**What is Java?**

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is widely used for building cross-platform applications.

**Explain the structure of a basic Java program.**

A basic Java program includes:

public class Main {

public static void main(String[] args) {

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

}

}

public class Main: Defines a public class named Main.

public static void main(String[] args): Main method where the program execution begins.

System.out.println("Hello, World!");: Prints "Hello, World!" to the console.

**What are the main data types in Java?**

Java has two main categories of data types:

Primitive types: byte, short, int, long, float, double, char, boolean

Reference types: Objects, Arrays, Interfaces

**What is the difference between int and Integer?**

int is a primitive data type, while Integer is a wrapper class in Java. Integer provides methods for manipulating integer values and can be used in collections that require objects.

**Explain the use of final keyword in Java.**

The final keyword can be used with classes, methods, and variables:

final class: Prevents the class from being subclassed.

final method: Prevents the method from being overridden by subclasses.

final variable: The value of the variable cannot be changed once initialized.

**What is the significance of the main method in Java?**

The main method is the entry point of any standalone Java application. It is where the JVM starts execution. The signature of the main method is public static void main(String[] args).

**What are Java identifiers?**

Identifiers are names given to elements such as variables, classes, and methods. They must begin with a letter (A-Z or a-z), currency character ($), or an underscore (\_). Subsequent characters can be letters, digits (0-9), currency characters, or underscores.

**Explain the System.out.println() statement.**

System.out.println() is used to print messages to the console. System is a class from the java.lang package, out is a static member of the System class, which is an instance of PrintStream, and println is a method of PrintStream that prints a message followed by a new line.

**What is the purpose of the package keyword in Java?**

The package keyword is used to group related classes and interfaces together. It helps organize code and avoid naming conflicts. For example:

package com.example.myapp;

**What are the rules for naming identifiers in Java?**

Identifiers can begin with a letter (A-Z or a-z), a currency character ($), or an underscore (\_).

Subsequent characters may be letters, digits (0-9), currency characters, or underscores.

Identifiers are case-sensitive.

They cannot be keywords or reserved words.

**Explain the difference between == and .equals() in Java.**

== compares the references (memory addresses) of objects, meaning it checks if both references point to the same object.

.equals() compares the values within the objects. For example, with String objects, .equals() checks the actual sequence of characters.

**What is type casting in Java?**

Type casting is converting one data type into another. There are two types:

Implicit (automatic) casting: Smaller type to a larger type, e.g., int to long.

Explicit casting: Larger type to a smaller type, e.g., double to float. This requires a cast operator.

int i = 100;

double d = i; // implicit casting

double d = 100.5;

int i = (int) d; // explicit casting

**What are Java keywords? Provide some examples.**

Keywords are reserved words in Java that have a predefined meaning in the language syntax. Examples include: class, public, static, void, if, else, while, for, return, int, double, new, try, catch, finally.

**What is a Java variable? Explain the different types of variables in Java.**

A variable is a container that holds data that can be changed during the execution of a program. There are three types of variables:

Local variables: Declared inside a method or block and accessible only within it.

Instance variables: Declared inside a class but outside any method, constructor, or block, and are accessible by all methods within the class.

Class/Static variables: Declared with the static keyword inside a class but outside any method, constructor, or block, and are shared among all instances of the class.

**Explain the concept of an array in Java.**

An array is a container that holds a fixed number of values of a single type. The length of an array is established when the array is created and cannot be changed. Arrays are zero-indexed.

int[] numbers = new int[5]; // declares an array of integers

numbers[0] = 10; // assigns a value to the first element

**What is the difference between break and continue statements?**

break: Exits from the loop or switch statement immediately.

for (int i = 0; i < 10; i++) {

if (i == 5) {

break; // exit loop

}

}

continue: Skips the current iteration of the loop and proceeds to the next iteration.

for (int i = 0; i < 10; i++) {

if (i == 5) {

continue; // skip the rest of the loop body for this iteration

}

}

**What are literals in Java?**

Literals are fixed values that appear directly in the code. They represent constant values of primitive data types. Examples include:

Integer literals: 10, 0xFF (hexadecimal)

Floating-point literals: 10.5, 2.5e3

Character literals: 'a', '\n'

String literals: "Hello, World!"

Boolean literals: true, false

**What is the difference between ++i and i++?**

++i is the pre-increment operator: It increments the value of i before using it in the expression.

i++ is the post-increment operator: It increments the value of i after using it in the expression.

int i = 5;

int pre = ++i; // pre is 6, i is 6

int post = i++; // post is 6, i is 7

**Explain the switch statement in Java.**

The switch statement allows for the selection of one of many code blocks to be executed based on the value of an expression. It provides a more readable alternative to a series of if-else-if statements.

int day = 3;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

default:

System.out.println("Invalid day");

}

**What are the main principles of Object-Oriented Programming?**

Encapsulation: Bundling the data (variables) and methods (functions) that operate on the data into a single unit, or class, and restricting access to some of the object's components.

Inheritance: A mechanism where one class (subclass or derived class) inherits the attributes and methods from another class (superclass or base class), promoting code reuse.

Polymorphism: The ability of different objects to respond, each in its own way, to identical messages (methods). This includes method overriding (runtime polymorphism) and method overloading (compile-time polymorphism).

Abstraction: The process of hiding the complex implementation details and showing only the necessary features of an object.

**Explain the concept of a class and an object in Java.**

Class: A blueprint for creating objects (a particular data structure), providing initial values for state (member variables) and implementations of behavior (member functions or methods).

public class Car {

String color;

int speed;

void accelerate() {

speed += 10;

}

}

Object: An instance of a class. It is created from a class and has its own state and behavior.

Car myCar = new Car();

myCar.color = "Red";

myCar.accelerate();

**What is inheritance in Java and how does it work?**

Inheritance allows a class to inherit fields and methods from another class. The class that inherits is called a subclass (or derived class), and the class being inherited from is called a superclass (or base class).

public class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

public class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

}

**What is method overriding in Java?**

Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The overridden method in the subclass should have the same signature as the method in the superclass.

class Animal {

void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Bark");

}

}

**What is method overloading in Java?**

Method overloading allows a class to have more than one method with the same name, as long as their parameter lists are different. It is a compile-time polymorphism.

class MathUtils {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

**What is the difference between an abstract class and an interface in Java?**

Abstract Class: A class that cannot be instantiated and may contain abstract methods (without implementation) as well as concrete methods (with implementation). It is used to provide a common base class for derived classes.

abstract class Animal {

abstract void makeSound();

void sleep() {

System.out.println("Sleeping");

}

}

Interface: A completely abstract class that contains only abstract methods. Interfaces are used to define a contract that classes can implement.

interface Animal {

void makeSound();

}

class Dog implements Animal {

public void makeSound() {

System.out.println("Bark");

}

}

**What is encapsulation and how do you achieve it in Java?**

Encapsulation is the technique of making the fields in a class private and providing access to them via public methods (getters and setters). This helps to protect the data and makes the code more maintainable.

public class Person {

private String name;

private int age;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**What is polymorphism and what are its types in Java?**

Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. The two types of polymorphism in Java are:

Compile-time polymorphism (method overloading): The method to be invoked is determined at compile time.

Runtime polymorphism (method overriding): The method to be invoked is determined at runtime.

class Animal {

void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Bark");

}

}

Animal myDog = new Dog();

myDog.makeSound(); // Outputs "Bark"

**What is the this keyword in Java?**

The this keyword is a reference variable that refers to the current object. It is used to differentiate between instance variables and parameters or to invoke current class methods and constructors.

public class Person {

private String name;

public Person(String name) {

this.name = name; // 'this.name' refers to the instance variable

}

public void display() {

System.out.println("Name: " + this.name); // Using 'this' to refer to instance variable

}

}

**Explain the super keyword in Java.**

The super keyword is used to refer to the immediate parent class object. It can be used to access parent class variables, methods, and constructors.

class Animal {

void eat() {

System.out.println("Animal eats");

}

}

class Dog extends Animal {

void eat() {

super.eat(); // Calls the parent class's eat method

System.out.println("Dog eats");

}

}

**What is an inner class in Java?**

An inner class is a class defined within another class. It can access all members (including private) of the outer class. There are four types of inner classes:

Nested (static) class

Member (non-static) inner class

Local inner class

Anonymous inner class

class OuterClass {

private int data = 30;

class InnerClass {

void display() {

System.out.println("Data is " + data);

}

}

}

**What is an abstract method in Java?**

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), within an abstract class or an interface. Subclasses or implementing classes must provide the implementation for these methods.

abstract class Animal {

abstract void makeSound();

}

class Dog extends Animal {

void makeSound() {

System.out.println("Bark");

}

}

**What is a constructor in Java?**

A constructor is a block of code used to initialize an object. It is called when an instance of a class is created. Constructors have the same name as the class and do not have a return type.

public class Car {

String color;

Car(String color) {

this.color = color;

}

}

Car myCar = new Car("Red");

**Can a class implement multiple interfaces in Java?**

Yes, a class can implement multiple interfaces in Java, providing a way to achieve multiple inheritance.

interface Drivable {

void drive();

}

interface Flyable {

void fly();

}

class FlyingCar implements Drivable, Flyable {

public void drive() {

System.out.println("Driving");

}

public void fly() {

System.out.println("Flying");

}

}

**What is the purpose of the instanceof keyword?**

The instanceof keyword is used to test whether an object is an instance of a specific class or an interface.

Dog myDog = new Dog();

System.out.println(myDog instanceof Animal); // Returns true

**What is the difference between composition and inheritance?**

Inheritance: Represents an "is-a" relationship. It is a mechanism where one class (subclass) inherits attributes and methods from another class (superclass). It is used to achieve polymorphism and code reuse.

class Animal {

void eat() {

System.out.println("Animal eats");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

Composition: Represents a "has-a" relationship. It is a design principle where a class contains an instance of another class to achieve code reuse. It is more flexible than inheritance.

class Engine {

void start() {

System.out.println("Engine starts");

}

}

class Car {

private Engine engine;

Car() {

engine = new Engine();

}

void startCar() {

engine.start();

}

}

**What are the access modifiers in Java?**

Public: The member is accessible from any other class.

Private: The member is accessible only within the class it is declared.

Protected: The member is accessible within the same package and by subclasses.

Default (package-private): If no access modifier is specified, the member is accessible only within the same package.

**What is a static method in Java?**

A static method belongs to the class rather than an instance of the class. It can be called without creating an instance of the class. Static methods can only access static variables and other static methods directly.

class Utility {

static int add(int a, int b) {

return a + b;

}

}

int sum = Utility.add(5, 10); // No need to create an instance of Utility

**What is a singleton class in Java? How do you implement it?**

A singleton class restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.

public class Singleton {

private static Singleton instance;

private Singleton() {} // Private constructor to prevent instantiation

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

Singleton single = Singleton.getInstance();

**What is the final keyword used for in Java?**

Final class: Cannot be subclassed.

public final class Constants {

// Class code

}

Final method: Cannot be overridden by subclasses.

class Parent {

final void display() {

System.out.println("Parent display");

}

}

Final variable: Its value cannot be changed once initialized.

final int MAX\_VALUE = 100;

**What is a constructor chaining in Java?**

Constructor chaining is the process of calling one constructor from another constructor with respect to the current object. It can be done within the same class using this() or from a parent class using super().

class Base {

Base() {

System.out.println("Base constructor");

}

}

class Derived extends Base {

Derived() {

this(10);

System.out.println("Derived no-arg constructor");

}

Derived(int x) {

super();

System.out.println("Derived parameterized constructor");

}

}

**What is the difference between this and super in Java?**

this: Refers to the current instance of the class.

class Car {

private String color;

Car(String color) {

this.color = color;

}

}

super: Refers to the immediate parent class instance. It is used to access parent class members and constructors.

class Animal {

void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

void makeSound() {

super.makeSound(); // Calls the parent class's makeSound method

System.out.println("Bark");

}

}

**What is the purpose of the abstract keyword in Java?**

The abstract keyword is used to declare a class or method as abstract. An abstract class cannot be instantiated and may contain abstract methods that must be implemented by subclasses.

abstract class Animal {

abstract void makeSound();

}

class Dog extends Animal {

void makeSound() {

System.out.println("Bark");

}

}

**How does Java achieve runtime polymorphism?**

Java achieves runtime polymorphism through method overriding and dynamic method dispatch. When a method is called on a superclass reference, the actual method that gets executed is determined at runtime based on the object that the reference points to.

class Animal {

void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

void makeSound() {

System.out.println("Bark");

}

}

Animal myAnimal = new Dog();

myAnimal.makeSound(); // Outputs "Bark" at runtime

**What is an interface in Java, and how is it different from an abstract class?**

An interface is a completely abstract class that contains only abstract methods and constants. In Java 8 and later, interfaces can also contain default and static methods.

Differences:

An abstract class can have instance variables and constructors, while an interface cannot.

A class can implement multiple interfaces but can only extend one abstract class.

interface Animal {

void makeSound();

}

class Dog implements Animal {

public void makeSound() {

System.out.println("Bark");

}

}

**Can you explain the concept of object cloning in Java?**

Object cloning is a process of creating an exact copy of an existing object. It is achieved by implementing the Cloneable interface and overriding the clone() method from the Object class.

class Person implements Cloneable {

String name;

Person(String name) {

this.name = name;

}

protected Object clone() throws CloneNotSupportedException {

return super.clone();

}

}

Person original = new Person("John");

Person copy = (Person) original.clone();

**What is a nested class and what are its types?**

A nested class is a class defined within another class. Types of nested classes include:

Static nested class: A static member of the enclosing class.

class Outer {

static class StaticNested {

void display() {

System.out.println("Static nested class");

}

}

}

Non-static nested class (Inner class): An instance member of the enclosing class.

class Outer {

class Inner {

void display() {

System.out.println("Inner class");

}

}

}

**What is the difference between shallow copy and deep copy in Java?**

Shallow Copy: Copies the object's reference pointers, so the new object points to the same memory locations as the original object.

class Person implements Cloneable {

String name;

Address address;

protected Object clone() throws CloneNotSupportedException {

return super.clone(); // Creates a shallow copy

}

}

Deep Copy: Creates a new instance for the referenced objects as well, ensuring a completely independent copy.

class Person implements Cloneable {

String name;

Address address;

protected Object clone() throws CloneNotSupportedException {

Person cloned = (Person) super.clone();

cloned.address = (Address) address.clone(); // Ensures deep copy

return cloned;

}

}

**What are default methods in interfaces, and why were they introduced in Java 8?**

Default methods in interfaces are methods with a body. They were introduced in Java 8 to allow the addition of new methods to interfaces without breaking the existing implementations.

interface Vehicle {

default void start() {

System.out.println("Vehicle started");

}

}

class Car implements Vehicle {

// Inherits default method from Vehicle

}

**Explain the concept of dynamic method dispatch in Java.**

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at runtime rather than compile-time. This allows Java to support runtime polymorphism.

class Animal {

void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

void makeSound() {

System.out.println("Bark");

}

}

class Cat extends Animal {

void makeSound() {

System.out.println("Meow");

}

}

Animal a;

a = new Dog();

a.makeSound(); // Outputs "Bark"

a = new Cat();

a.makeSound(); // Outputs "Meow"

**What is the difference between aggregation and composition in Java?**

Aggregation: A weaker form of association where the lifecycle of the contained objects is not tied to the lifecycle of the container object. Aggregated objects can exist independently.

class Department {

private List<Employee> employees;

}

class Employee {

// Employee class definition

}

Composition: A stronger form of association where the contained objects' lifecycle is tightly coupled with the container object's lifecycle. Composed objects cannot exist independently.

class House {

private Room room;

House() {

room = new Room();

}

}

class Room {

// Room class definition

}

**What is the finalize method in Java?**

The finalize method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. It is used to perform cleanup operations before the object is permanently discarded.

class Resource {

@Override

protected void finalize() throws Throwable {

try {

// Cleanup code

} finally {

super.finalize();

}

}

}

**What is a covariant return type in Java?**

Covariant return type allows an overriding method to return a subtype of the type returned by the overridden method.

class Animal {

Animal get() {

return this;

}

}

class Dog extends Animal {

@Override

Dog get() {

return this;

}

}

**Explain the difference between ArrayList and LinkedList.**

ArrayList: Resizable array implementation. Provides fast random access but slow insertions and deletions (except at the end).

tring> arrayList = new ArrayList<>();

LinkedList: Doubly linked list implementation. Provides fast insertions and deletions but slower random access.

List<String> linkedList = new LinkedList<>();

**What is an enum in Java?**

An enum is a special data type that enables for a variable to be a set of predefined constants. It is the most effective way to define a set of named values.

enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

}

**Explain the concept of synchronized keyword in Java.**

The synchronized keyword is used to control access to a critical section of code by multiple threads to ensure thread safety. It can be applied to methods or blocks.

public class Counter {

private int count = 0;

public synchronized void increment() {

count++;

}

}

**What is the difference between wait(), notify(), and notifyAll() in Java?**

wait(): Causes the current thread to wait until another thread calls notify() or notifyAll() on the same object.

notify(): Wakes up a single thread that is waiting on this object's monitor.

notifyAll(): Wakes up all threads that are waiting on this object's monitor.

synchronized void example() {

try {

wait(); // Wait for notification

} catch (InterruptedException e) {

e.printStackTrace();

}

notify(); // Notify one waiting thread

notifyAll(); // Notify all waiting threads

}

**What is the purpose of the transient keyword in Java?**

The transient keyword in Java is used to indicate that a field should not be serialized when the containing object is serialized.

class User implements Serializable {

private String username;

private transient String password; // password will not be serialized

}

**Explain the volatile keyword in Java.**

The volatile keyword in Java is used to mark a variable as being stored in main memory. This ensures that the value of the volatile variable is always read from the main memory and not from a thread's local cache, providing visibility of changes across threads.

class SharedResource {

private volatile boolean flag = true;

}

**What is the default method in interfaces, and how is it used?**

Default methods in interfaces are methods with a default implementation. They were introduced in Java 8 to allow interfaces to evolve without breaking existing implementations.

interface Vehicle {

default void start() {

System.out.println("Vehicle started");

}

}

class Car implements Vehicle {

// Inherits default method from Vehicle

}

**What is the difference between a static block and an instance block?**

Static block: A block of code that runs once when the class is loaded into memory. It is used for static initializations.

class Example {

static {

System.out.println("Static block executed");

}

}

Instance block: A block of code that runs every time an instance of the class is created. It is used for instance initializations.

class Example {

{

System.out.println("Instance block executed");

}

}

**How do you prevent a class from being subclassed in Java?**

To prevent a class from being subclassed, you declare the class as final.

public final class Constants {

// Class code

}

**What is a factory method in Java?**

A factory method is a method that returns an instance of a class. It can be used to abstract the creation process and encapsulate the logic of object creation.

class ShapeFactory {

static Shape getShape(String type) {

if ("Circle".equals(type)) {

return new Circle();

} else if ("Square".equals(type)) {

return new Square();

}

return null;

}

}

**What is the difference between throws and throw in Java?**

throws: Used in method signatures to declare that a method can throw one or more exceptions.

void method() throws IOException {

// Method code

}

throw: Used within a method to throw an exception explicitly.

void method() {

throw new IOException("Error message");

}

**Explain the purpose of a marker interface in Java.**

A marker interface is an interface with no methods or constants. It is used to mark a class with a specific property or capability. Examples include Serializable, Cloneable, and Remote.

class Example implements Serializable {

// Class code

}

**What is method hiding in Java?**

Method hiding occurs when a subclass defines a static method with the same name and signature as a static method in the superclass. The method in the subclass hides the method in the superclass.

class SuperClass {

static void display() {

System.out.println("SuperClass display");

}

}

class SubClass extends SuperClass {

static void display() {

System.out.println("SubClass display");

}

}

**What is the diamond problem and how does Java handle it?**

The diamond problem occurs in multiple inheritance scenarios where a class inherits from two classes that have a common base class, leading to ambiguity. Java handles this by not allowing multiple inheritance of classes. Instead, Java allows multiple inheritance through interfaces.

interface A {

void display();

}

interface B extends A {

void display();

}

class C implements B {

public void display() {

System.out.println("Display method");

}

}

**What is an immutable class in Java? How do you create one?**

An immutable class is a class whose instances cannot be modified after creation. To create an immutable class:

Declare the class as final.

Make all fields private and final.

Provide only getter methods for the fields.

Initialize all fields through the constructor.

public final class ImmutableClass {

private final String name;

private final int age;

public ImmutableClass(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

}

**What is method resolution order (MRO) in Java?**

Method resolution order (MRO) determines the order in which methods are inherited in the presence of multiple inheritance. In Java, since multiple inheritance is not allowed with classes, this concept primarily applies to interfaces and follows a depth-first, left-to-right approach.

interface A {

default void display() {

System.out.println("A");

}

}

interface B extends A {

default void display() {

System.out.println("B");

}

}

class C implements B {

public static void main(String[] args) {

new C().display(); // Outputs "B"

}

}

|  |  |  |
| --- | --- | --- |
| **S.No.** | **PUT** | **POST** |
| 1 | The PUT method is considered idempotent. In this method, if we send a request several times, it will be counted as a single request modification. | The POST method is considered idempotent. In this method, if we send a request multiple times, then multiple URLs will get created on the servers. |
| 2 | The PUT method is used to modify a single resource. | The POST method is used to add a child resource. |
| 3 | It can be cached. | It cannot be cached. |
| 4 | This method acts as specific. | This method acts as an abstract. |
| 5 | In the PUT method, we use the UPDATE query. | In the POST method, we use a CREATE query. |
| 6 | When it comes to the PUT method, the client finalises which URL resource should be updated. | When it comes to the POST method, the server finalies which URL resource should be added. |