Final OOPS Concept Note

1. What Is Oops in Java Explain?
    - OOPS stands For Object Oriented Programming Structure.
    - Write a program in real life Object Is known as Oops.
    - Object Oriented programming means Working with real word Objects.
    - An Object-Oriented programming is a technique through which we can develop and design the programs using CLASS and OBJECT.

2. What are the Four Pillars of OOP?
    - Encapsulation **(providing Security Restrict Direct Access)**
    - Abstractions **(Hiding the Complicity show the Essential Features to the users)**
    - Inheritance **(Providing Reusability)**
    - Polymorphism (provides **code flexibility, reusability, and easier maintenance**)

3. What is a Class and an Object?
    - Class
    - A class is a model/prototype/template/blueprint for creating Object Object Creation is Possible using class Template.
    - Class Is a logical Entity which contain fields and methods.
    - Object
    - An Object is a Physical Entity which exist in the Real World.
    - An object is Instance of the class which contains states and behaviours

4. Can a class exist without an object?
    - Yes, but it has no functional use until instantiated.

5. What is an instance?
    - An **instance** is an **object** created from a **class**.

6. What is instantiation?
    - The process of creating an object from a class.

7. Is java pure Object-Oriented language or not?
    - No, Java is not a pure object-oriented language because it is accepting primitive data type, Actually any language which accepts primitive data type is not a pure object oriented language.
    - Only Objects are moving in the network but not the primary data type so java has introduced Wrapper class concept to convert the primary data types into corresponding wrapper object.

==========Instance Variable==========

8. Instance Variable OR Non static Field?
    - Instance variable we can declare at class level.
    - If a non static variable is declared inside the class but outside of the method then it is called Instance Variable OR Non static field.

Final OOPS Concept Note

- An instance variable is automatically created and initialized with default value at the time of Object creation. [We can't even think about instance variable without object creation].
- An instance variable can be accessible anywhere within the class as well as outside of the class also using valid access modifier.

9. How are instance variables different from local variables?

- **Instance variables:** An instance variable belongs to objects, and it is initialized with a default value at the time of object creation.
- **Local variables** are declared inside methods, constructors, or blocks, and they exist only during the execution of that method.

10. How do instance variables differ from static variables?

- Instance variable
- An instance variable Belongs to Object When Object Is Created All instance variable initialize with default value and Each object has its own copy of these variables.
- Static Variable
- static variable Belongs to class when class in loaded All static variables are executed and initialize with Default value.
- Single Copy is Created for All Variables and Shared by All Objects:

11. Can you access instance variables from a static method?

- No, **static methods** belong to the class itself, not to instances (objects). Instance variables can only be accessed from non-static methods, as static methods do not have a reference to an instance.

12. Can instance variables be initialized in a static block?

- No, instance variables cannot be initialized inside a **static block**. Static blocks are executed only once when the class is loaded and are used for initializing **static variables**, not instance variables.

13. How many Ways to Initialize the Object?

- At the Time of Declaration
- By using Object Creation and Object Reference
- By using Method
- BY using Constructor
- By using Super ()
- By using Non-static Block

14. what is non-static block?

- An instance block is a very special block in java which is automatically executed at the time of creating the Object.
- A non static block is executed before the Constructor body execution
- The first line of any constructor is reserved for this() OR super(). If a contsructor contains super() as a first line of constructor then that constructor 2nd line is reserved for Non static block.

Final OOPS Concept Note

==========Constructor Topics==========

15. What is a constructor?

- Constructor is a special Method Which is used initialize the current Object.
- If the class name and method name same and that particular method Does not Contain any kind of return type then it is called Constructor.
- "A constructor is executed or called at the time of object creation.

16. What are the types of constructors in Java?

- **No-Argument Constructor (Default Constructor)**:
- A constructor that does not take any parameters.
- If no constructor is defined, Java provides a default constructor that initializes object fields to their default values (like null, 0, false).
- **Parameterized Constructor**:
- A constructor that takes one or more parameters to initialize the object with specific values when it is created.
- This allows flexibility in setting initial values for the object at the time of creation.
- **Copy Constructor**:
- A constructor that creates a new object by copying values from an existing object of the same class.
- It is not provided by default in Java, but can be defined by the programmer to enable object duplication.

17. What is constructor overloading in Java?

- Constructor overloading occurs when a class has multiple constructors with the same name but different parameter lists.

18. What is constructor chaining in Java?

- Constructor chaining is the process of calling one constructor from another constructor in the same class or from the superclass using this () or super ().

19. Can a constructor call another constructor in the same class?

- Yes, a constructor can call another constructor in the same class using this ().

20. Can constructors be final, synchronized, or abstract?

- final: Constructors cannot be final because they are used to initialize objects.
- synchronized: Constructors cannot be synchronized.
- abstract: Constructors cannot be abstract because they are not inherited.

21. What is the default constructor in Java?

- : A default constructor is a constructor that takes **no arguments**. If no constructors are defined in a class, Java automatically provides a default constructor that initializes the instance variables to their default values.

22. Can a constructor be private in Java?

- Yes, a constructor can be private. This is often used in design patterns like **Singleton**, where you want to prevent the creation of multiple instances of a class.

Final OOPS Concept Note

23. What is the difference between a constructor and a method in Java?

- Constructor: Has the same name as the class, does not have a return type, and is used to initialize objects.
- Method: Can have any name (except the class name), has a return type (even void), and is used to define behaviour.

24. What is the role of this keyword in constructors?

- This keyword refers to the **current instance** of the class. It is commonly used to differentiate between instance variables and constructor parameters that have the same name.

25. What is a copy constructor in Java?

- A copy constructor is a constructor that creates a new object as a copy of an existing object.

26. What is the significance of super () in a constructor?

- Super () is used to call the **constructor** of the **parent (superclass)**. It is called automatically in a subclass constructor if no explicit call is made, but it can also be used explicitly to invoke a specific constructor of the superclass.

27. Why compiler is adding default constructor to our class?

- We have 2 reasons that why compiler is adding default constructor:
- 1) Without default constructor, Object creation is not possible in java by using new keyword, if the class does not contain user-defined constructor.
- 2) As we know only class level variables are having default values so, new keyword will initialize all the non-static variable with the support of java compiler with default values as shown below.

28. What is the advantage of writing constructor in our class?

- If we don't write a constructor in our program then variable initialization and variable re-initialization both are done in two different lines.
- If we write constructor in our program then variable initialization and variable re-initialization both are done in the same line i.e at the time of Object creation.

29. Can we should declare a constructor with final keyword?

- **You can't use `final` with constructors** in Java because constructors are **not inherited** or **overridden**.
- The `final` keyword is for **methods** (to prevent overriding) and **classes** (to prevent inheritance), but constructors don't need that protection because they **can't be overridden**.

30. What is new keyword in java how its Works?

- **new** is a keyword used to **create objects** and allocate memory for them. It is used to instantiate (create) objects from classes.

31. Object Creation Process?

- ClassName ReferanceVariable s = New Constructor (ClassName)

==========Static Fields==========

Final OOPS Concept Note

32. What is a static field in Java?

- A static field (also known as a class variable) is a variable that belongs to the class, not to instances (objects), If you declare a variables inside a class and outside method with static keyword
- It is shared among all objects of the class.

33. How is a static field accessed?

- You can access it using the **class name** or an **object**, but using the class name is recommended.

34. Can static fields be private?

- Yes. Static fields can use any access modifier: private, public, protected, or default (package-private).

35. Can we use this keyword with static fields?

- **No**, because this refers to the current object, and static fields belong to the class.

36. What is static Block?

- It is a special block in java which is automatically executed at the time of loading the .class file.
- Static blocks are executed only once because in java we can load the .class files only once.

==========

37. **What is Data Hiding?

- Data hiding is nothing but declaring our data members with private access modifier so our data will not be accessible from outer world that means no one can access our data directly from outside of the class.
- We should provide the accessibility of our data through methods so we can perform VALIDATION ON DATA which are coming from outer world.

==========Encapsulation Topic==========

38. *** What is Encapsulation?

- Accessing our private data with public methods like setter and getter.
- Binding the private data with its associated method in a single unit is called Encapsulation.
- It provides security because our data is private (Data Hiding) and it is only accessible via public methods WITH PROPER DATA VALIDATION.

39. Why is encapsulation important?

- Protects data
- Makes code more secure
- Improves maintainability
- Enables control over data (validation)
- Helps in modularity

40. What is the real-life example of encapsulation?

- ATM Machine:

Final OOPS Concept Note

- You press buttons (public methods)
- You don't see how it processes inside (private data/logic)

41. What is the main advantage of encapsulation?

- It allows the **developer to change internal code** without affecting other parts of the program (outside users).
- It provides security because our data is private (Data Hiding) and it is only accessible via public methods WITH PROPER DATA VALIDATION.

-

✅ **6. What is the difference between encapsulation and abstraction?**

Feature	Encapsulation	Abstraction
Hides	Data (variables)	Implementation details (logic)
Achieved using	`private` fields + getters/setters	Abstract classes, interfaces
Focus	How data is accessed	What operations are performed

42. How does encapsulation improve code maintainability?

- You can change the inner logic or variables without affecting other classes, as long as the public interface (getters/setters) remains the same.

43. . How is encapsulation different from inheritance?

- **Encapsulation**: Protects data
- **Inheritance**: Reuses code from a parent class

44. ***Pass by Value?

- Java does not support pointers so java only works with pass by value.
- Pass by value means we are sending the copy of original data to the method.

==========Garbage Collector==========

45. What is a Garbage Collector (GC) in Java?

- It is an automatic memory management in java. It is used to delete un-used objects from the HEAP memory.
- Garbage Collector is a daemon thread, It will scan the HEAP area,
- identify which objects are eligible for Garbage Collector (Object which does not contain any references) then delete those objects from the HEAP me mory so we can create another object inside the HEAP memory.
- It follows Algorithm
- **Mark**: **Mark** means GC scans the heap and **marks all *reachable* (in-use) objects**.
- **Sweep**: then **removes the *unreachable* (unused) objects**.

46. Memory's in java?

- Heap Memory
- In java, whenever we create an object then Object and its content (properties and behavior) are stroed in a special memory called HEAP Memory. Garbage collector visits heap memory only.
- Stack memory
- All the local variables and parameters variables are executed in Stack Frame and available in Stack Memory.
- Method Area
- All static member allocate memory in method area.
-

47. How does Garbage Collection work?

Final OOPS Concept Note

- **Objects are created**: When you create objects in your program, they take up memory.
- **Unused objects**: If an object is no longer used (no references point to it), it's considered **garbage**.
- **Garbage collector runs**: The garbage collector automatically looks for objects that are no longer needed and frees up the memory they were using.
- **memory is reclaimed:** Once the garbage collector finds those unused objects, it removes them and gives back the memory for future use.

========== Blank final field==========

48. **What is a blank final field in Java?**
    - A **blank final field** is a final field that is declared but not initialized at the point of declaration. It must be assigned a value before the constructor completes.

49. Can a blank final field be initialized later?
    - Yes, either through the constructor or through an instance initializer block.

==========**Relation Between Classes**==========

50. Relationship between the classes?
    - In java, In between the classes we have 2 types of relation.
    - 1) IS-A Relation
    - 2) HAS-A Relation

51. What is IS-A Relation in java?
- IS-A represents inheritance, where a subclass inherits from a superclass.
- Tightly coupled means if you modify the superclass, it affects all its subclasses.
- Example
- • An Employee is a type of Person. Just like a person can be a teacher, doctor, or student, an employee is a specific type of person who works for a company.
- • In this case, the Employee inherits general traits from the Person (like having a name, address, and phone number).

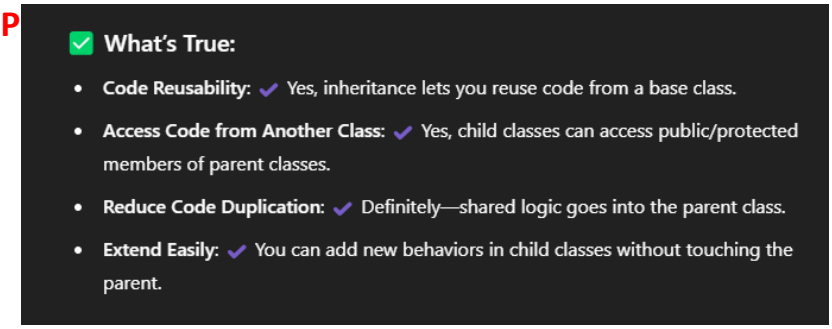
52. . **How is IS-A relationship implemented in Java?**
    - It is implemented using the extends keyword (for classes) or the implements keyword (for interfaces).

53. . **Why use IS-A relationship in Java?**
    - **Code reuse:** The subclass can use code from the parent class.
    - **Polymorphism:** You can treat a subclass object as a superclass object.
    - **Flexible design:** You can build a class hierarchy.

54. What is Inheritance In java?
    - **Inheritance** is one of the fundamental concepts of Object-Oriented Programming (OOP). It allows a **class (called the child or derived class)** to **inherit properties and behaviours (methods and variables)** from **another class (called the parent or base class).**
    - **Or**
    - Inheritance is one of the principles of object-oriented programming (OOP). It describes a relationship between classes where one class inherits all the accessible properties and behaviors of another class, excluding private data members. It provides reusability, helps avoid code duplication, and supports the creation of clean and maintainable programs."

55. Why do we use inheritance?
    - **Reuse Code** – No need to write the same code again.
    - **Organize** – Helps group related classes.
    - **Extend Easily** – Add new features without changing old code.
    - "Inheritance provides code reusability. You can access code from another class, reduce code duplication, and extend functionality easily."

Final OOP



Example:

Example: Employee Management System

- In an **Employee Management System**, you have a general `Employee` class that holds common attributes like `name`, `salary`, and `role`. Then, you have different types of employees like `Fulltime Employee`, `Parttime Employee`, and `Contract Employee`, which all inherit the basic properties from the `Employee` class but add their own specific details.
-

56. Types Of Inheritance?

- 5 types Inheritance in java
  1. **Single Inheritance**
  2. **Multilevel Inheritance**
  3. **Hierarchical Inheritance**
  4. **Multiple Inheritance** (through interfaces)
  5. **Hybrid Inheritance**

1- Single Inheritance

- One class in Inherits from another class is called single level inheritance.
- Ex- A Car is a type of Vehicle.

2- Multilevel Inheritance

- class inherits all the properties and behaviors from a **parent class**, and that parent class itself inherits from another class.
- Grandparent class- parent child
- Grandparent Class: Bank Account → Provides general banking features.
- Parent Class: Saving Account → Adds interest calculation.
- Child Class: DigitalSavingAccount → Adds digital banking features.

3- Hierarchical Inheritance

- Multiple child Class inherits All properties and behaviour one parent Class.
- Ex- Consider an Online Payment Platform where:
- Parent Class: Payment Gateway → Handles basic payment operations.
- Child Class 1: CreditCardPayment → Handles payments through credit cards.
- Child Class 2: UP Payment → Handles payments through UPI.
- Child Class 3: NetBankingPayment → Handles payments through net banking.

4- Multiple Inheritance (through interfaces or not Supported IN class)

- A class Wants To inherits more than One parent Class.

5- Hybrid Inheritance

- Hybrid Inheritance is a combination of multiple and hierarchical inheritance.
-

57. What is super in Java?

- Java, super is a reference variable used to access superclass methods and variables, and to call the superclass constructor.

58. **Why doesn't Java support multiple inheritance with classes?**

- To avoid **ambiguity** and **complexity** caused by the diamond problem.
- Multiple Inheritance is a situation where a sub class wants to inherit the properties more than one super classes .

Final OOPS Concept Note

- In every constructor we have super () or this (). When compiler will add super () to the first line of the constructor then we have an ambiguity issue that super () will call which super class. It is also known as Diamond Problem

59. Rule Of Inheritance?

- To inherit from a class, use the extends keyword.
- Constructor of Parent Class is Called First
- The constructor of the parent class is called before the child class constructor.

60. Different Between Super () and Super in java?

- **Super**
- is a reference variable that refers to the parent (superclass) object.
- To access super class variable (Variable Hiding)
- 2) To access super class method (Method Overriding)
- Super ()
- Super () is used to call the parent class constructor from the subclass
- It must be the first statement in the subclass constructor.
- It is used to initialize the parent class's instance variable

61. How many ways we can initialize the Object Properties?

- 1) At the time of declaration:
- 2)By using Object Reference:
- 3)By using methods:
- B) Second Approach (Method with Parameter)
- 4) By using Constructor
- B) Second Approach (Parameterized Constructor)
- C) Third Approach (Copy Constructor)
- d) By using instance block (Instance Initializer)
- 5) By using super ():

62. Describe Access modifiers in java?

- In order to define the accessibility level of the class as well as member of the class we have 4 access modifiers:
- **1) private (Within the same class)**
- It is the most restrictive access modifier because the member declared as private can't be accessible from outside of the class.
- **2) default (Within the same package/folder)**
- It is an access modifier which is less restrictive than private.
- As far as its accessibility is concerned, default members are accessible within the same folder(package) only. It is also known as private-package modifier.
-
- **3) protected (Within the same package OR even from another package by using Inheritance)**
- he protected access modifier is less restrictive than the default modifier. A protected member can be accessed:
- Within the same package (like default access).
- Outside the package but only through inheritance.
- **4) public (No Restriction)**
- It is an access modifier which does not contain any kind of restriction that is the reason the member declared as public can be accessible from everywhere without any restriction.

63. HAS-A Relation in Java?

Final OOPS Concept Note

- Has-A relationship represents Association concept where we use a class as a property of another class and HAS-A relation we can achieve by using Association concept.
- **ASSOCIATION**
- **Association** builds a relationship between two classes and describes how much one class knows about or interacts with another class.
- In Java, the association can have one-to-one, one-to-many, many-to-one and many-to-many relationships.
- In java Association Divided by 2 Types 1- Composition 2- Aggregation.

- **What is a One-to-Many Association?**
- **A One-to-Many association means that one object of a class is associated with multiple objects of another class.**
- **Example: A Teacher can teach multiple Students.**
- **What is a Many-to-One Association?**
- A **Many-to-One** association means that multiple objects of a class are associated with one object of another class.
  Example: Multiple `Employees` belong to one `Department`.
- **What is a Many-to-Many Association?**
- A **Many-to-Many** association means that multiple objects of one class are associated with multiple objects of another class.
  Example: A `Student` can enrol in many `Courses`, and a `Course` can have many `Students`.
-
- **COMPOSITION (Strong Reference)**
- Composition is a way to design classes where a class contains an object of another class and describes strong reference. The contained class object cannot exist without the container class object.
- Ex- **An e-commerce system**
- For example, in an e-commerce system, an Order contains Order Items. If the order is deleted, the items are deleted too."
- **Aggregation (Week Reference)**
- Aggregation is a way to design classes where a class contain object of contend class and it describe weak reference because in aggregation contend object can exist independently like Student teacher
- EX- A Teacher teaches Students, but Students can exist without a Teacher.
- **The Teacher might have many Students, but even if the Teacher is removed (or changed), the students still exist.**
- **The students are not dependent on the Teacher for their lifecycle.**
-

==========**JVM ARCHITECTURE**==========

64. Role Of JVM?

- JVM is responsible for loading and verify and execute the .class File it verify the .class file and convert byte-code into native code or OS Understanding machine code and ensuring platform independent and manage the memory Allocation.

65. Explain JVM Architecture?

- The entire JVM Architecture is divided into 3 sections
  1. Class Loader sub system
  2. Runtime Data areas (Memory Areas)
  3. Execution Engine

1- Class Loder Sub-System

Final OOPS Concept Note

- The main purpose of Class Loader sub system to load the required .class file into JVM Memory from different memory locations.
- In order to load the .class file into JVM Memory, It uses an algorithm called "Delegation Hierarchy Algorithm".
- Internally, Class Loader sub system performs the following Task

1. LOADING
2. LINKING
3. INITIALIZATION

1- LOADING

- In order to load the required .class file, JVM makes a request to class loader sub system. The class loader sub system follows delegation hierarchy algorithm to load the required .class files from different areas.
- To load the required .class from different area, we have 3 different kinds of class loaders.
- Bootstrap/ Primordial Class Loder
- Platform/Extension class Loader
- Application/System class Loader

1. Bootstrap/ Primordial Class Loder

- It is responsible for loading all the predefined .class files that means all API(Application Programming Interface) level predefined classes are loaded by Bootstrap class loader.
- It has the highest priority because Bootstrap class loader is the super class for Platform class loader.
- It loads the classes from the following path
- C -> Program files -> Java -> JDK -> lib -> jrt-fs.jar

2. Platform/Extension class Loader

- It is responsible to load the required .class file which is given by some 3rd party in the form of jar file.
- It is the sub class of Bootstrap class loader and super class of Application class loader so it has more priority than Application class loader.
- It loads the required .class file from the following path.
- C -> Program files -> Java -> JDK -> lib -> ext -> ThirdParty.jar

3. Application/System class Loader

- It is responsible to load all user-defined .class file into JVM memory.
- It has the lowest priority because it is the sub class Platform class loader.

66. If all the class loaders are failed to load the .class file into JVM memory?

- JVM will generate an exception i.e. java.lang.ClassNotFoundException.
-

67. How Delegation Hierarchy algorithm works?

- Whenever JVM makes a request to class loader sub system to load the required .class file into JVM memory, first of all, class loader sub system makes a request to Application class loader, Application class loader will delegate (by pass) the request to the platform class loader, platform class loader will also delegate the request to Bootstrap class loader.
- Whenever JVM makes a request to class loader sub system to load the required .class file into JVM memory, first of all, class loader sub system makes a request to Application class loader, Application class loader will delegate(by pass) the request to the platform class loader, platform class loader will also delegate the request to Bootstrap class loader.

Final OOPS Concept Note

68. Which class Is lode first in jvm Memory?

- java.lang.Object is the first class to be loaded into JVM Memory.

69. What is Method Channing in java?

- It is a technique through which we call multiple methods in a single statement.
- In this method chaining, always for calling next method we depend upon last method return type.

- **LINKING PHAGE**
- In linking phase, we have 3 modules:
- a) Verify b) Prepare c) Resolve
- Verify: -
- It ensures the correctness of the .class files, If any suspicious activity is there in the .class file then It will stop the execution immediately by throwing a runtime error i.e. java. lang. VerifyError.
- There is something called ByteCodeVerifier (Component of JVM), responsible to verify the loaded .class file i.e. byte code. Due to this verify module JAVA is highly secure language.
- prepare:
- Static variable memory allocation + static variable initialization with default value even the variable is final]
- It will allocate the memory for all the static data members, here all the static data member will get the default values so if we have static int x = 100; then for variable x memory will be allocated (4 bytes) and now it will be initialized with default value i.e. 0, even the variable is final.
- **Resolve:**
- All the symbolic references (#7) will be converted into direct references OR actual reference.
- **Initialization:**
- Here class initialization will take place. All the static data member will get their actual/original value and we can also use static block for static data member initialization.
- Here, in this class initialization phase static variable and static block is having same priority so it will be executed according to the order. (Top to bottom)

70. Can we write a java program without main method?

- It looks like we can write a java program by using static block (without main method) but this facility was available till JDK 1.6.
- From JDK 1.7V onwards we can't write a java program without main method because at the time of class loading JVM will verify the presence of main method.

71. How many ways to lode the .class file?

- 1) By using java command
- By using Constructor (new keyword at the time of creating object)
- By accessing static data member of the class.
- By using inheritance
- By using Reflection API
- Loading the .class file by using Reflection API :
- java.lang.Class class has provided a predefined static factory method called forName(String className), It is mainly used to load the given .class file at runtime, The return type of this method is java.lang.Class

72. What is the difference between java.lang.ClassNotFoundException and java.lang.NoClassDefFoundError?

Final OOPS Concept Note

- **java.lang.ClassNotFoundException**
- It occurs when we try to load the required .class file at RUNTIME by using Class.forName(String className) statement or loadClass() static of ClassLoader class and if the required .class file is not available at runtime then we will get an exception i.e java.lang.ClassNotFoundException
- We will get java.lang.ClassNotFoundException because at runtime JVM will try to load Sample.class file but Sample.class file is not available at runtime
- **java.lang.NoClassDefFoundError :**
- It occurs when the class was present at the time of COMPILATION but at runtime the required .class file is not available(manualy deleted by user) Or it is not available in the current directory (Misplaced) then we will get a runtime error i.e java.lang.NoClassDefFoundError.

73. A static method does not act on instance variable directly why?

- All the static members (static variable, static block, static method, static nested inner class) are loaded/executed at the time of loading the .class file into JVM Memory.
- At class loading phase object is not created because object is created in the 2nd phase i.e Runtime data area so at the TIME OF EXECUTION OF STATIC METHOD AT CLASS LOADING PAHSE, NON-STATIC VARIABLE WILL NOT BE AVAILABLE BY DEFAULT henec we can't access non static variable from static context [static block, static method and static nested inner class] without creating the object.

- Runtime Data areas :
- **HEAP AREA :**
- All the Objects and its member (NSV + NSM) are available in HEAP memory. The Objects which are stored in the HEAP Memory
- are eligible for GC when object does not contain any references.
- We have only one HEAP Area per JVM.
-
- **STACK AREA :**
- All the methods are executed in Stack Area.
- All the local and parameter variables are stored in Stack Area.
- Every time we call a method in java then a separate Stack Frame will be created and each stack frame contains 3 parts :
- a) Local Variable Array
- b) Frame Data
- c) Operand Stack
- We have multiple Stack Area per JVM.
-
- **PC Register :**
- It stands for Program Counter Register.
- In java environment, we can craete multiple threads.
- In order to hold the currently executing instruction of each thread we have a separate PC register. [13-MAR]
-
- **Native Method Stack :**
- Native method means, the java methods which are written by using native languages like C and C++. In order to write native method we need native method library support.
- Native method stack will hold the native method information in a separate stack.
- **Execution Engine: [Interpreter + JIT Compiler]**
- Interpreter

Final OOPS Concept Note

- In java, JVM contains an interpreter which executes the program line by line. Interpreter is slow in nature because at the time of execution if we make a mistake at line number 9 then it will throw the execption at line number 9 and after solving the exception again it will start the execution from line number 1 so it is slow in execution that is the reason to boost up the execution java software people has provided JIT compiler.
-
- JIT Compiler :
- It stands for just in time compiler. The main purpose of JIT compiler to boost up the execution so the execution of the program will be completed as soon as possible.
- JIT compiler holds the repeated instruction like method signature, variables, native method code and make it available to JVM at the time of execution so the overall execution becomes very fast

=======**POLYMORPHISM TOPICS**==========

74. What is POLYMORPHISM in java?

- Poly = many
- Morphs = forms
- So, Polymorphism means "many forms."
- Poly morphism is one of the four pillars of Object-Oriented Programming (OOP).
- The word Polymorphism is derived from two Greek words whose meaning is "same object having different behaviour".
- One object having different behaviour means it allows one object to tak e multiple forms.
- Or//
- Polymorphism allows the same method or function name to behave differently based on the object that is calling it. Thsere are two main types: **compile-time polymorphism** (like method overloading) and **runtime polymorphism** (like method overriding). This helps make code more flexible and easier to maintain.”

75. Why We use polymorphism?

- ✅ **Code Reusability**
- You can write code that works on the **parent class** and it will work for all **child classes** too.
- ✅ **Flexibility**
- You can change or extend behaviour **without changing existing code**.
- ✅ **Maintainability**
- Easier to maintain and upgrade applications with less risk of breaking code.
- ✅ **Method Overriding (Runtime Polymorphism)**
- You can **override methods** in subclasses to provide **custom behaviour**.
- ✅ **Dynamic Method Dispatch**
- JVM decides at runtime which method to call — based on the actual object, not reference type.
- **IN short---**
- "**Polymorphism** helps achieve **dynamic method dispatch**, **code reusability**, and **flexibility**. It lets the same method behave differently based on the object, which is useful for writing clean, extensible code.

76. How to achieve Polymorphism or types?

- Polymorphism can be divided into two types:
  1. Static polymorphism OR Compile time polymorphism OR Early binding
  2. Dynamic Polymorphism OR Runtime polymorphism OR Late binding
- **1) Static Polymorphism**

Final OOPS Concept Note

- The polymorphism which exists at the time of compilation is called Static OR compile time polymorphism.
- In static polymorphism, compiler has very good idea that which method is invoked depending upon METHOD PARAMETER.
- Here the binding of the method is done at compilation time so, it is known as early binding.
- We can achieve static polymorphism by using Method Overloading concept.

77. What is method-Overloading In java?

- **Method Overloading** in Java means **having two or more methods in the same class (or subclass)** with the **same name** but **different parameter lists** (type, number, or order).
- It can apply to **both static and non-static methods**, but the key thing is **same method name, different parameters**.
- **Why We Use Method-Overloading?**
- Method overloading allows us to define multiple methods with the same name but different parameters. It helps ls improve code readability and maintainability, makes our code more flexible, and supports compile-time polymorphism. It's especially useful when we want to perform similar operations with different types or numbers of inputs."
- Ex- **Real-Time Scenario: Payment System**
- an app (like Amazon, Swiggy, or Uber) where a customer can pay in different ways:
- Using a **credit card**
- Using a **debit card**
- Using **UPI**
- Using **wallet or coupon**

Why It's Overloading:

- All methods are named `make Payment ()`
- Parameters are different in type or number
- Helps handle **multiple payment options** using the **same method name** — neat and clean

Ex-2

When you use an ATM, there are **different ways to withdraw money**:

1. Withdraw using **just amount**
2. Withdraw using **amount + account type** (e.g., savings or current)
3. Withdraw using **amount + account type + PIN**
4. Withdraw using **amount + account type + PIN + location**

5. **Why It's Overloading:**
6. All methods are named `withdraw`
7. Same class, different parameter lists
8. This is **method overloading**, making the ATM system flexible depending on how much info is available.

78. What is the benefit of Static Polymorphism?

- Code readability
- Improved reusability
- Allows flexibility in calling methods with different arguments.

Final OOPS Concept Note

79. Ambiguity issue while overloading a method?

- Compiler always provide more priority to most specific data type or class type.
- 2) WAV [Widening -> Autoboxing -> Var Args]
- Ambiguity in method overloading occurs when the compiler can't decide which overloaded method to call due to equally valid matches from type conversion. It leads to a compile-time error
- **2) Dynamic Polymorphism**
- The polymorphism which exists at runtime is called Dynamic polymorphism Or Runtime Polymorphism.
- Here compiler does not have any idea about method calling, at runtime JVM will decide which method will be invoked depending upon CLASS TYPE OBJECT.
- Here method binding is done at runtime so, it is also called Late Binding.
- We can achieve dynamic polymorphism by using Method Overriding.
- **What is Method Overriding?**
  - Method Overriding involves writing two or more non-static methods in a superclass and subclass in such a way that the method signature (method name and parameter list) is the same. The main point is that the method in the subclass overrides the method in the superclass to provide a new implementation.

  Or /

  "The polymorphism which exists at runtime is known as dynamic or runtime polymorphism. It allows the JVM to decide which method to call or invoke at runtime, based on the actual object type, not the reference type. This refers to method overriding, where a method is defined in both the superclass and subclass in such a way that the method signature must be the same, or the return type must be the same or covariant. This allows the subclass to provide a specific implementation."

80. Why is Dynamic Polymorphism useful?

- **Code flexibility**
- **Promotes interface-based programming**
- **Enables loose coupling and extensibility**
- ✅ **Method Overriding (Runtime Polymorphism)**
- You can **override methods** in subclasses to provide **custom behaviour**.
- Method overriding allows a subclass to provide a specific implementation of a method already defined in its superclass. This enables dynamic polymorphism, where the JVM calls the appropriate method at runtime based on the actual object.

Ex – Notification System

- "A real-time example of method overriding is a **notification system** where the base class has a method like sendNotification (), and different notification types like **email**, **SMS**, or **push** override it to perform specific actions."

81. Role of access modifier while overriding a method?

- While overriding the method from super class, the access modifier of sub class method must be greater or equal in comparison to access modifier of super class method otherwise we will get compilation error.
- In terms of accessibility, public is greater than protected, protected is greater than default (public > protected > default) [default < protected < public]
- So, the conclusion is we can't reduce the visibility of the method while overriding a method.

82. Can we change return type of a method time of overriding in java?

- In Java, the return type **can** be changed while overriding a method, but there are **specific rules** to follow. This is known as **Covariant Return Type**.

83. What is a Covariant Return Type?

Final OOPS Concept Note

- **Covariant** in Java allows a method in a subclass to return a more **specific type** (subclass) instead of the general type (superclass) defined in the parent class.
- while overriding a method, the return type can be a **subclass** of the return type declared in the parent class.
- **covariance** allows a **subclass** to override a method from the **superclass** and change the **return type** to a **more specific type**.

84. What is method Hiding in java?

"Method hiding happens when you define static methods with the same signature in both the superclass and the subclass. Since static methods belong to the class and are not object-specific, they are resolved at compile-time based on the reference type, not the actual object type."

- **At the time of compiling**, the compiler will search for the method in the **superclass** if the reference type is of the superclass.
- **At runtime**, the JVM will execute the method based on the reference type, so if the reference is of the superclass type, the JVM will call the method in the superclass, even if the object is of the subclass type.

85. What is var- args in java?

- It was introduced from JDK 1.5 onwards.
- It stands for variable argument. It is an array variable which can hold 0 to n number of parameters of same type or different type by using Object class.
- var-args must be only one and last argument.
- Orr
- Varargs (Variable Arguments) in Java allows you to pass a variable number of arguments to a method. It is a feature introduced in Java 5 that provides a more flexible way to handle method parameters when you are unsure of how many arguments will be passed.
-
- With var-args, you can pass zero or more arguments to a method, and Java will automatically treat them as an array within the method.

86. Explain Wrapper classes in java and why we use?

- java is not fully Object-Oriented Programming Language because of 8 primitive data type the problem of 8 primitive datatype actually in the network only object is moving not 8 primitive datatype that's why java software people introduce wrapper class.
- Converting Primitive datatype to corresponding Wrapper Objects.
- From JDK 1.5 Onwards INTRODUCE 2 CONCEPTS.
- 1. AutoBoxing (Converting primitive datatype to corresponding Wrapper Object)
- Using Method: - Integer.ValueOf();
- 2. UnBoxing:- (Converting Wrapper Object to Corresponding Primitive Datatype)
- Using method:- xxxValue ();
- Integer x =3;
- Int x = x.IntValue ();

87. Are wrapper classes immutable in Java?

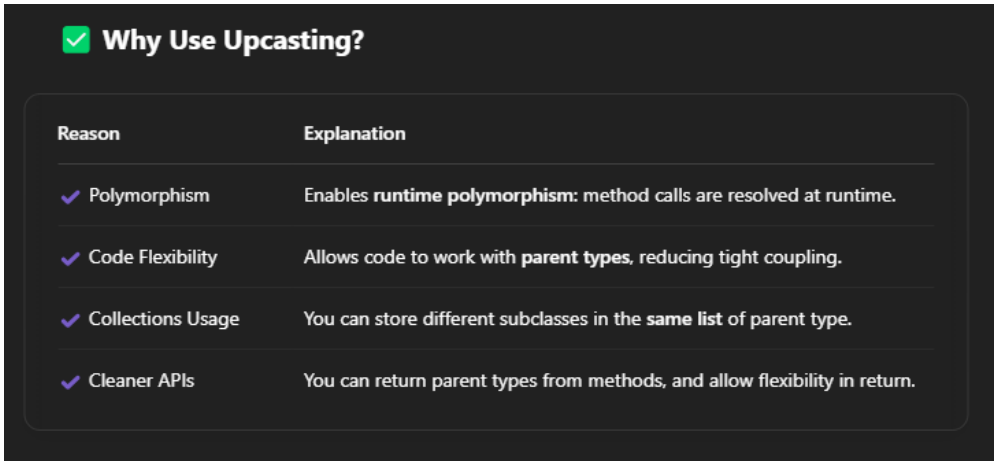
- Yes, all wrapper classes in Java are **immutable**, meaning their values **cannot be changed** after creation.

88. What is Upcasting and Down-casting in java?

- Assign Sub class object into superclass reference variable called Upcasting.

Final OOPS Concept Note

- when overriding a method, but it becomes necessary when you want to invoke the overridden method using a reference of the superclass type.
-



89. **What is Down-casting?**

- Assigning super class object into sub–Class reference it not possible without upcasting.
- Ex: - Super s1 = new Sub ();
- s1.m1();
- Sub s2 = (Sub) s1;
- s2.m3();
- **When we use Down-Casting?**
- In order to call specific method of sub class we need current class reference so down casting is required.
- Orr
- **Upcasting** is used to achieve **runtime polymorphism** and make code more **flexible** and **maintainable**.
  Down casting is used when you want to access **subclass-specific methods** or properties from a superclass reference, but it must be done **carefully** using instanceof to avoid runtime errors.

90. Assigning smaller datatype into bigger is known as **widening**, it is implicitly done by Java.
    Narrowing is when you explicitly assign bigger datatype into smaller datatype using castin

91. **instanceof Operator:**

- It is a relational operator as well as keyword It return true/false
- It is mainly used to verify whether a reference variable is pointing to a particular type of Object or not?
- In order to deal with multiple objects, we need to take the support of
- instance of operator.to a particular type of Object or not?

92. **What is final Keyword in java?**

- It is used to provide some kind of restriction in our program.
- We can use final keyword in ways 3 ways in java.
- 1-to declare a class (inheritance is not possible)
- 2-to declare a method (Overridden is not possible)
- 3-to declare a variable as final (reassign Is not possible)
- 1) To declare a class as a final. (Inheritance is not possible)
- whenever we declare a class as a final class then we can't extend or inherit that class otherwise we will get a compilation error.
- We should declare a class as a final if the composition of the class (logic of the class) is very important and we don't want to share the feature of the class to some other developer to modify the original behaviour of the existing class, in that situation we should declare a class as a final.
- for final class we can create object as well as we can modify the data.

Final OOPS Concept Note

- Whenever we declare a constructor as private then we should declare the class with final modifier. If constructor is private then we can't create a sub class because super class constructor is not visible from sub class constructor.
- 2) To declare a method as a final (Overriding is not possible)
- Whenever we declare a method as a final then we can't override that method in the sub class otherwise there will be a compilation error.
- We should declare a method as a final if the body of the method i.e the implementation of the method is very important and we don't want to override or change the super class method body by sub class method body then we should declare the super class method as final method.
- 3) To declare a variable (Field) as a final (Re-assignment is not possible)
- If we declare a variable as a final then we can't perform re-assignment (i.e nothing but re-initialization) of that variable.

==========**ABSTRACTION TOPICS**==========

93. What is SEALD-CLASS

- A sealed class in Java is a class that restricts which other classes can extend or inherit from it.
- We use it when we want to allow only specific classes to be its subclasses. This helps keep our code more secure and controlled.
- It was introduced in Java 17, and we use the sealed keyword along with permits to list the allowed subclasses.
- By using sealed keyword, we can declare classes and interfaces as sealed
- The class which is inheriting from the sealed class must be final, sealed or non-sealed.
- The sealed class must have at least one sub class.
- We can also create object for Sealed class.
- It provides the following modifiers:
- 1) sealed: Can be extended only through permitted class.
- 2) non-sealed: Can be extended by any sub class, if a user wants to give permission to its sub classes.
- 3) permits: We can provide permission to the sub classes, which are inheriting through Sealed class OR sealed interface
- 4) final: we can declare permitted sub class as final so, it cannot be extended further.

✅ **Why use sealed classes in Java?**

Sealed classes let you:

1. **Restrict inheritance** – only specific classes can extend them.
2. **Improve safety** – compiler knows all subclasses, great for switch.
3. **Make APIs cleaner** – you define a fixed set of subclasses.
4. **Enable better optimization** – JVM can optimize better.

94. What is Abstraction in java?

- If we display only the essential details without showing the background details (complexity) then it is called Abstraction.
- Abstraction in Java is a way to hide the complex details of a program and only show the essential features to the user. It allows you to focus on what an object does, not how it does it.
- Abstraction is a concept in Java where we hide unnecessary details and only expose the essential features of an object.
- **For example, when we use a method like `System.out.println`**
  -

95. Why do we use abstraction?

- Hides internal implementation
- Shows only important features
- Reduces complexity

Final OOPS Concept Note

- Increases security
- Makes code flexible and maintainable (Loose-coupling)
- Helps in designing clean, reusable APIs

96. How to achieve Abstraction in java?

- In java, we can achieve abstraction by using the following two concepts:
- 1) Abstract class and abstract method [Partial Abstraction, 0 to 100%]
- 2) Interface (Full Abstraction, 100%)

97. What is an abstract class in Java?

- A class that does not provide complete implementation (partial implementation) is defined as an abstract class.
- "An abstract class can contain abstract methods, concrete methods, or combination of both."
- if a class contains at least one method as an abstract method then we should compulsory declare that class as an abstract class.
- Once a class is declared as an abstract class we can't create an object for that class. Because an abstract class is a blueprint or template of another class or an abstract class itself incomplete.
- **The purpose of an abstract class is to provide a common blueprint for its subclasses, while allowing the subclasses to define their own specific behaviour by implementing the abstract methods.**
- Or
- An **abstract class** in Java is a class that **cannot be fully implemented** and is declared using the **abstract keyword**.
- You **cannot instantiate** an abstract class directly because it is **incomplete** — it serves as a **blueprint or template** for other classes.
- An abstract class can contain **abstract methods** (without a body), **concrete methods** (with implementation), or **a combination of both**.
- It helps define **"what to do"**, and the **subclass provides the "how to do"** part by implementing the abstract methods.
-

98. What is Abstract Method in java?

- An abstract method observation is very simple because every abstract method contains abstract keyword, abstract method does not contain any method body
- In java, whenever action is common but implementations are different then we should use abstract method, generally we declare abstract method in the super class and its implementation must be provided in the sub classes.
- if a class contains at least one method as an abstract method then we should compulsory declare that class as an abstract class.
- All the abstract methods declared in the super class must be overridden in the sub classes otherwise the sub class will become as an abstract class hence object can't be created for the sub class as well.
- An abstract class may or may not have abstract method but an abstract method must have abstract class.
- Or
- An **abstract method** is a method that is declared using the abstract keyword, **does not have a body**, and ends with a **semicolon (;)**.
  We use an abstract method **when the action is common, but the implementation differs** across subclasses.
  The **subclass must provide the implementation** of the abstract method.

99. What is the difference between abstraction and encapsulation?

- **Abstraction:**
- **Purpose:** Hides complex implementation details and shows only the essential features.
- What it does: Focuses on what an object does.

Final OOPS Concept Note

- **How:** Achieved through abstract classes and interfaces, allowing you to define methods without implementing them.
- **Example:** A car dashboard showing the "start engine" button without exposing how the engine works.
- **Encapsulation:**
- **Purpose:** Hides the internal state and protects data by restricting direct access.
- What it does: Focuses on how an object stores and protects its data.
- **How:** Achieved by using private variables and providing public methods (getters and setters) to access them.
- **Example:** A car having a private engine that can only be controlled by pressing buttons or using a key.
- **In summary:**
- Abstraction simplifies complex systems by hiding details.
- Encapsulation protects the internal state and ensures controlled access to it.

100. Example in Abstraction?

- **Abstraction in the ATM Example:**
- **What the user sees**: The user interacts with options like "Withdraw", "Deposit", and "Check Balance", without knowing the internal workings of how the ATM communicates with the bank or how transactions are processed.
- **What is hidden**: The complex internal logic such as validating the account, checking balance, processing network requests, updating the database, and handling security concerns like PIN verification.
- **Abstract Class**: The ATM system has an abstract class Transaction that defines the structure for different types of transactions like Withdraw, Deposit, and Balance Inquiry.

101. What is Anonymous inner class in java?

- **If we declare a class inside a method body without any class name then it is called Anonymous inner class.**
- The main purpose of anonymous inner class to extend a super class or to implement an interface that means for creating sub type.
- [Anonymous inner class is only used for creating the sub type]
- **Anonymous inner class declaration and Object creation by using new keyword both will be done in a single line i.e single statement.**
- **"We use anonymous inner classes to reduce boilerplate code and make the code more concise, especially for one-time implementations of interfaces or abstract classes."**

==========**Interface in java**==========

102. What is Interface In java?

- An interface is a Keyword in java which is similar to a class .
- An interface represents intermediate person between Client and Developer means an interface describe what to do and its implementer class Describe how to do.
- Interface methods are by default public + abstract.
- Interface variable's is by default public static final.
- Interfaces are used to achieve **abstraction** and **multiple inheritance** in Java."
- We can achieve loose coupling using interface.
- Up to 7 version interface contains only abstract methods from java 8 onwards we have a facility to write default and static methods. From **Java 9**, interfaces can have **private** methods too.

103. Why we can't create object an interface?

- **"We can't create an object of an interface because it doesn't have any implementation.** It only contains method declarations (abstract methods), and there's no actual code to run — so Java doesn't know what to execute."

104. Why we use Interface in java?

Final OOPS Concept Note

- "We use interfaces in Java to achieve **abstraction** and **loose coupling**, and to support **multiple inheritance**. An interface defines **a contract** that classes must follow, but it doesn't provide the actual implementation. This allows us to write flexible, modular, and reusable code."

105. What is Loose Coupling and Tightly Coupled IN java?

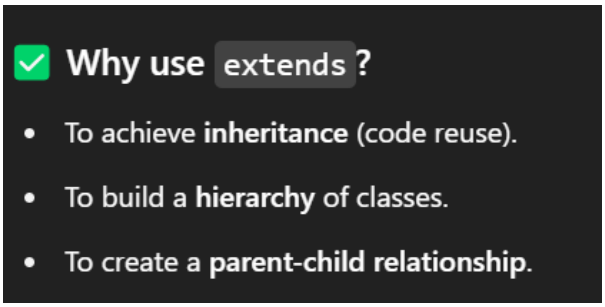
- If the degree of dependency from one class object to another class is very low then it is called loose coupling. [interface is reqd].
- According to IT industry standard we should always prefer loose coupling so the maintenance of the project will become easy.
- - If the degree of dependency of one class to another class is very high then it is called tightly Coupled.
- **High Cohesion [Encapsulation**
- Our private data must be accessible via public methods (setter and getters) so, in between data and method we must have high cohesion.

106. 73.How can we achieve multiple inheritance using interface?

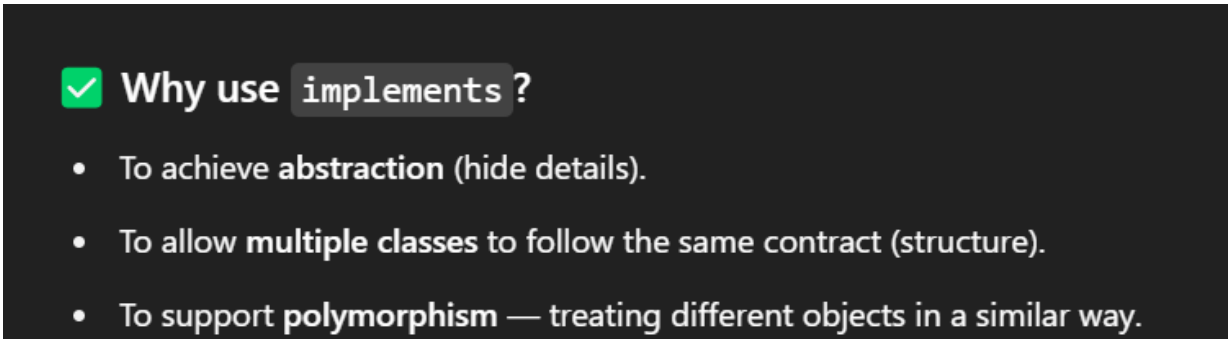
- interface never contains any constructor so the implementer class constructor will directly call the Object class Constructor hence multiple Inheritance is possible in java.

107. Different between Extends and Implements And uses in interface?

- **extends**:
- **Used with classes and interfaces.**
- When a class **extends** another class, it means the subclass inherits the **behaviour and properties** (fields and methods) of the superclass.
- When an interface **extends** another interface, it means the child interface inherits the **abstract methods** of the parent interface.
- **In Class Inheritance**: A class can only extend **one** class due to **Java's single inheritance model**.
- **In Interface Inheritance**: An interface can **extend multiple interfaces**. This supports **multiple inheritance** in Java, but only for interfaces.



- **implements**:
- **Used with classes**
- A class **implements** an interface, which means the class **agrees to provide the actual implementation** for all the abstract methods declared in the interface.
- A **class can implement multiple interfaces**. This is Java's way of achieving **multiple inheritance** of behavior.
- One interface can extend another interface, It cannot implement, implementation means we need to provide the body for abstract method.



Final OOPS Concept Note

108. Why they allowed to write method body inside an interface?

- The basic problem OR limitation with abstract method is, It must be implemented in all the implementer (sub) classes otherwise the implementer class will become as an abstract class.
- It leads to big problem for industry because, If we want to add any new feature (new abstract method) to the existing interface then it must be overridden in the sub class, whether it is required OR not required?
- From JDK 1.8V, Java has provided default method facility.

109. What is a static method inside an interface?

- It is allowed to write static method with method body inside an interface from JDK 1.8V.
- It is used to represent the common feature for all the classes which are going to access this static method.
- By default, the access modifier of static method is public.
- Static method of an interface can be accessible through interface name only because the static method of an interface is limited to interface only, it is not available inside the implementer classes.

110. difference between class static method and interface?

- interface static methods are by default public.
- interface static methods are available to interface only, it is not available in the implementer classes.

111. Different between interface and Abstraction?

Aspect	Abstract Class	Interface
Abstract Methods	Can have abstract and concrete methods.	Can only have abstract methods (until Java 8, which added default methods).
Fields	Can have instance variables.	Can only have constants (`public static final`).
Constructors	Can have constructors.	Cannot have constructors.
Access Modifiers	Can have any access modifiers (public, protected, etc.).	Methods are implicitly **public**.
Inheritance	A class can extend only one abstract class.	A class can implement multiple interfaces.
Purpose	Used to share code between related classes and define common behavior.	Used to define a contract for behavior without implementation.
Multiple Inheritance	Not allowed (only single inheritance).	Allowed (can implement multiple interfaces).

112. What is Functional Programming?

- It is introduced from Java 8v.
- In Functional Programming, First time java concentrated on functions i.e methods.
- The main purpose of learning functional programming is to reduce the length of the function OR method.

113. What is a Functional Interface?

- If an interface contains exactly one abstract method, and It may contain 'n' numbers of default and static method but it must contain only one abstract method.
- it may be represented by @FunctionalInterface annotation so the Java compiler will restrict the developer to take only one abstract method.

114. What is lambada Expression in java?

- It is a new feature introduced in java from JDK 1.8 onwards.
- It is an anonymous function i.e. function without any name.
- In java it is used to enable functional programming.
- It is used to concise our code as well as we can remove boilerplate code.
- Lambda will work only with functional interface.

115.