Name - Bhagyesh Deore
Roll No - 230940320030

JAVA Interview Questions

1) what do you know about JVM, JRE and JDK ?

→ JVM, JRE and JDK are essential components in the JAVA programming language ecosystem. Here's what each of them stands for and what they do:-

1) JVM (Java virtual machine):- Java virtual machine is an abstract machine that provides a runtime enviroment in which Java bytecode can be executed. when you compile a Java source file, it is translated into bytecode, which is platform-independent code. JVM is responsible for executing this bytecode on the host operating system, regardless of the underlying architecture and platform. It provides feature such as memory management, garbage collection and security

2) JRE (JAVA Runtime Enviroment):- Java Runtime Enviroment is a software package that provides everything that is required to run a Java-based application. It includes a JVM, class libraries and other supporting files to enable the execution of Java applications. In simple terms if you want to run a Java program on your computer, you needed to have the JRE installed Users who only want to run Java applications on thier systems typically need to install JRE

3) JDK (JAVA Development kit) :- Java developers kit is a software development kit used to develop Java applications. It includes the JRE, an interpreter / loader (JAVA), a compiler (javac), an archiver (Jar), a documentation generator (Javadoc), and other tools needed for Java development. In other words, JDK is a full-featured software development kit for Java, including everything that is in the JRE, as well as tools for compiling and debugging Java code. Developers use JDK to create JAVA applications, applets and servlets.

2) Is JRE platform dependent or independent?

→ Java Runtime Environment is designed to be platform independent. Once a Java program is compiled into bytecode, it can be executed on any system with the appropriate JRE installed, regardless of the underlying operating system and hardware architecture. This platform independence is one of the key features of Java, achieved through the use of the Java virtual machine (JVM), which interprets and executes the bytecode on different platforms.

3) which is ultimate base class in Java class hierarchy & list the names of methods of it?

→ In Java, the ultimate base class for all classes is the 'Object' class. Every class in Java is directly or indirectly derived from the 'Object' class. The Object class is a part of the "java.lang" package, and it contains several methods that are inherited by all Java classes. Some of the important methods of the Object class include:

1) toString() :- This method returns a string representation of the Object. It is often overridden in user-defined classes to provide a meaningful string representation of the object state.

2) equals(Object obj) :- This method compares the current object with the specified object for equality. By default, it compares memory addresses, but it is usually overridden in user-defined classes to compare the contents of the objects.

3) hashCode() : This method returns a hashCode value for the object. It is used in hash-based collections like HashMap.

4) getClass() :- This method returns the runtime class of an object. It is useful for obtaining information about the object's type

5) notify(), notifyAll(), and wait(): These methods are used for inter-thread communication and synchronization.

6) finalize(): This method is called by the garbage collector before the object is garbage collected. It can be overridden to provide cleanup operations.

4) Which are the reference types in java?

→ In Java, there are several types of references that you can use to refer to objects. These reference types determine how objects are managed in memory and how they can be accessed. The main types of references in Java are:

1) Strong References:- Strong references are the standard reference type in Java. When you create an object and assign it to a reference variable using the 'new' keyword, you are creating a strong refrence. As long as there is strong reference to an object, the object will not be garbage collected.

2) Weak References:- Weak references allow objects to be garbage collected when there are no strong references pointing to them. Weak reference are useful when you want

to have non-essential references to objects. Weak references are implemented using the 'WeakReference' class.

3) Soft Reference :- Soft references allow objects to be garbage collected only when the JVM is running low on memory. Soft references are useful for implementing memory-sensitive caches. Soft reference are implemented using the SoftReference class

4) Phantom References :- Phantom reference are used to schedule post-mortem clean-up actions before an object is removed from memory. Phantom references are often used in conjuction with the 'java.lang.ref.ReferenceQueue' class to implement custom cleanup logic.

5). Explain narrowing and widening ?

→ In the context of Java data types, narrowing and widening refer to the process of converting a value from one data type to another. These conversions can be implicit or explicit depending on whether they are done automatically by the compiler or require manual interrention by the programmer.

1) Widening (implicit) conversion:

Widening or implicit conversion, occurs when you convert a value from a smaller data type to a larger data type. Since there is no loss of data in this conversion. Java allows it to happen automatically. for example, converting an int to a double.

```
int value = 42;
double dv = value;
```

2) Narrowing (Explicit) conversion:-

Narrowing, or explicit conversion, occurs when you convert a value from a larger data type to a smaller data type since there is a potential loss of data in this conversion (due to the smaller range of the destination data type). Java requires explicit casting to perform narrowing conversions. for example, converting a double to an int.:

```
double dvalue = 42.56;
int ivalue = (int) dvalue
```

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

→ It is possible to write a statement without a semicolon by utilizing other language constructs. One common technique to print "HelloCDAC"

Without a semicolon at the end is by using a code block:-

```
Public class Main {

    Public Static void main (String [] args) {

        if (System.out.printf ("Hello CDAC" + "\n") = null)
        {
        }

    }
}
```

In this code, the "printf" method is used to print the string "Hello CDAC". The statement is inside an 'if' block and the condition is always 'false' (since 'system.out.printf' never returns 'null'). Therefore, the block of code inside the 'if' statement will be executed without the needed for a semicolon at the end of the print statement. Note that this approach is not recommended for actual code as it sacrificed readability for a trick. Using semicolons properly enhances code readability and maintainability.

7) can you write java application without main function? if yes, how?

→ Yes, we can execute a java program without a main method by using a static block.

Static block in java is a group of statements that gets executed only once when the class is loaded into the memory by Java classLoader, it is also known as a static initialization block. Static initialization block is going directly into the stack memory.

Example.

```
Class staticInitializationBlock {
    static {
        System.out.println("class without a main
                            method");
        System.exit(0);
    }
}
```

8) What will happen, if we call main method in static block?

→ When the Java virtual machine (JVM) starts running a Java program, it looks for the 'public static void main (String[] args)' method in the specified class and executes the code inside it. if you call the 'main' method from a static block, it will execute when the class is loaded, bypassing the normal entry

point for the program. Heres an example to illustrate :-

```
public class main {

    static{
        System.out.println("Inside static block");
        main(new String[]{});
    }

    public static void main(String[] args) {

        System.out.println("Inside main method");
    }
}
```

In this code, the main method is called from a static block. When you run this program, it will output :-

```
Inside static block
Inside main block
```

Q In System.out.println, Explain meaning of every word?

→

1) System :- System is a pre-defined class in Java and is a part of the java.lang package

2) out : 'out' is a static member of the 'System' class and represents the standard output stream. It is an instance of the PrintStream class, which provides methods to print data to the console or another output destination. out is an Object of type PrintStream and is initialized to the standard output

3) println : println is method of the PrintStream class. It stands for "print line". The println method is used to print a string or any other data type to the standard output, followed by a newline character ('\n'). After printing the data, it moves the cursor to the next line, so the next output appears on new line.

10) How will you pass object to the function by reference in Java. ?

→ When you pass an object reference to function, you are passing the value of the reference, not the actual object. However, you can modify the state of the object inside the function because you have a reference to it. Changes to the object's internal state are visible outside the function, but reassigning the reference inside the function will not affect the original reference outside the function.

Here's an example to illustrate this concept:-

```
class Myclass {
    int value;

        Myclass (int value) {
            this.value = value;
        }
    }

public class Main {
        public static void modifyObject (Myclass ob){
            Obj.value = 100;
            Obj = new Myclass (200);
        }

public static void main (String args[]) {

    Myclass myobject = new Myclass (10);
System.out.println ("Before: " + myobject.value);

modifyObject (myobject);

System.out.println ("After: " + myobject.value);

}

}
```

M T W T F S S

Page No.:	YOUVA
Date:	

11) Explain constructor chaining ? How can we achieve it in C++ ?

Constructor chaining, also known as constructor delegation is a concept in Object-oriented programming where one constructor calls another constructor within the same class. This allows you to reuse the code of one constructor in another constructor reducing redundancy and promoting code reusability. Constructor chaining is commonly used to initialize object state efficiently by avoiding duplicate code in multiple constructors.

In C++, constructor chaining is achieved through the use of member initializer lists. Here's an example to illustrate constructor chaining in C++.

```
#include <iostream>
using namespace std;

class MyClass {

    public:
        int value;

    MyClass() : MyClass(0) {
        cout << " Default constructor called" << endl;
    }

    MyClass (int val) : value (val) {
        cout << parameterized constructor call " << val << en
};
```

```
int main() {

    Myclass obj1;
    Myclass obj2(42);

cout << "value of obj1: " << obj1.value << endl;
cout << "value of obj2: " << obj2.value << endl;

return 0;

}
```

In this example, Myclass has two constructors: a default constructor and a parameterized constructor. The default constructor calls the parameterized constructor with an arguments of '0' using member initializer list syntax (: Myclass(0)). This way, the parameterized constructor is reused within the default constructor. When you create objects of Myclass, you can observe the constructor chaining in action

12) which are the rules to overload method in subclass

→ Method overloading allows you to define multiple methods in the same class with the same name but different parameters. When you extend a class (creating a subclass), you can also overload methods in the subclass. The rules for method overloading in the super class, Here are the key rules to keep in mind when overloading methods in a subclass.

1) Method signatures: The method signature consists of the method name and the parameter list. When overloading a method in a subclass the method must have a different parameter list than the methods with the same name in the superclass. This can include a different number of parameter, different types of parameters or both.

2) Access Modifiers:- The access level of the overloaded method in the subclass cannot be more restrictive than the access level of the method in the superclass. For example, if the superclass method is public, the subclass method can be public or protected, but not private.

3) Return type:- The return type of the overloaded method can be different from the return type of the method in the superclass. However, changing only the return type is not sufficient to distinguish overload methods. Overloaded methods must have different parameter lists.

13) Explain the difference among finalize and dispose

→ In the context of Java and C#, finalize() and dispose() serve similar purposes, but they are used in different programming paradigms (Java for garbage collection and C# for resource management). Let me explain the differences between these methods.

- finalize() in Java :-

In Java, the finalize() method is a special method provided by the Object class. It's called by the garbage collector before an object is collected, giving the object a chance to clean up any resources it holds. However, finalize() is considered deprecated in modern Java programming and it's not recommended to rely on it for resource cleanup. This is because the timing and guarantee of when finalize() will be called are not predictable and it might not be called at all in certain situations.

- dispose() in C# :-

In C#, the dispose() method is used for resource management, especially when dealing with unmanaged resources like file handles, database connections or graphic objects. It is part of the IDisposable pattern. Classes that implement the IDisposable interface should provide a dispose() method to release resources explicitly.

The dispose() method allows developers to release resources immediately rather than relying on the garbage collector. It's commonly used in conjuction with the using statement in C# to ensure proper resource clean up.

14) Explain the difference among final, finally and finalize?

→ Final, finally and finalize are three distinct concepts in Java programming, each serving a different purpose. Here's a breakdown of the differences among them:-

1) final :- final is a keyword in Java that can be applied to variables, methods and classes, indicating different levels of immutability and restriction.

- final variable :- When applied to a variable final means that the variable cannot be reassigned after its initial assignment.
- final Methods :- When applied to a method final indicates that the method cannot be overriden by subclasses.
- final Classes :- When applied to a class final means that the class cannot be subclassed.

2) finally :- finally is a block in Java associated with exception handling. It is used in a try-catch - finally block structure to ensure that certain code gets executed regardless of whether an exception is thrown or not. The finally block is often used for cleanup operations, such as closing files or releasing resources. For example:-

```
try {
}
catch (Exception e) {
}

finally {
}
```

3) finalize :- finalize() is method defined in the Object class in Java. It is called by the garbage collector before an Object is removed from memory. The finalize() Method can be overridden in a class to provide custom cleanup code for objects before they are garbage collected.

M T W T F S S

Page No.: YOUVA

Date:

11) Explain the difference among checked and unchecked exception?

→

	Checked Exception	Unchecked Exception
1	Checked exception happen at compile time when source code is transformed into an executable code	Unchecked exceptions happen at runtime when the executable program starts running
2	The checked exception is checked by the compiler	These type of exceptions are not checked by the compiler.
3	Checked exceptions can be created manually	They can also be created manually
4	This exception is counted as a sub-class of the class	This exception happens in runtime and hence it is not included in an exception class
5	Java virtual machine requires the exception to be caught or handled	Java virtual machine does not need the exception to be caught or handled

16) Explain exception chaining?

→ Exception chaining, also known by exception wrapping, is a technique used in programming where one exception is wrapped inside another exception. This approach is particularly useful when you want to capture and convey additional contextual information about an error without losing the original exception's stack trace or information.

In languages like Java, you can achieve exception chaining by passing the original exception as a parameter to the constructor of a new exception. This allows you to associate a specific context or reason for the new exception, while preserving the details of the original exception.

M T W T F S S

Page No.: YOUVA

Date:

11. Explain the difference among throw and throws?

	Throw	Throws
1)	The throw keyword is used inside a function. It is used when it is required to throw an Exception logically	The throws keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions.
2)	The throw keyword is used to throw an exception explicity. It can throw only one exception at a time	The throws keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs if matched with the declard one, is thrown automatically then
3)	syntax of throw keyword includes the instance of the exception to be thrown. Syntax wise throw keyword is followed by the instance variable	syntax of throws keyword includes the class names of the exceptions to be thrown. Syntax wise throws keyword is followed by exception class names.
4)	Throws keyword cannot propagate checked exceptions. It is only used to propogate the uncheck exceptions	throws keyword is used to propagate the checked Exceptions only.

18. In which case, finally block doesn't execute?

→ In Java the finally block associated with a try-catch-finally construct is designed to execute under most circumstances. However, there are a few scenarios in which the finally block may not execute:-

1) System.exit(): if your code invokes the System.exit() method with a status code before the finally block is reached, the Java Virtual Machine (JVM) terminates and the finally block does not get a chance to execute.

2) Infinite Loop or Hang: If your code gets stuck in an infinite loop or hangs indefinitely (e.g. due to deadlock), the finally block may never get a chance to execute because the program does not progress beyond the try block.

It's important to note that under normal program execution and exception handling scenarios, the finally block is designed to execute even if exceptions are thrown and caught. However, the two scenarios mentioned above can prevent the finally block from executing.

19. Explain upcasting?

→ Upcasting is a type of Object typecasting in which a child Object is typecasted to a parent class object. By using the Upcasting, we can easily acces the variables and methods of the parent class to the child class. Here we don't acces all the variables and the method. We acces only some specified variable and methods of the child class. Upcasting is also known as Generalization and widening.

20. Explain dynamic method dispatch?

→ Dynamic method dispatch is a mechanism in Object oriented programming language, such as Java, that enable the selection of method to be executed at runtime rather than compiler time. It allows you to call a method on an object and the specific implementation of that method is determined by the actual type of the Object. At runtime, Dynamic method dispatch is a fundamental concept in achieving polymorphism in object-oriented programming.

Key points about dynamic method dispatch:

1) Polymorphism
2) Inheritance and Overriding
3) Base and Derived Classes.

21. What do you know about final method?

→ When a method is declared as final, it cannot be overriden by a subclass. This is useful for methods that are part of a class's public API and should not be modified by subclasses.

22. Explain fragile base class problem and how can we overcome it?

→ The fragile base class problem is a software design issue that occurs in object-oriented programming when a class is widely used as a base class (parent class) for other classes and changes made to the base class can unintentionally break or introduce bugs in the derived classes. This problem arises when subclasses rely on the behaviour and implementation details of the base class and any changes to the base class can have unintended consequences.

Here are some common scenarios that can lead to the fragile base class problem:

- Adding new methods
- Changing method signatures
- Changing internal data structures
- To overcome the fragile base class problem, you can follow these design principles and best practices:

- Minimum coupling
- Use Abstraction
- LISKOV substitution principle
- Design by contract
- Interfaces and Abstract classes
- Interfaces and Abstract classes
- Testing and Regressing Testing
- Documentation

23) Why Java does not support Multiple implementation inheritence?

→ In Java, a class cannot extend more than one class, ~~therefore following illegal~~ to avoid the complexities and ambiguities that can arise from it. Multiple inheritance allows a class to inherit properties and behaviours from multiple classes while this might seem advantageous, it introduces several problems that can make the language more challenging to understand, maintain, and debug.

1) Diamond problem: One of the most well-known issues with multiple inheritance is the diamond problem. It occurs when a class inherits from two classes that have a common ancestor. If both parent classes have the same method and the child classes does not override it its ambiguous which method should be inherited. This ambiguity can lead to unexpected behaviour

2) Complexity and maintenance :- Multiple inheritance can make the code more complex and harder to maintain.

3) Readability and understandability.

4) Cyclic dependencies.

24) Explain marker interface? List the name of some marker interface?

→ A marker interface in Java is an interface with no methods or fields. It is used to indicate a certain capability or feature of a class that implements it. The presence of a marker interface signifies some meta-data about the class or its objects. Classes implementing marker interfaces can be treated differently by other classes or systems at runtime based on the interface they mark.

Here are some examples of marker interfaces in Java :-

1) Serialization interface
2) Cloneable interface
3) Remote interface
4) Annotation interfaces
5) EventListener interface

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

25. Explain the significance of marker interface?

→ The main use of marker interface in Java is to convey to the JVM that the class implementing some interface of this category has to be granted some special behavior

e.g when a class implements the serializable interface, which is a marker interface, then this is an indication to the JVM that the objects of this class can be serialized similarly, when a class implements cloneable interface, then it indicates to the JVM that the objects of this class can be cloned.