**JAVA IN ONE DOC**

**Java:** Java, OOP, OOP Pillars, constructor, finalize method, super, new, this, finally keywords, method overriding, overloading, interface, abstract class, aggregation, composition, association, dependency injection, data parsing, Diamond problem, Pointer

**Design Patterns:** Singleton, Consumer Producer

**DS:** array list, linked list, heap, stack, bt, bst, bfs, dfs

**Database:** Views, index, stored procedure, triggers, primary key, foreign key, composite key

**JAVA:**

**Java** is a class-based **object-oriented programming** language

**oop** is a concept based on structure which follows **class** and **object** model.

**Class** means: a theoretical concept in which we have methods which defines property & behavior of objects.  
**object** means: a real-world entity, to call a class we create its object by using **new** keyword.

**new**: provides dynamic memory.

**Static method:** it means the method belong to same class where it is created

**Void:** Method returns nothing

**main**: start of the program

**constructor:** it is a special method that is used to initialize objects. The constructor is called when an object of a class is created. The name of constructor should be as same as of the name of class.

**this:** it refers to current invoking object.

**Super:** target parent class constructor

**final:** used to define constant value to a variable. Or constant class, constant method for avoiding polymorphism and inheritance.

**finally:**  it is used with try catch block, no matter what happens in try catch !! finally block always executed.

**finalize:** before garbage collector! Finalize is call to free some space and clean the program.

**PILLARS**

**Polymorphism**: => many forms

**Overloading:** static binding => compile time

* create method/constructor with same name diff argument in same class will leads to method/ constructor overloading

**Overriding:** dynamic binding => runtime

* create method with same name same argument in diff class and use inheritance will leads to method overriding.

**Inheritance:** used to inherit properties of other class. By using extends keyword.

**Multiple inheritance:** only possible in **interface** which used implements keyword.

**Interface:** it is used to implement 100% **abstraction** in classes. This means that all methods in an interface are unable to have a body — the body must be implemented in the class that implements the interface. In this every method is by default abstract.

**Abstraction:** used to hide internal working and showing only relevant information.

* **Abstract Class**: In this we have both abstract and not abstract methods uses inheritance for implementing abstract methods.
* **Abstract methods**: body of abstract method is implements in the child class and also it will be called using instance of child class.

**Encapsulation:** It is used to wrap up the code within class. By making data members private and calling them using getter setter.

**Relationship Of Classes**

**Q #1) What is the difference between ‘****IS – A’ and ‘HAS – A’ relationship?**

Answer: IS-A relations represent inheritance in which a child class is a ‘kind of’ a parent class.

The Has-a relationship represents association which means composition and aggregation in which it denotes a whole-part relationship where a part cannot exist without the whole.

**Q #2) What is Aggregation and Composition in Java?**

Answer: In Composition, the two objects are tightly coupled i.e., when one object destroyed, then the other containing object will automatically destroy as it cannot function independently on its own.

In Aggregation, one object uses another object by this they can function independently.

**Q #3) Why do we use Composition in Java?**

Answer: Composition allows us to reuse only what we need from the object and also control its visibility. It also reduces code complexity as well as the bugs in the code.

**Q #4) What is the use of Association?**

Answer: Association represents the ‘has-a’ relationship in Java. Association in Java is used when one object needs to use the functionality and services of another object. Composition and aggregation are two forms of association.

**Q #5) Is Composition better than Inheritance?**

Answer: It depends on the OOP system that we are coding. If we can represent the objects with IS-A relationship throughout its lifetime in the application then inheritance is a better option. When we want to reuse the code and cannot represent the object system with the IS-A relationship then the composition is better. Also, when we want to frequently change front-end classes without disturbing the back-end classes, then the composition is better because in the case of inheritance slight changes mostly affect the entire inheritance hierarchy.

**dependency injection**: It removes the dependency of the programs. It makes our code loosely coupled and easier for testing. We can do it with Method / constructor / properties .

coupling: In this we have dependency / relation one thing rely on other due to which security issue occurs so we try to do less coupling

cohesion: In this we don’t have dependency / relation and the components are independent so we try to do more cohesion.

**data parsing**: it is the process of taking data in one format and transforming it to another format.

**Diamond problem**: It leads to the ambiguity as the compiler doesn't know which superclass method to execute. Because of the diamond-shaped class diagram, it's referred to as Diamond Problem in java. The diamond problem in Java is the main reason java doesn't support multiple inheritances in classes

**Pointer:** It is just the address of some location in memory. In Java, pointers play an important role behind the scenes in the form of references to objects. A Java variable of object type stores a reference to an object, which is just a pointer giving the address of that object in memory.

**Design Patterns:**

**Singleton:** It involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

**Consumer Producer:** The producer consumer pattern is a concurrency design pattern where one or more producer threads produce objects which are queued up, and then consumed by one or more consumer threads. The objects enqueued often represent some work that needs to be done. Decoupling the detection of work from the execution of work means you can control how many threads at a time that are engaged in detecting or executing the work.

**DS:**

**Array:** It is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

**Array list:** It can change their size dynamically. If array is not big enough to store new elements, a new double size array is created to replace the old one.

**linked list:** It stores data in the form of node. Each node has a data part and an address part. The address part stores the link to the next node. To do insertion you have to create a new node and link it with the linked list node using the address part.

**Stack Vs Heap Java**

Stack is used to store the order of method execution and local variables while the heap memory stores the objects and it uses dynamic memory allocation and deallocation.

**Binary tree:** A tree in which each node (parent) has at most two-child nodes (left and right) is called binary tree.

**Binary search tree:** It isOrdered Binary Tree. In BST, all the nodes in the left subtree have values that are less than the value of the root node. Similarly, all the nodes of the right subtree of the BST have values that are greater than the value of the root node. This ordering of nodes has to be true for respective subtrees as well.

**bfs:** it goes for shallowest node search neighbors before children.

**dfs:** it goes deep in each branch before moving to explore another branch.

**Database:**

**Views**: it’s a virtual table, it’s a copy of original table, whatever changes we do in view does not affect original table we create it so that if we do any change that will be reflected on the view.

**Index**: it is a lookup display of database; it helps us in speed retrieval of rows from table or view due to which our database gets optimized.

**stored procedure**: it provides security to our data and also help us in reusability, we create procedure and reuse it whenever we want by just passing the parameters.

**triggers**: when any event call it perform action, basically trigger perform action.

**primary key**: uniquely defined the record, it cannot be null, only one pk in table.

**foreign key**: primary key of one table, used in another table as ref is said to be a foreign key

**composite key:** group of columns in the database, whose values together make a unique value is known as composite key.

example: The primary key first Name + last Name for students is a composite key.