Q1. History of java?

Ans=Java was developed by James Gosling at Sun Microsystems in 1991 and officially released in 1995. It introduced platform independence with its "Write Once, Run Anywhere" (WORA) capability, supported by the Java Virtual Machine (JVM). Now maintained by Oracle, Java is widely used for web, enterprise, and mobile applications.

Q2. Features of Java.

Ans= Features of Java is:

1 Platform Independent: Code runs on any platform using the Java Virtual Machine (JVM).

2 Object-Oriented: Focuses on objects and classes, supporting principles like inheritance, polymorphism, and encapsulation.

3 Simple: Easy to learn with a syntax similar to C++ but without complexities like pointers.

4 Secure: Offers runtime checks and eliminates vulnerabilities like explicit memory handling.

5 Portable: Code is architecture-neutral and can run on different systems without modification.

6 Robust: Features strong memory management, exception handling, and garbage collection.

7 Multithreaded: Supports concurrent execution of programs, improving performance.

8 Dynamic: Supports dynamic linking and runtime flexibility.

Q3. Understanding JVM, JRE, and JDK.

Ans=JVM (Java Virtual Machine): Executes Java bytecode, enabling platform independence by converting it to machine code at runtime.

JRE (Java Runtime Environment): Provides libraries, JVM, and other components needed to run Java applications. It does not include development tools.

JDK (Java Development Kit): A complete development package that includes JRE, development tools (e.g., compiler, debugger), and libraries for writing and building Java programs.

Q4. Setting up the Java environment and IDE (e.g., Eclipse, IntelliJ).

Ans=Setup:

Step=1 Go to "Environment Variables" in System Settings.

Step=2 Add the JDK bin folder in path.

Step=3 Download Eclipse.

Step=4 Install Eclipse.

Step=5 Start a new Java project.

Q5. Java Program Structure (Packages, Classes, Methods).

Ans=

package core; // Package declaration

public class MyClass { // Class declaration

public static void main(String[] args) { // Main method

System.out.println("Hello, World!"); // Print statement

}

}

Q1. Install JDK and set up environment variables?

Ans=

Step.1: Download JDK

Step.2: Install JDK

Step.3: Set Environment Variables

* Open System Properties → Environment Variables →Add Path.

Step.4: Open Command Prompt and run: using javac -version.

Step.5: Create and run a simple Java program.

Create and run a simple Java program

Q1. Primitive Data Types in Java?

Ans=Primitive Data type is 8 in Java.

* byte: 1 byte, range -128 to 127
* short: 2 bytes, range -32,768 to 32,767
* int: 4 bytes, range -2,147,483,648 to 2,147,483,647
* long: 8 bytes, range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
* float: 4 bytes, precision ~6-7 digits
* double: 8 bytes, precision ~15 digits
* char: 2 bytes, Unicode range 0 to 65,535
* boolean: 1 bit, values true or false

Q2. Variable Declaration and Initialization?

Ans=In Java, variables are declared to store data with a specified type.

1. Declaration: Specifies the data type and variable name.
2. Initialization: Assigns a value to the variable.

Q3. Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise?

Ans=Operators in Java

1. Arithmetic Operators: Perform basic mathematical operations like addition, subtraction, multiplication, division, and modulus.
2. Relational Operators: Compare two values and return a boolean result (e.g., greater than, less than, equals).
3. Logical Operators: Combine multiple boolean expressions (AND, OR, NOT).
4. Assignment Operators: Assign values to variables, including combined operations like += and -=.
5. Unary Operators: Operate on a single operand (e.g., increment, decrement, negation).
6. Bitwise Operators: Perform operations at the bit level (AND, OR, XOR, complement, shifts).

Q4. Type Conversion and Type Casting?

Ans=Type Conversion and Type Casting in Java

1. Type Conversion:
   * Implicit Conversion (Widening): Automatic conversion from a smaller to a larger data type.
   * Explicit Conversion (Narrowing): Manual conversion from a larger to a smaller data type using casting.
2. Type Casting:
   * Widening: Automatically done by Java (smaller to larger types).
   * Narrowing: Requires explicit casting (larger to smaller types)

Q1. If-Else Statements?

And=If-Else Statements in Java

1. If Statement: Executes a block of code if the specified condition is true.
2. If-Else Statement: Executes one block of code if the condition is true and another block if the condition is false.
3. If-Else: If-Else Statement Allows multiple conditions to be checked in sequence, executing a block of code based on the first true condition.
4. Nested If-Else: An if-else statement inside another if or else block, used to check multiple levels of conditions

Q2. Switch Case Statements?

Ans=Switch-Case Statements in Java

1. Switch Statement: Evaluates an expression and matches it with a case value to execute the corresponding block of code.
2. Case: Defines a block of code for each possible value of the expression.
3. Default: An optional block executed if no case matches the expression.
4. Break: Exits the switch statement to prevent "fall through" to the next case.

Q3.Loop?

Ans=Loops in Java

1. For Loop: Repeats a block of code a specific number of times with initialization, condition, and update expressions.
2. While Loop: Executes a block of code as long as a condition is true, with the condition checked before each iteration.
3. Do-While Loop: Executes a block of code at least once, with the condition checked after each iteration.

Q4. Break and Continue Keywords?

Ans=. Break and Continue:

1. Break: Exits from the current loop or switch statement, stopping further iterations or cases.

2. Continue: Skips the current iteration of a loop and proceeds with the next iteration.

Q1. Defining a Class and Object in Java?

Ans=

1.Class: Class is a structure in with we can have member function and member variable are there.

2.Object: Object is instance of class using which we can excises properties of class.

->Object have state (data) and behavers.

Q2. Constructors and Overloading?

Ans= 1. Constructors ->It is special member function of class with have same name as class

->It have no return type.

-> It well call automatically when object is created.

-> There are three type of construction:

a. default

b. Parameters

c. Copy

2. Overloading ->Same function name with different parameters into same class.

Q3. Object Creation, Accessing Members of the Class?

Ans=Using NEW keyword we can create object of class

Syntex is -> class name object =NEW constructer();

->For accessing members of class we use (.)operator with object name.

Syntex-> object . member of class();

Q4. this Keyword?

Ans=To refer current class property in same class we use THIS key word.

Q1. Defining Methods?

Ans= In Java, methods are blocks of code that perform specific tasks and can be executed when called. Methods are defined within a class and can take input (parameters), perform actions, and optionally return a value.

Q2. Method Parameters and Return Types?

Ans= Method Parameters:The inputs specified in a method's signature, which define the data a method can accept when it is called.

Return Type:The data type of the value a method returns to the caller. If no value is returned, the return type is void.

Q3. Method Overloading?

Ans= Method Overloading: In Java, method overloading is a feature that allows multiple methods to have the same name but different parameter lists.

-> Methods must have the same name.

-> Methods must have different parameter lists:

1. Different number of parameters.
2. Different types of parameters.
3. Different order of parameters (if types differ).

Q4. Static Methods and Variables?

Ans=Static Methods: IF function is static we can call it direct by the class, and for static method we use static key word.

-> Static Variable: To same memory allocate we use static variable.

Q1. Basics of OOP: Encapsulation, Inheritance, Polymorphism, Abstraction?

Ans=OOPS: Object-Oriented Programming (OOP) is a programming method that organizes code into units called objects. It helps in creating modular, reusable, and maintainable code by modeling real-world entities.

->Inheritance: To access property of one class to another class is called inheritance.

->Polymorphism: Same function name but having different functionality that is polymorphism.

->Abstraction: Hiding internal details and showing essential information to user.

a. For using abstract class we use abstract keyword.

b. It can not create object of abstract class.

c. If function id=s abstract than class also should be abstract.

->Encapsulation: Reaping data in single unit is called encapsulation, and in encapsulation property should be private. Use To String function in class to get data by object.

Q2. Inheritance: Single, Multilevel, Hierarchical?

Ans=Inheritance:

1. Single: In Single Inheritance, a child class inherits from a single parent class.
2. Multilevel: In Multilevel Inheritance, a class inherits from another class, and that class in turn inherits from another class.
3. Hierarchical: In Hierarchical Inheritance, multiple child classes inherit from a single parent class.

Q3. Method Overriding and Dynamic Method Dispatch?

Ans=Method Overriding: Same function name, same parameter, same return type into different class and inheritance is compulsory.

Dynamic Method: Also known as runtime polymorphism, it is the process by which a call to an overridden method is resolved at runtime rather than compile-time.

Q1. Constructor Types (Default, Parameterized)?

Ans=Constructor is special member function of class which has same name as class name.

1. Default Constructor: default constructor is a constructor that does not take any parameters.
2. Parameterized Costructor: parameterized constructor is a constructor that takes arguments to initialize an object with specific values.

Q2. Copy Constructor (Emulated in Java)?

Ans= A copy constructor is a special type of constructor that creates a new object by copying the values from an existing object of the same class. It is used to create a duplicate of an object.

In Java, you need to manually create a constructor that copies the fields from one object to another.

Q3. Constructor Overloading?

Ans= In Constructor overloading multiple constructors with the same name exist, but with different parameters. It allows you to create objects in different ways by providing different sets of arguments

Q4. Object Life Cycle and Garbage Collection?

Ans= Object Life Cycle in Java:

1. Creation: An object is created using the new keyword and initialized by a constructor.
2. Use: The object is used in the program (methods called, values modified).
3. Reachability: An object is reachable as long as there are references to it.
4. Destruction: When no references point to the object, it becomes eligible for garbage collection.

Garbage Collection in Java:

* Garbage collection is the automatic process of reclaiming memory used by unreachable objects.
* The garbage collector identifies and removes objects that are no longer in use, freeing memory.
* The process is automatic, but System.gc () can suggest garbage collection, though it's not guaranteed

Q1. One-Dimensional and Multidimensional Arrays?

Ans= One-Dimensional Arrays

A one-dimensional array is a list of elements of the same type, stored in contiguous memory locations. You can think of it as a single row or a sequence of elements

Multidimensional Arrays

A multidimensional array is an array of arrays, where each element can be another array. The most common form is the two-dimensional array (a matrix), but arrays can have more than two dimensions

Q2. String Handling in Java: String Class, StringBuffer, StringBuilder?

1. Ans= String Class:
   * Immutable. Once created, its value cannot be changed.
   * Suitable for constant or unchanging strings.
   * Example: String str = "Hello";
2. StringBuffer Class:
   * Mutable and thread-safe.
   * Used when you need to modify a string frequently in multi-threaded environments.
   * Example: StringBuffer sb = new StringBuffer("Hello"); sb. append (" World");
3. StringBuilder Class:
   * Mutable but not thread-safe (faster than StringBuffer).
   * Used for frequent string modifications in single-threaded environments.
   * Example: StringBuilder sb = new StringBuilder("Hello"); sb. append (" World");

Q3. Array of Objects?

Ans= An array of objects is a collection of references to objects of a particular class. Instead of storing primitive values like integers or characters, it stores references (memory addresses) to objects.

* Creation: You can create an array of objects by defining the array with a specific class type.
* Storage: The array holds references to objects, not the objects themselves.
* Access: Each element in the array can be used to access the corresponding object and its methods or attributes.

Q4. String Methods (length, charAt, substring, etc.)?

Ans= String Methods in Java:

1. length(): Returns the length of the string.
2. charAt(int index): Returns the character at the specified index.
3. substring(int start): Returns a substring starting from the given index.
4. substring(int start, int end): Returns a substring from the start index to the end index.
5. toLowerCase(): Converts the string to lowercase.
6. toUpperCase(): Converts the string to uppercase.
7. equals(Object obj): Compares two strings for equality (case-sensitive).
8. equalsIgnoreCase(String str): Compares two strings for equality (case-insensitive).
9. indexOf(int ch): Returns the index of the first occurrence of a character.
10. replace(char oldChar, char newChar): Replaces all occurrences of a character with a new character.
11. trim(): Removes leading and trailing spaces from the string

Q1. Inheritance Types and Benefits?

Ans=Types of Inheritance in Java

1. Single Inheritance: A class inherits from one parent class.
2. Multilevel Inheritance: A class inherits from a class that is itself derived from another class.
3. Hierarchical Inheritance: Multiple classes inherit from a single parent class

Benefits of Inheritance

1. Code Reusability: Reuse common code in multiple classes.
2. Extensibility: Add new features without modifying existing code.
3. Maintainability: Centralized code updates are reflected in derived classes.
4. Polymorphism: Supports method overriding for flexible behavior.
5. Logical Hierarchy: Models real-world relationships between entities.

Q2. Method Overriding?

Ans=When a subclass provides its own implementation of a method already defined in its parent class, using the same method name, parameters, and return type.

Q3. Dynamic Binding (Run-Time Polymorphism)?

Ans=It is the process of determining which method to invoke at runtime, based on the actual object type rather than the reference type. This allows different behaviors for the same method call, enabling flexibility and polymorphism in Java.

Q4. Super Keyword and Method Hiding?

Ans=Super Keyword:

* Definition: Refers to the parent class and is used to access parent class methods, constructors, and variables.
* Usage:
  1. super. Method() – Calls the parent class method.
  2. super() – Calls the parent class constructor.
  3. super. Variable – Accesses parent class variables.

Method Hiding:

* Definition: In Java, if a subclass defines a static method with the same name as a static method in the parent class, the parent method is "hidden" (not overridden).
* Key Point: The method call is determined by the reference type, not the object type, because static methods are resolved at compile time.

Q1. Abstract Classes and Methods?

Ans= Abstract Class:  
A class that cannot be instantiated and is meant to be inherited by other classes. It can have abstract methods (without implementation) and concrete methods (with implementation).

 Abstract Method:  
A method declared in an abstract class without a body. It must be implemented by any subclass that is not abstract.

Q2. Interfaces: Multiple Inheritance in Java?

Ans= Interface:  
An interface is a reference type, similar to a class, that can contain only method declarations (no implementation) and constants. Classes implement interfaces to provide method definitions.

 Multiple Inheritance:  
Java does not support multiple inheritance with classes to avoid ambiguity. However, Java allows multiple inheritance through interfaces. A class can implement multiple interfaces, thus inheriting the abstract methods of all the interfaces

Q3. Implementing Multiple Interfaces?

Ans=interface Interface1 {

Public void fun();

}

interface Interface2 {

Public void fun1();

}

class inter implements Interface1, Interface2 {

public void fun() {

}

public void fun1() {

}

}

Q1. Java Packages: Built-in and User-Defined Packages?

Ans=A package in Java is a collection of related classes and interfaces used to organize code, prevent naming conflicts, and improve reusability.

Built-in Packages ->Java provides predefined packages for common tasks:

* java.lang – Core classes (default)
* java.util – Collections and utilities
* java.io – File handling
* java.net – Networking
* java.sql – Database connectivity

User-Defined Packages

*  Created by developers to organize custom classes.
*  Declared using the package keyword.
*  Accessed using the import statement.

Q2. Access Modifiers: Private, Default, Protected, Public?

Ans=Access Modifiers:

Private: A modifier that makes a member accessible only within the same class.

Default: A modifier (when no access modifier is specified) that makes a member accessible

within the same package.

Public: A modifier that makes a member accessible from anywhere in the program.

Protected: A modifier that makes a member accessible within the same package and in

subclasses outside the package.

Q3. Importing Packages and Classpath?

Ans= Importing allows access to classes from different packages in Java.

import java.util.ArrayList;

-> Classpath is the path where Java searches for classes and packages.

java -cp /path/to/classes MyProgram.

Q1. Types of Exceptions: Checked and Unchecked?

Ans= 1. Checked Exceptions

* Exceptions that are checked at compile-time.
* Must be handled using try-catch or declared with throws.
* Examples:
  + IOException (file not found)
  + SQLException (database error)
  + ClassNotFoundException (missing class)

2. Unchecked Exceptions

* Exceptions that occur at runtime and are not checked at compile-time.
* Usually caused by programming errors.
* Examples:
  + NullPointerException (accessing null object)
  + ArrayIndexOutOfBoundsException (invalid array index)
  + ArithmeticException (division by zero)

Q2. try, catch, finally, throw, throws?

Ans=Exception Handled:

try – Defines a block where exceptions may occur. It must be followed by either a catch or finally block.

catch – Handles the exception thrown in the try block. It specifies the type of exception to be caught.

finally – A block that always executes after try and catch, used for resource cleanup.

throw – Used to manually throw an exception inside a method or block.

throws – Declares exceptions that a method might throw, allowing the caller to handle them.

Q3. Custom Exception Classes?

Ans= A custom exception is a user-defined exception that extends the Exception or RuntimeException class. It is used when built-in exceptions are not sufficient for specific error handling.

1. Extending Exception (Checked Exception) – Must be handled using try-catch or throws.
2. Extending RuntimeException (Unchecked Exception) – Does not require explicit handling

Q1. Introduction to Threads?

Ans= A thread is the smallest unit of execution in a program, allowing multiple tasks to run simultaneously. Java provides built-in support for multithreading using the Thread class and Runnable interface.

Q2. Creating Threads by Extending Thread Class or Implementing Runnable Interface?

Ans= Threads in Java can be created in two ways:

1. Extending the Thread Class  
   A class can inherit from the Thread class and override its run() method. The thread is started using the start() method.
2. Implementing the Runnable Interface  
   A class can implement Runnable and define the run() method. The Thread class is then used to execute the Runnable object.

Q3. Thread Life Cycle?

Ans= A thread in Java goes through several states during its execution:

1. New – The thread is created but not yet started.
2. Runnable – The thread is ready to run but waiting for CPU time.
3. Running – The thread is currently executing.
4. Blocked/Waiting – The thread is paused, waiting for resources or a signal.
5. Terminated – The thread has finished execution or was stopped.

Q4. Synchronization and Inter-thread Communication?

Ans= Synchronization ensures that multiple threads do not access shared resources simultaneously, preventing data inconsistency. It is achieved using the synchronized keyword on methods or blocks.

Inter-thread Communication allows threads to communicate and coordinate execution. It is implemented using wait(), notify(), and notifyAll() methods within synchronized blocks.

Synchronization maintains data integrity, while inter-thread communication enables efficient task execution in multithreading.

Q1. Introduction to File I/O in Java (java.io package)?

Ans= File I/O (Input/Output) in Java allows reading and writing data to files. Java provides the java.io package, which includes classes like File, FileReader, FileWriter, FileInputStream, and FileOutputStream for handling file operations.

Q2. FileReader and FileWriter Classes?

Ans= FileReader and FileWriter are used for reading and writing character data in Java. They are part of the java.io package and work with text files.

* FileReader – Reads character data from a file.
* FileWriter – Writes character data to a file

Q3. BufferedReader and BufferedWriter?

Ans= BufferedReader and BufferedWriter are used for efficient reading and writing of character data. They are part of the java.io package and work with text files by using buffers to improve performance.

* BufferedReader – Reads text efficiently using a buffer and provides methods like readLine().
* BufferedWriter – Writes text efficiently by buffering data before writing to a file

Q4. Serialization and Deserialization?

Ans= Serialization is the process of converting an object into a byte stream so it can be saved to a file or transferred over a network. It is done using the ObjectOutputStream class.

Deserialization is the process of converting a byte stream back into an object. It is done using the ObjectInputStream class

Q1. Introduction to Collections Framework?

Ans= The Collections Framework in Java is a unified architecture for storing, manipulating, and processing groups of objects. It is part of the java.util package and provides various data structures and algorithms

1. List (Ordered, Allows Duplicates)

* ArrayList
* LinkedList
* Vector

2. Set (Unique Elements, No Duplicates)

* HashSet
* LinkedHashSet
* TreeSet

3. Map (Key-Value Pairs)

* HashMap
* LinkedHashMap
* TreeMap
* Hashtable

4. Queue (FIFO Order)

* PriorityQueue
* LinkedList

Q2. List, Set, Map, and Queue Interfaces?

Ans= List – Ordered, allows duplicates (ArrayList, LinkedList, Vector).

Set – Unique elements, no duplicates (HashSet, LinkedHashSet, TreeSet).

Map – Key-value pairs, unique keys (HashMap, LinkedHashMap, TreeMap, Hashtable).

Queue – FIFO order, processes elements sequentially (PriorityQueue, LinkedList).

Q3. ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap?

Ans= ArrayList – Dynamic array, fast random access, slower insert/delete (O(1) access).

LinkedList – Doubly linked list, fast insert/delete, slower access (O(n) access).

HashSet – Unordered set, no duplicates, uses hashing (O(1) operations).

TreeSet – Sorted set, no duplicates, uses Red-Black Tree (O(log n) operations).

HashMap – Unordered key-value store, fast lookup (O(1) operations).

TreeMap – Sorted key-value store, maintains ascending order (O(log n) operations).

Q4. Iterators and ListIterators?

Ans= Iterator: Used to traverse elements in Collection (e.g., List, Set). Supports next(), hasNext(), and remove().

ListIterator: Used to traverse List in both forward and backward directions. Supports next(), hasNext(), previous(), hasPrevious(), add(), and set().

Q1. Streams in Java (InputStream, OutputStream)?

Ans= Streams in Java are used for handling input and output operations in a sequential flow of data.

* InputStream – Abstract class for reading byte data (FileInputStream, BufferedInputStream).
* OutputStream – Abstract class for writing byte data (FileOutputStream, BufferedOutputStream)

Q2. Reading and Writing Data Using Streams?

Ans= Reading Data: Done using InputStream (e.g., FileInputStream, BufferedInputStream). Reads byte-by-byte or in chunks.

Writing Data: Done using OutputStream (e.g., FileOutputStream, BufferedOutputStream). Writes byte data to files or other destinations

Q3. Handling File I/O Operations?

Ans= File I/O in Java is managed using the java.io package, which provides classes for reading and writing data.

* Reading Files: FileReader, BufferedReader, FileInputStream.
* Writing Files: FileWriter, BufferedWriter, FileOutputStream.
* Managing Files: File class (create, delete, check existence).