Java Software Engineer Job Interview Questions And Answers

Interview Questions Answers

https://interviewquestionsanswers.org/

About Interview Questions Answers

Interview Questions Answers . ORG is an interview preparation guide of thousands of Job Interview Questions And Answers, Job Interviews are always stressful even for job seekers who have gone on countless interviews. The best way to reduce the stress is to be prepared for your job interview. Take the time to review the standard interview questions you will most likely be asked. These interview questions and answers on Java Software Engineer will help you strengthen your technical skills, prepare for the interviews and quickly revise the concepts.

If you find any **question or answer** is incorrect or incomplete then you can **submit your question or answer** directly with out any registration or login at our website. You just need to visit Java Software Engineer Interview Questions And Answers to add your answer click on the *Submit Your Answer* links on the website; with each question to post your answer, if you want to ask any question then you will have a link *Submit Your Question*; that's will add your question in Java Software Engineer category. To ensure quality, each submission is checked by our team, before it becomes live. This Java Software Engineer Interview preparation PDF was generated at **Wednesday 29th November, 2023**

You can follow us on FaceBook for latest Jobs, Updates and other interviews material.
www.facebook.com/InterviewQuestionsAnswers.Org

Follow us on Twitter for latest Jobs and interview preparation guides.
https://twitter.com/InterviewQA

If you need any further assistance or have queries regarding this document or its material or any of other inquiry, please do not hesitate to contact us.

Best Of Luck.

Interview Questions Answers.ORG Team
https://InterviewQuestionsAnswers.ORG/
Support@InterviewQuestionsAnswers.ORG

Java Software Engineer Interview Questions And Answers Guide.

Question - 1:

Tell us why java is not 100% Object-oriented?

Ans:

Java is not 100% Object-oriented because it makes use of eight primitive datatypes such as boolean, byte, char, int, float, double, long, short which are not objects.

View All Answers

Question - 2:

Tell us why do you need to use synchronized methods or blocks?

Ans:

If threads are being used and a number of threads have to go through a synchronized section of code, only one of them may be executed at a time. This is used to make sure shared variables are not updated by multiple threads.

View All Answers

Question - 3:

Explain me what is the difference between HashMap and ConcurrentHashMap?

Ans:

ConcurrentHashMap is thread-safe; that is the code can be accessed by single thread at a time while HashMap is not thread-safe. ConcurrentHashMap does not allow NULL keys while HashMap allows it.

View All Answers

Question - 4:

Do you know why Java is platform independent?

Ans:

Platform independent practically means "write once run anywhere". Java is called so because of its byte codes which can run on any system irrespective of its underlying operating system.

View All Answers

Question - 5:

Explain me when do you use volatile variables?

Ans:

When a member variable is accessed by multiple threads and want the value of a volatile field to be visible to all readers (other threads in particular) after a write operation completes on it.

View All Answers

Question - 6:

Explain me how are Annotations better than a Marker Interfaces?

Ans:

Annotations lets one achieve the same purpose of conveying metadata about the class to its consumers without creating a separate type for it. Annotations are more powerful, too, letting programmers pass more sophisticated information to classes that "consume" it.

View All Answers

Question - 7:

Explain me what is singleton class and how can we make a class singleton?

Ans:

Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

View All Answers

Question - 8:

Tell me what is the difference between an Iterator and a ListIterator?

Ans:

This question tests the proper usage of collection iterators. One can only use ListIterator to traverse Lists, and cannot traverse a Set using ListIterator.
What's more, one can only traverse in a forward direction using Iterators. Using ListIterator, one can traverse a List in both the directions (forward and backward).
One cannot obtain indexes while using Iterator. Indexes can be obtained at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.

View All Answers

Question - 9:

Tell us what is the marker interface in Java?

Ans:

The marker interface in Java is an interfaces with no field or methods. In other words, it an empty interface in java is called a marker interface. An example of a marker interface is a Serializable, Clonable and Remote interface. These are used to indicate something to the compiler or JVM.

View All Answers

Question - 10:

Explain me how does the JVM handle storing local variables vs storing objects?

Ans:

Objects are stored on the heap. Variables are a reference to the object.
Local variables are stored on the stack.

View All Answers

Question - 11:

Do you know what is a Service?

Ans:

A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services.

View All Answers

Question - 12:

Please explain and compare fail-fast and fail-safe iterators. Give examples?

Ans:

The main distinction between fail-fast and fail-safe iterators is whether or not the collection can be modified while it is being iterated. Fail-safe iterators allow this; fail-fast iterators do not.
Fail-fast iterators operate directly on the collection itself. During iteration, fail-fast iterators fail as soon as they realize that the collection has been modified (i.e., upon realizing that a member has been added, modified, or removed) and will throw a ConcurrentModificationException. Some examples include ArrayList, HashSet, and HashMap (most JDK1.4 collections are implemented to be fail-fast).
Fail-safe iterates operate on a cloned copy of the collection and therefore do not throw an exception if the collection is modified during iteration. Examples would include iterators returned by ConcurrentHashMap or CopyOnWriteArrayList.

View All Answers

Question - 13:

Tell me why is the code printing true in the second and false in the first case?

Ans:

JVM's cache behavior can be confusing, so this question tests that concept. The second output is true as we are comparing the references, because the JVM tries to save memory when the Integer falls within a range (from -128 to 127). At point 2, no new reference of type Integer is created for 'd'. Instead of creating a new object for the Integer type reference variable 'd', it is only assigned with a previously created object referenced by 'c'. All of these are done by JVM.

View All Answers

Question - 14:

Tell us what is the difference between equals() and == ?

Ans:

Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.
"==" or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. public boolean equals(Object o) is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

View All Answers

Question - 15:

Tell me what is reflection? Give an example of functionality that can only be implemented using reflection?

Ans:

Reflection allows programmatic access to information about a Java program's types. Commonly used information includes: methods and fields available on a class, interfaces implemented by a class, and the runtime-retained annotations on classes, fields and methods.
Examples given are likely to include:
* Annotation-based serialization libraries often map class fields to JSON keys or XML elements (using annotations). These libraries need reflection to inspect those fields and their annotations and also to access the values during serialization.
* Model-View-Controller frameworks call controller methods based on routing rules. These frameworks must use reflection to find a method corresponding to an action name, check that its signature conforms to what the framework expects (e.g. takes a Request object, returns a Response), and finally, invoke the method.
* Dependency injection frameworks lean heavily on reflection. They use it to instantiate arbitrary beans for injection, check fields for annotations such as @Inject to discover if they require injection of a bean, and also to set those values.
* Object-relational mappers such as Hibernate use reflection to map database columns to fields or getter/setter pairs of a class, and can go as far as to infer table and column names by reading class and getter names, respectively.

View All Answers

Question - 16:

Tell me what is multiple inheritance? Is it supported by Java?

Ans:

If a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.
The problem with multiple inheritance is that if multiple parent classes have a same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.
Therefore, Java doesn't support multiple inheritance. The problem is commonly referred as Diamond Problem.

View All Answers

Question - 17:

Please explain what is the ThreadLocal class? How and why would you use it?

Ans:

A single ThreadLocal instance can store different values for each thread independently. Each thread that accesses the get() or set() method of a ThreadLocal instance is accessing its own, independently initialized copy of the variable. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or transaction ID). The example below, from the ThreadLocal Javadoc, generates unique identifiers local to each thread. A thread's id is assigned the first time it invokes ThreadId.get() and remains unchanged on subsequent calls.

```
public class ThreadId {
    // Next thread ID to be assigned
    private static final AtomicInteger nextId = new AtomicInteger(0);
    // Thread local variable containing each thread's ID
    private static final ThreadLocal<Integer> threadId =
        new ThreadLocal<Integer>() {
            @Override protected Integer initialValue() {
                return nextId.getAndIncrement();
        }
    };
    // Returns the current thread's unique ID, assigning it if necessary
    public static int get() {
        return threadId.get();
    }
}
```

Each thread holds an implicit reference to its copy of a thread-local variable as long as the thread is alive and the ThreadLocal instance is accessible; after a thread goes away, all of its copies of thread-local instances are subject to garbage collection (unless other references to these copies exist).

View All Answers

Question - 18:

Explain me why aren't you allowed to extend more than one class in Java but are allowed to implement multiple interfaces?

Ans:

Extending classes may cause ambiguity problems. On the other hand, in terms of interfaces, the single method implementation in one class can serve more than one interfaces.

View All Answers

Question - 19:

As you know ArrayList, LinkedList, and Vector are all implementations of the List interface. Which of them is most efficient for adding and removing elements from the list? Explain your answer, including any other alternatives you may be aware of?

Ans:

Of the three, LinkedList is generally going to give you the best performance. Here's why:
ArrayList and Vector each use an array to store the elements of the list. As a result, when an element is inserted into (or removed from) the middle of the list, the elements that follow must all be shifted accordingly. Vector is synchronized, so if a thread-safe implementation is not needed, it is recommended to use ArrayList rather than Vector.
LinkedList, on the other hand, is implemented using a doubly linked list. As a result, an inserting or removing an element only requires updating the links that immediately precede and follow the element being inserted or removed.
However, it is worth noting that if performance is that critical, it's better to just use an array and manage it yourself, or use one of the high performance 3rd party packages such as Trove or HPPC.

View All Answers

Question - 20:

Explain me how can you catch an exception thrown by another thread in Java?

Ans:

This can be done using Thread.UncaughtExceptionHandler.
Here's a simple example:

```
// create our uncaught exception handler
Thread.UncaughtExceptionHandler handler = new Thread.UncaughtExceptionHandler() {
   public void uncaughtException(Thread th, Throwable ex) {
      System.out.println("Uncaught exception: " + ex);
   }
};
// create another thread
Thread otherThread = new Thread() {
   public void run() {
      System.out.println("Sleeping ...");
      try {
         Thread.sleep(1000);
      } catch (InterruptedException e) {
         System.out.println("Interrupted.");
      }
      System.out.println("Throwing exception ...");
      throw new RuntimeException();
   }
};
// set our uncaught exception handler as the one to be used when the new thread
// throws an uncaught exception
otherThread.setUncaughtExceptionHandler(handler);
// start the other thread - our uncaught exception handler will be invoked when
// the other thread throws an uncaught exception
otherThread.start();
```

View All Answers

Question - 21:

Tell me what are static initializers and when would you use them?

Ans:

A static initializer gives you the opportunity to run code during the initial loading of a class and it guarantees that this code will only run once and will finish running before your class can be accessed in any way.
They are useful for performing initialization of complex static objects or to register a type with a static registry, as JDBC drivers do.
Suppose you want to create a static, immutable Map containing some feature flags. Java doesn't have a good one-liner for initializing maps, so you can use static initializers instead:

```
    public static final Map<String, Boolean> FEATURE_FLAGS;
    static {
       Map<String, Boolean> flags = new HashMap<>();
       flags.put("frustrate-users", false);
       flags.put("reticulate-splines", true);
       flags.put(...);
       FEATURE_FLAGS = Collections.unmodifiableMap(flags);
    }
```

Within the same class, you can repeat this pattern of declaring a static field and immediately initializing it, since multiple static initializers are allowed.

View All Answers

Question - 22:

How to override a private or static method in Java?

Ans:

You cannot override a private or static method in Java. If you create a similar method with same return type and same method arguments in child class then it will hide the super class method; this is known as method hiding. Similarly, you cannot override a private method in sub class because it's not accessible there. What you can do is create another private method with the same name in the child class. Let's take a look at the example below to understand it better.

```
class Base {
private static void display() {
System.out.println("Static or class method from Base");
}
public void print() {
System.out.println("Non-static or instance method from Base");
}
class Derived extends Base {
private static void display() {
System.out.println("Static or class method from Derived");
}
public void print() {
System.out.println("Non-static or instance method from Derived");
}
public class test {
public static void main(String args[])
{
Base obj= new Derived();
obj1.display();
obj1.print();
}
}
```

View All Answers

Question - 23:

Can you compare the sleep() and wait() methods in Java, including when and why you would use one vs. the other?

Ans:

sleep() is a blocking operation that keeps a hold on the monitor / lock of the shared object for the specified number of milliseconds.
wait(), on the other hand, simply pauses the thread until either (a) the specified number of milliseconds have elapsed or (b) it receives a desired notification from another thread (whichever is first), without keeping a hold on the monitor/lock of the shared object.
sleep() is most commonly used for polling, or to check for certain results, at a regular interval. wait() is generally used in multithreaded applications, in conjunction with notify() / notifyAll(), to achieve synchronization and avoid race conditions.

View All Answers

Question - 24:

Tell us what is the Java Classloader? List and explain the purpose of the three types of class loaders?

Ans:

The Java Classloader is the part of the Java runtime environment that loads classes on demand (lazy loading) into the JVM (Java Virtual Machine). Classes may be loaded from the local file system, a remote file system, or even the web.
When the JVM is started, three class loaders are used:
1. Bootstrap Classloader: Loads core java API file rt.jar from folder.
2. Extension Classloader: Loads jar files from folder.
3. System/Application Classloader: Loads jar files from path specified in the CLASSPATH environment variable.

View All Answers

Question - 25:

Tell me public static void main(String args[])?

Ans:

* public : Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
* static : It is a keyword in java which identifies it is class based i.e it can be accessed without creating the instance of a Class.
* void : It is the return type of the method. Void defines the method which will not return any value.
* main: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
* String args[] : It is the parameter passed to the main method.

View All Answers

Question - 26:

Explain me what is runtime polymorphism or dynamic method dispatch?

Ans:

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let's take a look at the example below to understand it better.

```
class Car {
void run()
{
System.out.println("car is running");
}
}
class Audi extends Car {
void run()
{
System.out.prinltn("Audi is running safely with 100km");
}
public static void main(String args[])
{
Car b= new Audi();    //upcasting
b.run();
}
}
```

View All Answers

Question - 27:

Explain me what is the difference between String s = "Test" and String s = new String("Test")? Which is better and why?

Ans:

In general, String s = "Test" is more efficient to use than String s = new String("Test").
In the case of String s = "Test", a String with the value "Test" will be created in the String pool. If another String with the same value is then created (e.g., String s2 = "Test"), it will reference this same object in the String pool.
However, if you use String s = new String("Test"), in addition to creating a String with the value "Test" in the String pool, that String object will then be passed to the constructor of the String Object (i.e., new String("Test")) and will create another String object (not in the String pool) with that value. Each such call will therefore create an additional String object (e.g., String s2 = new String("Test") would create an addition String object, rather than just reusing the same String object from the String pool).

View All Answers

Question - 28:

Explain me what does it mean for a collection to be "backed by" another? Give an example of when this property is useful?

Ans:

If a collection backs another, it means that changes in one are reflected in the other and vice-versa.
For example, suppose we wanted to create a whitelist function that removes invalid keys from a Map. This is made far easier with Map.keySet, which returns a set of keys that is backed by the original map. When we remove keys from the key set, they are also removed from the backing map:

```
    public static <K, V> Map<K, V> whitelist(Map<K, V> map, K... allowedKeys) {
        Map<K, V> copy = new HashMap<>(map);
        copy.keySet().retainAll(asList(allowedKeys));
        return copy;
    }
```

retainAll writes through to the backing map, and allows us to easily implement something that would otherwise require iterating over the entries in the input map, comparing them against allowedKey, etcetera.
Note, it is important to consult the documentation of the backing collection to see which modifications will successfully write through. In the example above, map.keySet().add(value) would fail, because we cannot add a key to the backing map without a value.

View All Answers

Question - 29:

Explain me what is the volatile keyword? How and why would you use it?

Ans:

In Java, each thread has its own stack, including its own copy of variables it can access. When the thread is created, it copies the value of all accessible variables into its own stack. The volatile keyword basically says to the JVM "Warning, this variable may be modified in another Thread".
In all versions of Java, the volatile keyword guarantees global ordering on reads and writes to a variable. This implies that every thread accessing a volatile field will read the variable's current value instead of (potentially) using a cached value.
In Java 5 or later, volatile reads and writes establish a happens-before relationship, much like acquiring and releasing a mutex.
Using volatile may be faster than a lock, but it will not work in some situations. The range of situations in which volatile is effective was expanded in Java 5; in particular, double-checked locking now works correctly.
The volatile keyword is also useful for 64-bit types like long and double since they are written in two operations. Without the volatile keyword you risk stale or invalid values.
One common example for using volatile is for a flag to terminate a thread. If you've started a thread, and you want to be able to safely interrupt it from a different thread, you can have the thread periodically check a flag (i.e., to stop it, set the flag to true). By making the flag volatile, you can ensure that the thread that is checking its value will see that it has been set to true without even having to use a synchronized block. For example:

```
public class Foo extends Thread {
    private volatile boolean close = false;
    public void run() {
        while(!close) {
            // do work
        }
    }
    public void close() {
        close = true;
        // interrupt here if needed
    }
}
```

View All Answers

Question - 30:

Tell us why would it be more secure to store sensitive data (such as a password, social security number, etc.) in a character array rather than in a String?

Ans:

In Java, Strings are immutable and are stored in the String pool. What this means is that, once a String is created, it stays in the pool in memory until being garbage collected. Therefore, even after you're done processing the string value (e.g., the password), it remains available in memory for an indeterminate period of time thereafter (again, until being garbage collected) which you have no real control over. Therefore, anyone having access to a memory dump can potentially extract the sensitive data and exploit it.
In contrast, if you use a mutable object like a character array, for example, to store the value, you can set it to blank once you are done with it with confidence that it will no longer be retained in memory.

View All Answers

Question - 31:

Can you give real world examples of when to use an ArrayList and when to use LinkedList?

Ans:

ArrayList is preferred when there are more get(int), or when search operations need to be performed as every search operation runtime is O(1).
If an application requires more insert(int) and delete(int) operations, then LinkedList is preferred, as LinkedList does not need to maintain back and forth to preserve continued indices as arraylist does. Overall this question tests the proper usage of collections.

View All Answers

Question - 32:

int a = 1L;
won't compile and int b = 0;
b += 1L;
compiles fine. Explain me why?

Ans:

When += is used, that's a compound statement and the compiler internally casts it. Whereas in the first case, the compiler straightaway shouts at you since it is a

direct statement.
Compiler behavior and statement types can be confusing, so questions like this will test a candidate's grasp of these concepts.

View All Answers

Question - 33:

Explain me what is association?

Ans:

Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationship can be one to one, One to many, many to one and many to many.

View All Answers

Question - 34:

Explain me what is method overloading and method overriding?

Ans:

* Method Overloading:
In Method Overloading, Methods of the same class shares the same name but each method must have different number of parameters or parameters having different types and order.
Method Overloading is to "add" or "extend" more to method's behavior.
It is a compile time polymorphism.
The methods must have different signature.
It may or may not need inheritance in Method Overloading.
Let's take a look at the example below to understand it better.

```
class Adder {
Static int add(int a, int b)
{
return a+b;
}
Static double add( double a, double b)
{
return a+b;
}
public static void main(String args[])
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

* Method Overriding:
In Method Overriding, sub class have the same method with same name and exactly the same number and type of parameters and same return type as a super class.
Method Overriding is to "Change" existing behavior of method.
It is a run time polymorphism.
The methods must have same signature.
It always requires inheritance in Method Overriding.
Let's take a look at the example below to understand it better.

```
class Car {
void run(){
System.out.println("car is running");
}
Class Audi extends Car{
void run()
{
System.out.prinltn("Audi is running safely with 100km");
}
public static void main( String args[])
{
Car b=new Audi();
b.run();
}
}
```

View All Answers

Question - 35:

Can you explain me what is Polymorphism?

Ans:

Polymorphism is briefly described as "one interface, many implementations". Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of polymorphism:
* Compile time polymorphism
* Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

View All Answers

Question - 36:

Tell us what are constructors in Java?

Ans:

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.
There are two types of constructors:
* Default constructor
* Parameterized constructor

View All Answers

Question - 37:

Tell us how can you swap the values of two numeric variables without using any other variables?

Ans:

You can swap two values a and b without using any other variables as follows:
a = a + b;
b = a - b;
a = a - b;

View All Answers

Question - 38:

Tell us what are checked and unchecked exceptions? When do you use them?

Ans:

A checked exception is an exception that must be catch, they are checked by the compiler. An unchecked exception is mostly runtime exception, and is not required to be catch. In general, use checked exception when the situation is recoverable (retry, display reasonable error message).

View All Answers

Question - 39:

Explain me when do you need to override the equals and hashCode methods in Java?

Ans:

By defining equals() and hashCode() consistently, the candidate can improve the usability of classes as keys in hash-based collections such as HashMap.

View All Answers

Question - 40:

Tell me can an enum be extended?

Ans:

No. Enum types are final by design.

View All Answers

Question - 41:

Tell us what do you mean by aggregation?

Ans:

Aggregation is a specialized form of Association where all object have their own lifecycle but there is ownership and child object can not belongs to another parent object. Let's take an example of Department and teacher. A single teacher can not belongs to multiple departments, but if we delete the department teacher object will not destroy.

View All Answers

Question - 42:

Do you know what is the advantage of generic collection?

Ans:

They enable stronger type checks at compile time.
A Java compiler applies strong type checking to generic code, and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

View All Answers

Question - 43:

Explain me why doesn't the following code generate a NullPointerException even when the instance is null?

Ans:

```
Test t = null;
t.someMethod();
public static void someMethod() {
  ...
}
```
There is no need for an instance while invoking a static member or method, since static members belongs to a class rather than an instance.
A null reference may be used to access a class (static) variable without causing an exception.

View All Answers

Question - 44:
Explain me what is a good usecase of calling System.gc()?

Ans:
One may call System.gc() when profiling an application to search for possible memory leaks. All the profilers call this method just before taking a memory snapshot.
View All Answers

Question - 45:
Do you know how threadsafe is enum in Java?

Ans:
Creation of an enum is guaranteed to be threadsafe. However, the methods on an enum type are not necessarily threadsafe
View All Answers

Java Programing Most Popular & Related Interview Guides