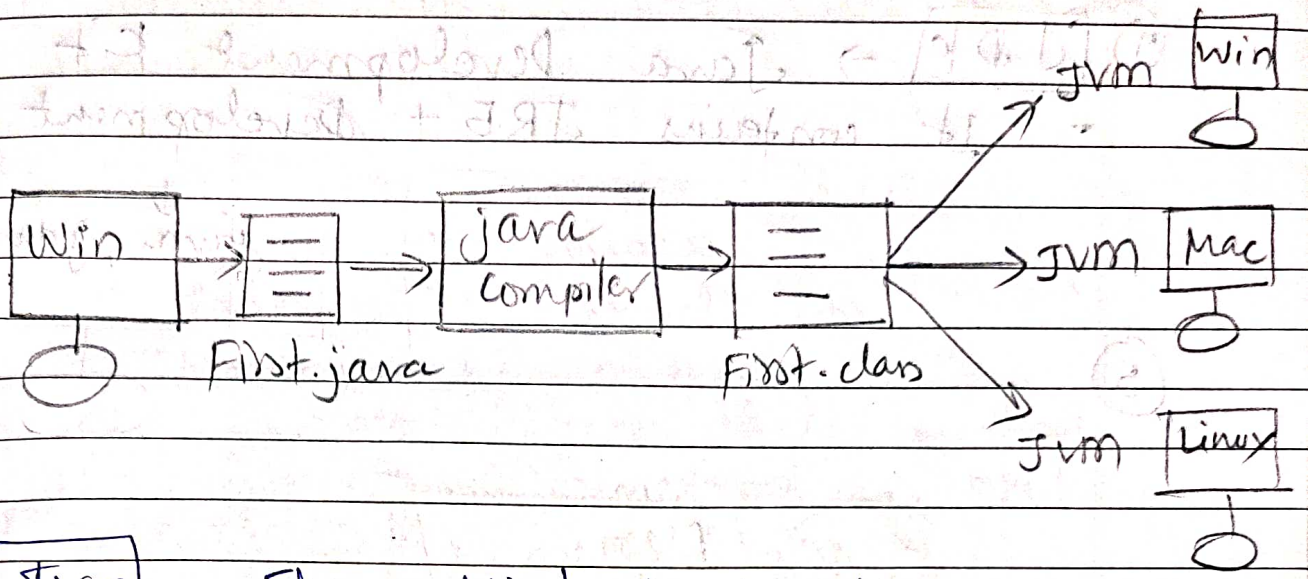
Java Interview Questions

Q.① What do you know about JVM, JRE and JDK?

→

①



② JVM → Java Virtual Machine

- It is an abstract machine
- Specification that provides run-time environment in which java bytecode can be executed.

- It is platform dependent.
- It does the following tasks:
  1. loads code
  2. Verifies code
  3. Executes code
  4. Provides runtime environment.

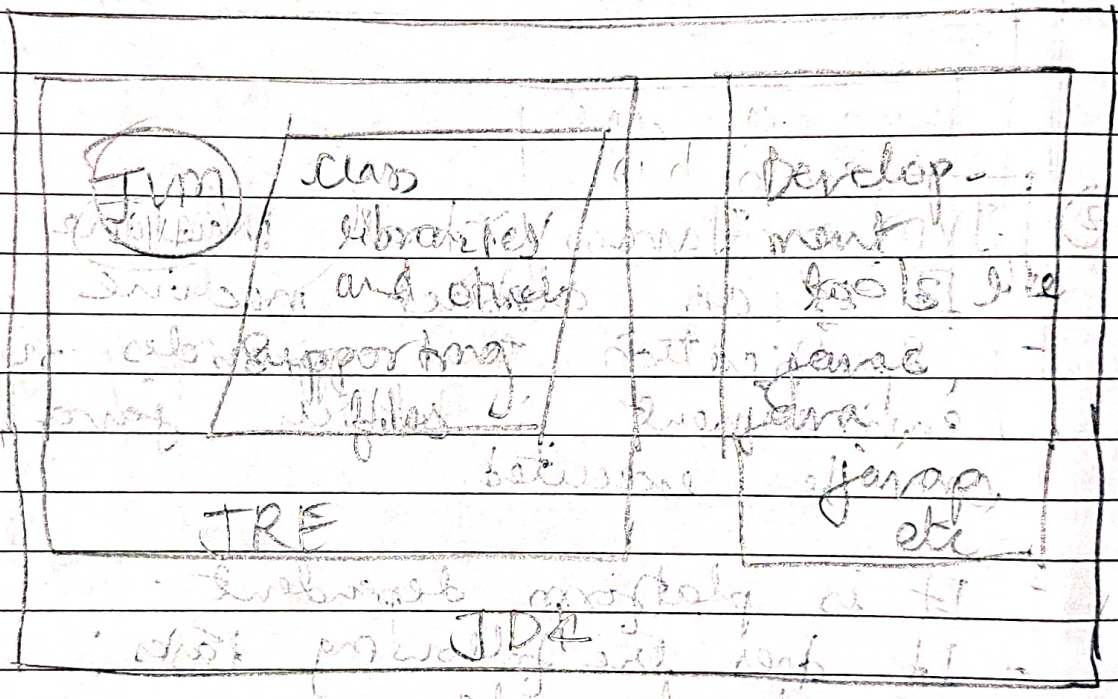
③ JRE → Java Runtime Environment

- It is an implementation of the JVM that actually executes our java programs.
- It contains JVM, class libraries and other supporting files.

- Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE.

(4) JDK → Java Development Kit
- It contains JRE + Development tools.
↓
java, javac, jar

(5)



Q.2 Is JRE platform dependant or independent?

→ (1) JRE is platform dependent software which we can download from oracle site.

(2) To run Java application on any machine, the machine must be equiped with JRE.

Q) Which is ultimate base class in java class hierarchy? List the name of methods of it?

→ ① Object class is base class of all the classes in java.

② Object class provides multiple methods which are as follows:
1. toString() method
2. hashCode() method
3. equals(Object obj); method
4. finalize() method
5. getClass() method
6. clone() method
7. wait(), notify(), notifyAll() methods.

① public String toString()
- returns the string representation of this object.

② public int hashCode()
- returns the hashcode number for this object

③ public boolean equals(Object obj)
- compares the given object to this object

④ protected void finalize() throws Throwable
- is invoked by the Garbage collector before object is being garbage collected

(5) public final Class getClass()
- returns the Class class object. The Class class can further be used to get the metadata of this class.

(6) protected Object clone() throws CloneNotSupportedException
- creates and returns the exact copy of this object.

(7) public final void wait(long timeout) throws InterruptedException
- causes current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).

(8) public final void notify()
- wakes up single thread, waiting on this object's monitor.

(9) public final void notifyAll()
- wakes up all the threads, waiting on this object's monitor.

(Q.4) Which are the reference types in java?

⇒ (1) There are 4 non-primitive/reference types in java:
(1) Interface
(2) Class
(3) Enum
(4) Array

Q.5 Explain narrowing and widening ?

→ ① Narrowing : Widening

- Proces of converting, value of variable of narrower type into wider type is called as Widening.

- ex:- int num1 = 10;
→ double num2 = (double) num1;
(widening)

- In case of widening, explicit typecast is optional.

② Narrowing

- Proces of converting value of variable of wider type into narrower type is called narrowing.

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

- ex:- double num1 = 10.5d.
→ int num2 = (int) num1;
narrowing

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

→

```
public static void main(String[] args) {
    if (System.out.printf("Hello CDAC") != null)
    {
    }
}
```

Q.1 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.

- ex:-

```
class StaticInitializationBlock {
    static {
        System.out.println("Hello");
        System.exit(0);
    }
}
```

O/P :- Hello

Q.2 What will happen, if we call main method in static block ?

⇒

① The static blocks always execute first before the main() method in Java because the compiler stores

them on memory at the time of class loading and before object creation.

- ex 1

```
public class StaticBlockExample {
    static {
        System.out.println("static block 1");
    }
    static {
        System.out.println("static blok 2");
    }

    public static void main(String[] args) {
        System.out.println("main method");
    }
}
```

→ static block 1
static block 2
main method
} o/p

Q.9 In System.out.println(); Explain meaning of every words ?

→

① println() is responsible for printing the argument and printing a new line.

② System refers to a final class that we can find in the java.lang package and this System class has a public and static member field called PrintStream.

③ All instances of class PrintStream have a public method called println()

④ out is an instance of the PrintStream class.

Q.10 How will you pass object to the function by reference?

→
```
public class Example{
    int a=10;
    void call (Example eg){
        eg.a = eg.a+10;
    }

    public static void main(String[] args){
        Example eg = new Example();
        System.out.println("Before
            call by reference: " + eg.a);

        eg.call(eg);
        System.out.println("After pass by
            reference: " + eg.a);
    }
```

Output:

Before call-by-reference: 10
After call-by-reference: 20

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

→

(1) Constructor chaining is the process of calling one constructor from another constructor.

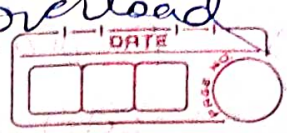
(2) In C++, this feature is called Constructor Delegation, was introduced in C++ 11.

(3)
```
class A {
    int x, y, z;
public:
    A() {
        x = 0;
        y = 0;
        z = 0;
    }
    A(int z) {
        // x = 0;
        // y = 0;
        this -> z = z;
    }
    void show() {
        cout << x << endl;
        cout << y << endl;
        cout << z << endl;
    }
};
int main() {
    A obj(2);
    obj.show();
    return 0;
}
```

o/p :- 0
0
2

This 2 lines are redundant

(12) Which are the rules to overload method in sub-class?



→

(1) In the subclass we can overload the methods inherited from the superclass such overloaded methods neither hide nor override the superclass instance methods - they are new methods, unique to the subclass.

(2)
```
public static OverloadingTest {
    public static void main(String[] args){
        ChildClass cc = new ChildClass();
        SuperClass sc = cc;
        sc.method("lol");
        cc.method("lol");
    }

static class SuperClass {
    public void method(Object o){
        System.out.println("Superclass called.");
    }

static class ChildClass extends SuperClass{
    public void method(String s){
        System.out.println("Childclass called");
    }
}
}
```

o/p:- Superclass called
Childclass called

(3) The o/p clearly states that there is no overloading in this case.

(13) Explain difference between finalize and disposeg.

	Dispose() Method	Finalize() method
①	It is defined in the interface IDisposable interface	① It is defined in java.lang.object class.
②	It is used to close or release unmanaged resources stored by an object like files or streams.	② It is used to clear up unmanaged resources owned by the current object before it is destroyed
③	Syntax:- public void Dispose(){ }	③ syntax. protected void finalize(){ }
④	Access specifier: public	④ private
⑤	Invoked by the user	⑤ Invoked by the garbage collector

(14) Explain the difference bew. final, finally and finalizeg

⇒ ① final

It is a keyword and access modifier which is used to apply restrictions on a class, variable or method.

ex:-

```
public class test2
{
    final int test = 1;
}
```

⑤ finally

It is the block in java exception handling to execute the important code whether the exception occurs or not.

ex:-
```
public class Test2 {
    public static void main(String[] args)
    {
        try {
        } catch( ) {
        }
    finally {
        S.o.println("finally block is always executed");
    }
}
```

o/p :-

finally block is always executed.

⑥ finalize

It is the method in java which is used to perform clean up processing just before object is garbage collected.

ex:-
```
public class Test3 {
    public static void main(String[] args)
    {
    }
```

protected void finalize()
{
    Sop ("called finalize");
}
}

→ o/p :- called finalize

(15) Explain the difference among checked and unchecked exception ?

→ (1) Checked Exceptions

- these are the exceptions which are checked at compile time.

- If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the throws keyword.

- Types :-

① fully checked exceptions :-
It is a checked exception where all its child classes are also checked, like IOException, and InterruptedException

② Partially checked exceptions :-
It is a checked exception where some of its child classes are unchecked, like an exception

- ex :-

```
import java.io.*;
class FG2
{
    p s v m (String[] args)
    {
        FileReader fre = new FileReader
                ("C:\\test\\a.txt");

        BufferedReader fileInput = new
                BufferedReader(fr);

        for (int i=0; i<3; i++)
            S.o.println(fileInput.readLine());

        fileInput.close();
    }
}
```

o/p error: unreported exception FileNotFoundException;
FileReader fr = new FR...
error: -11- ↑ FNFException
S.o.println(fileInput.readLine());
error: -11- ↑ IOException
fileInput.close();
↑

(2) Unchecked Exceptions :-
- these are the exceptions that are not checked at compile time. In C++ all exceptions are unchecked. In java, exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.

- ex: class GFG2

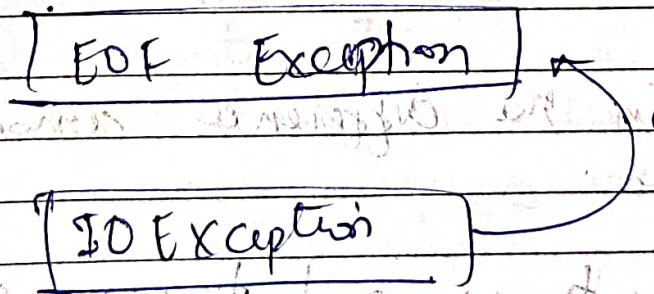
```
{ p s v m (String[] args)
    int x=0;
    int y=10;
    int z = y/x;
  }
}
```

Q.15 Exception in thread "main" java.lang.ArithmeticException : / by zero at Main.main(Main.java:5)

Q.16 Explain exception chaining?

⇒ Exception chaining :- It occurs when one exception causes another exception.

- In java, chained exception is the exception that is caused by another exception.
- It helps to debug, as it can help us track down root cause of an error.



Exception chaining

- ex:-

```
public class ExceptionHandling {
    p s v m (String[] args)
    {
        try {
            NumberFormatException ex =
        new NumberFormatException ("Exception");

        ex.initCause(new NullPointerException(
            "this is actual cause of exception"));
        throw ex;
    }

    catch (NumberFormatException ex)
    {
        S.o.println(ex);
        S.o.println(ex.getCause());
    }
    }
}
```

⇒

java.lang.NumberFormatException: Exception
java.lang.NullPointerException: This is
the actual cause of exception

Q.17 Explain the difference among throw & throws.

⇒

① The throw and throws are the concepts of exception handling in Java where the throw keyword throws the exception explicitly from a method or a block of code, whereas the throws keyword is used in the signature of the method.

ex:①

```
public class XYZ {
    p s v m (String[] args) {
        try {
            throw new ArithmeticException();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        }
    }
}
```

java.lang.ArithmeticException at XYZ.main(XYZ.java:10)

ex②

```
public class QFG {
    p s v m (String[] args) throws Exception {
        try {
            writeToFile();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void writeToFile()
    throws Exception {
        BufferedWriter bw = new BufferedWriter(new FileWriter("myFile.txt"));
        bw.write("Test");
        bw.close();
    }
}
```

Q.8 In which case finally block doesn't executed

→ ① When the System.exit() method is called in the try block before the execution of finally block, finally block will not be executed.

ex:- public class Vishal {
    void m1()
    {
        try {
            System.out.println("I am in try block");
            System.exit(0);
        }
        finally {
            System.out.println("I am in finally block");
        }
        System.out.println("statement after finally");
    }
    public static void main(String[] args)
    {
        Vishal obj = new Vishal();
        obj.m1();
    }
}

o/p:- I am in try block.

Q.9 Explain up-casting ?

→ ① Upcasting is the typecasting of a child object to a parent object.
② 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 class members using this feature.

ex:
```
class Animal{
    String name;
    void name()
    {
        s.o.pln("Animal");
    }
}

class fish extends Animal{
    String color;
    @Override
    void nature()
    {
        sopln("Aquatic Animal");
    }
}
public class GFG{
    p s v m(){ String[] args){
        Animal a = new Fish();
        a.name = "Gold Fish";

        Fish f = new Fish();
        f.name = "Whale";
        f.color = "Blue";

        s.o.pln("Obj a);
        s.o.pln("Name: " + a.name);
```

```
a.nature();
System.out.println("Object f");
System.out.println("Name : " + f.name);
System.out.println("Color : " + f.color);
f.nature();
}
}
```

Ans Object a
Name : Goldfish
Aquatic Animal

Object f
Name : Whale
Color : Blue
Aquatic Animal

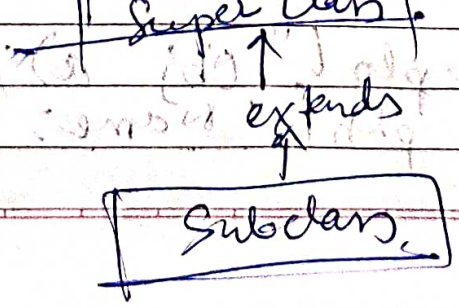
Q.20 Explain dynamic method dispatch in java?

→ (M) Dynamic method Dispatch or Run-time Polymorphism

(1) Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

(2) Upcasting

SuperClass obj = new SubClass



⑤ ex:-

```
class A {
    void m1() {
        S.o.println("Inside A's m1 method");
    }
}
class B extends A {
    void m1()  // overriding m1()
    {
        S.o.println("Inside B's m1 method");
    }
}
class C extends A {
    void m1(){
        S.o.println("Inside C's m1 method");
    }
}
class Dispatch {
    p s v m (S[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A ref;  // obtain reference of type A
        ref = a;  // ref refers to an A object
        ref.m1();  // calling A's version of m1()
        ref = b;  // now ref refers to a B obj
        ref.m1();  // calling B's version of m1()
        ref = c;
        ref.m1();
    }
}
```

O/P:- Inside A's m1 method
-11- B's -11-
-11- C's -11-

Q.2) What do you know about final method?

⇒

① When a method is declared as final, it cannot be overridden by a subclass.

② This is useful for methods that are part of a class's public API and should not be modified by subclasses.

③ Similarly, when class is declared as final, it cannot be extended by a subclass.

Q.3) Explain fragile base class problem and how to overcome it?

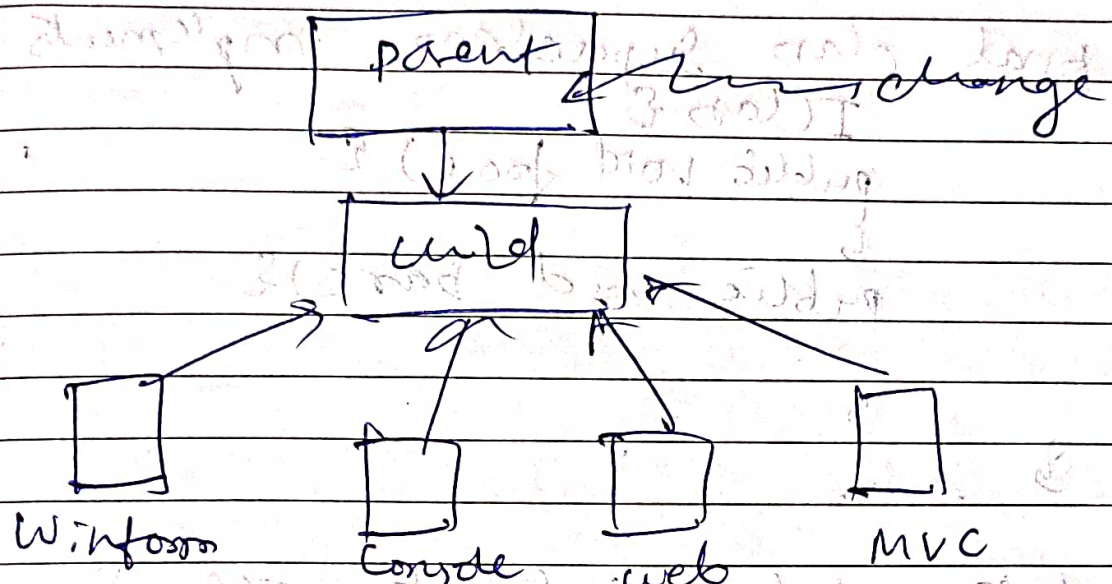
⇒

① Take a simple scenario where we have a simple parent child class relationship. Assume that this child class is used in lot of projects and installed at lots of places. Now assume after the child class has been deployed across locations, after some months, there are some changes in the parent class.

(5) These changes can have cascading and unexpected behavior in child class as well.

(6) This unexpected behavior on child classes is termed as "Fragile class problem".

(4)



(6) Code with fragile problem

```
class superclass1
    void foo(){
    }
    void bar(){
    }
}

class subclass extends superclass1
    @Override
    void bar(){
    }
}
```

any changes in his method will change

(7) Solution for fragile base problem

(1)

```
public interface IClass {
    void foo();
    void bar();
}

final class Superclass implements
    IClass {
    public void foo() {
    }
    public void bar() {
    }
}

class Subclass implements IClass {
    private Superclass helper = new
        Superclass();
    public void foo() {
        this.helper.foo();
    }
    public void bar() {
    }
}
```

Soln (1) Make all your concrete classes final by default.

(2) Try not to use inheritance ("Bar is a Foo") relationship. Instead use a helper ("Bar uses a Foo") relationship b/w your classes.

(3) use interfaces rather than abstract classes to ensure that classes using these helper have a uniform interface.

(4) Remember that almost * every extends can be replaced by implements.

(5) Why java does not support multiple implementation inheritance!

⇒

2 Reasons:-

(1) (1) In java every class is a child of object class; when it inherits from more than one superclass. sub class gets the ambiguity to acquire the property of Object class.

(2) ex:-
Consider a case where class B extends class A and class C and both class A and C have the same method display().

(3) Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java.

(4)
```
class Class1 {
    public void display() {
    }
}
class Class2 {
    public void display() {
```

public class Test extends class1, class2 {

p s v m (S[] a) {

Test obj = new Test();

obj.display();

}

}

which display() to call

Q.19 Exception in thread "main" java.lang. error : Unresolved compilation problem.

Q.20 In java every class has a constructor, if we write it explicitly or not at all the first statement is calling super() to invoke the super class constructor. If the class has more than one super class, it gets confused.

Q.21 Explain marker interface? Legt the name of some marker interfaces?

⇒

① Marker interface is an empty (no fields or methods) interface.

② Examples of marker interface are Serializable, Cloneable and Remote interface. All these interfaces are empty interfaces

③ ex:-

public interface Serialization {

// nothing here

}

Q.25 Explain the significance of marker interface ?

⇒ ① We can use a marker interface in java to indicate that a class can be serialized, to indicate a class can be clonable and to indicate a class can be remotely accessed.

② A marker interface is an interface that doesn't have any methods or constants inside it. It provides run-time type information about objects, so that the compiler and JVM have additional information about the object.

③ A marker interface is also called a tagging interface.