means it's implemented in C or C++ or any other native language. **23) Is ++ operator is thread-safe in Java?** (answer) No it's not a thread safe operator because its involve multiple instructions like reading a value, incriminating it and storing it back into memory which can be overlapped between multiple threads. **24) Difference between a = a + b and a += b ?** (answer) The += operator implicitly cast the result of addition into the type of variable used to hold the result. When you add two integral variable e.g. variable of type byte, short, or int then they are first promoted to int and them addition happens.

If result of addition is more than maximum value of a then a + b will give compile time error but a += b will be ok as shown below

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

25) Can I store a double value in a long variable without casting? (answer) No, you cannot store a double value into a long variable without casting because the range of double is more that long and you we need to type cast. It's not dificult to answer this question but many develoepr get it wrong due to confusion on which one is bigger between double and long in Java. **26) What will this return 3*0.1 == 0.3? true or false?** (answer) This is one of the really tricky questions. Out of 100, only 5 developers answered this question and only of them have explained the concept correctly. The short answer is false because some floating point numbers can not be represented exactly. **27) Which one will take more memory, an int or Integer?** (answer) An Integer object will take more memory an Integer is the an object and it store meta data overhead about the object and int is primitive type so its takes less space. **28) Why is String Immutable in Java?** (answer) One of my favorite Java interview question. The String is Immutable in

java because java designer thought that string will be heavily used and making it immutable allow some optimization easy sharing same String object between multiple clients.

See the link for the more detailed answer. This is a great question for Java programmers with less experience as it gives them food for thought, to think about how things works in Java, what Jave designers might have thought when they created String class etc. **29) Can we use String in the switch case?** (answer) Yes from Java 7 onward we can use String in switch case but it is just syntactic sugar. Internally string hash code is used for the switch. See the detailed answer for more explanation and discussion. **30) What is constructor chaining in Java?** (answer) When you call one constructor from other than it's known as constructor chaining in Java. This happens when you have multiple, overloaded constructor in the class.

3. 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.

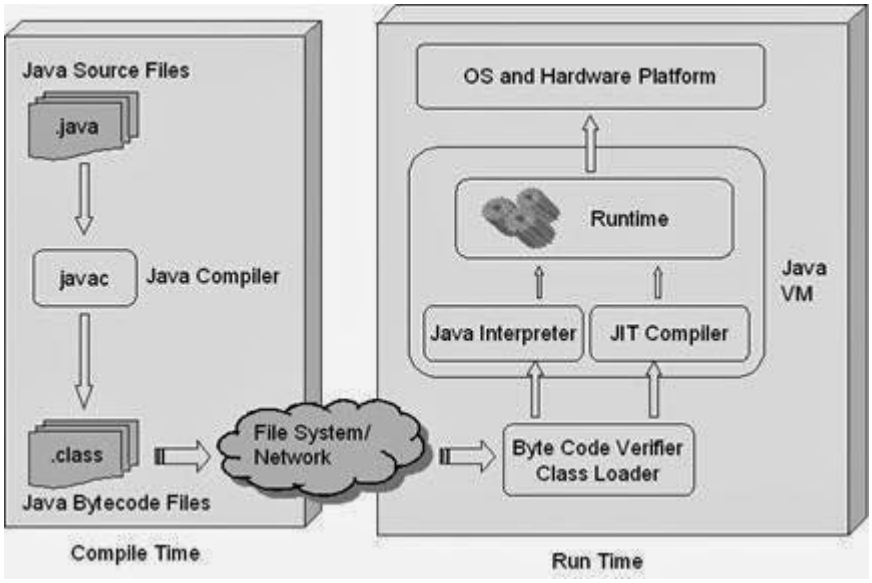
This is actually become the prime topic for interviewing any experienced Java developer for senior position e.g. technical lead, VP or team lead.

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. I vote goes to Java Performance, The Definitive guide by Scott. **31) What is the size of int in 64-bit JVM?** The size of an int variable is constant in Java, it's always 32-bit irrespective of platform. Which means the size of primitive int is same in both 32-bit and 64-bit Java virtual machine. **32) The difference between Serial and Parallel Garbage Collector?** (answer) Even though both the serial and parallel collectors cause a stop-the-world pause during Garbage collection. The main difference between them is that a serial collector is a default copying collector which uses only one GC thread for garbage collection while a parallel collector uses multiple GC threads for garbage collection. **33) What is the size of an int variable in 32-bit and 64-bit JVM?** (answer) The size of int is same in both 32-bit and 64-bit JVM, it's always 32 bits or 4 bytes. **34) A difference between WeakReference and SoftReference in Java?** (answer) Though both WeakReference and SoftReference helps garbage collector and memory efficient, WeakReference becomes eligible for garbage collection as soon as last strong reference is lost but SoftReference even thought it can not prevent GC, it can delay it until JVM absolutely need memory. **35) How do WeakHashMap works?** (answer) WeakHashMap works like a normal HashMap but uses WeakReference for keys, which means if the key object doesn't have any reference then both key/value mapping will become eligible for garbage collection. **36) What is -XX:+UseCompressedOops JVM option? Why use it?** (answer) When you go migrate your Java application from 32-bit to 64-bit JVM, the heap requirement suddenly increases, almost double, due to increasing size of ordinary object pointer from 32 bit to 64 bit.

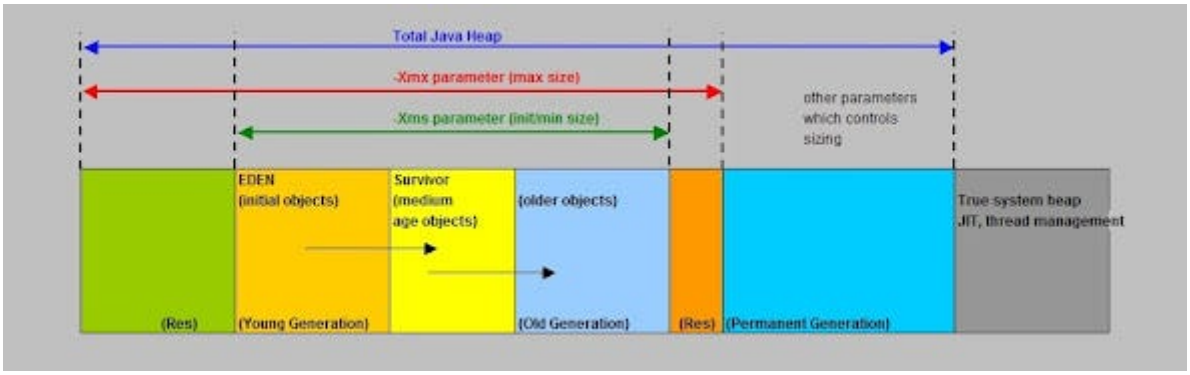
This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP. **37) How do you find if JVM is 32-bit or 64-bit from Java Program?** (answer) You can find that by checking some system properties like sun.arch.data.model or os.arch **38) What is the maximum heap size of 32-bit and 64-bit JVM?** (answer) Theoretically, the maximum heap memory you can assign to a 32-bit JVM is ^{232} 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 ^{264} which is quite large but practically you can specify heap space up to 100GBs. There are even JVM e.g. Azul where heap space of 1000 gigs is also possible. **39) What is the difference between JRE, JDK, JVM and JIT?** (answer) JRE stands for Java run-time and it's required to run Java application. JDK stands for Java development kit and provides tools to develop Java program e.g. Java compiler. It also contains JRE. The JVM stands for Java virtual machine and it's the process responsible for running Java application.

The JIT stands for Just In Time compilation and helps to boost the performance of Java application by converting Java byte code into native code when the crossed certain threshold i.e. mainly hot code is converted into native code.



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

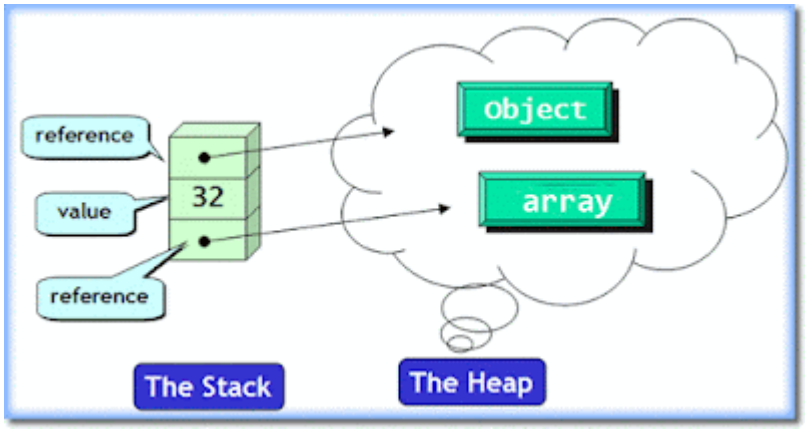


41) Can you guarantee the garbage collection process? (answer) No, you cannot guarantee the garbage collection, though you can make a request using System.gc() or Runtime.gc() method. **42) How do you find memory usage from Java program? How much percent of the heap is used?** You can use memory related 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 precents of the heap is used and how much heap space is remaining.

`Runtime.freeMemory()` return amount of free memory in bytes, `Runtime.totalMemory()` returns total memory in bytes and `Runtime.maxMemory()` returns maximum memory in bytes. **43) What is the difference between stack and heap in Java?** (answer) Stack and heap are different memory areas in the JVM and they are used for different purposes. The stack is used to hold method frames and local variables while objects are always allocated memory from the heap.

The stack is usually much smaller than heap memory and also didn't shared between multiple threads, but heap is shared among all threads in JVM.



4. 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) The a = b does object reference matching if both a and b are an object and only return true if both are pointing to the same object in the heap space, on the other hand, `a.equals(b)` is used for logical mapping and its expected from an object to override this method to provide logical equality.

For example, String class overrides this equals() method so that you can compare two Strings, which are the different object but contains same letters. **45) What is a.hashCode() used for? How is it related to a.equals(b)?** (answer) hashCode() method returns an int hash value corresponding to an object. It's used in hash based collection classes e.g Hashtable, HashMap, LinkedHashMap and so on. It's very tightly related to `equals()` method. According to Java specification, two objects which are equal to each other using equals() method must have same hash code. **46) Difference between final, finalize and finally?** (answer) 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.

The `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?** The `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.

5. Java Collections Framework Interview Questions

It also contains Data structure and algorithm Interview question in Java, questions on array, linked list, HashMap, ArrayList, Hashtable, Stack, `Queue`, `PriorityQueue`, `LinkedHashMap` and `ConcurrentHashMap`. **48) The difference between List, Set, Map, and Queue in Java?** (answer) The list is an ordered collection which allows duplicate. It also has an implementation which provides constant time index based access, but that is not guaranteed by List interface. Set is unordered collection which **49) Difference between poll() and remove() method?** Both `poll()` and `remove()` take out the object

from the Queue but if `poll()` fails then it returns `null` but if remove fails it throws `Exception`. **50) The difference between LinkedHashMap and PriorityQueue in Java?** (answer) PriorityQueue guarantees that lowest or highest priority element always remain at the head of the queue, but `LinkedHashMap` maintains the order on which elements are inserted. When you iterate over a `PriorityQueue`, iterator doesn't guarantee any order but iterator of `LinkedHashMap` does guarantee the order on which elements are inserted. **51) Difference between ArrayList and LinkedList in Java?** (answer) The obvious difference between them is that 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) You can either use the Sorted collection like TreeSet or TreeMap or you can sort using the ordered collection like a list and using Collections.sort() method. **53) How do you print Array in Java?** (answer) You can print an array by using the Arrays.toString() and `Arrays.deepToString()` method. Since array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but Arrays.toString() will print each element.

54) LinkedList in Java is doubly or singly linked list? (answer) It's a doubly linked list, you can check the code in JDK. In Eclipse, you can use the shortcut, Ctrl + T to directly open this class in Editor. **55) Which kind of tree is used to implement TreeMap in Java?** (answer) A Red Black tree is used to implement TreeMap in Java.

56) What is the difference between Hashtable and HashMap? (answer) There are many differences between these two classes, some of them are following: a) Hashtable is a legacy class and present from JDK 1, HashMap was added later. b) Hashtable is synchronized and slower but HashMap is not synchronized and faster. c) Hashtable doesn't allow null keys but HashMap allows one null key. See the answer for more differences between HashMap and Hashtable in Java. **57) How HashSet works internally in Java?** (answer) HashSet is internally implemented using an HashMap. Since a Map needs key and value, a default value is used for all keys. Similar to HashMap, HashSet doesn't allow duplicate keys and only one null key, I mean you can only store one null object in HashSet. **58) Write code to remove elements from ArrayList while iterating?** (answer) Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Here is the sample code which uses right way o remove elements from ArrayList while looping over and avoids ConcurrentModificationException. **59) Can I write my own container class and use it in the for-each loop?** Yes, you can write your own container class. You need to implement the Iterable interface if you want to loop over advanced for loop in Java, though. If you implement Collection then you by default get that property. **60) What is default size of ArrayList and HashMap in Java?** (answer) As of Java 7 now, default size of ArrayList is 10 and default capacity of HashMap is 16, it must be power of 2. Here is code snippet from ArrayList and HashMap class :

```
 // from ArrayList.java JDK 1.7
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) No, because hashcode of an object should be always same. See the answer to learning more about things to remember while overriding hashCode() method in Java. **64) What is the difference between Comparator and Comparable in Java?** (answer) The Comparable interface is used to define the natural order of object while Comparator is used to define custom order. Comparable can be always one, but we can have multiple comparators to define customized order for objects. **65) Why you need to override hashcode, when you override equals in Java?** (answer) Because equals have code contract mandates to override equals and hashcode together .since many container class like HashMap or HashSet depends on hashcode and equals contract.

6. 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)

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) 72) What is the memory mapped buffer in Java? (answer) 73) What is TCP NO DELAY socket option? 74) What is the difference between TCP and UDP protocol? (answer) 75) The difference between `ByteBuffer` and StringBuffer in Java? (answer)

7. Java Best Practices Interview question

Contains best practices from different parts of Java programming e.g. Collections, String, IO, Multi-threading, Error and Exception handling, design patterns etc. This section is mostly for experience Java developer, technical lead, AVP, team lead and coders who are responsible for products. If you want to create quality products you must know and follow the best practices. **76) What best practices you follow while writing multi-threaded code in Java?** (answer) Here are couple of best practices which I follow while writing concurrent code in Java: a) Always name your thread, this will help in debugging. b) minimize the scope of synchronization, instead of making whole method synchronized,

only critical section should be synchronized. c) prefer volatile over synchronized if you can can. e) use higher level concurrency utilities instead of `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) This is similar to the previous question and you can use the answer given there. Particularly with thread, you should: a) name your thread b) keep your task and thread separate, use Runnable or Callable with thread pool executor. c) use thread pool d) use volatile to indicate compiler about ordering, visibility, and atomicity. e) avoid thread local variable because incorrect use of ThreadLocal class in Java can create a memory leak. Look there are many best practices and I give extra points to the developer which bring something new, something even I don't know. I make sure to ask this question to Java developers of 8 to 10 years of experience just to gauge his hands on experience and knowledge. **79) Name 5 IO best practices?** (answer) IO is very important for performance of your Java application. Ideally you should avoid IO in critical path of your application. Here are couple of Java IO best practices you can follow:

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

b) Use classes from NIO and NIO2

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

d) use memory mapped file for faster IO.

If a Java candidate doesn't know about IO and NIO, especially if he has at least 2 to 4 years of experience, he needs some reading. **80) Name 5 JDBC best practices your follow?** (answer) Another good Java best practices for experienced Java developer of 7 to 8 years experience. Why it's important? because they are the ones which set the trend in the code and educate junior developers. There are many best practices and you can name as per your comfort and convenience. Here are some of the more common ones: a) use batch statement for inserting and updating data. b) use PreparedStatement to avoid SQL exception and better performance. c) use database connection pool d) access resultset using column name instead of column indexes. e) Don't generate dynamic SQL by concatenating String with user input. **81) Name couple of method overloading best practices in Java?** (answer) Here are some best practices you can follow while overloading a method in Java to avoid confusion with auto-boxing: a) Don't overload

method where one accepts int and other accepts Integer. b) Don't overload method where number of argument is same and only order of argument is different. c) Use varargs after overloaded methods has more than 5 arguments.

8. Date, Time and Calendar Interview questions in Java

82) Does SimpleDateFormat is safe to use in the multi-threaded program? (answer) No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a ThreadLocal variable.

If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **joda-time** library. **83) How do you format a date in Java? e.g. in the ddMMyyyy format?** (answer) You can either use SimpleDateFormat class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. dd-MM-yyyy or ddMMyyyy. 84) How do you show timezone in formatted date in Java? (answer) 85) The difference between java.util.Date and java.sql.Date in Java? (answer) 86) How to you calculate the difference between two dates in Java? (program) 87) How do you convert a String(YYYYMMDD) to date in Java? (answer)

9. Unit testing JUnit Interview questions

89) How do you test static method? (answer) You can use PowerMock library to test static methods in Java. 90) How to do you test a method for an exception using JUnit? (answer) 91) Which unit testing libraries you have used for testing Java programs? (answer) 92) What is the difference between `@BeforeEach` and `@BeforeClass` annotation? (answer)

10. Programming and Coding Questions for Java Interviews

93) How to check if a String contains only numeric digits? (solution) 94) How to write LRU cache in Java using Generics? (answer) 95) Write a Java program to convert bytes to long? (answer) 96) How to reverse a String in Java without using StringBuffer? (solution) 97) How to find the word with the highest frequency from a file in Java? (solution) 98) How do you check if two given String are anagrams? (solution) 99) How to print all permutation of a String in Java? (solution) 100) How do you print duplicate elements from an array in Java? (solution) 101) How to convert String to int in Java? (solution) 102) How to swap two integers without using temp variable? (solution)

11. Java Interview questions from OOP and Design Patterns

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

also contains questions from GOF design patterns. **103) What is the interface? Why you use it if you cannot write anything concrete on it?** The interface is used to define API. It tells about the contract your classes will follow. It also supports abstraction because a client can use interface method to leverage multiple implementations e.g. by using List interface you can take advantage of random access of ArrayList as well as flexible insertion and deletion of LinkedList.

The interface doesn't allow you to write code to keep things abstract but from Java 8 you can declare static and default methods inside interface which are concrete. **104) The difference between abstract class and interface in Java?** (answer) There are multiple differences between abstract class and interface in Java, but the most important one is Java's restriction on allowing a class to extend just one class but allows it to implement multiple interfaces.

An abstract class is good to define default behavior for a family of class, but the interface is good to define Type which is later used to leverage Polymorphism. Please check the answer for a more thorough discussion of this question. **105) Which design pattern have you used in your production code? apart from Singleton?** This is something you can answer from your experience. You can generally say about dependency injection, factory pattern, decorator pattern or observer pattern, whichever you have used. Though be prepared to answer follow-up question based upon the pattern you choose. **106) Can you explain Liskov Substitution principle?** (answer) 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) Believe it or not, Java is all about application programming and structuring code. If you have good knowledge of common coding best practices, patterns and what not to do than only you can write quality code.

Law of Demeter suggests you "talk to friends and not stranger", hence used to reduce coupling between classes. **108) What is Adapter pattern? When to use it?** Another frequently asked Java design pattern questions. It provides interface conversion. If your client is using some interface but you have something else, you can write an Adapter to bridge them together.

This is good for Java software engineer having 2 to 3 years experience because the question is neither difficult nor tricky but requires knowledge of OOP design patterns. **109) What is "dependency injection" and "inversion of control"? Why would someone use it?** (answer) **110) What is an abstract class? How is it different from an interface? Why would you use it?** (answer) One more classic question from Programming Job interviews, it is as old as chuck Norris. An abstract class is a class which can have state, code and implementation, but an interface is a contract which is totally abstract.

Since I have answered it many times, I am only giving you the gist here but you should read the article linked to answer to learn this useful concept in much more detail. **111) Which one is better constructor injection or setter dependency injection?** (answer) Each has their own advantage and disadvantage. Constructor injection guaranteed that class will be initialized with all its dependency, but setter injection provides flexibility to set an optional dependency.

Setter injection is also more readable if you are using an XML file to describe dependency. Rule of thumb is to use constructor injection for mandatory dependency and use setter injection for optional dependency. **112) What is difference between dependency injection and factory design pattern?** (answer) Though both patterns help to take out object creation part from application logic, use of dependency injection results in cleaner code than factory pattern.

By using `dependency injection`, your classes are nothing but POJO which only knows about dependency but doesn't care how they are acquired.

In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern. Please see the answer for a more detailed discussion on this topic. **113) Difference between Adapter and Decorator pattern?** (answer) Though the structure of Adapter and Decorator pattern is similar, the difference comes on the intent of each pattern.

The adapter pattern is used to bridge the gap between two interfaces, but Decorator pattern is used to add new functionality into the class without the modifying existing code. **114) Difference between Adapter and Proxy Pattern?** (answer) Similar to the previous question, the difference between Adapter and Proxy patterns is in their intent.

Since both Adapter and Proxy pattern encapsulate the class which actually does the job, hence result in the same structure, but `Adapter` pattern is used for interface conversion while the Proxy pattern is used to add an extra level of indirection to support distribute, controlled or intelligent access. **115) What is Template method pattern?** (answer) Template pattern provides an outline of an algorithm and lets you configure or customize its steps. For examples, you can view a sorting algorithm as a template to sort object.

It defines steps for sorting but let you configure how to compare them using Comparable or something similar in another language. The method which outlines the algorithms is also known as template method. **116) When do you use Visitor design pattern?** (answer) The visitor pattern is a solution of problem where you need to add operation on a class hierarchy but without touching them. This pattern uses double dispatch to add another level of indirection. **117) When do you use Composite design pattern?** (answer) Composite design pattern arranges objects into tree structures to represent part-whole hierarchies. It allows clients treat individual objects and container of objects uniformly. Use Composite pattern when you want to represent part-whole hierarchies of objects.

118) The difference between Inheritance and Composition? (answer) Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies. **119) Describe overloading and overriding in Java?** (answer) Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity.

Though you can overload a method in the same class, but you can only override a method in child classes. Inheritance is necessary for overriding. **120) The difference between nested public static class and a top level class in Java?** (answer) You can have more than one nested public static class inside one class, but you can only have one top-level public class in a Java source file and its name must be same as the name of Java source file. **121) Difference between Composition, Aggregation and Association in OOP?** (answer) If two objects are related to each other, they are said to be associated with each other. `Composition` and `Aggregation` are two forms of association in object-oriented programming. The composition is stronger association than Aggregation.

In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object.

If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist. **122) Give me an example of design pattern which is based upon open closed principle?** (answer) This is one of the practical questions I ask experienced Java programmer. I expect them to know about OOP design principles as well as patterns. Open closed design principle asserts that your code should be open for extension but closed for modification.

Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code.

There are several design patterns which are based upon open closed design principle e.g. Strategy pattern if you need a new strategy, just implement the interface and configure, no need to modify core logic.

One working example is `Collections.sort()` method which is based on Strategy pattern and follows the open-closed principle, you don't modify sort() method to sort a new object, what you do is just implement Comparator in your own way. **123) Difference between Abstract factory and Prototype design pattern?** (answer) This is the practice question for you, If you are feeling bored just reading and itching to write something, why not write the answer to this question.

I would love to see an example the, which should answer where you should use the Abstract factory pattern and where is the Prototype pattern is more suitable. **124) When do you use Flyweight pattern?** (answer) This is another popular question from the

design pattern. Many Java developers with 4 to 6 years of experience know the definition but failed to give any concrete example.

Since many of you might not have used this pattern, it's better to look examples from JDK. You are more likely have used them before and they are easy to remember as well.

Now let's see the answer.

Flyweight pattern allows you to share object to support large numbers without actually creating too many objects.

In order to use `Flyweight` pattern, you need to make your object Immutable so that they can be safely shared. String pool and pool of Integer and Long object in JDK are good examples of Flyweight pattern.

12. Miscellaneous Java Interview Questions Answers

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) 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) 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.

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) The 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) 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) 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) 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) 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) 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) 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.

13. Modern Java Interview Questions (records, sealed classes, virtual threads)

Here are **7 modern Java interview questions** focused on recent features introduced in Java 14 through Java 21, such as **records, switch expressions, virtual threads, pattern matching**, and more:

134) What are Java Records, and when should you use them? Java Records (introduced in Java 14) are a special kind of class that act as transparent carriers for immutable data. They automatically generate constructors, `equals()`, `hashCode()`, and

`toString()` methods.

Use Case: When you need a simple data holder class without boilerplate code, like DTOs or value objects.

135) How does the new switch expression differ from the traditional switch statement in Java? Traditional `switch` is a statement and doesn't return a value, whereas **switch expressions** (Java 14) can return a value, support multiple labels (`case 1, 2 ->`), and avoid fall-through.

```
 String result = switch (day) {
    case MONDAY, FRIDAY -> "Weekend coming";
    case TUESDAY -> "Taco day";
    default -> "Just another day";
};
```

136) What are virtual threads in Java, and how are they different from platform threads?

Virtual threads (introduced as a preview in Java 19 and stable in Java 21) are lightweight threads managed by the JVM rather than the OS. They allow high concurrency (millions of threads) and make it easier to write scalable code using familiar blocking style APIs.

137) How does Pattern Matching for instanceof improve code readability?Pattern matching for `instanceof` (Java 16) eliminates redundant type casting by allowing you to declare a variable within the check:

```
 if (obj instanceof String s) {
    System.out.println(s.toUpperCase());
}
```

No need for `(String) obj` casting.

138) What is the advantage of sealed classes in Java, and how do they work? Sealed classes (Java 17) restrict which other classes or interfaces may extend or implement them. This provides better control over class hierarchies and aids in exhaustive pattern matching.

```
 public sealed class Shape permits Circle, Rectangle {}
```

139. What are text blocks, and how do they improve working with multi-line strings in Java?

Text blocks (introduced in Java 15) allow you to write multi-line string literals more cleanly using `"""`.

```
String query = """
   SELECT *
   FROM users
   WHERE status = 'ACTIVE';
   """;
```

No more messy string concatenations or escape characters.

140) How do structured concurrency and scoped values improve thread lifecycle management in Java?

Structured concurrency (Java 21 Incubator) makes it easier to manage the lifecycle of concurrent tasks by treating them as a unit. It helps with canceling related tasks together and handling errors predictably.

`Scoped values` (Java 21 preview) offer a safer and more efficient alternative to ThreadLocal for passing data to virtual threads.

That's all guys, **lots of Java Interview questions?** isn't it? I am sure if you can answer this list of Java questions you can easily crack any core Java or advanced Java interview.

Though I have not included questions from Java EE or J2EE topics e.g. Servlet, JSP, JSF, JPA, JMS, EJB, or any other Java EE technology or from major web frameworks like Spring MVC, Struts 2.0, Hibernate, or both SOAP and RESTful web services, it's still useful for Java developers preparing for Java web developer position, because every Java interview starts with questions from fundamentals and JDK API.

If you think, I have missed any popular Java question here and you think it should be in this list then feel free to suggest me. My goal is to create the best list of Java Interview Questions with the latest and greatest questions from recent interviews.