Interview Questions

FREEMIND

Date

Page

Q1 What do you know about JVM, JRE and JDK?

JVM: Java Virtual Machine

→ The JVM is responsible for interpreting the bytecode and translating it into machine code. (.class file contain bytecode, bytecode is object oriented assembly code (because assembly code is in class) designed for JVM) that can be executed by the underlying Operating System.

→ It also provides memory management, security, and other runtime services necessary for executing Java Applications.

JRE: Java Runtime Environment

→ JRE is platform dependent software to run Java Application on Client's machine. To run Java application we must deploy/install JRE.

JRE = rt.jar + Java Virtual Machine

JDK: Java Development Kit

→ JDK is a Java Specific SDK. JDK is necessary for developing Java Applications.

→ JDK consists of :-

JDK = Java Development Tools + Java Docs + rt.jar + JVM

- Java Development Tools: javac, java, javap, javadoc etc.
- Java Docs: HTML Pages, which contains help of core Java API
- rt.jar: It contains core Java API.
- Java Virtual Machine: It is abstract computer which manages execution of bytecode.

→ JDK is a platform dependent software.

→ Developer must install JDK to develop Java Application.

FREEMIND

Date

Page

Q2. Is JRE platform dependant or independent?

→ The Java Runtime Environment (JRE) itself, as a software is provided by Oracle, is indeed platform-dependent in the sense that different versions of the JRE are available for various operating sys. & architectures.

→ However, the key point of platform independance in Java lies in the fact that Java source code is compiled into platform-independent bytecode. This bytecode can be executed on any platform that has a compatible JRE without requiring the source code to be modified for each platform.

Q3. Which is ultimate base class in java class hierarchy? List name of methods of it?

→ In Java, the ultimate base class in the class heirarchy is the 'java.lang.Object' class, which provides a set of common methods :-

1. 'equals(Object obj)': Used for comparing objects for equality.
2. 'hashCode()': Returns a hash code value for the object.
3. 'toString()': Returns a string representation of the Object.
4. 'getClass()': Returns the runtime class of the Object.
5. 'notify()': Wakes up a single thread waiting on this object's monitor.
6. 'notifyAll()': Wakes up all threads waiting on this object's monitor.
7. 'wait()': Causes the current thread to wait until another thread notifies it or until a specified amount of time has passed.

FREEMIND

Date

Page

Q4. Which are the references types in Java?

→ References Types in Java are used for creating and manipulating objects, by storing references to their memory locations rather than the actual data.

→ Following are the reference types in Java

1. class type :- This reference type point to an obj. of class
2. array type :- points to an array.
3. interface type :- points to an object of a class implements an interface.

→ These variables only store the address of these values

→ Default values of any reference var. is null.

→ A reference variable can be used to refer any object of the declared type or any compatible type.

eg:- Animal animal = new Animal ("elephant");

Q5. Explain narrowing and widening?

Widening:-

Process of converting, value of variable of narrower type into wider type is called as widening.

```
public static void main (String [] args) {
    int num1 = 10;
    double num2 = (double) num1;
    System.out.println ("Num2: " + num2); // Num2: 10.0
}
```

In case of widening conversion, explicit typecasting is optional

```
int num1 = 10
double num2 = num1; // OK : Widening.
```

Date

Page

Narrowing

→ Process of converting value of variable of wider type into narrower type is called as narrowing.

```
double num1 = 10.5d;
int num2 = (int) num1; //Narrowing
```

→ In case of narrowing, explicit type casting is mandatory.

```
double num1 = 10.5d;
int num2 = num1; // error : incompatible type possible lossy conversion from double to int.
```

Q6. How will you print "Hello CDAC" statement on screen without semicolon?

→ Program 1 :- Using If statement.

```
# include <stdio.h>
int main () {
if (printf("Hello CDAC")) { }
return 0;
}
```

Program 2 :- Using switch statement

```
# include <stdio.h>
int main () {
switch (printf("Hello CDAC")) { }
return 0;
}
```

Program 3 :- Using while loop.

```
# include <stdio.h>
int main () {
while (!printf("Hello CDAC")) { }
return 0;
}
```

FREEMIND

Date

Page

Q7. Can You Write Java application without main fn()? 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 get executed only once when the class is loaded into the memory by Java class Loader, it is also known as Static Initialization Block.

Static initialization block is going directly into the heap memory area.

Example:-

```
class StaticInitBlock {
    static {
        System.out.println("class w/o main method");
        System.exit(0);
    }
}
```

In above example, we can execute a Java Program with out a main method (Works until Java 1.6 version) Java 7 and newer version don't allow this because JVM checks the presence of the main method before initializing the class.

Output : class without a main method.

Q8. What will happen, if we call main method in static block?

→ In Java, the main method is the entry point of a a Java Application when you execute a Java Program, the JVM starts by invoking the 'main' method of the class specified as the main class.

→ If you attempt to call the main method from within a static block of the same class or any other class, it will not have any special behaviour compared to calling any other static method. In others words calling main in a static block / method is just invoking a static method and won't change the entry point of your Java application.

Example :-

```
public class MainClass {
    static {
    // This is static block
System.out.println("Inside static block");
main(new String[] {"arg1", "arg2"});
//calling main method from the static block.
}
public static void main(String[] args) {
System.out.println("Inside main method");
for (String arg : args) {
System.out.println("Argument : "+arg);
}
}
}
```

Output :- Inside static block
Inside main method
Argument : arg1
Argument : arg2

FREEMIND

Date

Page

Q9. In System.out.println. Explain meaning of every word

→ System :- System is a final class declared in java.lang package.

out :- out is a reference of java.io.PrintStream, declared as public static final field inside System class.
PrintStream is a class declared in java.io package

println :- println is a non-static method of java.io PrintStream class. It is a overloaded method.

Q10. How will you pass Object to the function by reference?

→ In Java, objects are passed by value of the reference

→ This means that while the reference to the object is passed to the function, modifications to the reference itself will not affect the original reference outside the function.

→ However modifications to the objects properties/state will be reflected outside the function.

Example :-

```
Class MyObject {
    int value;
}

Void modifyObject (MyObject obj) {
    obj.value = 42;
}

Public static void main (String[] args) {
    MyObject myObj = new MyObject();
    myObj.value = 6;
    modifyObject(myObj);
}
```

myObj.value is now 42 because it was passed by value of the reference

Q11. Explain Constructor chaining & How can we achieve it in C++?

→ Constructor chaining is a concept in OOP where one constructor of a class call to another constructor of the same class to avoid code duplication & ensure that common initialization code is executed.

We can use constructor chaining using this keyword

```
eg- Date () {
        this (0,0,0);
    }
    Date (int a, int b, int c) { }
```

→ In C++ constructor chaining is achieved using member initializer list & the use of one constructor to call another constructor of the same class.

```
eg - myCon () {//todo}
     myCon (int val) : myCon () {
         //todo
     }
```

Q12. Which are the rules to overload method in sub class?

→ According to the OOPs, if implementation of any method is logically same equivalent then we should give same name to the method.

If we want to give same name to the method then we should follow some rules:

FREEMIND

Date

Page

1. If we want to give same name to the method and if type of all the parameters are same then number of parameters passed to the method must be different.

```
Static Void sum (int num1, int num2){
        // 2 method parameters
    }
Static Void sum (int num1, int num2, int num3){
        // 3 method parameters
    }
```

2. If we want to give same name to the methods and if number of parameters passed to the method are same then type of at least one parameter must be different.

```
Static Void sum (int num1, int num2) {
        // 2 method parameter (int, int)
    }
Static Void sum (int num1, double num2) {
        // 2 method parameter (int, double)
    }
```

3. If we want to give same name to the method and if no. of parameters passed to the method are same then order of type of parameters must be different

```
Static Void sum (int num1, float num2) {
        // 2 method parameter (int, float)
Static Void sum (float num1, int num2) {
        // 2 method parameter (float, int)
    }
```

Date
Page

4. Only on the basis of different return type, we can not give same name to methods.

5. When we define multiple methods with same name using above rules then it is called as method overloading.

Q13. Explain the difference among finalize and dispose?

Features	Dispose( ) Method	Finalize( ) method
1. Defined	It is defined in the interface disposable interface	It is defined in java.lang.Object class
2. Basic	It is used to close or release unmanaged resources stored by an object, like files or streams.	It is used to clean up unmanaged resources owned by the current object before it is destroyed.
3. Syntax	The syntax of dispose () method is Public Void dispose(){ //Todo }	The syntax of finalize() method is: protected void finalize(){ //Todo }
4. Access Modifier	It is declared as Public.	It is declared as protected
5. Invoked	Invoked by user	Invoked by garbage collector.
6. Speed	Very Quickly	It is invoked more slower than dispose() method

FREEMIND

Date

Page

7.	Performance	It executes immediate action and has no impact on the performance of the site.	It has an impact on the performance of the site.
8.	Implementation	Everytime whenever there is a close() fn. the dispose() method should be implemented.	Unmanaged resources must be implemented using the finalize() method.

14. Explain difference among final, finally and finalize?

→ Final

1. It is keyword, we can use with local variable field method and class.
2. If we use final modifier with method local variable and field then it is considered as constant.
3. If we use final modifier with method then we can not override/redefine method inside sub class.
4. If we use final modifier with class then we can not extend it.

Finally

1. It is a block that we can use with try/catch.
2. If we want to close local resources then we should use finally block.

Date
Page

Finalize

1. It is non final method of java.lang.Object class.

2. If we want to release class level (which is declared as field) resources then we should use finalize method.

Q15. Explain the difference among checked and unchecked exception?

→ Checked exception and unchecked exception are types of exception in Java, which are designed for Java compiler (Not for JVM)

1. Checked Exception :-

→ Java.lang.Exception is considered as superclass of all the checked Exception.

→ java.lang.Exception and all its subclasses except java.lang.RuntimeException (and its subclasses) are considered as checked exceptions.

Example of checked Exceptions

- java.lang.CloneNotSupportedException
- java.lang.InterruptedException
- java.io.IOException
- java.sql.SQLException.

→ Compiler force developer to handle or to use try-catch block for check-exception.

2. Unchecked Exception.

→ java.lang.RuntimeException is considered as superclass of all the unchecked exception

Date

Page

→ java.lang.RuntimeException and all its subclasses are considered as unchecked exception.

→ Example of unchecked exception
- Runtime Exception.
- Number Format Exception
- Null Pointer Exception.
- Negative Array Size Exception
- Array Index Out of Bounds Exception.
- Array Store Exception
- Class Cast Exception
- Arithmetic Exception.

→ Compiler do not force developer to handle or to use try-catch block for unchecked Exception.

Q16. Explain Explain Chaining?

→ Process of handling exception by throwing new type of exception is called as exception ~~handling~~ chaining

→ Chained exception are associated such that the previous exception causes ~~both~~ each exception in the chain.

→ It can help debug, as it can help us track down the root cause of an error.

→ We can use exception chaining in situation where is another exception. However its important to note that chaining can make our code more difficult to read and understand. Therefore we should use

Date

Page

exception chaining sparingly and only when necessary.

→ If we use chained exceptions, it is a good idea to document the chain in our code. It'll help others understand the code and make it easier to debug if an error occurs.

→ For example, if an InputStream throws an IOException and the InputStream's read() method throws an EOFException, then the EOFException is the cause of the IOException. The following code demonstrates the use of chained exception.

```
import java.io.*;
class ChainedExceptionDemo {
public static void main (String[] args) throws
IOException {
        try {
            throw new IOException("IOException Encountered")
.initCause (new EOFException ("root cause is EOF Exception"));
        catch (Throwable e) {
        // Handle the IOException
    System.out.println ("Caught exception→ "+e);
        // Handle the EOF Exception here
EOFException eof = (EOFException) e.getCause();
System.out.println ("The cause is : "+eof);
}
}
}
```

FREEMIND

Date

Page

Q17. Explain the difference among throw & throws?

throw :-

→ It is keyword in Java

→ If we want to generate new exception then we should use throw keyword

→ Only Objects that are instances of Throwable class (or one of its subclasses) are thrown by the JVM. Or can be thrown by the Java throw statement

String ex = new String ("Divide by zero exception");
throw ex; //No exception of type String can be thrown an exception type must be a subclass of Throwable.

→ Throw statement is a jump statement.

throws :-

→ It is a keyword in Java.

→ If we want to delegate exception from method to caller method then we should use throws keyword clause

→ Public class Program {
public static void displayRecord() throws InterruptedException{
for (int count = 1; count <= 10; ++count) {
System.out.println("Count : " + count);
Thread.sleep(500);
}
}

Public static void main (String[] args) {
try { program.displayRecord();
} catch (InterruptedException e) {
e.printStackTrace();
}}

Date
Page

Q18. In which case finally block doesn't execute?

→ The finally block may not execute if the JVM exists while the try or catch code is being executed

The try block of the writeList method that you've been working with here opens a PrintWriter. The Program should close that stream before existing the writeList method.

Q19. Explain Upcasting

→ Super Class reference variable can contain reference of instance of subclass. It is called as Upcasting

→ Upcasting is the typecasting of a child object to a parent object.

→ Upcasting can be done implicitly. Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child members using this feature.

→ Instead of all the members, we can access some specified members of the child class.

→ For instance, we can access the overridden methods

→ Example:-

Let there will be an animal class. There can be many different classes of animals. One such class is Fish.

So lets assume that the Fish class extends the Animal class

FREEMIND

Date

Page

→ Therefore the two way of inheritance in this case as:

```
Object o1 = new Boolean (true); // Upcasting
Number n1 = new Byte (123); // Upcasting
Byte b1 = new Byte (123); //OK But not Upcasting
Object o = new Integer (123456); //Upcasting.
```

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 at compile time. This is an important concept because of how Java Implements Run-Time Polymorphism.

→ Java uses the principle of a superclass reference variable can refer to "subclass object". To resolve calls to overridden methods at run-time.

→ When a superclass reference is used to call on overridden method. Java determines which version of the method to execute based on the type of the object being reffered to at the time call.

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

Advantages of Dynamic method Dispatch.

→ It allows Java to support overridding of methods which are important for run-time polymorphism.

→ It allows a class to define methods 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 those methods.

→ It allows subclasses to incorporate their own methods and define their implementation.

Q21. What do you know about final method?

→ Same as Q. no. 14.

Q22. Explain fragile base class problem and how can we even come it?

→ If we make changes in the body super-class then we must recompile super class as well as all its subclasses. This problem is called as fragile base class problem.

→ We can solve fragile base class problem by defining super type as interface.

Q23. Why Java does not support multiple implementation inheritance?

→
```
Class A { }
Class B { }
Class C { }        // NOT OK
      ↓
   extends A, B
// Multiple implementation inheritance.
```

Date
Page

→ In C++, combination of two or more than two types of inheritance is called Hybrid Inheritance

→ If we combine hierarchical & multiple inheritance together then it becomes diamond inheritance which creates same problem.

→ To avoid diamond problem, Java do not support multiple implementation inheritance / multiple class inheritance

→ Conclusion : In Java, class can extend only one class.

Q24. Explain marker Interface ? List the name of some marker interface?

→ An interface which do not contain any member is called as marker/ tagging interface.

→ Marker Interface are used to generate metadata. It helps JVM to perform some operations e.g. to do done serializing state of Java instance etc

Examples

- java.lang.Cloneable
- java.util.EventListener
- java.util.RandomAccess
- java.rmi.Remote

Q25. Explain the significance of marker interface.

→ Its significance lies in its ability to mark or tag a class as having a specific characteristics or behavior without providing any implementation details. Here's a brief explanation of the significance of marker interface.

1. Identification: Marker interfaces are used to identify or classify classes based on certain criteria or characteristics. When a class implements a marker interface it indicates that the class possesses specific traits or compabilities associated with that interface

2. Semetic Information: They provide semetic information to both programmers and the runtime environments. This can help developers understand the intended use or behaviour of a class without needing to examine its source code.

3. Compile Time and Runtime Checks: Marker interfaces are often used in situation where you want to perform compile-time or run-time checks based on the class's characteristics. For example in Java, the 'serializable' marker interface is used to indicate that a class can be serialized, and this information is used during object serialization and deserialization.

4. Reflection and Framework: Marker interfaces are sometimes used in reflection and by frameworks or libraries to discover and work with classes that meet specific criteria. For instance, a

Date

Page

framework might search for classes implementation of a particular marker interface and apply certain behaviours or rules to them.

5. Customization and Extension: Developers can create their own marker interface to customize or extend the behaviour of their classes within a framework or application. By implementing a specific marker interface, classes can opt into additional functionality or behaviour.

6. Documentation and Contracts: They serve as a form of documentation or contract between developers. By adhering to marker interfaces, developers signal their intention to follow certain conventions or requirements.

In summary, marker interfaces are a way to provide metadata about a class's capabilities or characteristics aiding in code organization, communication and the application of specific behaviours or rules.

They are simple but effective mechanism for adding meaning to classes in object-oriented programming