CORE JAVA

INTERVIEW

QUESTIONS

- **What is Java?**

  Java is an object-oriented, high-level programming language developed by Sun Microsystems (now Oracle). It is platform-independent due to its "write once, run anywhere" nature, enabled by the Java Virtual Machine (JVM).

- **What are the main features of Java?**
  - Object-Oriented: Java follows object-oriented principles like inheritance, encapsulation, polymorphism, and abstraction, which makes code reusable, modular, and easier to maintain.
  - Platform-Independent: Java’s "write once, run anywhere" capability allows Java code to run on any platform with a compatible JVM, making it highly portable.
  - Simple and Familiar: Java is designed to be easy to learn, especially for those familiar with C or C++, with features like automatic memory management and a simple syntax.
  - Secure: Java provides a secure environment through features like bytecode verification, sandboxing, and the Security Manager, protecting against threats like viruses and tampering.

- **What is the difference between JVM, JRE, and JDK?**
    1. JVM (Java Virtual Machine):
        - Purpose: Executes Java bytecode, making Java platform-independent.
        - Role: Runs the compiled Java program by converting bytecode into machine code.
        - Availability: Part of the JRE and JDK.
    2. JRE (Java Runtime Environment):
        - Purpose: Provides the environment to run Java applications.
        - Components: Includes the JVM along with libraries and other components required for execution.
        - Role: Used to run, but not develop, Java applications.
    3. JDK (Java Development Kit):
        - Purpose: A complete package for developing and running Java applications.
        - Components: Contains the JRE, JVM, and additional development tools like the compiler (javac), debugger, and other utilities.
        - Role: Needed for both development and execution of Java applications.

- **What is bytecode in Java?**
  In Java, bytecode is the intermediate representation of a Java program. When a Java source file (.java) is compiled, it is transformed by the Java compiler into bytecode (.class files) instead of machine-specific code. This bytecode is a low-level set of instructions that can be executed by the Java Virtual Machine (JVM).

- **What are access modifiers in Java?**
  Access modifiers in Java are keywords used to set the accessibility or scope of classes, methods, and variables. They control where these members can be accessed in an application.

public :-

Accessible from any class, package, or subclass.

protected :-

Accessible within the same package and by subclasses, even if they are in different packages.

default (no modifier) :-

Accessible only within the same package.

private:

Accessible only within the class in which it is declared.

- **What is an object in Java?**

  In Java, an object is an instance of a class that represents a specific entity in a program with state and behavior. Objects are created from classes, which act as blueprints defining their attributes (fields) and behaviors (methods).

- **What is a class in Java?**

  In Java, a class is a blueprint or template for creating objects. It defines the structure (attributes or fields) and behavior (methods) that the objects created from it will have. Essentially, a class encapsulates data for the object and methods to manipulate that data, following the principles of object-oriented programming.

- **What is different between global & local Variable?**

	Global Variable		Local Variable
1	Scope of global variable is outside the method / across the method	1	Scope of the variable is within a method
2	Global variable can be static	2	Local variable cannot be static
3	Global variable can be private	3	Local variable cannot be private
4	variable are present within the class and outside the method	4	Variable are present inside the method

- **What is inheritance in Java?**

  In Java, inheritance is an object-oriented programming concept that allows one class to inherit the properties (fields) and behaviors (methods) of another class. This promotes code reusability and establishes a parent-child relationship between classes.

  There are 3 types :-

  1) Single Inheritance :-

  A class inherits from one superclass only, forming a simple parent-child relationship.

  2) Multilevel inheritance :-

  A chain of inheritance where a class inherits from another class, which in turn inherits from another.

  3) Hierarchical Inheritance :-

  Multiple classes inherit from a single superclass.

- **What is the difference between a constructor and a method?**

	Constructor		Method
1	Constructor is used for Initialize the object	1	Method is used for Perform operations/behaviors
2	Constructor having same as class name	2	Method can have any name
3	Constructor having no return type (not even void)	3	Method Must have a return type
4	Constructor automatically called on object creation	4	Method explicitly called on object
5	Constructor can be overloaded	5	Method can be overloaded and overridden

- **How is the static keyword used in Java?**

  In Java, the static keyword is used to define class-level members (variables and methods) that are shared across all instances of the class, rather than being tied to individual objects. This means that a static member belongs to the class itself, not to specific instances of the class. Here's how the static keyword is used:

  1) Static Variable
  2) Static Method
  3) Static Blocks

- **How is the String class different from other data types?**

  The `String` class in Java is different from other data types because it is immutable (its value cannot be changed after creation), stored in a special String pool to optimize memory usage, and provides many built-in methods for string manipulation. Unlike primitive types, `String` is an object, allowing advanced operations but making it slightly less memory-efficient for modifications. For frequent modifications, `StringBuilder` or `StringBuffer` (mutable alternatives) are recommended.

- **What all data types you know in java ?**
  There are two categories of data types:
    1) Primitive data type
    2) User defined data type

  > Primitive Data type :-

    1) byte
    2) short
    3) int
    4) long
    5) float
    6) double
    7) char
    8) boolean

- **How does the final keyword work with variables, methods, and classes?**

  The final keyword in Java has specific uses depending on its context:

  1) Final Variable :- A final variable's value cannot be changed once assigned, making it a constant.

  Example :

  final int MAX_SIZE = 100;

  2) Final Method :- A final method cannot be overridden by subclasses, preserving its original behavior

  Example :

  public final void display() { }

3) Final Class: A final class cannot be subclassed, preventing inheritance.

Example :

public final class Utility { }

- **Explain the use of if, else, switch, and break statements in Java ?**

1) if Statement :- Executes a block of code if a specified condition is true.

Example : if (condition) { /* code */ }

2) else Statement :- Executes an alternative block if the if condition is false.

Example : if (condition) { /* code if true */ } else { /* code if false */ }

3) switch Statement :- Selects and executes code based on the value of an expression, useful for multiple possible values.

Example :
```
switch (value) {
    case 1: /* code */ break;
    case 2: /* code */ break;
    default: /* code if no match */
}
```

4) break Statement :- Exits a loop or switch block immediately, stopping further execution in that block.

- **How does the continue statement work in a loop?**

  In Java, the continue statement skips the current iteration of a loop and immediately moves to the next iteration. It's commonly used to skip specific values or steps in for, while, or do-while loops without terminating the entire loop.

```
Example :   for (int i = 1; i <= 5; i++) {
                if (i == 3) continue; // Skip when i is 3
                System.out.println(i);
            }
            // Output: 1, 2, 4, 5
```

- **What is the enhanced for-each loop? When should it be used?**

  The enhanced for-each loop in Java simplifies iteration over arrays and collections, automatically handling indexing. It's ideal for situations where you need to access each element without modifying the underlying collection.

```
Example :   int[] numbers = {1, 2, 3};
            for (int num : numbers) {
                System.out.println(num);
            }
```

- **Explain the try-catch-finally block. How does it work in Java?**

  In Java, the try-catch-finally block is used for handling exceptions:

  1) try: Contains code that might throw an exception.

  2) catch: Catches and handles the exception if one occurs.

  3) finally: Executes code after try and catch, regardless of whether an exception was thrown, often used for cleanup.

  Example :

```
try {
    int result = 10 / 0; // risky code
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero."); // handles exception
} finally {
    System.out.println("Execution complete."); // always runs
}
```

- **Explain the concepts of inheritance and polymorphism in Java.**

  1) Inheritance :- Allows a class (subclass) to inherit properties and methods from another class (superclass), promoting code reuse and establishing an "is-a" relationship.

  Example :

```
class Animal { }
class Dog extends Animal { } // Dog inherits from Animal
```

  2) Polymorphism :- Allows objects to be treated as instances of their parent class, enabling dynamic method behavior. It comes in two forms:

  - Compile-time (Method Overloading): Same method name, different parameters.
  - Runtime (Method Overriding): Subclass provides its own implementation of a superclass method.

  Example :

```
Animal animal = new Dog(); // Dog behaves as an Animal
 animal.sound(); // Calls Dog's overridden method if available
```

- **What is method overloading? How is it different from method overriding?**

  1) Method Overloading :- Occurs when multiple methods in the same class have the same name but different parameters (different type, number, or order). It’s a form of compile-time polymorphism and allows methods to perform similar tasks with different inputs.

  Example :

  ```
  void print(int num) { }
  void print(String text) { } // Overloaded print method
  ```

  2) Method Overriding :- Occurs when a subclass provides a specific implementation of a method that is already defined in its superclass, with the same name, return type, and parameters. It’s a form of runtime polymorphism and allows the subclass to modify inherited behavior.

  Example :

  ```
  class Animal { void sound() { } }
  class Dog extends Animal { void sound() { } } // Overridden sound method
  ```

- **What is an interface in Java? How is it different from an abstract class?**

  1) Interface :- An interface is a reference type that can contain abstract methods (without a body) and static/final constants. It defines a contract that implementing classes must follow, enabling multiple inheritance of behavior.

  Example :

  interface Drawable { void draw(); }

  2) Difference from Abstract Class :-

  - An abstract class can have both abstract methods and concrete (implemented) methods, while an interface (prior to Java 8) only allowed abstract methods.
  - Classes can implement multiple interfaces but can extend only one abstract class.
  - Abstract classes are best for shared state and behavior; interfaces are for defining capabilities or contracts.

- **Can a Java class extend multiple classes? Why or why not?**

  No, a Java class cannot extend multiple classes because Java does not support multiple inheritance to avoid ambiguity from inheriting behaviors from multiple classes (the "diamond problem").

  Instead, Java allows a class to implement multiple interfaces, which provides flexibility without inheriting conflicting implementations.

- Explain the super and this keywords in Java.

  1) this :- Refers to the current instance of a class. It’s used to access instance variables, methods, or constructors of the current object.

  Example :

  ```
  this.name = name; // Refers to current object's 'name' field
  ```

  2) super :- Refers to the superclass (parent class) of the current object. It’s used to access superclass variables, methods, or constructors.

  Example :

  ```
  super.display(); // Calls display() method in superclass
  ```

- **What are exceptions in Java? How are they different from errors?**

  1) Exceptions :- Exceptions are events that disrupt normal program flow, typically due to issues like invalid input or file not found. They are recoverable and can be handled using try-catch blocks.

  Example :

  try { /* code */ } catch (Exception e) { /* handle exception */ }

  2) Errors: Errors are serious issues, often related to system resources (like OutOfMemoryError), and are non-recoverable. They usually indicate problems outside the application's control, so they are not meant to be caught or handled in code.

- **Explain the purpose of the throw and throws keywords.**

  1) throw: Used to explicitly throw an exception in the code. It is followed by an instance of an exception class.

  Example :

  throw new IllegalArgumentException("Invalid argument");

2) throws: Used in a method signature to declare that a method may throw exceptions, allowing the caller to handle or propagate them.

- **What is a NullPointerException, and how can it be avoided?**

  A NullPointerException occurs in Java when you attempt to dereference (access methods or fields of) a null object reference. It happens when an object is not initialized or assigned null.

  How to Avoid:

  1. Check for null before accessing object methods or fields.

```
if (object != null) {
    object.method();
}
```

2. Initialize objects properly before use.

3. Use Optional (Java 8+) to handle potential null values more safely.

- **What is the difference between checked and unchecked exceptions?**

  1) Checked Exceptions :-

  - These are exceptions that are explicitly checked at compile-time.
  - The programmer is required to handle or declare them using the throws keyword.
  - Examples: IOException, SQLException.

    Example :

    ```
    public void readFile() throws IOException { /* code */ }
    ```

  2) Unchecked Exceptions :-

  - These are exceptions that are not checked at compile-time and are inherited RuntimeException.
  - The programmer is not required to handle or declare them.
  - Examples: NullPointerException, ArrayIndexOutOfBoundsException.

    Example :

    ```
    int[] arr = new int[5];
    arr[10] = 5; // ArrayIndexOutOfBoundsException
    ```

- **What is the Java Collections Framework?**
  The Java Collections Framework is a set of interfaces, classes, and algorithms that provide a standard way to store, manipulate, and access collections of objects, such as lists, sets, and maps.
    - Interfaces: Define the contract for different types of collections (e.g., List, Set, Map).
    - Implementations: Concrete classes that implement the collection interfaces (e.g., ArrayList, HashSet, HashMap).
    - Algorithms: Utility methods for sorting, searching, and manipulating collections (e.g., Collections.sort()).
- **Explain the difference between List, Set, and Map in Java.**
  1) List :-
    - Ordered collection: Elements are stored in a specific order (insertion order).
    - Allows duplicates: Multiple identical elements can exist.
    - Common Implementations: ArrayList, LinkedList.

Example:

```
List<String> list = new ArrayList<>();
 list.add("A");
list.add("A"); // Allowed
```

2) Set :-

- Unordered collection: No guaranteed order of elements.
- No duplicates: Only unique elements are stored.
- Common Implementations: HashSet, LinkedHashSet.

Example:

```
Set<String> set = new HashSet<>();
set.add("A");
 set.add("A"); // Duplicate not allowed
```

3) Map :-

- Key-Value pair collection: Stores data in key-value pairs.
- Keys are unique: No two identical keys.
- Common Implementations: HashMap, TreeMap.

Example:

```
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("B", 2);
```

- **How does HashSet work internally in Java?**
  In Java, a HashSet is implemented using a HashMap internally. It stores elements as keys in the map with a constant dummy value (PRESENT).

  How it works:

  1) Hashing :- When an element is added, its hash code is computed using the hashCode() method. This determines where the element is stored in the underlying hash table.
  2) Uniqueness :- If two elements have the same hash code (a hash collision), the equals() method is used to compare the elements. If they are equal, the new element is not added (ensuring no duplicates).
  3) No Order :- HashSet does not maintain the order of elements because it stores them based on their hash codes.

- **What is different between Instance Variable and Static Variable ?**

Instance Variable	Static Variable
A variable declared inside the class but outside the body of the method.	A variable that is declared as static is called a static variable.
We can call instance variable by using object creation.	We can call static variable by using class name directly.
Instance variable not give updated value.	Static variable gives updated values for variable.
Instance variable will load after constructor calling	Static variable will load at the same class loaded
Instance variable allocate different address for all copies of variable.	Static variable allocate single space for all copies of variable.

- **What is a thread in Java? How do you create and start a thread?**

  In Java, a thread is a lightweight process that allows multiple tasks to run concurrently within a program. Each thread has its own execution path but shares the same memory space.

  How to Create and Start a Thread:

  1) By Extending the Thread class :-
     - Override the run() method to define the task.
     - Create an instance of the class and call start() to begin execution.

  Example:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

MyThread t = new MyThread();
t.start(); // Starts the thread
```

2) By Implementing the Runnable interface :-

- Implement the run() method and pass it to a Thread object.
- Call start() to begin execution.

Example:

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread is running");
    }
}

Thread t = new Thread(new MyRunnable());
t.start(); // Starts the thread
```

- **Explain the purpose of synchronized in Java.**

  In Java, the synchronized keyword is used to ensure that only one thread can access a critical section of code at a time, preventing race conditions and ensuring thread safety in concurrent environments.

Purpose:

- Method Level: Ensures that only one thread can execute a synchronized method at a time.

  Example:

```
synchronized void method() {
   // critical code
}
```

- Block Level: Restricts access to a specific block of code, providing finer control over synchronization.

  Example:

```
synchronized (object) {
   // critical code
}
```

- **Explain the concept of lambda expressions in Java?**

  In Java, a lambda expression is a concise way to represent an anonymous function (or method) that can be passed around as an argument to methods or stored in variables. It allows for more readable and compact code, especially when working with functional interfaces.

  Syntax:

  ```
  (parameters) -> expression
  ```

  Example:

  ```
  // Lambda expression for a simple addition
  (int a, int b) -> a + b;
  ```

  Uses:

  - Functional Interfaces: Lambda expressions work with functional interfaces (interfaces with a single abstract method).
  - Streams API: Commonly used with Java's Streams API to perform operations like filtering, mapping, etc.