# Java interview question

Q1) what do you know about JVM, JRE & JDK?

→ 1. JVM

i) JVM is stand for the Java Virtual Machine.
ii) It is integral part of Java Environment (JRE) Runtime
iii) JVM responsible for executing Java byte code.
iv) Provides platform independency by translating byte code into native machine code at Runtime.
v) Ensure memory management, garbage collection & excepting handling.

2. JRE.

i) JRE is stand for Java Runtime Enviroment.
ii) consider of JVM & libraries necessary to Run Java application.
iii) Its what you need to run Java application on your computer.
iv) JRE doesn't include development tool like compilers.

8. JDK
i) JDK is stand for the Java develop ment kit.
ii) JDK contains the tools needed for developing Java application
iii) it include the JRE for runtime running Java application
iv) contains the Java compiler ('javac') debugger & other development utilities
v) used by developers to create, compile & run Java application

* JDK is for Java development, JRE is for running Java application & Jvm is the runtime enviroment that executes Java byte code.

Q 2.) is JRE platform dependant or independant?

→ the Java Runtime enviroment (JRE) is designed to be platform-independent one of the key goals of Java is platform independance, & the JRE plays a crucial role in achieving this goal

Java accomplishes platform idde pedence by using Jvm which interprets & executes Java bytecode. the Jvm is

specific to each platform (eg. windows, macOS, linux.) but it ensure that Java bytecode, compiled from Java source code can run on any platform with the appropriate JVM implementation.

So, while the JVM is platform-dependant, the JRE which includes the JVM & necessary libraries, allows Java application to be platform-independent by providing the necessary runtime enviroment for executing Java bytecode on different platform.

93) which is ultimate base class in Java class hierarchy? list name of method of it?

→ In Java class hierarchy the ultimate base class is the "object" class the "object" class is the root of class hierarchy & it is super class of all other class in Java

| | method | description |
|---|---|---|
| 1. | Public final class getclass() | return the class class object of this object. classclass can be use to get metadata. |
| 2. | Public int hashcode() | return the hashcode number for this object |
| 3. | Public string to string() | return string represe ntation of this object |
| 4. | public final void notify() | wake up single thread waiting on this object monitor. |

Q4) which are the reference type in Java ?

→ Reference data type in java are these which contain reference / address of dynamically created object

i. class types :- this reference type point to object of class.
ii. array type :- this reference type point to an array.
iii. Interface class :- this reference type point to an array.
iv. Enum type :- Enumeration are the reference type that represent a fix set of constant

Q5) Explain narrowing & widening :-
→ narrowing conversion.
defn :-
Narrowing conversion in the content of Java refer to explic type conversion of a value form a large data to smaller data type this conversion often involve loss of data
concept :-
It involve changing the data type of value to a smaller data type which may result in data loss if original value cannot be accurately represented in smaller type
Example :-
converting 'double' value to an 'int'

Real life example:-
Imagine measuring weight of object in kg (floating pain value) & needing to represent in gram this involve narrowing conversion where you cast kg in gram but loss decimal precision

Q6) How will you print "Hello CDAC" statement on screen, without semicolon

→
```
class HelloWorld {
    Public static void main (String [] args) {
        System.out.println ("Hellow CDAC");
    }
}
```

Q7) can you write java application without main function? if you how?

→ yes, we can execute a java Program without a main method by using static block.
static block in java is a group statement that get executed only once when class is loaded in memory by java class

loder. It is known as static initialization block.

```
class staticInitializationBlock {
    static {
        System.out.println("class without main");
    }
}
```

Q8) what will happen, if we call main method in static block?

→ calling the main method form within a static block in Java is technically possible but it can lead concexpected behaviours & is generally not a good practice. the 'main' method is meant to be entry point of Java application & it is typically called JVM when you run a program.

If you called 'main' method from static block in java technically possible but it can lead to unexpected behaviour

i) the static block will executable when the class is loaded by Jvm. this mean that the code within the static block executed before main method.
ii) when the 'main' method is called from within static block executed before main method.
iii) when the 'main' method is called from within static block it will execute before main method.
iv) If there is code or logic in the 'main' method that terminate the Programme.

Q9) In System.out.Println, explain meaning of every coord?

→ i) System :- 'System' is a class in java that is part of 'Java.lang' Package. It provide access to various system-relable function & resource such as I/P and O/P.

ii) Out :- 'Out' is a static various within the system class. A represent the standard O/P stream

which is typically connected to console where you see program o/p.

iii) Println :- 'Println' is a method that is part of 'print stream' class which is associated with standard output stream. It is use to print text or data to standard o/p with a newline character at the end which means that each call to 'Println' will display o/p on the new line.

Q10) How will you pass object to the function by refrence ?

→ In Java, objects are passed to functions by refrence by default. this means that when you pass an object to a function, you are passing reference to the object in memory. & any changes made to the object within the function will affect the original object outside the function. this behavior is because java uses refrences to object rather than passing object by value.

Q11) Explain constructor chaining ? How we achive it in c++ ?

→ constructor chaining is also knoms as constructor delegation is a concept in oops where one constructor of class can call another constructor within the same class this is particularly useful when you have multiple constructor in class with different set of parameter. & you want to avoid duplicating code by reusing the initialising logic from one constructor in another constructor

In c++ we can achieve constructor in another chaining by using constructor initializer list to call another constructor of same class

Q12) which are the rules to overload method in sub class ?

→ 1.) the overload method in the subclass must have a different method signature than the method in the superclass. this typically mean a different no. or type of parameter.

ii) the return type of overload method can be the same or subtype of the return type of the method in the superclass.
iii) you cannot override a method in a subclass if it is marked as 'final' in the superclass beacouse 'final' method can not override.
iv) you can not override a method in the subclass if it has a more restrictive access modifier (eg. changing from 'public' to 'private') than the method in the superclass
v) the access modifier of overloaded method in the subclass can be the same or less restrictive (eg. changing from 'protected' to 'public'), than the method in the superclass.
vi) you can overload a static method in the subclass but it won't be consider an override of the superclass's static method are not subject to method overide

Remember the overloading & overring are different different concept in Java. Overloading involve defining multiple method in the same class with different parameter lists while overriding involve prouding a specific implementation of method in a subclass that has the same method signature as method in superclass

Q13) Explain the different among finalize & dispose?

→

| feature | dispose() method | finalize() method |
|---|---|---|
| define | It is define in the interface disposable interface | it is define in Java.lang.object class |
| Basic | It is use to class or release unmanaged resources store by object, like file or stream. | It is use to clear up unmanged resources owned by current object before it is destoyed |
| Syntax | the syntax of dispose() method is<br>Public void dispose(){<br>// Dispose code here<br>} | the syntax of finalize method is<br>Protected void finalize() {<br>// Here code is finalization<br>} |
| Access specifi | It is declare as public | It is declare as private |

Q14) Explain the difference among final, finally & finalize ?

→

| Basic comparison | final | finally | finalize |
|---|---|---|---|
| defn | final is a keycoord used in Java to restrict the modification of a variable, method or class | finally is a block used in Java to ensure that a section of code is always executed, even if an exception is thrown | finalize is a method in java used to perform cleanup process ing on the object before it is garbage collected. |
| uses | final is used to declare variable, method or class as unchangeble. | finally is used after a try a catch block to execute code that must be run regardless of whether an exception is thrown or not | finalize is used to perform clean up operation on an object. Such as closing files or releasing other resources, before it is garbage collected |
| Execution | final is executed at compile time | finally is executed at runtime | finalize is executed by the garbage collector before an object is destroyed |

| Basis of comparison | final | finally | finalize |
|---|---|---|---|
| Exceptional handling | final is not used for exceptional handling | finally is used as for a exceptional handling | finalize is not used for exceptional handling |

Q.15) Explain the difference among checked & unchecked exception?

| | check Exceptions | uncheck Exceptions |
|---|---|---|
| defn | checked at compile-time | Not check at compile time |
| Exception Type | IOException, ClassNotFoundException, etc | NullPointerException, ArrayIndexOutOfBounds Exception, etc |
| Handling Requirement | Must be caught or declared using throws keyword | Optional to catch or declare using throws keyword |
| Exceptional Handling | Handled using try-catch blocks or throws keyword | Not required to handle explicitly |

| | check Exceptions | uncheck Exceptions |
|---|---|---|
| flow control | Program flow is interrupted & transferred to catch block | Program execution is halted with an error message |
| eg. | file operation, database operation etc | logical error, invalid argument etc. |

Q16) Explain exception chaining ?

→ Exception chaining, also known as exception wrapping, is a practice in exception handling where a new exception is thrown with a reference to the original (or underlying) exception this allows you to add context or additional information to the exception without losing the original exception's stack trace & details

In Java, you can chain exceptions using the following constructor

Public YourException (String message, throwable cause)

1. You create a new exception (eg "YourException") with a message & reference to the original exception ("cause").
2. this new exception retains the stack trace & details of the original exception while allowing you to provide additional information in the message or context

chaining exception is useful for debugging & logging purpose, as it helps preserve the complete history of what went wrong starting from the root cause. it also helps in better error reporting & diagonosing problem in your application

Q17) Explain the difference among throw & throws ?

| | Throw | Throws |
|---|---|---|
| 1. | Java throw keyword is used to exception | Java throws keyword is used to declare an exception |
| 2. | checked exception can not be propagated using throw only | checked exception can be propagated with throws. |

| | Throw | Throws |
|---|---|---|
| 3. | throw is followed by an instance | throws is followed by class |
| 4. | throw is used with in the method | throws is used with the method signature |
| 5. | You cannot throw multiple exception | You can declare multiple exception |

Q18) In which case, finally block doesn't execute ?

→ A 'finally' block in Java doesn't execute in following case

i) If the JVM exits abruptly, for example, due to 'System.exit()' or a fatal error

ii) If the thread executing the 'try' or 'catch' block is interrupted by another thread using 'Thread.interrupt()'.

iii) If there is an infinite loop or an endless recursion inside the 'try' or 'catch' block preventing it from completing normally.

iv) If the system crashes or the Program is forcefully terminated from the outside (eg by the operating system)

Q19) Explain up casting?

→ upcasting in Java is the process of casting a subclass object to its superclass type. this is done implicitly & doesn't require an explicit cast operator. upcasting is safe & common in polymorphism scenarios, where you want to treat objects of derived classes as object of their base class

```
class Animal {
    Void makeSound() {
        System.out.Println("some sound");
    }
}

class Dog extends Animal {
    Void makeSound() {
        System.Out.Println("Bark");
    }
}
```

```
Public class Main {
    Public static void main (String [] args) {
        Animal myAnimal = new Dog(); // upcasting
        myAnimal.makeSound(); // calls Dog's makeSound() due to dynamic bind
    }
}
```

In the example above 'myAnimal' is upcasted from a 'Dog' object to an 'Animal' Refrence. this allows you to access the common properties + behaviours of the base class while still invoking the overriden method from the derived class when appropriate

Q20) Explain dynamic method dispatch?

→ Dynamic method dispatch is the mechanism in which a call to an overridden method is resolved at run time instead of compile time this is an important concept because of how java implements runtime polymorphism.

Java uses the principle of 'a super class reference variable can refer to a subclass object' to resolve calls to overridden method at run time. When a superclass reference is used to call an overridden method, Java determines which version of the method to execute based on the type of the object being referred to at the time call.

In other words, it is the type of object being referred to that determines which version of an overridden method will be executed.

Advantages of dynamic method dispatch.

1. It allows java to support "overriding of method", which are important for runtime polymorphism.
2. It allows a class to define method that will be shared by all its derived classes, while also allowing these sub-classes to define their "specific implementation" of a few or all of these methods
3. It allows subclasses to incorporate their own methods & define their implementation.

Q21) what do you know about final method ?

→ A final method in the Java is a method that cannot be overridden or modified by any subclass. You use 'final' keyword to declare a method as final. final method are used when you want to ensure that a method's implementation remain the same across all subclasses providing method security and preventing accidental changes to critical behaviors.

Q22) Explain fragile base class problem and how can we overcome it ?

→ the fragile base class problem occurs in object-oriented Programing when a class (the "Base class" or "Parent class") is extented by multiple subclasses. & changes made to the base class can unintentionally break or introduce bugs in derived class.

To overcome the fragile base class problem you can consider these strategies

1. Avoid Overly complex Base classes :- keep base classes simple and focused on a single responsibility, adhering to the single responsibility Principle. this reduce the likelihood of unintented consequences when modifying the base class.
2. Use abstraction :- favor interface and abstract classes over concerte base classes. this way, you can add new behavior in subclasses without altering the base class.
3. Encapsulation :- make use of encapsulation to hide implementation details & protect the base class's internals from subclasses. minimize the exposure of internal data.
4. Use composition :- Instead of inheritance, use composition. Create helper classes that can be composed within subclasses, reducing the need for extensive inheritance hierarchies.
5. Document Changes :- If you must make changes to a base class, thoroughly document these changes and their potential impacts on derived classes to alert developers to potential issues.

6. Version control and testing :- Implement version control and extensive testing to detect and prevent issues arising from changes to the base class

Q23) why java does not support multiple implementation inheritance ?

→ Java does not support multiple implementation inheritance (where a class can inherit from multiple classes with implementations) to avoid the "diamond problem" and maintain a simpler, more predictable and safe language. multiple inheritance can lead to issues such as ambiguity in method resolution and complex inheritance hierarchies. Instead, Java uses interfaces to achieve a form of multiple inheritance through interface implementation while preventing the ambiguities associated with multiple implementation inheritance.

Q24) Explain marker interface? list the name of some marker interface?

→ A marker interface in java is an interface that does not declare any methods. It's used to mark or tag classes that implement it, indicating that they possess a certain capability or property. marker interface are often used for reflection, serialization, or to indicate a contract without adding any methods. Some commonly used marker interface in Java are:

1. 'Serializable' :- Indicate that a class's objects can be serialized to be stored or transmitted.
2. 'Cloneable' :- Indicate that a class's objects can be cloned using the 'clone' method.
3. 'RandomAccess' :- Indicates that a list or collection supports efficient random access.
4. 'Remote' :- Used in Java RMI (Remote method Invocation) for distributed computing.
5. 'Readable' & 'Appendable' :- used in Java.nio package for input & output operation.

M T W T F S S
Page No.:
Date:
YOUVA

6. 'SingleThreadModel': Used in Java Servlet to indicate that only one thread should execute service method for servlet at a time

Q2e) Explain the significance of marker interface ?

→ the significance of a marker interface in Java lies in its ability to mark or tag classes to indicate a specific capability, property, or contract without adding any method.

1. Documentation :- They serve as a form of documentation, clearly indicating what a class can or cannot do.
2. Runtime information :- They provide runtime information that can be used for reflection & runtime decision-marking
3. Serialization and Persistence :- marker interfaces like 'Serializable' & 'cloneable' enable specific behaviors for object during serialization, cloning or other operation
4. Contract Enforcement :- They help enforce contracts and coding standards without introducing additional method signatures.

5. Framework Integration:- marker interfaces are often used in frameworks & libraries to apply specific functionalities or behaviors to classes implementing the marker

6. Simplicity:- Marker interface are simple and lightweight, marking them a convenient way to convey specific class characteristic