Q1) Explain OOPS?

As the name suggests, Object-Oriented Programming or OOPs refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks assigned by you. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

The popular object-oriented languages are Java, C#, PHP, Python, C++, etc.

Q2) Explain an abstraction? Real life example.

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or non-essential units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
Data Abstraction may also be defined as the process of identifying only the required characteristics of an object, ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the object.
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the car speed or applying brakes will stop the car, but he does not know how on pressing the accelerator, the speed is actually increasing. He does not know about the inner mechanism of the car or the implementation of the accelerators, brakes etc. in the car. This is what abstraction is.
In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

3) Explain encapsulation? Real life example.

It is defined as the wrapping up of data under a single unit. It is the mechanism that binds together the code and the data it manipulates. Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically, in encapsulation, the variables or the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.
In encapsulation, the data in a class is hidden from other classes, which is similar to what data-hiding does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.
Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.

Real-life example-

Example 1: School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc. Realtime

Example 2: When you log into your email accounts such as Gmail, Yahoo Mail, or Rediff mail, there is a lot of internal processes taking place in the backend and you have no control over it.

Q4) Explain the relationship among abstraction and encapsulation?

Abstraction	Encapsulation
1. Abstraction is the process or method of gaining the information.	1.While encapsulation is the process or method to contain the information.
2. In abstraction, problems are solved at the design or interface level.	2. While in encapsulation, problems are solved at the implementation level.
3. Abstraction is the method of hiding the unwanted information.	3. Whereas encapsulation is a method to hide the data in a single entity or unit

	along with a method to protect information from outside.
4. We can implement abstraction using abstract class and interfaces.	4. Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
5. In abstraction, implementation complexities are hidden using abstract classes and interfaces.	5. While in encapsulation, the data is hidden using methods of getters and setters.
6. The objects that help to perform abstraction are encapsulated.	6. Whereas the objects that result in encapsulation need not be abstracted.
7. Abstraction provides access to specific part of data.	7. Encapsulation hides data and the user can not access same directly (data hiding).
8. Abstraction focus is on “what” should be done.	8. Encapsulation focus is on “How” it should be done.

Q5) Explain polymorphism?

Polymorphism is the ability of any data to be processed in more than one form OR Polymorphism in Java is a concept by which we can perform a *single action in different ways*. The word itself indicates the meaning
as poly means many and morphism means types. Polymorphism is one of the most important concept of object oriented programming language. The most common use of polymorphism in object-oriented programming occurs when a parent class reference is used to refer to a child class object. Here we will see how to represent any function in many types and many forms.

Real life example of polymorphism, a person at the same time can have different roles to play in life. Like a woman at the same time is a mother, a wife, an employee and a daughter. So the same person has to have many features but has to implement each as per the situation and the condition. Polymorphism is considered as one of the important features of Object Oriented Programming.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism.

Q6) Explain Inheritance?

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features (fields and methods) of another class.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a *parent-child* relationship.

Uses of inheritance in java-

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability

Q7) How composition is better than inheritance?

Favouring Composition over Inheritance is a principle in object-oriented programming (OOP). Classes should achieve polymorphic behaviour and code reuse by their composition rather than inheritance from a base or parent class. To get the higher design flexibility, the design principle says that composition should be favoured over inheritance.
Inheritance should only be used when subclass 'is a' superclass. Don't use inheritance to get code reuse. If there is no 'is a' relationship, then use composition for code reuse.

Reasons to Favour Composition over Inheritance in Java and OOP:

1. The fact that Java does not support multiple inheritances is one reason for favouring composition over inheritance in Java. Since you can only extend one class in Java, but if you need multiple features, such as reading and writing character data into a file, you need Reader and Writer functionality. It makes your job simple to have them as private members, and this is called Composition.

2. Composition offers better test-ability of a class than Inheritance. If one class consists of another class, you can easily construct a Mock Object representing a composed class for the sake of testing. This privilege is not given by inheritance.
3. Although both Composition and Inheritance allow you to reuse code, one of Inheritance's disadvantages is that it breaks encapsulation. If the subclass depends on the action of the superclass for its function, it suddenly becomes fragile. When super-class behaviour changes, sub-class functionality can be broken without any modification on its part.
4. In the timeless classic Design Patterns, several object-oriented design patterns listed by Gang of Four: Elements of Reusable Object-Oriented Software, favour Composition over Inheritance. Strategy design pattern, where composition and delegation are used to modify the behaviour of Context without touching context code, is a classical example of this. Instead of getting it by inheritance, because Context uses composition to carry strategy, it is simple to have a new implementation of strategy at run-time.
5. Another reason why composition is preferred over inheritance is flexibility. If you use Composition, you are flexible enough to replace the better and updated version of the Composed class implementation. One example is the use of the comparator class, which provides features for comparison.

Q 8) Which OOPS concept is used as a reuse mechanism?

Inheritance is the object-oriented programming concept where an object is based on another object. Inheritance is the mechanism of code reuse. The object that is getting inherited is called the superclass and the object that inherits the superclass is called a subclass.

Q9) Which OOPS concept exposes only the necessary information to the calling functions?

Data hiding is a technique used in object-oriented programming.

It means hiding the internal details. (Abstraction)

Data hiding makes sure that the internal details are restricted to class members.

Data integrity is maintained in data hiding.

ata hiding reduces the complexities and increases the robustness.

Another main advantage is that it reduces the interdependencies between two software.

Abstraction is a process of hiding implementation details and exposes only the functionality to the user. In abstraction, we deal with ideas and not events. This means the user will only know “what it does” rather than “how it does”.

There are two ways to achieve abstraction in Java

1. Abstract class (0 to 100%)
2. Interface (100%)

Real-Life Example: *A driver will focus on the car functionality (Start/Stop -> Accelerate/ Break), he/she does not bother about how the Accelerate/ brake mechanism works internally. And this is how the abstraction works.*

Q10) Explain a class? Create a class.

- Class is a set of object which shares common characteristics/ behaviour and common properties/ attributes.
- Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.
- Class does not occupy memory.
- Class is a group of variables of different data types and group of methods.
- A class in java can contain:
  - data member
  - method
  - constructor
  - nested class and
  - interface

Syntax to declare a class:

```
access_modifier class<class_name>
{
   data member;
method;
constructor;
nested class;
```

```
        interface;
    }
```

Eg:

- Animal
- Student
- Bird
- Vehicle
- Company

Q11) Write in brief abstraction and encapsulation.

Encapsulation: Encapsulation in Java is the process by which data (variables) and the code that acts upon them (methods) are integrated as a single unit. By encapsulating a class's variables, other classes cannot access them, and only the methods of the class can access them

Abstraction: Abstraction is a technique of hiding unnecessary details from the user. The user is only given access to the details that are relevant. Vehicle operations or ATM operations are classic examples of abstractions in the real world.

Q12) Explain difference among class and object?

Class	Object
A class is a blueprint from which you can create the instance, i.e., objects.	An object is the instance of the class, which helps programmers to use variables and methods from inside the class.
A class is used to bind data as well as methods together as a single unit.	Object acts like a variable of the class.

Classes have logical existence.	Objects have a physical existence.
A class doesn't take any memory spaces when a programmer creates one.	An object takes memory when a programmer creates one.
The class has to be declared only once.	Objects can be declared several times depending on the requirement.

Q13) Define access modifiers?

Access modifiers are keywords used to specify the accessibility of a class (or type) and its members. These modifiers can be used from code inside or outside the current application.

Access modifiers in .NET are used to control the accessibility of each of the members of a type from different possible areas of code. This can be handled from within the current assembly or outside it. An assembly represents a logical unit of functionality and consists of types and resources located in one or more files.

The purpose of using access modifiers is to implement encapsulation, which separates the interface of a type from its implementation. With this, the following benefits can be derived:

- Prevention of access to the internal data set by users to invalid state.
- Provision for changes to internal implementation of the types without affecting the components using it.
- Reduction in complexity of the system by reducing the interdependencies between software components.

Q14) Explain an object? Create an object of above class

A Java object is a member (also called an instance) of a Java class. Each object has an identity, a behaviour and a state. The state of an object is stored in fields (variables), while methods (functions) display the object's behaviour. Objects are created at runtime from templates, which are also known as classes.

Q15) Give real life examples of object

Objects have states and behaviours. Example: A dog has states - colour, name, breed as well as behaviours – wagging the tail, barking, eating.

- 1. Class - Human being
- Man - an object of Human being
- Woman - an object of Human being

- 2. Class - colour
- Red - an object of colour
- Blue - an object of colour

- 3. Class - pets
- Dog - an object of pets
- Cat - an object of pets

Q16) Explain a Constructor.

A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

A constructor is like an instance method that usually has the same name as the class, and can be used to set the values of the members of an object, either to default or to user-defined values. However, although it resembles it, a constructor is not a proper method since it doesn't have a return type. Instead of performing a task by executing code, the constructor initializes the object, and it cannot be static, final, abstract, and synchronized.

Q17) Define the various types of constructors?

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. <class_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
1. //Java Program to create and call a default constructor
2. class Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. public static void main(String args[]){
7. //calling a default constructor
8. Bike1 b=new Bike1();
9. }
10. }
```

Output:

```
Bike is created
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. //Java Program to demonstrate the use of the parameterized constructor.
2. class Student4{
3.     int id;
4.     String name;
5.     //creating a parameterized constructor
6.     Student4(int i,String n){
7.     id = i;
8.     name = n;
9.     }
10.    //method to display the values
11.    void display(){System.out.println(id+" "+name);}
12.
13.    public static void main(String args[]){
14.    //creating objects and passing values
15.    Student4 s1 = new Student4(111,"Karan");
16.    Student4 s2 = new Student4(222,"Aryan");
17.    //calling method to display the values of object
18.    s1.display();
19.    s2.display();
```

```
20.    }
21. }
```

Output:

111 Karan

222 Aryan

Q18) Whether static method can use non static members?

A static method can only access static data members and static methods of another class or same class but cannot access non-static met hods and variables. Also, a static method can rewrite the values of any static data member.

Q19) Explain Destructor?

A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete. A destructor has the same name as the class, preceded by a tilde (~). For example, the destructor for class String is declared: ~String()

Q20) Explain an Inline function?

An inline function is one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a separate

set of instructions in memory. This eliminates call-linkage overhead and can expose significant optimization opportunities.

Q21) What is a virtual function?

A virtual function or virtual method in an OOP language is a function or method used to override the behaviour of the function in an inherited
class with the same signature to achieve the polymorphism.

When the programmers switch the technology from C++ to Java, they think about where the virtual function is in Java. In C++, the virtual function is
defined using the virtual keyword, but in Java, it is achieved using different techniques. See Virtual function in C++.

Java is an object-oriented programming language; it supports OOPs features such as polymorphism, abstraction, inheritance, etc.
These concepts are based on objects, classes, and member functions.

By default, all the instance methods in Java are considered as the Virtual function except final, static, and private methods as these methods
can be used to achieve polymorphism.

How to use Virtual function in Java
The virtual keyword is not used in Java to define the virtual function; instead, the virtual functions and methods are achieved using the following techniques:

We can override the virtual function with the inheriting class function using the same function name. Generally, the virtual function is defined in the parent class and overrides it in the inherited class.
The virtual function is supposed to be defined in the derived class. We can call it by referring to the derived class's object using the reference or pointer of the base class.
A virtual function should have the same name and parameters in the base and derived class.
For the virtual function, an IS-A relationship is necessary, which is used to define the class hierarchy in inheritance.
The Virtual function cannot be private, as the private functions cannot be overridden.
A virtual function or method also cannot be final, as the final methods also cannot be overridden.
Static functions are also cannot be overridden; so, a virtual function should not be static.

By default, Every non-static method in Java is a virtual function.
The virtual functions can be used to achieve oops concepts like runtime polymorphism.

Q 22) Explain a friend function?

friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class.
Even though the prototypes for friend functions appear in the class definition, friends are not member functions. java does not provide the friend keyword it is used in c++.

Q 23) Explain function overloading?

Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overloading is also known as Compile-time Polymorphism, Static Polymorphism, or Early binding in Java. In Method overloading compared to parent argument, child argument will get the highest priority.

Different Ways of Method Overloading in Java
Changing the Number of Parameters.
Changing Data Types of the Arguments.
Changing the Order of the Parameters of Methods

Q 24) Explain a base class, sub class, super class?

Ans=

base class:

a base class is an existing class from which the other classes are determined and properties are inherited. It is also known as a superclass or parent class.

Super class:

A superclass is the class from which many subclasses can be created. The subclasses inherit the characteristics of a superclass.
The superclass is also known as the parent class or base class.

Subclasses :

A subclass is a class derived from the superclass. It inherits the properties of the superclass and also contains attributes of its own.

Q 25) write in brief linking of base class ,Subclass and base object and super object ?

Object relation of Superclass (parent) to Subclass (child) exists while child to parent object relation never exists. This means that reference of parent class could hold the child object while child reference could not hold the parent object.

In case of overriding of non static method the runtime object would evaluate that which method would be executed of subclass or of super class. While execution of static method depends on the type of reference that object holds.
Other basic rule of inheritance is related to static and non static method overriding that static method in java could not be overridden while non static method can be. However subclass can define static method of same static method signature as superclass have hut that would not be consider as overriding while known as hiding of static method of superclass.

Q 26) Explain an abstract class?

Ans = Abstract class in Java

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods.
It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember
An abstract class must be declared with an abstract keyword.
It can have abstract and non-abstract methods.
It cannot be instantiated.
It can have constructors and static methods also.
It can have final methods which will force the subclass not to change the body of the method.

Q 27) What is Operator Overloading in C++?

Ans = We can easily apply operators on primitive data types (like integers, characters, floating point numbers) but not on user-defined data types like Complex numbers, fractions, etc.

We can also make operators work for user-defined data types by explicitly defining the operation performed by a given operator on a given data type,
known as Operator overloading in C++.

note : there is no concept like operator overloading in java.

Q 28) Define different types of arguments? (Call by value/Call by reference) .

Functions can be invoked in two ways: Call by Value or Call by Reference. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called actual parameters whereas the parameters received by function are called formal parameters.

Call By Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations.

So any changes made inside functions are not reflected in actual parameters of the caller.

Call by Reference: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

Call By Value =

While calling a function, we pass values of variables to it. Such functions are known as "Call By Values".

In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function.

With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.

Call By refrance =
While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References.
In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function.
With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.

Q 29) Explain the super keyword?

Ans = The super keyword refers to superclass (parent) objects.

It is used to call superclass methods, and to access the superclass constructor.

The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

Q 30) Explain method overriding?

Ans = Declaring a method in the subclass which already exists there in the parent class is known as method overriding.
When a class is inheriting a method from a superclass of its own, then there is an option of overriding the method provided it is not declared as final.
The advantage of using overriding is the ability to classify a behaviour that's specific to the child class, and the child class can implement a parent class method based on its necessity.

There are certain rules that a programmer should follow to implement overriding. These are:

In Java, a method can only be written in the child class and not in same class.
Argument list should be the same as that of the overridden method of that class.
Instance methods can also be overridden if they are inherited by the child class.
A constructor cannot be overridden.
Final - declared methods cannot be overridden.
Any method that is static cannot be used to override.
The return type must have to be the same, or a subtype of the return type declared in the original overridden method in the parent class.
If a method cannot be inherited, then it cannot be overridden.

A child class within the same package as the instance's parent class can override any parent class method that is not declared private or final.
A child class in a different package can only override the non-final methods declared as public or protected.

Q31) Difference among overloading and overriding?

Overriding and *overloading* are the core concepts in Java programming. They are the ways to implement polymorphism in our Java programs. Polymorphism is one of the OOPS Concepts.

When the method signature (name and parameters) are the same in the superclass and the child class, it's called *overriding*. When two or more methods in the same class have the same name but different parameters, it's called *overloading*.

Overriding	Overloading
Implements "runtime polymorphism"	Implements "compile time polymorphism"
The method call is determined at runtime based on the object type	The method call is determined at compile time
Occurs between superclass and subclass	Occurs between the methods in the same class
Have the same signature (name and method arguments)	Have the same name, but the parameters are different
On error, the effect will be visible at runtime	On error, it can be caught at compile time

Q32) Whether static method can use non-static members?

A static method can only access static data members and static methods of another class or same class but cannot access non-static methods and variables. Also, a static method can rewrite the values of any static data member.

A non-static method can access static data members and static methods as well as non-static members and methods of another class or same class, also can change the values of any static data member.

Q33) Explain a base class, sub class super class?

In an object-oriented programming language, a base class is an existing class from which the other classes are determined and properties are inherited. It is also known as a superclass or parent class. In general, the class which acquires the base class can hold all its members and some further data as well.

A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

Q34) Write in brief linking of base class, sub class and base object, sub object?

There are two approaches to refer a subclass object. Both have some advantages/disadvantages over the other. The declaration affect is seen on methods that are visible at compile-time.

First approach (Referencing using Superclass reference):

A reference variable of a superclass can be used to a refer any subclass object derived from that superclass. If the methods are present in Super Class, but overridden by Sub Class, it will be the overridden method that will be executed.

Second approach (Referencing using subclass reference):

A subclass reference can be used to refer its object.

Q35) Explain an interface?

In interface in Java is a blueprint of a class. It has static constants and abstract methods.

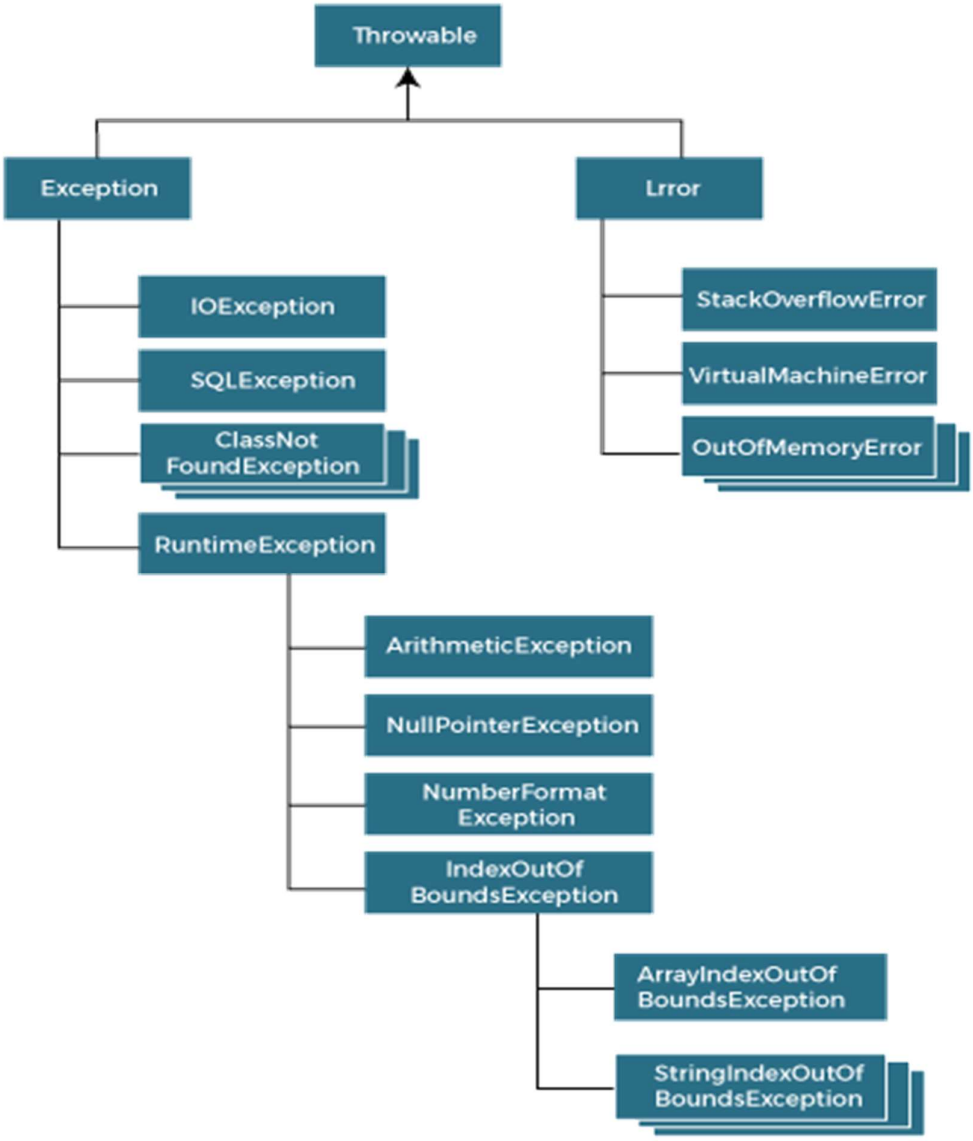
The interface in Java is *a mechanism to achieve* abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship.

Q36) Explain exception handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.



Q37) Explain the difference among structure and a class?

Structures and classes differ in the following particulars:

Structures are *value types*; classes are *reference types*. A variable of a structure type contains the structure's data, rather than containing a reference to the data as a class type does.

Structures use stack allocation; classes use heap allocation.

All structure elements are Public by default; class variables and constants are Private by default, while other class members are Public by default. This behaviour for class members provides compatibility with the Visual Basic 6.0 system of defaults.

A structure must have at least one nonshared variable or nonshared, non custom event element; a class can be completely empty.

Structure elements cannot be declared as Protected; class members can.

A structure procedure can handle events only if it is a Shared Sub procedure, and only by means of the AddHandler Statement; any class procedure can handle events, using either the Handles keyword or the AddHandler statement. For more information, see Events.

Structure variable declarations cannot specify initializers or initial sizes for arrays; class variable declarations can.

Q38) Explain the default access modifier in a class?

When no access modifier is specified for a class, method, or data member – it is said to be having the default access modifier by default.

The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.

Q39) Explain a pure virtual function?

A pure virtual function or pure virtual method is a virtual function that is required to be implemented by a derived class if the derived class is not abstract. Classes containing pure virtual methods are termed "abstract" and they cannot be instantiated directly. A subclass of an abstract class can only be instantiated directly if all inherited pure virtual methods have been implemented by that class or a parent class. Pure virtual methods typically have a declaration (signature) and no definition (implementation).

Q40) Explain dynamic or runtime polymorphism?

Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as runtime polymorphism or dynamic method dispatch. We can achieve dynamic polymorphism by using the method overriding.

In this process, an overridden method is called through a reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Properties of Dynamic Polymorphism

- It decides which method is to execute at runtime.
- It can be achieved through dynamic binding.
- It happens between different classes.
- It is required where a subclass object is assigned to a super-class object for dynamic polymorphism.
- Inheritance involved in dynamic polymorphism.

Q41) .Do we require a parameter for constructors?

The default constructor does not accept any parameter. It is used if we want to initialize the instance variables with certain values. Every Java class has a default constructor, invisibly. So, we need not to define it, separately.

Q42). Explain static and dynamic binding?

The static binding uses Type information for binding while Dynamic binding uses Objects to resolve to bind. Overloaded methods are resolved (deciding which method to be called when there are multiple methods with the same name) using static binding while overridden methods use dynamic binding, i.e, at run time.

Q43). How many instances can be created for an abstract class?

The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure.

Q44). Explain the default access specifiers in a class definition?

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access control modifier is available to any other class in the same package.

Q45). Which OOPS concept is used as reuse mechanism?

Inheritance is the feature that provides a reuse mechanism.
This mechanism provides reusability to the user. While abstraction, encapsulation, and dynamic binding have different functionalities in the OOP paradigm.

Q46). Define the Benefits of Object Oriented Programming?

Object-oriented programming is ultimately about taking a huge problem and breaking it down to solvable chunks. For each mini-problem, you write a class that does what you require. And then — best of all — you can reuse those classes, which makes it even quicker to solve the next problem.

Q47). Explain method overloading?

Method overloading allows a class to define multiple methods with the same name, but different signatures. That is, it allows you to define different methods that have the same name, but that respond to correspondingly different messages sent to an instance of the class.

Q48). Explain the difference among early binding and late binding?

Early binding reduces the number and severity of run-time errors because it allows the compiler to report errors when a program is compiled. Late binding can only be used to access type members that are declared as Public . Accessing members declared as Friend or Protected Friend results in a run-time error.

Q49). Explain early binding? Give examples?

This is compile time polymorphism. Here it directly associates an address to the function call. For function overloading it is an example of early binding.

Q50).Explain loose coupling and tight coupling?

Tight Coupling means one class is dependent on another class. Loose Coupling means one class is dependent on interface rather than class. In tight coupling, there are hard-coded dependency declared in methods. In loose coupling, we must pass dependency externally at runtime instead of hard-coded.

Q51) Give an example among tight coupling and loose coupling.

Tight coupling: In general, Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.
Example: If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI (Remote Method Invocation).

Loose coupling: In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to overcome from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection it's possible to achieve loose coupling.
Example: If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can't do that, then you have tight coupling. The examples of Loose coupling are Interface, JMS.

Q52) Write in brief abstract class.

Abstract class concept is one of the basic concepts of Object-Oriented Programming. It gives us the possibility of modelling concepts from the real world and facilitates the use of one of the OOP principles: code reuse. An abstract class in Object-Oriented Programming (OOP) is a class that cannot be instantiated.

An abstract class is a template definition of methods and variables of class (category of objects) that contains one or more abstracted methods. Abstract classes are used in all object-oriented programming (OOP) languages, including java (see java abstract class), C++, C#. Objects or classes may be abstracted, which means that they are summarized into characteristics that are relevant to the current program's operation.

Q53) Define the Benefits of oops over pop?

The benefits of oops over procedural programming:

- The benefits of OOPs are quick and easy to execute.
- The benefits of OOPs provide a clear structure for the programs.
- Benefits of OOPs help to keep the C code Do not Repeat Yourself, (DRY) and make the code easier to maintain, modify and rectify.
- The benefits of OOPs make it possible to produce full applicable operations with lower code and shorter development time

Q54) Explain Generalization and Specialization?

Generalization and specialization are the Enhanced Entity Relationship diagram (EER-diagram)

1. **Generalization:** It works on the principle of bottom up approach. In Generalization lower level functions are combined to form higher level function which is called as entities.
2. **Specialization:**
   We can say that Specialization is opposite of Generalization. In Specialization things are broken down into smaller things to simplify it further. We can also say that in Specialization a particular entity gets divided into sub entities and it's done on the basis of it's characteristics. Also in Specialization Inheritance takes place.

Q55) Write in brief Association, Aggregation and Composition?

Association, Aggregation, and Composition are terms that represent relationships among objects. They are very basic stuff of Object Oriented Programming.

Association

Association is a relationship among the objects. Association is "*a*" relationship among objects. In Association, the relationship among the objects determines what an object instance can cause another to perform an action on its behalf. We can also say that an association defines the multiplicity among the objects. We can define a one-to-one, one-to-many, many-to-one and many-to-many relationship among objects. Association

is a more general term to define a relationship among objects. Association means that an object "uses" another object.

For example Managers and Employees, multiple employees may be associated with a single manager and a single employee may be associated with multiple managers.

Aggregation

Aggregation is a special type of Association. Aggregation is "*the*" relationship among objects. We can say it is a direct association among the objects. In Aggregation, the direction specifies which object contains the other object. There are mutual dependencies among objects.

For example, departments and employees, a department has many employees but a single employee is not associated with multiple departments.

Composition

Composition is special type of Aggregation. It is a strong type of Aggregation. In this type of aggregation the child object does not have their own life cycle. The child object's life depends on the parent's life cycle. Only the parent object has an independent life cycle. If we delete the parent object then the child object(s) will also be deleted. We can define the Composition as a "Part of" relationship.

For example, the company and company location, a single company has multiple locations. If we delete the company then all the company locations are automatically deleted. The company location does not have their independent life cycle, it depends on the company object's life (parent object)

Q56) Write in brief Object Composition vs. Inheritance.

Inheritance and Composition both are design techniques. The Composition is a way to design or implement the "has-a" relationship whereas, the Inheritance implements the "is-a" relationship. The "has-a" relationship is used to ensure the code reusability in our program. In Composition, we use an instance variable that refers to another object.

Although both inheritance and composition provide the code reusability, the difference between both terms is that in composition, we do not extend the class. The composition relationship of two objects is possible when one object contains another object, and that object is fully dependent on it. The composition represents the part of relationship. For instance, A house ***has a*** living room (living room is a part of house), A person ***has a*** heart (heart is a part of the human body), and many more.

Composition has various benefits like it allows code reusability, it is helpful in achieving the multiple inheritance, it provides better test-ability of a class. It also allows us to easily replace the composed class implementation with a better and improved version. Using composition concept, we can dynamically change the behaviour of our program by changing the member objects at run time.

Q57) Explain cohesion?

Cohesion in Java is the principle of Object-Oriented programming. Cohesion is closely related to ensuring that the purpose for which a class is getting created in Java is well-focused and single. In other words, the more closely related stuff is grouped in a class, the higher will be the cohesiveness.

Q58) Explain “black-box-reuse” and “white-box-reuse”?

Black box reuse is using a class/function/code unmodified in a different project
White box reuse is taking a class/function/code from one project and modifying it to suit the needs of another project.
The pros to black-box reuse are that once the code has been written, debugged, and tested, you can reuse it countless times in different circumstances. The downside is that truly black-box-reusable code is rare and can take time and effort to format the API and calling code and make it consistent with the black box approach (no context leaking).
The pros to white-box reuse are that you can indeed use your code more than once without having to first extricate it from the original project. You simply copy and modify and you're on your way. This type of reuse is much more common, but it also has a few downsides. Mostly, if you discover a bug in one implementation, you need to check to make sure that it's fixed in all the other implementations. This can be difficult if they diverge widely, as often happens.

Q59) Explain “this”

‘this’ is a reference variable that refers to the current object. 1. Using ‘this’ keyword to refer current class instance variables 2. Using this () to invoke current class constructor 3. Using ‘this’ keyword to return the current class instance

Q60) Write in brief static member and member functions.

The static member functions are special functions used to access the static data members or other static member functions. A member function is defined using the static keyword. A static member function shares the single copy of the member function to any number of the class objects.

A static data member(s) is the data member(s) that gets memory only once for the whole class, no matter how many object(s) of a class is created. All objects of its class share this common copy. The static data member is declared within the class by prefixing the data member declaration in the class body with a keyword static and Static Data Members always have Default values as \0 for int and Null for Strings and needs to be defined explicitly outside the class using the scope resolution operator, which identifies the class to which it belongs. They will Never Store any Garbage values. Always remember that Static Data Members are always used in the Static Member Functions. If a Member Functions wants to use Static Data, then we must have to declare that Member Function as Static. And the Static Data Members are always Assigned Some values from the outside from the Class.

Q61). How will you relate unrelated classes or how will you achieve polymorphism without using the base class?

Polymorphism: Polymorphism is a characteristic of being able to assign a different behaviour or value in a sub class which was something to declare in a parent class. Run Time Polymorphism is done by using inheritance & interface. Run Time Polymorphism is a process in which a call to an overridden method is resolve at run time rather than at compile time. In this process an overridden method is called through the reference variable of a superclass So yes, you can achieve polymorphism without inheritance through interface implementation. Two elements are polymorphic with respect to a set of behaviours if they realize the same interfaces. An object of a class can be casted to the type of each interface it implements.

Q62). Explain the Diamond problem?

In Java, the diamond problem is related to multiple inheritance. Sometimes it is also known as the deadly diamond problem or deadly diamond of death. A diamond problem is an ambiguity that arises only in multiple inheritances that occurs when a child class has the values inherited from the two parents. Wherein these parent classes are inherited from a common grandparent class. For example, consider an example in which we have a Child class inherited from the classes of Mother and Father. These classes inherit a third-class named “person”: Child > Mother> Person > Father > Person So, according to the given scenario, the child class inherits the “person” class two times in the program. Once, it is from the mother, and again, the second time is from the father. This creates confusion for the compiler to execute which constructor first. This situation causes a diamond-shaped inheritance graph. Hence, it is known as “The Diamond Problem”.

Q63). Explain the solution for diamond problem?

The solution to the diamond problem is default methods and interfaces. We can achieve multiple inheritance by using these two things. The default method is similar to the abstract method. The only difference is that it is defined inside the interfaces with the default implementation. We need not to override these methods. Because they are already implementing these interfaces.

Q64). Explain the need of abstract class?

Abstract class in Java is similar to interface except that it can contain default method implementation. An abstract class can have an abstract method without body and it can have methods with implementation also. An abstract class is mostly used to provide a base for subclasses to extend and implement the abstract methods and override or use the implemented methods in abstract class. Template The abstract class in Java enables the best way to execute the process of data abstraction by providing the developers with the option of hiding the code implementation. It also presents the end-user with a template that explains the methods involved. Loose Coupling Data abstraction in Java enables loose coupling, by reducing the dependencies at an exponential level. Code Reusability Using an abstract class in the code saves time. We can call the abstract method wherever the method is necessary. Abstract class avoids the process of writing the same code again. Abstraction Data abstraction in Java helps the developers hide the code complications from the end-user by reducing the project's complete characteristics to only the necessary components. Dynamic Resolution Using the support of dynamic method resolution, developers can solve multiple problems with the help of one abstract method.

Q65). Why can’t we instantiate abstract class?

An abstract class is a class which doesn’t have an implementation for one or more methods. It means that the jvm doesn’t have any direction of what to do in case if someone calls the method which is abstract. So if you are able to create an object for the abstract class and call any abstract method of it the jvm will not be able to decide what to do and hence it may be crashed. So to avoid this situation we are restricted to instantiate an abstract class. If you need an object for that abstract class create a concrete subclass and create an object for it and use it.

Q66). Can abstract class have constructors?

Yes it can have a constructor and it is defined and behaves just like any other class's constructor. Except that abstract classes can't be directly instantiated, only extended, so the use is therefore always from a subclass's constructor. Each abstract class must have a concrete subclass which will implement the abstract methods of that abstract class. When we create an object of any subclass all the constructors in the corresponding inheritance tree are invoked in the top to bottom approach. When we create an object of a class which is concrete and subclass of the abstract class, the constructor of the abstract class is automatically invoked. Hence we can have a constructor in abstract classes.

Q67). How many instances can be created for an abstract class?

The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure. For an abstract class in the OOP example, we cannot instantiate it.

Q68). Which keyword can be used for overloading?

If both parent & child classes have the same method, then the child class would override the method available in its parent class. By using the super keyword we can take advantage of both classes (child and parent) to achieve this.

69). Explain the default access specifiers in a class definition?

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Q70). Define all the operators that cannot be overloaded?

Unlike C++, Java doesn't support operator overloading. Java doesn't provide freedom to programmers, to overload the standard arithmetic operators e.g. +, -, * and / etc. However with one exception, + and += are overloaded for String object.

Q 71). Explain the difference among structure and a class?

Class

It is defined using 'class' keyword.
When data is defined in a class, it is stored in memory as a reference.
It gets memory allocated only when an object of that class is created.
The reference type (before creating an object) is allocated on heap memory.
They can have constructors and destructors.
It can use inheritance to inherit properties from base class.
The 'protected' access modifier can be used with the data members defined inside the class.

Structure

The 'struct' keyword is used to define a structure.
Every member in the structure is provided with a unique memory location.
When the value of one data member is changed, it doesn't affect other data members in structure.
It helps initialize multiple members at once.
Total size of the structure is equivalent to the sum of the size of every data member.
It is used to store various data types.

It takes memory for every member which is present within the structure.
A member can be retrieved at a time.
It supports flexible arrays.
Its instance can be created without a keyword.
It doesn't support protected access modifier.
It doesn't support inheritance.
It doesn't have a constructor or destructor.
The values allocated to structures are stored in stack memory.

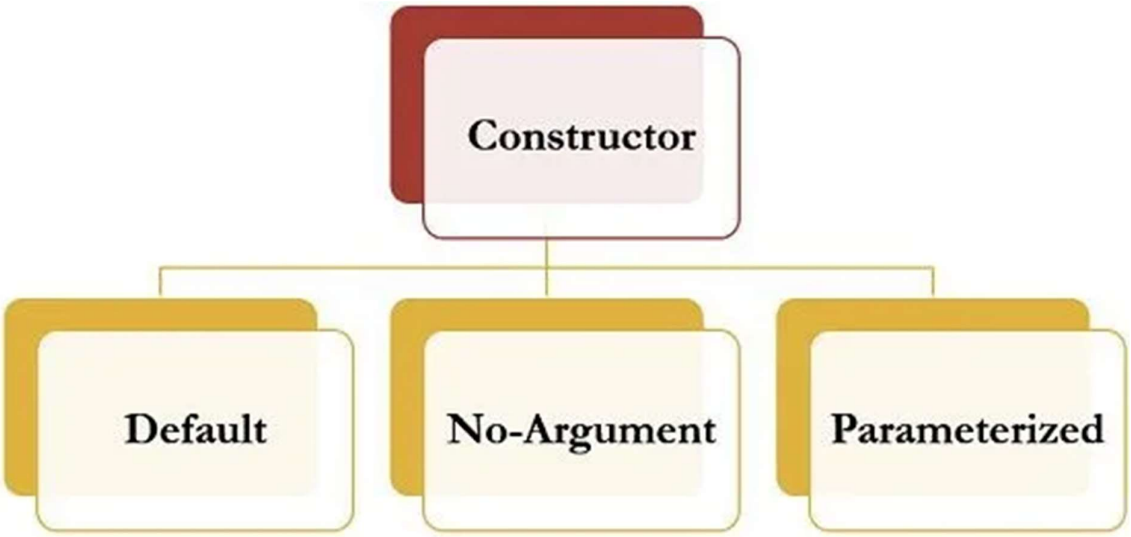
Q 72). Explain the default access modifier in a class?

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public

Q 73). Can you list out the different types of constructors?

Types of Constructors in Java



In general, there are three types of constructors:

1. Default Constructor
2. No-Argument Constructor
3. Parameterized Constructor

Q 74). Explain a ternary operator?

In Java, the ternary operator is a type of Java conditional operator. In this section, we will discuss the ternary operator in Java with proper examples.

The meaning of ternary is composed of three parts.

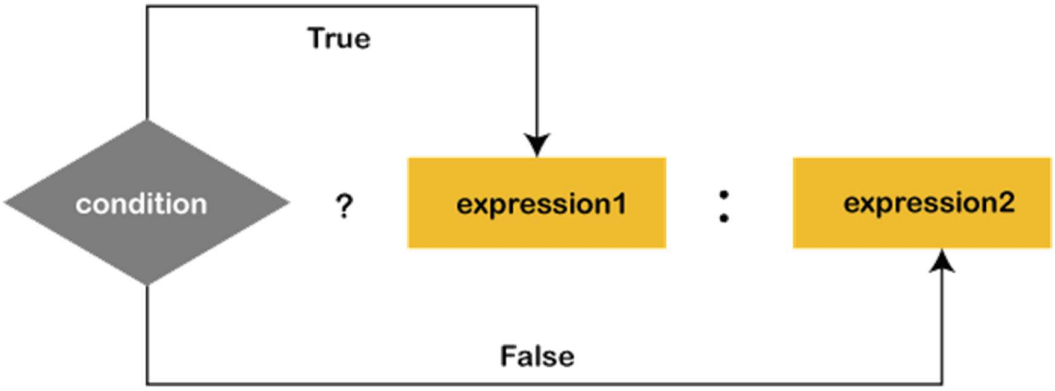
The ternary operator (? :) consists of three operands. It is used to evaluate Boolean expressions. The operator decides which value will be assigned to the variable. It is the only conditional operator that accepts three operands. It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.

Note: Every code using an if-else statement cannot be replaced with a ternary operator.

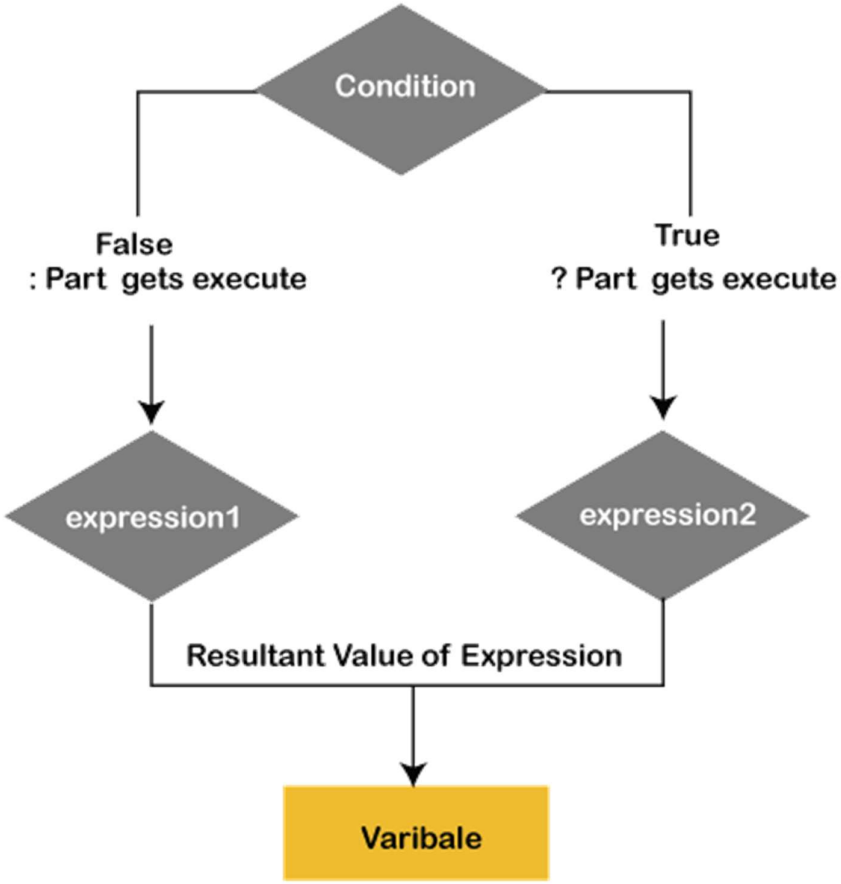
Syntax:

```
variable = (condition) ? expression1 : expression2
```

The above statement states that if the condition returns true, expression1 gets executed, else the expression2 gets executed and the final result stored in a variable.



Let's understand the ternary operator through the flowchart.



Q75). Do We Require Parameter for Constructors?

It is not necessary to have a constructor block in your class definition. If you don't explicitly write a constructor, the compiler automatically inserts one for you.

Q76). Explain Sealed Modifiers?

We can say that the class that cannot be inherited but can be instantiated is known as the sealed class. It allows classes and interfaces to have more control over their permitted subtypes. It is useful both for general domain modelling and for building a more secure platform for libraries.

Q77). Explain the Difference Between New And Override.

Override: Override implements Polymorphism. Which says I want to use the same method but in various classes with the same signature of methods.

New: The new keyword hides the definition of the base class method and gives a new definition in the derived class.

Q78). How Can We Call the Base Method Without Creating An Instance?

Yes, it is possible,

1. If it is a static method.
2. By inheriting from that class.
3. From derived classes using base keyword.

Q 79). Define The Various Types Of Constructors?

Types of Constructors in Java

Now is the correct time to discuss the types of the constructor, so primarily there are two types of constructors in java:

- No-argument constructor
- Parameterized Constructor

No-argument constructor: A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler creates a **default constructor(with no arguments)** for the class. And if we write a constructor with arguments or no arguments then the compiler does not create a default constructor.
Note: *Default constructor provides the default values to the object like 0, null, etc. depending on the type.*

Parameterized Constructor: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

Q 80). Define Manipulators?

Ans:

A Java manipulator is your own code in Java that takes records from any number of pipeline components in Forge or, optionally, your source data, and changes it according to your processing requirements. A Java manipulator can then write any records you choose to its output.

For example, a Java manipulator can write the "transformed" records into its output, so that the records can be passed to the next pipeline component in Forge.

Java manipulators are the most generic way of modifying your data and records in the pipeline. In other words, content adapters represent a specific case of Java manipulators. You create Java manipulators with the Content Adapter Development Kit API.

A Java manipulator has the following characteristics:

It adheres to the Adapter interface provided with the CADK. It uses the interface described in the Content Adapter API (Javadoc), and requires you to import all the Java classes supplied with the Content Adapter Development Kit.

It can have any number of inputs. It can take either source data, or, most importantly, it can take the records from record adapters or other components in the pipeline, and transform them further, according to your needs.

It can have only one output. Do not emit empty records from your Java manipulator, that is record objects with no PVal objects specified. This can cause Forge to abort processing after acquiring the empty record.

It has a simplified debugging mechanism. You import the JVM java.util.logging.Logger into your Java manipulator code. You can then make calls to the Logger object in different parts of your code to print the logging messages that are logged in the Forge log.

It can transform your records in any way you want; it isn't limited to just changing Pval values.

Q81) Can you give some examples of tokens?

In Java, a program is a collection of classes and methods, while methods are a collection of various expressions and statements. Tokens in Java are the small units of code which a java compilier uses for constructing those statements and expressions. Java supports 5 types of tokens which are:

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. Special symbols

Q82) Explain structured programming and its disadvantage?

Structured programming encourages dividing an application program into a hierarchy of modules or autonomous elements, which may, in turn, contain other such elements. Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability.

The following are the disadvantages of structured programming:

A high level language has to be translated into the machine language by translator and thus a price in computer time is paid.

The object code generated by a translator might be inefficient compared to an equivalent assembly language program.

Data type are proceeds in many functions in a structured program. When changes occur in those data types, the corresponding change must be made to every location that acts on those data types within the program. This is really a very time consuming task if the program is very large.

Let us consider the case of software development in which several programmers work as a team on an application. In a structured program, each programmer is assigned to build a specific set of functions and data types. Since different programmers handle separate functions that have mutually shared data type. Other programmers in the team must reflect the changes in data types done by the programmer in data type handled. Otherwise, it requires rewriting several functions.

Q83) When to use interface over abstract class.

When to use an interface

If the functionality we are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing a common functionality to unrelated classes.

Interfaces are a good choice when we think that the API will not change for a while.

Interfaces are also good when we want to have something similar to multiple inheritances since we can implement multiple interfaces.

If we are designing small, concise bits of functionality, use interfaces. If we are designing large functional units, use an abstract class.

Q84) Explain a private constructor? Where will you use it?

A private constructor in Java is used in restricting object creation. It is a special instance constructor used in static member-only classes. If a constructor is declared as private, then its objects are only accessible from within the declared class. You cannot access its objects from outside the constructor class.

Private Constructor Use-Cases

Private constructors in Java are accessed only from within the class. You cannot access a private constructor from any other class. If the object is yet not initialised, then you can write a public function to call the private instructor. If the object is already initialised, then you can only return the instance of that object. A private constructor in Java has the following use-cases:

You can use it with static members-only classes.

You can use it with static utility or constant classes.

You can use it to serve singleton classes.

You can use it to assign a name, for instance, creation by utilising factory methods.

You can use it to prevent subclassing.

Q85) Can you override private virtual methods?

No, we cannot override private virtual methods in Java.

Private methods in Java are not visible to any other class which limits their scope to the class in which they are declared.

Q86) Can you allow class to be inherited, but prevent from being over-ridden?

In a base class, the sealed keyword is only used to prevent a class from being derived, but in inherited classes it can be used to prevent another inherited class from overriding the method.

Q87) Why can’t you specify accessibility modifiers for methods inside interface?

All members of an interface is marked as public because:

1) If Private:- The members cannot be implemented into child class (as private members are derived into child class).

2) If Protected :- The members cannot be accessed by instance of the class implementing members of that interface (as protected members become private into inherited class) even the members will not be inherited into other child classes of class implementing interface.

3) If Public :- The members can be implemented into child classes and can be accessed by instances of that class as well as can be inherited into child class of that class.

So All the members of Interface are by default considered as PUBLIC.

Q88) Can static members use non static members? Give reasons

A static method can only access static data members and static methods of another class or same class but cannot access non-static methods and variables. Also, a static method can rewrite the values of any static data member.

To access any Non-Static Data member, you need the Reference ID of the object containing them everytime. Whereas static things are a part of class, they are accessed by Class's name. Also, static methods are loaded at compile time, and non-static things at Run time, while compiling, if you call a non-static data member, it would need an object which is created at Run time, that's why it throws an error.

Q89) Define different ways a method can be overloaded?

If a class has multiple methods having same name but parameters of the method should be different is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int, int) for two parameters, and b(int ,int, int) for three parameters then it may be difficult for you to understand the behaviour of the method because its name differs.

Method Overloading in java is based on the number and type of the parameters passed as an argument to the methods. We can not define more than one method with the same name, Order, and type of the arguments. It would be a compiler error. The compiler does not consider the return type while differentiating the overloaded method. But you cannot declare two methods with the same signature and different return types. It will throw a compile-time error. If both methods have the same parameter types, but different return types, then it is not possible.

Java can distinguish the methods with different method signatures. i.e. the methods can have the same name but with different parameters list (i.e. the number of the parameters, the order of the parameters, and data types of the parameters) within the same class.

Q90) Can we have an abstract class without having any abstract method?

An abstract class is a class that is declared abstract —it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed. When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class.

Q91) Explain the default access modifier of a class?

Default access modifier

When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class.

Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package.

Q92) Can function overriding be explained in same class?

No, Function Overriding can only be defined in different class because it treated as Base and Derived or parent-child relationship.

Q93) Does function overloading depends on Return Type?

No, It does not depend on return type. Because if return type is different and function name as well as parameter also same. Then it will give compile time error.

Q94) Can abstract class have a constructor?

Yes, an abstract class can have a constructor in java The compiler automatically adds the default constructor in every class either it is an abstractor class or concrete class. You can also provide a constructor to abstract class explicitly. The constructor is always used to initiate a class or use it to create an object of the class. But we can't instantiate it with the new() operator or any other ways.

Q95) Define rules of Function overloading and function overriding?

Method Overloading

1)For overloading to come into picture, there must be at least two methods of the same name.

2) The methods must have different number of parameters. If both the methods have the same number of parameters, then their type must be different.

Method Overriding

1)For overriding to work, we need to have at least one method with the same name in both the parent class as well as the child class.

2) Both the methods must have the same number of parameters with the same type.

Method Overloading Vs. Method Overriding

Method Overloading	Method Overriding
Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
Access specifier can be changed.	Access specifier must not be more restrictive than original method(can be less restrictive).
Static methods can be overloaded which means a class can have more than one static method of same name.	Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.
private and final methods can be overloaded	private and final methods cannot be overridden.
Return type of method does not matter in case of method overloading, it can be same or different.	*Return type must be same or covariant* in method overriding.
overloading is a compile-time concept	Overriding is a run-time concept

---