Q. What is Java, and why is it not considered a pure oop language?

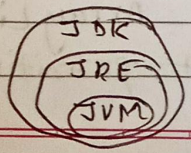
—

- Java is class based oop language
- Write Once, Run Anywhere → compile java code, it can run on any platform that supports Java without recompiling
- But, Not everything is object. instead they are primitive like int or char. Eventhough it has wrapper class like Integer or Float that converts primitive into objects it still allow direct operations on primitive. Therefore, not pure oop language.

Q. Key components involved in running a Java program?

— To write Java program → first create Java source file (.java file)

- On compilation .java —converted→ bytecode and store in .class file
  Bytecode is platform independent
- Execution :
  → JDK Java Development Kit provides all the tools to create Java app
  → JRE Java Runtime Environment is needed to run compiled bytecode
  → JVM Java Virtual Machine runs the program across different platforms. by interpreting and executing line by line bytecode.



Q. What are main features of Java?

— · Simple and Secure
· Runs on any platform
· Organizes code through objects → easier to write and maintain
· Supports multithreading

Q. What is Java String Pool?

— It is a memory where all the string values you create are stored. It prevents wasting memory on duplicates as when a string is created Java checks if that exact string is present in String Pool and gives reference to the one that's already there, Unless created using new keyword as it stores in heap.

Q. What are Wrapper classes and why needed in Java?

— Primitive → byte (8), Short (16), int (32), long (64), float (32), double (64), char (16), boolean

- Wrapper class converts primitive data type to Objects
- Wrapper → Byte, Short, Integer, Long, Float, Double, Character, Boolean
- Allows them to store null values unlike primitive
- Use methods like .parseInt(), .valueOf(), .toString()
- Works with collections like ArrayList<>() as it only stores objects not primitive
- int ⇄ Integer (Autoboxing: int → Integer; Unboxing: Integer → int)

Q. Scenario based question on collections in Java?

— ArrayList → dynamic array that grows as we add elements

HashSets → stores unique elements, no duplicates

HashMap → used for mapping just like dictionary key : values

Using collections helps managing large amount of complex data with built in functions like adding, removing, searching.

Q. Explain use of "this" and "super" keywords in Java?

— "this" → refers to current object or class pointing to the object that is calling method or constructor.

"super" → refers to parent class / method / constructor used to access method or variable from parent class

Q. What is different between static and instance method?

— static → method that belongs to class itself and can be called without creating object of class

→ just call using ClassName.methodName()

→ only access other static variables or methods and cannot use this keyword as they donot belong to any object.

instance → method that belongs to object of a class
→ create object first then call the method on that object
→ can access both static and instance variable and can use this keyword.

Q. What are constructors and their types ?

Special method used to initialize object in Java
- Same Name as class
- No Return Type
- Automatic call when object is created using new keyword

Types are
→ Default : takes no arguments
initialize object with default value

→ Parameterized : takes parameters to initialize object fields

```
class Bike{
  Bike(){
    S.o.p("created");
  }
  public static void main(String[]args){
    Bike b1 = new Bike();
  }
}
```

```
class Car{
  Car(String m, int y){
    model = m;
    year = y;
  }
  public static void main(String[]args){
    Car c1 = new Car("Tesla", 2022);
  }
}
```

Q. Difference between StringBuffer & StringBuilder?

— StringBuffer → synchronized, two threads can't call method of StringBuffer simultaneously just like mutex

StringBuilder → non synchronized, multiple threads can call methods of StringBuilder simultaneously

Q. Difference between Abstract & Interfaces?

— Both allow us to define methods & subclass must implement

Abstract class → "extends"

- can have both abstract & non abstract methods
- supports single inheritance
- can have various types of variables (final, static or non static)
- can extend other classes & implement interfaces & provide implementations

Interface → "implements"

- can only have abstract methods
- supports multiple inheritance
- variables are public, static, final by default
- can only extend other interfaces.
- cannot provide implementation

Q. What is method overloading & can we overload Main() method?

— Allows multiple methods with same name : Methods must differ in number of type of parameters. (Compile time Polymorphism (Static)).

We can overload main() method but JVM will only call original (String[] args) main method.

Q. What is Method Overriding?

– occurs when subclass provides a specific implementation of a method already defined in its superclass.
But method signature (name, parameters) should be same.
Run time Polymorphism (Dynamic)
We can't override static or private method.

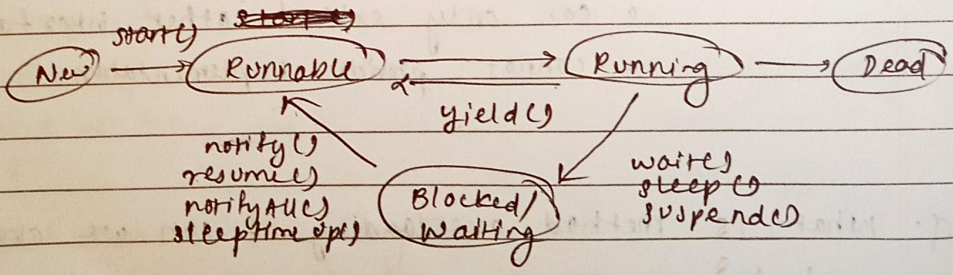
Q. How is Exception Handling done in Java?

–

Try block → contains code that might throw exception
Catch block → captures and handles specific exceptions
multiple catch blocks can be used for different exception types
Finally block → always executes regardless of exception there or not.

Q. What do you mean by life cycle of thread in Java?

–



Q. What is Singleton class & how do we create it?

Design pattern that ensures a class has only one instance throughout the program and provide global access point to that instance.

→ Private constructor - prevents direct instantiation prevent other classes from creating new instance

→ Static method (getInstance()) - provides global access to single instance

→ Static instance variable - stores single instance as static variable and shared across all calls to instance

Q. Difference between Aggregation & Composition in Java?

Aggregation → "Has a"
objects can exist independently
child objects can outlive parent object
used when objects have their own lifecycle

Composition → "part of"
child object depends on parent object
child objects are destroyed with parent
used when objects are strongly bound together

Q. What is Anonymous Inner class?

Class without a name that is defined and instantiated all at once, typically used to override method or add extra behaviour and only one object of that class is created.

Used using abstract or interface

→ interface Age {
    void getAge();
}

public static void main (String [] args){
    Age obj = new Age() {
        public void getAge(){
            SOP ("Age");
        };
    obj.getAge();
}

} → Anonymous Inner Class

19. Difference between Implicit & Explicit Type Conversion.

—

Implicit → Type coercion
automatic conversion of smaller datatype to larger data type by compiler
→ int num = 10; double bignum = num;

Explicit → programmer manually converts a variable from one data type to another
→ double bignum = 10.5;
int num = (int) bignum;

Q. Explain purpose of volatile keyword in Java?

— Ensure that changes to variable are immediately visible to all threads, preventing threads from caching the variables from their local cache. It tells JVM that variables value will always be read from and written to main memory.

Q. Difference between system.out, system.err and system.in ?

System.out → Standard output for regular messages
System.out.println()
normal output
console (eg. black)

System.err → Standard output for error messages
System.err.println()
Error or diagnostic output
Console (eg. red)

System.in → Standard input for reading user input (System.in) used with Scanner or InputStream
Input stream
Reads from keyboard

Q. ~~What~~ are Access Specifier & their purpose?

~~Public~~	Same Class	Same Package	Subclass (diff pkg)	Everywhere
Public	✓	✓	✓	✓
Protected	✓	✓	✓	X
default	✓	✓	X	X
private	✓	X	X	X

Q. Difference between final, Finally, Finalize keyword?

Final → restrict user from changing the value of a variable from extending a class or overriding a method.

Finally → used with try catch block and is executed regardless of exception thrown or not.

Finalize → special method provided by object class that is called by garbage collector before reclaiming an object memory. sc. finalize();

Q. Difference between deep copy & shallow copy?

Deep Copy → creates new object and copies all fields ensuring changes made to one object doesn't affect other
- new memory is allocated, each object has its own independ reference

Shallow Copy → copy the reference of an object, not actual object
- changes made to one reference affect the other
- no new memory is allocated, both references point to same memory location.

Q. What is OOP and why is it important?

Programming paradigm that organizes software design around objects rather than functions or logic.
→ Modularity : smaller modules (broaked into)
→ Code Reusability : (inheritance)
→ Data security : (encapsulation)
→ Scalability
→ Easier maintenance

Q. What are class & objects?

Class → blueprint for creating objects
defines data (attributes) & methods (functions)
Ex: class Dog {...}
Object → instance of a class
real world entity with properties & behavior
Ex: Dog d1;

Q. What are four pillars of OOPs?

→ Encapsulation — bundles data & methods
private field, accessed via getters and setters
→ Abstraction — hides complexity, showing only essential features
→ Inheritance — one class inherits properties from another
"extends"
→ Polymorphism - same method behaves differently for different objects.

Q. How to call one constructor from another?

- Using constructor chaining.
- constructor chaining is process of calling one constructor from another within the same class (using this() keyword) or from base class (using super() keyword)
- To reuse code and avoid duplication when initializing an object.

```
class Student {
    int id
    String name;
    Student() {
        this (1, "Riya");
        SOP ("Default");
    }
    Student (int id, String name){
        this.id = id;
        this.name = name;
        SOP ("Parameterized");
    }
}
```

→ Output:
Parameterized
Default

```
class Person {
    Person () {
        SOP ("Person");
    }
}
class Student extends Person{
    Student() {
        super();
        SOP ("Student");
    }
}
```

→ Output:
Person
Student

Q. What are Manipulators in Java?

Methods or functions that modify the state or behavior of object via setters or other class methods.

Q. What do you mean by abstract class & can we intantiate the abstract class?

- Abstract class is blueprint for other class
- cannot create objects directly
- can contain both abstract and regular methods.
- provide basic structure for subclasses to build on
- ensure subclasses implement required methods.

Q. What is interface in Java and how does it promote abstraction?

Interface is a reference type that contains only abstract methods (without body) and constant variables.
It provides a blueprint for classes to follow without implementing any specific behavior

Q. Exception Handling in Java?

Types : User defined Exception
Built in Exception

Built in Exception types:

→ Checked Exception
- class not found
- IO
- Interrupted
- Instantiation
- SQL
- File not found

→ Unchecked Exception
- Arithmetic
- class cast
- NULL Pointer
- Array Store
- Array Index Out of Bounds
- Illegal thread State

Q. Explain types of inheritance?

→ Single - One child class inherits from one parent class
```
- class Animal {...}
  class Dog extends Animal {...}
```

→ Multilevel - class derived from a class which is already derived from another class
```
- class Animal {...}
  class Dog extends Animal {...}
  class Puppy extends Dog {...}
```

→ Hierarchical - multiple child classes inherit from one parent class
```
- class Animal {...}
  class Dog extends Animal {...}
  class Cat extends Aimal {...}
```

Q. Explain virtual keyword?

When a base class (eg. A) is inherited twice in a hierarchy (via B & C to D), ambiguity arises in D.
To prevent duplication and ensure only one instance of A in D, use virtual with A in B and C.

```
class A {...}
class B : virtual public A {...}
class C : virtual public A {...}
class D : public B, pubic C {...}
```

Q. What is difference between DBMS & RDBMS?

Database Management System →
- Software to store, retrieve, update and manage data in database
- Data stored as files (single files)
- No connection between files (data ✗ → relationship)
- Less data security
- Supports single user mostly

Relational Database Management System →
- Type of DBMS
- Data stored in the form of tables (relations) with row and columns
- Has relationships between tables (using keys)
- High data security & integrity
- Support multiple users

Q. Keys in DBMS?

~~Primary~~ →

Super → uniquely identifies records
may contain extra attributes
a table can have many super keys

Candidate → minimal super key (eligible for primary)
no extra attribute
a table can have multiple candidate

Primary → cannot be null
one candidate key selected to identify record
only one primary key per table

Alternate → candidate keys not selected as primary key

Foreign → attribute in one table that refers to primary key of another table

Composite → key made of more than one attribute

Unique → unique values, allow null value (one)

Surrogate → artificially created
usually auto incremented

Natural → key formed from real world data
AdharNo., EmailId

Q. What are constraints and their types?

Rules you set up for data in tables.
Helps in keeping data accurate and reliable

- → Not Null
- → UNIQUE
- → CHECK
  - restrict invalid data entry to be stored by applying condition
- → PRIMARY KEY
- → FOREIGN Key
- → DEFAULT
  - prevent null values by automatically inserts a predefined value

Q. Explain DDL and DML commands ?

Data Definition Language : structure of database
- → Create
- → Drop
- → Alter

(related to table)

Data Manipulation Language : structure data
- → Insert
- → Update
- → Delete

(related to value)

Delete → removes specific rows from table
Truncate → removes all rows from table
Drop → deletes entire table including its structure

Q. Keywords in SQL ?

Group by → group rows that have same values and apply aggregate functions COUNT(), SUM(), AVG(), MIN(), MAX()
Order by → sort result set (rows) by ASC | DESC
Where → filters rows before grouping works on individual rows
Having → filters groups after group by works on grouped data.

Q. Difference between Union & Union all in SQL?

Union → combines both lists into one remove any duplicate
Union all → combines two lists but keeps every attribute even if they show up more than once.

Q. What are views in SQL?

Virtual table that doesn't store data itself, only the query is stored but allows you to see data from one or more table in a specific way.

Q. What are Triggers in SQL?

They are like reflex actions which allows you to set up an automatic action everytime a certain event happens in database
Adding, Updating, Deleting

Ex:
```
CREATE TRIGGER before_insert_emp
BEFORE INSERT ON Employee
FOR EACH ROW
BEGIN
    SET NEW.Salary = NEW.Salary + 100;
END;
```

Types : Before, After, Instead OF

Q. Explain Indexing in SQL and clustered Index?

Indexing is technique used to speed up data retrieval from a table by creating datastructure on one or more columns.

→ `CREATE INDEX idx ON Employee(Name);`

Clustered Index

It determines the physical order of data in table where table data is stored in sorted order
only one clustered index per table

→ `CREATE CLUSTERED INDEX idx ON Employee(Name);`

Q. Explain Normalization and their types?

Prows of organizing data

→ Reduce data redundancy

→ Avoid data anomalies

Types (Normal forms)

→ 1NF : Each table cell should contain single value and each column must have unique name

→ 2NF : All non key attributes must depend on the primary key

→ 3NF : Every non key attribute must be independent of other non-key attributes

→ BCNF : Boyce Codd Normal Form
Every determinant (an attribute that can determine other attributes) must be a candidate key.

Imp Notes

- API Rate Limiting
  Restricting the number of requests that user or client can make to an API in a given time period to prevent abuse

- Caching
  Storing frequently accessed data in memory to reduce latency and load on backend systems (eg. Redis)

- Database Sharding
  Splitting a database into a smaller pieces (shards) to distribute the load and improve performance.

- Load Balancing
  Distributing traffic across multiple servers to avoid overloading a single server and to improve performance.

- Replication
  Copying data across multiple servers or locations to ensure redundancy, reliability, and availability.

- Scalability
  the ability of a system to handle growth, either by scaling vertically (adding resources to a single server) or horizontally (adding more servers).

- Authentication & Authorization
  verifying user identity (authentication) and controlling access (authorization), often using OAuth / JWT.

- Consistency vs. Availability (CAP Theorem)
  A trade off between consistency (all nodes see the same data) and availability (the system remains operational).

- Data Integrity
  Ensuring that data remains accurate and consistent across the system, even during failures.

- Event Sourcing
  Storing state changes as a series of events, allowing the system to rebuild state from those events.

- Event Driven Architecture
  Systems that react to events or messages asynchronously, improving scalability and responsiveness.

- Fault Tolerance
  The ability of a system to continue functioning despite partial failures (eg. failover mechanisms).

- Message Queues
  Using queues (eg. Kafka, RabbitMQ) for asynchronous communication between components in a system.

- Microservices Architecture
  Breaking down a system into smaller, independent services that communicate over a network.

- Rate Limiting & Throttling
  Controlling the flow of requests to prevent overloading a system or service.

- Service Discovery
  Automatically finding the location and availability of services within a distributed system.

- Asynchronous Processing
  Handling tasks asynchronously to avoid blocking processes and improve performance.

- Circuit Breaker Pattern
  Preventing repeated failure of operations by stopping execution when a failure is detected.

- Containerization (eg. Docker)
  Packaging applications with their dependencies into containers for consistent deployment.

- Content Delivery Network (CDN)
  A system of distributed servers that deliver content to users from locations closest to them.

- Database Indexing
  Creating indexes on columns of a database to improve search and retrieval times.

- Distributed Transactions
  Ensuring consistency across distributed systems, often using two phase commits or Saga patterns

- Horizontal vs Vertical Scaling
  Horizontal scaling adds more machines, vertical scaling adds more power to existing machines.

- Read / Write splitting
  Separating read operations from write operations to improve database performance.

- Sharding strategies
  Techniques for dividing data across multiple databases or servers to improve scalability and performance.

- Transactional System & ACID Properties
  Ensuring reliability and consistency through Atomicity, Consistency, Isolation, and Durability.

- Big Data & Analytics
  Managing and processing large volumes of data (eg. using Hadoop, Spark) for analysis and insights.

- Data Partitioning
  Dividing data into smaller, manageable partitions for optimization and performance improvement

- Distributed Systems & Fault Isolation
  Architecting systems in a way that allows for faults to be isolated without bringing down the entire systems.

- Real time Systems
  Systems that respond to inputs or events within a very short time frame (eg. messaging, trading systems).

- Versioning (API & Data)
  Handling changes to APIs and data formats in a way that maintains compatibility with existing systems.

- Video Streaming (Adaptive Bitrate)
  Streaming video content with dynamic adjustment to the available bandwidth and device capabilities.

- Blockchain & Decentralized Systems
  Systems built on blockchain principles for decentralized and secure transactions.

- Distributed Caching
  Caching data across multiple servers to improve performance and availability in large systems.

- Global Distribution & Multi Region Architecture
  Architecting systems to ensure global availability and low latency by deploying across multiple regions.

- Video Encoding & Transcoding
  Converting video content into different formats and quality levels for playback across various devices.