Oops concepts.

Modifies:access, non access(static,final).

constructors

Oops concepts  
**Object-Oriented Programming (OOP) Concepts Explained with Answers to Interview Questions**

**OOP Concepts**

**1. Encapsulation**

Encapsulation bundles data (variables) and methods (functions) that operate on the data into a single unit (class) and restricts direct access to data members using access modifiers.

**Example**:

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public double getBalance() {

return balance;

}

}

**Q&A**:  
**Q: What is encapsulation? How is it implemented?**  
**A:** Encapsulation is the mechanism of wrapping data and methods in a single unit, ensuring controlled access using access modifiers like private, public, or protected. It is implemented by making class variables private and providing public getter and setter methods for controlled access.

**2. Inheritance**

Inheritance allows a class (child) to acquire properties and behaviors from another class (parent), promoting code reusability.

**Example**:

class Animal {

void eat() {

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

}

}

class Dog extends Animal {

void bark() {

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

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited method

dog.bark(); // Method specific to Dog

}

}

**Q&A**:  
**Q: What is inheritance? What are its types?**  
**A:** Inheritance is the process by which one class acquires the properties and methods of another class.

* **Single inheritance**: One class inherits from another.
* **Multilevel inheritance**: A class inherits from a derived class.
* **Hierarchical inheritance**: Multiple classes inherit from a single base class.  
  (Note: Java does not support multiple inheritance with classes to avoid ambiguity; it uses interfaces instead.)

**3. Polymorphism**

Polymorphism means "many forms" and allows methods to perform differently based on the object or input.

**Method Overloading (Compile-Time Polymorphism):**

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

**Method Overriding (Run-Time Polymorphism):**

class Animal {

void sound() {

System.out.println("This animal makes a sound.");

}

}

class Cat extends Animal {

@Override

void sound() {

System.out.println("The cat meows.");

}

}

**Q&A**:  
**Q: What is polymorphism? How is it achieved?**  
**A:** Polymorphism allows the same method or operator to behave differently. It is achieved through:

* **Compile-Time Polymorphism**: Using method overloading (methods with the same name but different parameters).
* **Run-Time Polymorphism**: Using method overriding (subclass method overrides the superclass method).

**4. Abstraction**

Abstraction hides implementation details and exposes only the essential features.

**Abstract Class Example:**

abstract class Shape {

abstract void draw(); // Abstract method

}

class Circle extends Shape {

void draw() {

System.out.println("Drawing a circle.");

}

}

**Interface Example:**

interface Vehicle {

void start();

}

class Car implements Vehicle {

public void start() {

System.out.println("Car starts with a key.");

}

}

Explain about modifiers ?

Absolutely! Let's go in-depth on **Java Modifiers** and then cover the most common **interview questions** specifically related to modifiers.

**✅ Java Modifiers Explained**

Java modifiers are **keywords** used to define the **scope**, **behavior**, and **access level** of classes, methods, constructors, and variables.

**🔹 1. Access Modifiers – Control Visibility**

| **Modifier** | **Class** | **Package** | **Subclass (Different Pkg)** | **World** |
| --- | --- | --- | --- | --- |
| public | ✔ | ✔ | ✔ | ✔ |
| protected | ✔ | ✔ | ✔ | ✖ |
| *default* | ✔ | ✔ | ✖ | ✖ |
| private | ✔ | ✖ | ✖ | ✖ |

**📌 Usage**

* public: Accessible from anywhere.
* protected: Accessible within package + subclasses.
* *(no modifier)*: Accessible only within the package.
* private: Accessible only within the class.

**🔹 2. Non-Access Modifiers**

**➤ For Classes**

* final: Class cannot be inherited.
* abstract: Class cannot be instantiated (may contain abstract methods).
* strictfp: Restricts floating-point precision consistency across platforms.

**➤ For Methods**

* final: Cannot be overridden by subclasses.
* abstract: Must be implemented by a subclass.
* static: Belongs to the class, not the instance.
* synchronized: Used in multithreading to lock access.
* native: Declares that the method is implemented in native code (like C/C++).
* strictfp: Ensures consistent floating-point calculations.

**➤ For Variables**

* final: Value cannot be changed once assigned.
* static: Shared among all instances of a class.
* transient: Excluded during serialization.
* volatile: Ensures changes made by one thread are visible to others.

.

**🚧 What is a Constructor?**

A **constructor** in Java is a **special method** that gets called **automatically** when an object of a class is created. It is used to **initialize the object**.

**✅ Key Characteristics**

| **Feature** | **Description** |
| --- | --- |
| **Name** | Same as class name |
| **Return type** | No return type, not even void |
| **Access** | Can be public, private, protected, or default |
| **Called when** | A new object is created using new |
| **Overloading** | Allowed (multiple constructors with different parameters) |
| **Inheritance** | Not inherited by subclasses |
| **Chaining** | You can call one constructor from another using this() |

**🔧 Types of Constructors**

**1. Default Constructor**

* No parameters
* If not explicitly written, Java provides one automatically

class Dog {

Dog() {

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

}

}

If you define **any constructor**, Java **does not** generate a default one for you.

**2. Parameterized Constructor**

* Allows you to pass values during object creation

class Dog {

String name;

Dog(String name) {

this.name = name;

}

}

**3. Copy Constructor (Manual)**

* Java does not provide one by default, but you can define it yourself

class Dog {

String name;

Dog(String name) {

this.name = name;

}

Dog(Dog d) {

this.name = d.name;

}

}

**4. Private Constructor**

* Used in **Singleton Pattern**
* Prevents external instantiation

class Singleton {

private static Singleton instance = new Singleton();

private Singleton() {} // private constructor

public static Singleton getInstance() {

return instance;

}

}

**5. Constructor Chaining**

* Calling one constructor from another in the same class using this()

class Car {

int speed;

Car() {

this(50); // calls the parameterized constructor

}

Car(int speed) {

this.speed = speed;

}

}

**🧠 Java Modifier Interview Questions (with Answers)**

**🔸 Access Modifier Questions**

1. **What’s the difference between public, private, protected, and default access?**
   * Explains how each controls visibility.
2. **Can a class be private in Java?**
   * Only **inner classes** can be private, top-level classes cannot.
3. **What’s the default access modifier for a class?**
   * *Package-private* (no modifier): accessible within the same package.

**🔸 Final Keyword Questions**

1. **What happens when you declare a variable as final?**
   * It becomes a constant – value can’t change once assigned.
2. **Can a final method be overridden?**
   * ❌ No, final methods can't be overridden.
3. **Can a final class be extended?**
   * ❌ No, final classes cannot be subclassed.

**🔸 Static Keyword Questions**

1. **What is the purpose of the static keyword?**
   * Creates class-level methods or variables shared by all instances.
2. **Can static methods access instance variables?**
   * ❌ No, they can only access other static data.
3. **Can we override a static method?**
   * No, it’s called *method hiding*, not overriding.

**🔸 Abstract Keyword Questions**

1. **Can we instantiate an abstract class?**
   * ❌ No, it must be subclassed and implemented.
2. **What happens if a class has an abstract method but is not declared abstract?**
   * Compilation error.
3. **Can an abstract class have a constructor?**
   * ✅ Yes, and it's called when a subclass is instantiated.

**🔸 Synchronized, Volatile, Transient Questions**

1. **What does synchronized do in Java?**
   * Prevents multiple threads from accessing the method/block simultaneously.
2. **What is the difference between volatile and synchronized?**
   * volatile ensures visibility; synchronized ensures mutual exclusion and visibility.
3. **What is transient in Java?**
   * Marks a field to be **excluded** from serialization.

**🔸 Other Trick Questions**

1. **Can a method be both abstract and final?**
   * ❌ No. Abstract means “must be overridden”; final means “cannot be overridden.”
2. **Can a method be static and abstract?**
   * ❌ No. Static methods belong to the class, abstract requires overriding in subclasses.
3. **Can a constructor be declared final, static, or abstract?**
   * ❌ No. Constructors cannot be final, static, or abstract.

If you'd like, I can also give you:

* A one-pager **Java Modifiers Cheat Sheet**
* Mock interview questions & answers (PDF or quiz format)
* Flashcards for quick revision

Ans,

Here's a concise **Java Modifiers Cheat Sheet** – perfect for interviews or quick revision 👇

**✅ Java Modifiers Cheat Sheet**

**🔹 Access Modifiers**

| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| --- | --- | --- | --- | --- |
| public | ✅ | ✅ | ✅ | ✅ |
| protected | ✅ | ✅ | ✅ | ❌ |
| *default* | ✅ | ✅ | ❌ | ❌ |
| private | ✅ | ❌ | ❌ | ❌ |

* **public**: Accessible everywhere
* **protected**: Accessible in same package + subclass (even in other packages)
* **default**: (No modifier) Only within same package
* **private**: Only within the class

**🔹 Non-Access Modifiers**

**➤ For Classes**

| **Modifier** | **Meaning** |
| --- | --- |
| final | Class can't be extended (no subclassing) |
| abstract | Class can't be instantiated, may have abstract methods |
| strictfp | Enforces floating-point consistency |

**➤ For Methods**

| **Modifier** | **Meaning** |
| --- | --- |
| final | Can't be overridden |
| abstract | Must be implemented in subclass |
| static | Belongs to class, not instance |
| synchronized | Thread-safe access |
| native | Implemented in non-Java (native) code |
| strictfp | Applies IEEE floating-point precision |

**➤ For Variables**

| **Modifier** | **Meaning** |
| --- | --- |
| final | Constant – value can't change |
| static | Shared across all instances |
| transient | Not serialized |
| volatile | Always read from main memory (used in multithreading) |

**🔹 Illegal Modifier Combinations**

| **Combination** | **Result** |
| --- | --- |
| abstract + final | ❌ Not allowed (conflict of intent) |
| abstract + static | ❌ Not allowed |
| abstract + private | ❌ Not allowed |
| final constructor | ❌ Not allowed (constructors can't be final) |

**Q1: Can a class be both abstract and final?**

**Answer:** ❌ No. abstract means the class **must be extended**, while final means it **cannot be extended**. Hence, it's a contradiction.

**Q2: What’s the difference between final, finally, and finalize()?**

| **Term** | **Type** | **Purpose** |
| --- | --- | --- |
| final | Modifier | Prevent change |
| finally | Block | Executes after try-catch |
| finalize() | Method | Called by GC before object is deleted (deprecated) |

**Q3: What is the use of the volatile keyword?**

**Answer:** It ensures that **multiple threads** see the most recent value of a variable. Prevents caching of variable values.

java

CopyEdit

volatile boolean running = true;

**Q4: Difference between static and final?**

| **Keyword** | **Meaning** |
| --- | --- |
| static | Belongs to class |
| final | Cannot be changed or overridden |

**Q5: What is transient?**

**Answer:** Used to **exclude** variables from serialization. When an object is serialized, transient fields are **skipped**.

**Q6: What happens if a method is both synchronized and static?**

**Answer:** The method locks the **class object**, not the instance. This is useful for **class-level locking** in multithreading.

**Q7: Can a constructor be static, final, or abstract?**

**Answer:**

* static: ❌ Not allowed. Constructors are tied to object creation.
* final: ❌ Not allowed. Constructors are not inherited.
* abstract: ❌ Not allowed. Constructors must be concrete.

**Q8: Can you override a method that is final or static?**

**Answer:**

* final: ❌ Cannot override
* static: ❌ Hides the method (doesn't override)

**Q9: What does strictfp do?**

**Answer:** Ensures **consistent floating-point calculations** across platforms by following IEEE standards.

**Q10: Can an interface have final, static, or private methods?**

**Answer:**

* final: ❌ (not allowed in interface)
* static: ✅ (Java 8+)
* private: ✅ (Java 9+)

**🔍 Quick Summary Table**

| **Modifier** | **Meaning** | **Can apply to** |
| --- | --- | --- |
| public | Accessible anywhere | Class, Method, Variable |
| private | Class-only access | Method, Variable |
| protected | Subclass + package | Method, Variable |
| default | Package-only access | Class, Method |
| final | Cannot change/extend | Class, Method, Variable |
| static | Class-level member | Method, Variable |
| abstract | No implementation | Class, Method |
| synchronized | Thread-safety | Method, Block |
| volatile | Ensure latest value | Variable |
| transient | Skip serialization | Variable |
| strictfp | IEEE FP consistency | Class, Method |
| native | Code in other lang | Method |

**Final, Finally, and Finalize in Java**

These three terms may sound similar, but they serve different purposes in Java. Let's break them down:

**1. final**

**Purpose:**

The final keyword is used to define constants, prevent inheritance, or prohibit method overriding.

**Use Cases:**

1. **Final Variables:**
   * The value of a final variable cannot be changed after initialization.
   * Must be initialized either at the time of declaration or in the constructor.

**Example:**

java

CopyEdit

public class FinalVariableExample {

final int MAX\_VALUE = 100; // Constant

void display() {

// MAX\_VALUE = 200; // Error: Cannot reassign a final variable

System.out.println("MAX\_VALUE: " + MAX\_VALUE);

}

}

1. **Final Methods:**
   * A final method cannot be overridden by subclasses.

**Example:**

java

CopyEdit

class Parent {

final void display() {

System.out.println("This method cannot be overridden.");

}

}

class Child extends Parent {

// void display() {} // Error: Cannot override final method

}

1. **Final Classes:**
   * A final class cannot be extended (i.e., no subclasses can be created).

**Example:**

java

CopyEdit

final class FinalClass {

void display() {

System.out.println("This class cannot be inherited.");

}

}

// class SubClass extends FinalClass {} // Error: Cannot inherit from a final class

**2. finally**

**Purpose:**

The finally block is used in exception handling to execute a block of code regardless of whether an exception is thrown or not.

**Key Points:**

* It is **optional** and must be used with a try block.
* Commonly used for **cleanup operations**, like closing resources.

**Example:**

java

CopyEdit

public class FinallyExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // Exception occurs here

} catch (ArithmeticException e) {

System.out.println("Exception caught: " + e.getMessage());

} finally {

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

}

}

}

**Output:**

vbnet

CopyEdit

Exception caught: / by zero

Finally block executed.

**When Does finally NOT Execute?**

* If the JVM terminates abruptly (e.g., using System.exit(0)).

**Example:**

java

CopyEdit

try {

System.exit(0);

} finally {

System.out.println("This will not execute.");

}

**3. finalize**

**Purpose:**

The finalize method is called by the Garbage Collector (GC) before an object is destroyed to allow cleanup operations.

**Key Points:**

* Defined in the Object class.
* Rarely used in modern Java due to the unpredictability of when garbage collection occurs.

**Syntax:**

java

CopyEdit

@Override

protected void finalize() throws Throwable {

System.out.println("Object is being garbage collected.");

}

**Example:**

java

CopyEdit

public class FinalizeExample {

public static void main(String[] args) {

FinalizeExample obj = new FinalizeExample();

obj = null; // Make the object eligible for garbage collection

System.gc(); // Request garbage collection

}

@Override

protected void finalize() throws Throwable {

System.out.println("Finalize method called before object destruction.");

}

}

**Output (example):**

sql

CopyEdit

Finalize method called before object destruction.

Explain super keyword?

A,

**When and Why Do We Use super?**

1. **Access Parent Class Members:**
   * In cases where a subclass has variables or methods with the same name as those in the parent class, super is used to explicitly refer to the parent class members, avoiding ambiguity.
2. **Call Parent Class Constructors:**
   * When creating an object of a subclass, the constructor of the parent class is called first. super is used to explicitly invoke a specific constructor in the parent class.
   * This is useful when the parent class does not have a no-argument constructor or when specific parameters need to be passed to the parent class.
3. **Override Parent Class Methods:**
   * If a method in the subclass overrides a method from the parent class, super can be used within the overriding method to call the original method from the parent class.

Ans,

class Parent {

int num = 100;

}

class Child extends Parent {

int num = 200;

void display() {

System.out.println("Child class variable: " + num); // Refers to Child class variable

System.out.println("Parent class variable: " + super.num); // Refers to Parent class variable

}

}

public class SuperExample1 {

public static void main(String[] args) {

Child obj = new Child();

obj.display();

}

}

What is serializable?  
a,

**Serializable:**

* **Definition**: The Serializable interface in Java is a marker interface (it doesn't contain any methods) that indicates a class can be serialized. Serialization is the process of converting an object's state (its instance variables) into a byte stream so that it can be saved to a file, sent over a network, or stored in memory.

package javafiles;  
import java.io.\*;  
  
public class SerialDemo {  
 public static void main(String[] args) throws Exception {  
 Save obj = new Save();  
 obj.i = 4;  
 // Serialization: Writing object to file  
 File f = new File("Obj.txt");  
 FileOutputStream fos = new FileOutputStream(f);  
 ObjectOutputStream oos = new ObjectOutputStream(fos);  
 oos.writeObject(obj);  
 oos.close();  
 // Deserialization: Reading object from file  
 FileInputStream fis = new FileInputStream(f);  
 ObjectInputStream ois = new ObjectInputStream(fis);  
 Save obj1 = (Save) ois.readObject();  
 ois.close();  
 // Printing the deserialized object's value  
 System.*out*.println("Value of Obj1: " + obj1.i);  
 }  
};  
class Save implements Serializable { // Implement Serializable interface  
 int i;  
}

5. What is the transient keyword in Java?

• Used to indicate that a variable should not be serialized.

**transient:**

* **Definition**: The transient keyword in Java is used to indicate that a field in a class should not be serialized. When an object is serialized, any fields marked as transient are ignored and not written to the output stream.

Example:

package javafiles;  
import java.io.\*;  
  
public class Serialtransient {  
 public static void main(String[] args) throws Exception {  
 Save1 obj = new Save1();  
 obj.i = 4;  
 obj.j = 10; // Adding a new field `j`  
  
 // Serialization: Writing object to file  
 File f = new File("Obj1.txt");  
 FileOutputStream fos = new FileOutputStream(f);  
 ObjectOutputStream oos = new ObjectOutputStream(fos);  
 oos.writeObject(obj);  
 oos.close();  
  
 // Deserialization: Reading object from file  
 FileInputStream fis = new FileInputStream(f);  
 ObjectInputStream ois = new ObjectInputStream(fis);  
 Save1 obj1 = (Save1) ois.readObject();  
 ois.close();  
  
 // Printing the deserialized object's values  
 System.*out*.println("Value of obj1.i: " + obj1.i); // This will be printed  
 System.*out*.println("Value of obj1.j: " + obj1.j); // This will be 0 because `j` is transient  
 }  
};  
  
class Save1 implements Serializable {  
 int i;  
 transient int j; // Marking field `j` as transient  
}

**🔥 Common Constructor-Related Interview Questions**

**1. What is the purpose of a constructor in Java?**

**Answer:**  
To initialize objects of a class. It sets default or user-defined values when an object is created.

**2. What is constructor overloading?**

**Answer:**  
Creating multiple constructors with different parameter lists in the same class.

public class Box {

Box() {}

Box(int a) {}

Box(int a, int b) {}

}

**3. Can constructors be private in Java? Why use them?**

**Answer:**  
Yes, often used in **Singleton Pattern** to restrict object creation.

**4. Can a constructor be final, static, or abstract?**

**Answer:**

| **Keyword** | **Allowed in Constructor?** | **Reason** |
| --- | --- | --- |
| final | ❌ | Constructors can't be inherited or overridden |
| static | ❌ | Constructors are not tied to the class, but to objects |
| abstract | ❌ | Constructors can’t be abstract since they must be implemented |

**5. What happens if a class has only a parameterized constructor? Can you still use new ClassName()?**

**Answer:**  
No. If you define a parameterized constructor and no default one, Java **does not provide** a default constructor. So, new ClassName() would cause a **compile-time error** unless you define a no-arg constructor explicitly.

**6. What’s the difference between constructor and method?**

| **Feature** | **Constructor** | **Method** |
| --- | --- | --- |
| Purpose | Initializes object | Performs action |
| Name | Same as class | Can be anything |
| Return type | None | Has return type |
| Called | Automatically on new | Manually |

**7. Can we call a constructor explicitly?**

**Answer:**  
Not directly like a method. It’s automatically invoked using new. However, constructors can be called **from other constructors** using this().

**8. What if we make the constructor protected or package-private?**

**Answer:**  
You restrict object creation:

* protected allows subclass/within-package instantiation.
* Default (no modifier) allows instantiation within the same package.

**🧠 Pro Tip (Advanced Interviews)**

You can use constructors in combination with **Builder Pattern**, **Immutable classes**, and **Dependency Injection** for cleaner, testable code. For example:

public class User {

private final String name;

private final int age;

public User(String name, int age) {

this.name = name;

this.age = age;

}

}

Immutable classes rely heavily on **constructors + final fields**.

**⚡ Summary Table**

| **Type** | **Syntax** | **Use** |
| --- | --- | --- |
| Default | ClassName() {} | Basic init |
| Parameterized | ClassName(int x) | Custom init |
| Copy | ClassName(ClassName obj) | Cloning |
| Private | private ClassName() {} | Singleton |
| Chained | this() | Constructor reuse |

Want to follow up with **object creation flow**, **memory model**, or **design patterns using constructors** (like Singleton, Builder)?

**Important Interview Questions based on oops:**

**Q: What is Object-Oriented Programming? What are its advantages?**  
**A:** OOP is a programming paradigm that uses objects and classes to structure software. Its advantages include:

* + Reusability through inheritance.
  + Improved code readability and maintainability via encapsulation.
  + Flexibility using polymorphism.

**Q: What are classes and objects?**  
**A:**

* + A **class** is a blueprint for objects (e.g., Car).
  + An **object** is an instance of a class (e.g., Car myCar = new Car();).

**Q: what is inheritance OOP?**  
**A:** process of sharing properties of one class with another

For example:

class Parent { void display() { } }

class Child extends Parent { }

Q.what is polymorphism?

A,one method acting in diffent forms.

Ex:man can act as husband,son,father.

Q,what is encapsulation ?

A,it is process of binding data as getters and and setters.

What is abstraction ?

**Abstract Classes**: These are classes that cannot be instantiated on their own and are meant to be subclassed. They can have both abstract methods (without a body) and concrete methods (with a body).

**What is interface**?

**A, Interfaces: Interfaces are a way to achieve abstraction in Java. They can only contain abstract methods (until Java 8, after which they can also contain default and static methods). A class that implements an interface must provide implementations for all its methods**

**Q: What is the difference between an abstract class and an interface?**  
**A:**

* + **Abstract class**: Can have abstract and concrete methods; supports constructors.
  + **Interface**: Only abstract methods (until Java 8), multiple inheritance supported.

**Q, where we will use extends and implements ?**.

extends-class->class

Implements-interface->class

**Q: Explain access modifiers in OOP.**  
**A:**

* + **Private**: Accessible only within the same class.
  + **Default**: Accessible within the same package.
  + **Protected**: Accessible within the same package and subclasses.
  + **Public**: Accessible everywhere.

**Q: What is the diamond problem in inheritance? How is it resolved in Java?**  
**A:**

* + The diamond problem occurs in multiple inheritance when a class inherits from two classes with the same method, causing ambiguity.
  + In Java, it is resolved by using interfaces instead of classes. If a conflict arises, you must explicitly specify which method to use.

**Q: What is the difference between composition and inheritance? Which is preferred?**  
**A:**

* + **Inheritance**: A "is-a" relationship (e.g., Dog is an Animal).
  + **Composition**: A "has-a" relationship (e.g., Car has an Engine).
  + **Preferred**: Composition is preferred for flexibility and reduced coupling.

1,what is interface?

A,

An **interface**

It specifies a set of methods that a class must provide but does not include any implementation details. Interfaces are a key part of object-oriented programming and are used to achieve abstraction and enforce a consistent structure across different classes.

**Key Features of Interfaces**

1. **Abstraction**: Interfaces provide a way to define abstract methods (methods without a body) that must be implemented by a class.
2. **No Implementation**: Methods in an interface do not have bodies (implementation); only the method signatures are defined.
3. **Multiple Inheritance**: Interfaces allow a class to implement multiple interfaces, enabling multiple inheritance-like behavior in languages that don't support it directly.
4. **Polymorphism**: Interfaces enable polymorphism by allowing different classes to be accessed through the same interface type.
5. **Standardization**: Interfaces ensure a consistent structure across classes, making it easier to maintain and extend code.

In below code eat added in interface only bit it can be called when a class is instatntiated also.

Ex:eat().

Code:

// Define the interface

public interface Animal {

void makeSound(); // Abstract method

void eat();

}

// Implement the interface in the Dog class

public class Dog implements Animal {

@Override

public void makeSound() {

System.out.println("Woof!");

}

@Override

public void eat() {

System.out.println("Dog is eating.");

}

}

// Implement the interface in the Cat class

public class Cat implements Animal {

@Override

public void makeSound() {

System.out.println("Meow!");

}

@Override

public void eat() {

System.out.println("Cat is eating.");

}

}

// Test the interface and its implementations

public class Main {

public static void main(String[] args) {

Animal dog = new Dog(); // Polymorphism

Animal cat = new Cat();

dog.makeSound(); // Output: Woof!

dog.eat(); // Output: Dog is eating.

cat.makeSound(); // Output: Meow!

cat.eat(); // Output: Cat is eating.

}

}

2,what is abstract class?

A, An **abstract class**

It serves as a blueprint for other classes and can contain both abstract methods (methods without implementation) and concrete methods (methods with implementation). Subclasses must provide implementations for the abstract methods.

Code:

// Define an abstract class

abstract class Animal {

// Abstract method (no body)

abstract void makeSound();

// Concrete method (has a body)

void eat() {

System.out.println("This animal is eating.");

}

}

// Subclass Dog extends the abstract class Animal

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Woof!");

}

}

// Subclass Cat extends the abstract class Animal

class Cat extends Animal {

@Override

void makeSound() {

System.out.println("Meow!");

}

}

// Test the abstract class and its subclasses

public class Main {

public static void main(String[] args) {

Animal dog = new Dog(); // Polymorphism

Animal cat = new Cat();

dog.makeSound(); // Output: Woof!

dog.eat(); // Output: This animal is eating.

cat.makeSound(); // Output: Meow!

cat.eat(); // Output: This animal is eating.

}

3,if we use static or default methos in interface it is equal to abstraction then what is difference ?

A,

**Key Differences Between Interface and Abstract Class**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| **Inheritance** | A class can inherit only one abstract class (single inheritance). | A class can implement multiple interfaces (multiple inheritance). |
|  |  |  |
| **Constructors** | Can have constructors. | Cannot have constructors. |
|  |  |  |
|  |  |  |
| **When to Use** | Use when you want to share code among related classes. | Use to define a contract or capability for unrelated classes. |
| **State (Fields)** | Can maintain state (fields that are non-final). | Cannot maintain state (fields are constants). |
|  |  |  |

**15,comparable vs comparator in interface ?**

**A, 1. Comparable Interface**

* **Definition: Belongs to the java.lang package.**
* **Purpose: Used to define the natural ordering of objects in a class.**
* **How to Use:**
  + **The class itself implements Comparable.**
  + **It overrides the compareTo method to define the default comparison logic.**

**Key Features of Comparable:**

1. **Single Sorting Logic:**
   * **Can only define one default sorting order for a class.**
2. **Modify the Class:**
   * **Requires modification of the class whose objects need to be sorted.**

**class Employee implements Comparable<Employee> {**

**int id;**

**String name;**

**public Employee(int id, String name) {**

**this.id = id;**

**this.name = name;**

**}**

**@Override**

**public int compareTo(Employee other) {**

**return this.id - other.id; // Natural order by id**

**}**

**}**

**// Usage**

**List<Employee> employees = new ArrayList<>();**

**employees.add(new Employee(2, "Alice"));**

**employees.add(new Employee(1, "Bob"));**

**Collections.sort(employees); // Sorts by id**

**Comparator Interface**

* **Definition: Belongs to the java.util package.**
* **Purpose: Used to define custom sorting logic outside of the class.**
* **How to Use:**
  + **Create a separate class (or use a lambda/anonymous class) that implements Comparator.**
  + **It overrides the compare method to define the custom comparison logic.**

**class Employee {**

**int id;**

**String name;**

**public Employee(int id, String name) {**

**this.id = id;**

**this.name = name;**

**}**

**}**

**// Custom Comparator**

**class NameComparator implements Comparator<Employee> {**

**@Override**

**public int compare(Employee e1, Employee e2) {**

**return e1.name.compareTo(e2.name); // Custom order by name**

**}**

**}**

**// Usage**

**List<Employee> employees = new ArrayList<>();**

**employees.add(new Employee(2, "Alice"));**

**employees.add(new Employee(1, "Bob"));**

**Collections.sort(employees, new NameComparator()); // Sorts by name**

23,what is binding in polymorphism?

A, Binding in Polymorphism

Binding refers to the process of associating a method call with the method definition or implementation. In the context of polymorphism, it determines which method will be executed at runtime or compile-time.

Types of Binding

1. Static Binding (Early Binding):

• Occurs at compile-time.

• The method to be invoked is determined based on the reference type of the object.

• It is used for method overloading, private, static, and final methods since these methods cannot be overridden.

• Example:

class Demo {

void show() {

System.out.println("Static Binding Example");

}

}

public class Test {

public static void main(String[] args) {

Demo obj = new Demo();

obj.show(); // Resolved at compile-time

}

}

2. Dynamic Binding (Late Binding):

• Occurs at runtime.

• The method to be executed is determined based on the actual object (not the reference type).

• It is used for method overriding, allowing polymorphic behavior.

• Example:

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

void sound() {

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

}

}

public class Test {

public static void main(String[] args) {

Animal obj = new Dog(); // Reference type is Animal, but object is Dog

obj.sound(); // Resolved at runtime, output: "Dog barks"

}

}

Key Differences Between Static and Dynamic Binding

Aspect Static Binding Dynamic Binding

Timing Compile-time Runtime

Methods involved private, static, and final methods Overridden methods

Polymorphism Type Compile-time polymorphism Runtime polymorphism

Decision based on Reference type Object type

Polymorphism and Binding in Action

class Shape {

void draw() {

System.out.println("Drawing a shape");

}

}

class Circle extends Shape {

@Override

void draw() {

System.out.println("Drawing a circle");

}

}

class Test {

public static void main(String[] args) {

Shape s = new Circle(); // Polymorphism

s.draw(); // Dynamic binding, resolved at runtime

}

}

Output:

Drawing a circle

In the example above:

• The reference s is of type Shape.

• At runtime, the actual object Circle determines which draw() method to call.

Conclusion

• Binding in polymorphism is a mechanism that allows dynamic method invocation based on the object at runtime.

• Static binding supports method overloading, while dynamic binding supports method overriding.

Some more oops interview questions?

A,  
**Q8. What is meant by static polymorphism?** Static polymorphism, also known as compile-time polymorphism, is achieved through method overloading or operator overloading. It allows the same method to have different implementations based on the parameters passed to it. The method to be invoked is determined at compile time.

**Q9. What is meant by dynamic polymorphism?** Dynamic polymorphism, or runtime polymorphism, is achieved through method overriding. It allows a subclass to provide a specific implementation of a method that is already defined in its superclass. The method to be invoked is determined at runtime based on the object type.

**Q10. What is the difference between overloading and overriding?**

* **Overloading**: This is a compile-time polymorphism where multiple methods have the same name but different parameters (method signature).
* **Overriding**: This is a runtime polymorphism where a subclass provides a specific implementation of a method that is already defined in its superclass.

**Q11. What is method hiding?** Method hiding occurs when a subclass defines a static method with the same name and signature as a static method in its superclass. The method in the superclass is hidden by the method in the subclass.

**Q12. What is variable hiding?** Variable hiding happens when a subclass declares a variable with the same name as a variable in its superclass. The variable in the superclass is hidden by the variable in the subclass.

**Q13. What is an abstract class?** An abstract class is a class that cannot be instantiated and is meant to be subclassed. It can contain abstract methods (methods without implementation) that must be implemented by subclasses, as well as concrete methods (methods with implementation).

**Q14. What is an interface?** An interface is a reference type in Java that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields. The methods in interfaces are abstract by default.

**Q15. Advantages and disadvantages of using Abstraction?**

* **Advantages**:
  + Reduces complexity by hiding implementation details.
  + Increases code reusability and maintainability.
  + Enhances security by exposing only necessary details.
* **Disadvantages**:
  + Can lead to performance overhead due to additional layers of abstraction.
  + May increase the complexity of the codebase if overused.

**Q16. Is it possible for a single class to implement two different interfaces?** Yes, a single class can implement multiple interfaces in Java. This allows the class to inherit the abstract methods of multiple interfaces and provide implementations for them.

**Q17. Why are default methods allowed in interfaces?** Default methods were introduced in Java 8 to allow the addition of new methods to interfaces without breaking the existing implementations. They provide a default implementation that can be overridden by implementing classes.

**Q18. Inheritance - We have 3 classes A, B, and C. Class C extends Class B, and Class B extends Class A. Each class has a method add(). Is there a way to call A's add() method from class C?** Yes, you can call A's add() method from class C using the super keyword. Here is an example:

class A {

void add() {

System.out.println("A's add()");

}

}

class B extends A {

void add() {

System.out.println("B's add()");

}

}

class C extends B {

void add() {

super.add(); // Calls B's add()

((A)this).add(); // Calls A's add()

}

}

**Q21. What is the difference between composition and inheritance?**

* **Composition**: A design principle where a class is composed of one or more objects of other classes, allowing for flexible code reuse. It represents a "has-a" relationship.
* **Inheritance**: A mechanism where a new class inherits properties and behavior from an existing class, representing an "is-a" relationship.

**Q22. What are the differences between an abstract class and an interface?**

* **Abstract Class**:
  + Can have both abstract and concrete methods.
  + Can have instance variables.
  + Can provide constructors.
  + Supports single inheritance.
* **Interface**:
  + Can only have abstract methods (until Java 8, which introduced default and static methods).
  + Cannot have instance variables.
  + Cannot provide constructors.
  + Supports multiple inheritance.

**Q23. What is the use of the default keyword in interfaces?** The default keyword in interfaces allows the definition of default methods, which provide a default implementation that can be overridden by implementing classes. This helps in adding new methods to interfaces without breaking existing implementations.

**Q24. Polymorphism - In an online shopping system, you have different types of users such as customers and admins. Both users need to log in but have different post-login actions.** In this scenario, you can use polymorphism to define a common interface or abstract class for users with a login method. Each user type (customer, admin) can then provide its own implementation of the post-login actions.

**Q25. Composition - Design a system for a car dealership where cars can have different engines (e.g., petrol engine, electric engine). You want to model cars and engines such that engines can be easily replaced without needing to modify the car class.** In this scenario, you can use composition to design the system. Define an Engine interface with different implementations (PetrolEngine, ElectricEngine). The Car class will have an Engine object, allowing the engine to be replaced easily.

interface Engine {

void start();

}

class PetrolEngine implements Engine {

public void start() {

System.out.println("Petrol engine starting...");

}

}

class ElectricEngine implements Engine {

public void start() {

System.out.println("Electric engine starting...");

}

}

class Car {

private Engine engine;

public Car(Engine engine) {

this.engine = engine;

}

public void setEngine(Engine engine) {

this.engine = engine;

}

public void startCar() {

engine.start();

}

}

Some missing questions in oops?

Here are some additional object-oriented programming (OOP) interview questions that are missing from the list you provided, which can help in further understanding OOP concepts:

**1. What is the difference between deep copy and shallow copy in OOP?**

* **Answer**:
  + **Shallow Copy**: A shallow copy creates a new object but does not recursively copy nested objects. Instead, it just copies references to the original nested objects.
  + **Deep Copy**: A deep copy creates a new object and recursively copies all nested objects, ensuring that the copy is completely independent of the original.

**2. What is the use of the super keyword in Java?**

* **Answer**:
  + The super keyword in Java is used to refer to the superclass (parent class) of the current object. It can be used to call superclass methods, access superclass constructors, or reference superclass fields.

**3. What is method overloading and how does it differ from method overriding?**

* **Answer**:
  + **Method Overloading**: Occurs when two or more methods in the same class have the same name but different parameters (different number of parameters or parameter types). It is resolved at compile time (static polymorphism).
  + **Method Overriding**: Happens when a subclass provides its specific implementation of a method that is already defined in its superclass. It is resolved at runtime (dynamic polymorphism).

**4. What is the difference between a class and an object?**

* **Answer**:
  + **Class**: A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.
  + **Object**: An object is an instance of a class. It is created based on the class template and can hold specific values for its attributes and can invoke methods defined in the class.

**5. What is the purpose of the final keyword in OOP?**

* **Answer**:
  + **Final Variable**: A variable that cannot be reassigned after its initial value is set.
  + **Final Method**: A method that cannot be overridden by subclasses.
  + **Final Class**: A class that cannot be subclassed.

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

* **Answer**:
  + **this**: Refers to the current instance of the class. It is used to access instance variables, methods, and constructors within the class.
  + **super**: Refers to the superclass (parent class) and is used to call the superclass’s methods, constructors, and fields.

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

* **Answer**:
  + An inner class is a class defined within another class. It has access to the outer class's private members. Inner classes can be used for logical grouping of classes, to enhance encapsulation, or to implement event listeners.

**8. What is the difference between static and non-static methods in OOP?**

* **Answer**:
  + **Static Methods**: Belong to the class rather than to instances of the class. They are invoked without creating an object and can access only static variables and methods.
  + **Non-Static Methods**: Belong to instances of the class. They require an object to be invoked and can access both static and non-static variables.

**9. What is the concept of "has-a" and "is-a" relationship in OOP?**

* **Answer**:
  + **"is-a" Relationship**: Represents inheritance, where a subclass is a type of its superclass. For example, a Dog "is-a" Animal.
  + **"has-a" Relationship**: Represents composition, where a class contains references to other objects as its members. For example, a Car "has-a" Engine.

**10. What is the purpose of the instanceof operator in Java?**

* **Answer**:
  + The instanceof operator is used to test whether an object is an instance of a specific class or subclass. It returns true if the object is an instance of the class or any subclass of that class, and false otherwise.

**11. What is the difference between the equals() method and the == operator in Java?**

* **Answer**:
  + **== Operator**: Compares memory references (whether two references point to the same object).
  + **equals() Method**: Compares the actual contents of two objects. It is overridden in many classes (like String) to compare the actual values rather than the memory references.

**12. What is an anonymous class in Java?**

* **Answer**:
  + An anonymous class is a class defined without a name, usually used to instantiate a class for a single use. It is commonly used to implement interfaces or extend classes in a concise manner.

**13. What is a singleton class in OOP?**

* **Answer**:
  + A singleton class ensures that only one instance of the class is created during the entire runtime of the application. It provides a global point of access to that instance.

**14. What are the differences between composition and aggregation?**

* **Answer**:
  + **Composition**: Represents a "has-a" relationship, but the lifecycle of the contained objects is controlled by the container object. If the container object is destroyed, the contained objects are also destroyed.
  + **Aggregation**: Also represents a "has-a" relationship, but the contained objects can exist independently of the container object. The container object does not manage the lifecycle of the contained objects.

**15. What is method hiding in OOP?**

* **Answer**:
  + Method hiding occurs when a subclass defines a static method with the same signature as a static method in its superclass. The method in the superclass is hidden by the method in the subclass, and it is not an example of method overriding because static methods cannot be overridden.

**16. What is the clone() method in Java and how does it work?**

* **Answer**:
  + The clone() method is used to create a copy of an object. It is part of the Object class, and classes can override it to perform deep copying if needed. By default, the clone() method creates a shallow copy of the object.

**17. What is the difference between a constructor and a method in OOP?**

* **Answer**:
  + **Constructor**: Special method used to initialize an object when it is created. It has the same name as the class and does not have a return type.
  + **Method**: A function defined in a class to perform some operations. It can have a return type and may be called after an object is created.

**18. What are the advantages of using OOP?**

* **Answer**:
  + Code reusability via inheritance.
  + Modular structure due to encapsulation.
  + Easier to maintain and scale due to abstraction.
  + Flexibility in programming via polymorphism.
  + Improved code organization and readability.

**19. Can we have an abstract class with a constructor?**

* **Answer**: Yes, abstract classes can have constructors. While an abstract class cannot be instantiated, its constructor is called when an object of a subclass is created, allowing initialization of fields in the abstract class.

**20. What is the difference between this() and super() in constructors?**

* **Answer**:
  + **this()**: Refers to the current class's constructor and is used to call another constructor in the same class.
  + **super()**: Refers to the superclass's constructor and is used to call a constructor in the parent class.

These additional questions cover a wide range of topics in OOP and will help deepen your understanding for interviews or general knowledge of OOP principles.

**Important Interview Questions based on modifiers:**

**1. How many types of modifiers in Java?**

Ans: There are two types of modifiers in Java:

* Access modifiers
* Non- access modifiers

**2. What are Access modifiers in java?**

Ans: [Access Modifiers](https://www.scientecheasy.com/2020/06/access-modifiers-in-java.html/) are those modifiers that are used to restrict the visibility of classes, fields, methods, and constructors.

Java supports four types of access modifiers:

* Private
* Default
* Protected
* Public

a) **Private:** Private members of a class can be accessed only within the class. It cannot be accessed from outside the class.

b) **Default:** Default members of a class are accessible within the same package due to visible only within the package. They cannot be accessed from outside the package.

c) **Protected:** Protected members of a class are visible within the package. Therefore, we can only access within the package but can be accessed to the subclasses outside the package through the inheritance only.

d) **Public:** Public members are visible anywhere. So, we can access it anywhere within or outside the package.

**3. Can we have a private constructor in Java?**

Ans: Yes, we can have a private constructor in Java.

* The private constructor is used when we do not want to create the object of that class.
* We cannot create a subclass of that class.
* It is also used in Singleton design and Factory method design patterns.

**4. Which access modifiers can be used with a class?**

Ans: Public and Default access modifiers can be used with a class.

**5. Can we instantiate the object of derived class if the parent constructor is protected?**

Ans: No, we cannot instantiate the object of derived class if the parent constructor is protected.

**6. What are non-access modifiers in Java?**

Ans: There are four [non-access modifiers in Java](https://www.scientecheasy.com/2020/06/non-access-modifiers-in-java.html/). They are as follows:

* Static
* Final
* Abstract
* Synchronized

a) **Static:** This modifier is used to check that a member is a class member or instance member. If you declare a class as static, this class will be executed first.

b) **Final:** Final is a keyword that is used to restrict the users. In other words, it is used to restrict further modification of a class, field, or method. If a class is declared as ‘final’, the class cannot be subclassed.

c) **Abstract:** Abstract is a keyword that is used with a class or a method. An abstract class or abstract method is used for further modification. If a class is declared as ‘abstract’, the class cannot be instantiated.

d) **Synchronized:** It is used to achieve thread safeness. Only one thread can enter in a synchronized method or block at a given time.

**7. Can we declare a top-level class as private?**

Ans: No, we cannot declare a top-level or outer class as private. It can have either “public” or no modifier.

If you declare a top-level class as a private, the compiler will complain that the “modifier private is not allowed here” but an inner class can be private.

Inner class means class as a member of another class. The same is the case with protected.

**8. Can we declare an abstract method as private?**

Ans: No, an abstract method cannot be private. They must be declared as public, protected, or default so that they can be further modified.

**9. Can we declare a top-level class as protected?**

Ans: No, we cannot declare a class as protected. An inner class can be protected but not an outer class.

**10. Can a method or a class be final and abstract at the same time?**

Ans: No, it is not possible. A class or a method cannot be final or abstract at the same time because the final method or final class cannot be further modified whereas an abstract class or an abstract method must be modified further.

**11. Why are access modifiers used?**

Ans: The access modifiers are used to restrict the access of a class and its members. Access modifiers are used to reduce the visibility of the members of a class.

**12. Which is the default access modifier?**

Ans: ‘Internal’ is the default access modifier if no access modifier is mentioned with a class or its members. Internal is a keyword that is used for declaration.

**13. What is the default access modifier for Interface?**

Ans: The public is the default access modifier for the interface. No other access modifier is allowed for them

**14. Can we define struct members as protected?**

Ans: No, we cannot define struct members as protected because struct does not support inheritance.

**15. What is the default access specifier for a class, an interface, and struct declared directly with a namespace?**

Ans: Internal

**16. What is access modifier for enumeration?**

Ans: Enumeration members are always public. No other access modifiers are allowed.

**17. What is the role of private constructor in Java?**

Ans: If you declare any constructor of a class as private, we cannot create the object of a class from outside the class. In other words, we cannot create the subclass of that class.

**18. Which is the least restrictive access modifier in Java?**

Ans: Public

**19. Which is the most restrictive access modifier in Java?**

Ans: Private

**20. Which access modifier is also known as Universal access modifier?**

Ans: Public

**21. Explain visibility control in Java.**

Ans: Visibility control in Java is implemented by the access modifiers.

**Important Interview Questions based on Static Keyword**

**1. What is static in Java?**

Ans: In Java, static is a keyword that is used for memory management mainly. Static means single copy storage for variables or methods.

The members that are marked with the static keyword inside a class are called static members.

**2. Can we access static members if no instance of the class is constructed?**

Ans: Yes, we can access the static members if no instance of class exists because they are not tied to a specific instance. They are shared across all instances of the class.

**3. Can we apply static keyword with a top-level class?**

Ans: No, static keyword cannot be applied with outer or top-level class but an inner class can be static.

**4. Will the following code snippet compile successfully? If yes, what is the output of the following program?**

public class Myclass

{

private int x = 10;

static int m1() {

int y = x;

return y;

}

public static void main(String[] args) {

m1();

}

}

JAVACopy code

Ans: No, the above code will not be compiled because x is an instance variable and instance member cannot be accessed from static region.

**5. Identify the error in the following code snippet. If there is no error then what will be the output of the program?**

public class Myclass

{

private int x = 10;

static int m1()

{

Myclass obj = new Myclass();

int y = obj.x;

return y;

}

public static void main(String[] args) {

System.out.println(m1());

}

}

JAVACopy code

Ans: There is no error in the above code snippet. Output: 10.

**6. What is the main use of static keyword in java?**

Ans: The main use of java static keyword is as follows:

* The static keyword can be used when we want to access the data, method, or block of the class without any object creation.
* It can be used to make the programs more memory efficient.

**7. Can we mark a local variable as static?**

Ans: No, we cannot mark a local variable with a static keyword.

**8. When does a static variable get memory?**

Ans: When a class is loaded into the memory at runtime, the static variable is created and initialized into the common memory location only once.

**9. What will be the output of the following program?**

public class Myclass

{

static int a = 20;

static int b = 30;

static int c = 40;

Myclass()

{

a = 200;

}

static void m1() {

b = 300;

}

static {

c = 400;

}

public static void main(String[] args) {

System.out.println(a);

System.out.println(b);

System.out.println(c);

}

}

JAVACopy code

Ans: Output: 20, 30, 400.

**10. What will be the output of the following code?**

public class Myclass {

static int a = 20;

Myclass() {

a = 200;

}

public static void main(String[] args) {

new Myclass();

System.out.println(a);

}

}

JAVACopy code

Ans: Output: 200.

**11. In which part of memory static variables are stored?**

Ans: All static variables are stored in PermGen space of the heap memory.

**12. How static variable is different from the instance variable?**

Ans: The difference between static variable and instance variable is as follows:

a) A static variable is also called class variable whereas, an instance variable is also called non-static variable.

b) Class variable can be accessed inside a static block, instance block, static method, instance method, and method of the inner class whereas, instance variable can be accessed only inside the instance members, and method of the inner class.

c) Class variable is always resolved during compile time whereas, instance variable is resolved during the runtime.

4) Static variable cannot be serialized in Java whereas, instance variable can be serialized.

**13. Will the following code snippet compile fine? If yes, what will be the output of the following program?**

public class Myclass {

static int a = 20;

Myclass() {

a++;

}

void m1() {

a++;

System.out.println(a);

}

public static void main(String[] args)

{

Myclass obj = new Myclass();

Myclass obj2 = new Myclass();

Myclass obj3 = new Myclass();

obj3.m1();

}

}

JAVACopy code

Ans: Output: 24.

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

Ans: When a method is declared with the keyword ‘static’, it is called static method in java.

**15. Why is a static method also called a class method?**

Ans: A static method is also called a class method because it ties to a class rather than an individual instance of a class. Therefore, we need not to create an object of the class to call and execute static method.

**16. Can we access static members (such as static variables and static methods) from an instance method?**

Ans: Yes, we can access static members from an instance method in java.

**17. Is it possible to access instance members from a static method?**

Ans: No, it is not possible to access instance members like instance variable and instance method from a static method.

**18. Identify the error in the following code.**

public class Test

{

Test() {

m2();

}

void m1() {

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

}

static void m2() {

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

m1();

}

public static void main(String[] args)

{

new Test();

}

}

JAVACopy code

Ans: There is an error inside the static method because we cannot make a static reference to the non-static method m1() from the type Test.

**19. Will the below code compile successfully? If yes, what will be the output of the following program?**

public class Test

{

Test(Test t) {

m1();

System.out.println("Constructor");

}

void m1() {

m2();

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

}

static void m2() {

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

}

public static void main(String[] args)

{

new Test(null);

}

}

JAVACopy code

Ans: Yes, the above code will be compiled successfully. There is no problem. Output: Static method, Instance method, Constructor.

**20. Is there any error in the below code snippet? If yes, identify the error and give the reason behind it.**

public class Test

{

void m1(Test test) {

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

}

static void m1(Test t) {

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

}

}

JAVACopy code

Ans: Yes, Duplicate method error. This is because we cannot declare a static method and instance method with the same signature in the same class.

**21. What is the difference between static method and instance method?**

Ans: Go to this tutorial: [*Static method in Java*](https://www.scientecheasy.com/2020/06/java-static-method.html/)

**22. Can we have a static method in an interface?**

Ans: Yes, from Java 8 and onwards, the interface allows to define a static method with body.

**23. Can we use this or super keyword in static method in Java?**

Ans: No, In the entire core java, this and super keywords are not allowed inside the static region.

**24. Is it possible to overload static methods in a class?**

Ans: Yes, we can overload static methods but override them. This is because they are bound with class, not instance.

**25. Is it possible to override static methods of a class?**

Ans: No, we cannot override static methods because static methods belong to a class, not individual objects, and are resolved at compile time by java compiler.

For more detail, go to this tutorial: [*Can we override static methods in Java?*](https://www.scientecheasy.com/2019/05/can-we-override-static-method-in-java.html/)

**26. Can we override an instance method as static?**

Ans: No.

**27. What is the output of the following program code?**

public class Myclass {

static int a = 20;

static void m2() {

a++;

}

public static void main(String[] args) {

System.out.println(a);

}

}

JAVACopy code

Ans: Output: 20.

**28. Why static block is executed before the main method in java?**

Ans: When the dot class file is loaded into memory, static block is executed. After executing the static block, JVM calls the main method to start execution. Therefore, static block is executed before the main method.

**29. What is the output of the following program below?**

public class Myclass {

static int a = 20;

static {

a++;

}

{

a++;

System.out.println(a);

}

public static void main(String[] args)

{

Myclass obj = new Myclass();

Myclass obj2 = new Myclass();

Myclass obj3 = new Myclass();

}

}

JAVACopy code

Ans: Output: 22, 23, 24.

**30. What is the use of static block in java?**

Ans: A static block can be used when

* we want to write that logic inside static block that is executed during the class loading.
* we want to change the default value of static variables.
* we want to initialize static variable of the class.

**31. What is the output of the following code snippet?**

public class Myclass {

Myclass() {

System.out.println("constructor");

}

static void m1() {

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

}

void m2(){

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

}

static {

System.out.println("static block");

}

{

System.out.println("instance block");

}

public static void main(String[] args)

{

Myclass obj = new Myclass();

m1();

obj.m2();

}

}

JAVACopy code

Ans: Output: static block, instance block, constructor, static method, instance method.

**32. How static block is different from an instance block in java?**

Ans: Static block is different from an instance block by the following key points:

a) Static block is also called a static initialization block whereas instance block is also called instance initialization block or non-static block.

b) Static block gets executed before the instance block whereas, instance block executes after the static block.

c) Only static variables can be accessed inside the static block whereas, both static and non-static variables can be accessed inside the instance block.

d) Static block executes when the class is loaded into the memory whereas instance block executes only when an instance of the class is created.

e) We cannot use this keyword inside the static block whereas this keyword can be used in the instance block.

**33. Can we declare a static block inside a method?**

Ans: No, we cannot declare a static block inside a method.

**34. What will be the output of the following code snippet after execution?**

public class Myclass {

static {

System.out.println("static block");

}

{

System.out.println("instance block");

}

public static void main(String[] args) {

Myclass obj = new Myclass();

Myclass obj2 = new Myclass();

Myclass obj3 = new Myclass();

}

}

JAVACopy code

Ans: Output: static block, instance block, instance block, instance block.

**35. Will the following code snippet compile fine? If yes, what will be output after execution?**

public class Myclass {

private static int x = 10;

static {

x++;

}

static {

++x;

}

{

x--;

}

public static void main(String[] args) {

Myclass obj = new Myclass();

Myclass obj2 = new Myclass();

Myclass obj3 = new Myclass();

System.out.println(x);

}

}

JAVACopy code

Ans: Yes, the code will be compiled fine. The output is 9.

**Constructor Interview Questions in Java**

**1. What is a constructor in Java?**

Ans: A constructor is a block of code, similar to a method that is used to initialize the state of an object (i.e. instance variable) in a class through a new operator. It is automatically called and executed at the time of object creation by JVM.

**2. What is the main objective of a constructor in java?**

Or, Why do we need a constructor in a class as a member?

Ans: The main objective/purpose of a constructor in java is to initialize instance variables in a class (or to set the initial state of an object).

**3. When a constructor is called/invoked in Java?**

Ans: The constructor of a class is called every time an object is created with a new keyword. For example, in the below code, two objects of class are created with the new keyword, therefore, constructor is called two times.

public class Test {

Test() {

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

}

public static void main(String[ ] args){

Test t1 = new Test();

Test t2 = new Test();

}

}

JAVACopy code

**4. Is it possible for a class to have multiple constructors in java?**

Ans: Yes, a class can have multiple constructors with different parameters. Which constructor gets called for object creation depends on the arguments passed while constructing different objects.

**5. Does a constructor return any value?**

Ans: The constructor can not have any return type even void also because if there is a return type then JVM would consider as a method, not a constructor.

**6. Can a constructor be marked with the final keyword?**

Ans: No, a constructor cannot be marked with the final keyword.

**7. Is it possible to inherit a constructor?**

Ans: No, a constructor cannot be inherited in java.

**8. Can we use this() and super() inside the constructor?**

Ans: No, this() and super() cannot be used together inside the constructor.

**9. Will the below code compile successfully? If not, why?**

class Test {

Test() {

}

public void display() {

Test();

}

}

JAVACopy code

Ans: On compilation of the above code, compile-time error will generate: The method Test() is undefined for the type Test. This is because it is illegal to invoke a constructor like this.

**10. What are possible access modifiers that can be marked for a constructor?**

Ans: The possible access modifiers for a constructor is as follows:

a) **private**: The constructor marked with private can be accessed only from its class.

b) **protected**: The constructor declared with protected access modifier is accessible from any class which resides in the same package.

c) **public**: The public constructor is accessible from any class within or outside the package.

**11. Is it possible to invoke a constructor of a class more than once for an object?**

Ans: No, it is not possible to call a constructor of a class more than once for an object. It is invoked only once per object at the time of object creation.

**12. Does a constructor of the class get called, before or after creating an object?**

Ans: A constructor gets called concurrently when the object creation is going on. JVM first allots memory space for the object in the heap and then executes the constructor to initialize instance variables. By the time object creation is completed, the execution of a constructor is completed.

**13. How many types of constructors are in Java?**

Ans: There are two types of constructors in java that are as follows:

* Default constructor (Non-parameterized constructor)
* Parameterized constructor

**14. What is a default constructor?**

Ans: A constructor that takes no parameter is called default constructor or parameterized constructor. For example:

A a = new A(); *// It will create an object of class A by invoking default constructor.*

JAVACopy code

**15. What is a parameterized constructor?**

Ans: A constructor that contains one or more parameters is called parameterized constructor. The parameterized constructor allows us to initialize different values to distinct objects.

**16. What is the main purpose of default constructor in java?**

Ans: The main purpose of default constructor is to initialize default values (null or zero value) to the objects.

Java compiler creates a default constructor at compile time only if there is no constructor in a class. After constructing the default constructor, default values are initialized to objects.

**17. Is it necessary to define a constructor as the same name as the name of class?**

Ans: Yes, a constructor must have the same name as that of class name. If the name of constructor is different, Java compiler will treat it as a normal method.

**18. Give an example that proves the definition of constructor.**  
Ans:

public class Test

{

String name;

int age;

Test() *// constructor function.*

{

name = "John";

age = 20;

}

}

JAVACopy code

**19. Write a program to define a constructor and pass data through parameters.**

Ans:

public class Person

{

String name;

int age;

Person(String n)

{

name = n;

age = 25;

}

}

JAVACopy code

**20. Will the below code compile successfully? If yes, what will be the output of the following program?**

public class Test

{

Test() {

System.out.println("Calling default constructor");

}

Test(Test test) {

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

}

public static void main(String[] args)

{

Test t = new Test(new Test());

}

}

JAVACopy code

Ans: Yes, the above code will be compiled successfully. The output of the program is as follows:

* Calling default constructor
* Calling parameterized constructor

**21. Is it possible to call a constructor from another constructor if multiple constructors are defined in a class?**

Ans: If multiple constructors are defined inside a class, it is possible to call a constructor from another constructor using this keyword.

**22. What will be the output of the following program?**

public class Test

{

Test(Object object) {

System.out.println("Hello");

}

Test(Test test) {

System.out.println("World");

}

public static void main(String[] args)

{

Test t = new Test(null);

}

}

JAVACopy code

Ans: Output: world.

**23. What is the use of constructor in java?**

Ans: The use of constructor in java is:

a) to assign the default value of instance variables.  
b) to execute a particular code at the time of object creation, we can write them inside the constructor.

**24. Can we declare a constructor as private?**

Ans: Yes, we can declare a constructor with a private access modifier. It is mainly done not to allow users to create an object of class from outside of the class. Basically, we use a [*private constructor*](https://www.scientecheasy.com/2021/02/java-private-constructor.html/) in a singleton design pattern.

**25. When does Java compiler define the default constructor?**

Ans: Java compiler defines a default constructor only if there is no explicit constructor declared by the programmer.

**26. Why a constructor defined by Java compiler is always called as default constructor?**

Ans: A constructor defined by Java compiler is always called as default constructor because it obtains all its default properties from its class. They are:

a) Its access modifier is same as its class access modifier.  
b) Its name is same as class name.  
c) It has no parameters and logic.

**27. What is constructor overloading in Java?**

Ans: [*Constructor overloading*](https://www.scientecheasy.com/2020/06/java-constructor-overloading.html/) is a technique in which a class can have more than one constructor having the same name but different parameter lists. A parameter list consists of order and types of arguments.

**28. How does Java compiler differentiate among multiple constructors in a class?**

Ans: Java compiler differentiates multiple constructors based on the number of parameter lists and their types. Therefore, the signature of each constructor must be different.

**29. What is the use of overloading constructor in java?**

Ans: Overloading constructors can be used for performing different tasks based on different data.

**30. What is the output of the below code snippet?**

public class Test

{

Test(int i) {

System.out.println("Hello");

}

Test(long l) {

System.out.println("World");

}

public static void main(String[] args)

{

Test t = new Test(20);

}

}

JAVACopy code

Ans: Output: Hello.

**31. What will the output of the following program?**

public class Test

{

Test(int i) {

System.out.println("Hello");

}

Test(Test test) {

System.out.println("World");

}

public static void main(String[] args)

{

Test t = new Test(new Test(20));

}

}

JAVACopy code

Ans: Output: Hello, World.

**32. Will the below code snippet compile successfully? If yes, what will be the output of the following program?**

public class Test

{

Test() {

this(20);

System.out.println("One");

}

Test(int i) {

this(null);

System.out.println("Two");

}

Test(Test test) {

System.out.println("Three");

}

public static void main(String[] args)

{

new Test();

}

}

JAVACopy code

Ans: Output: Three, Two, and One.

**33. What are the advantages of constructor overloading in Java?**

Ans: There are several advantages of using constructor overloading in a program. They are as follows:

* Java constructor overloading helps to achieve static polymorphism.
* The main advantage of constructor overloading is to allow an instance of a class to be initialized in various ways.
* It allows defining of the multiple constructors of a class with different signatures.
* It helps to perform different tasks for different purposes.

**34. Can we have more than one constructor with same signature in a class?**

Ans: No, we cannot have more than one constructor with same signature in a class.

**35. What is constructor chaining in java?**

Ans: [*Constructor chaining in Java*](https://www.scientecheasy.com/2020/06/java-constructor-chaining.html/) is a technique of calling one constructor from within another constructor by using this and super keywords.

**36. Why do we use/need constructor chaining?**

Ans: Constructor chaining can be used when we want to perform multiple tasks in a separate constructor for each task and make their order by chaining. It is useful to make the program more readable and easy to understand for everyone.

**37. How to call one constructor from another constructor in Java?**

Ans: Using this(), we can call the current class constructor within the “same class”.

Using super(), we can call the superclass constructor from the “base class”.

**38. What is a copy constructor?**

Ans: A constructor which is used to copy the data of one object to another object of the same class type is called copy constructor in Java.

**39. Can we create an object of class within the same class if a constructor is marked with private?**

Ans: Yes, we can create an object of class within the same class if constructor is marked with private but not outside the class.

**40. Can class be extended when a constructor is declared private?**

Ans: No.

**41. What is the difference between constructor and method?**

Ans: Refer to this tutorial: [*Constructor in Java*](https://www.scientecheasy.com/2020/06/constructor-in-java.html/)

Hope that this tutorial has covered almost all the important constructor interview questions in java with the best possible answers. I hope that you will have understood all answers to constructor interview questions nicely.

**Java Final Interview Questions and Answers**

**1. What is final in Java?**

Ans: Final is a keyword that is used to restrict the user in Java programming. It can be applied with variables, methods, or classes. It is a non-access modifier.

**2. Why do we need final keyword in Java?**  
Or, what is the use of final keyword in Java?

Ans: [*Final keyword*](https://www.scientecheasy.com/2020/07/final-keyword-in-java.html/) is used in Java program for three different purposes that are as follows:

* To declare a constant or to stop the value change. For example, private final int const\_value = 50;
* To prevent inheritance. When a class is marked with final keyword, it cannot be subclassed. String, Integer, and other wrapper classes are examples of final classes.
* To prevent a method from being overridden. When a method is marked with final keyword, it cannot be overridden by subclasses. A method declared with final keyword is faster than any other method because it is resolved at compile time.

**3. Why a constructor cannot be final in Java?**

Ans: If a method is marked as final it means we do not want any class to override it. As per Java Language Specification, a constructor cannot be overridden. So, there is no use in declaring a constructor as final.

**4. What is final variable in Java?**

Ans: A variable declared as final is known as a final variable in Java. Once the final variable is initialized, the value of the final variable can never be changed. In other words, we cannot be assigned a new value.

**5. When to use final variable in java?**

Ans: A final variable can be used when we want to make the value of variable constant throughout the execution of a program.

**6. What is the difference between normal variable and final variable?**

Ans: The only difference between a normal variable and a final variable is that we can re-assign the value to a normal variable but we cannot re-assign the value of a final variable once assigned it.

**7. Will the below code compile fine? If yes, what will be the output?**

public class Myclass {

private final static int x = 10;

static {

x++;

}

public static void main(String[] args) {

Myclass obj = new Myclass();

System.out.println(x);

}

}

JAVACopy code

Ans: No, the above code will not compile successfully because the final field cannot be re-assigned.

**8. Is there any error in the below code snippet? If not, what will be the output of the following program?**

public class Myclass {

private final int x = 10;

void m1()

{

final int x = 20;

System.out.println(x);

}

public static void main(String[] args) {

Myclass obj = new Myclass();

obj.m1();

}

}

JAVACopy code

Ans: No, there is no error in the above code snippet. Output: 20.

**9. What is a blank final variable in Java?**

Ans: A variable that is declared as final and not initialized at a time of declaration is called blank final variable.

**10. In which line of code snippet, there is an error?**

public class Myclass {

final int x; *// line 2*

Myclass() {

x = 10; *// line 4*

System.out.println(x++); *// line 5*

}

public static void main(String[] args) {

Myclass obj = new Myclass();

}

}

JAVACopy code

Ans: There is an error in line 5 because the final field cannot be re-assigned.

**11. Will the following code compile fine? If yes, what is the output of the following program?**

public class Myclass

{

final int x;

Myclass()

{

x = 10;

int y = x;

y++;

System.out.println(y);

System.out.println(y++);

}

public static void main(String[] args) {

Myclass obj = new Myclass();

}

}

JAVACopy code

Ans: Yes, the code will compile fine. There is no error in the code snippet. Output: 11, 11.

**12. What is the output of the following snippet code?**

public class Myclass

{

final static int x;

static {

x = 10;

}

public static void main(String[] args) {

System.out.println(x);

}

}

JAVACopy code

Ans: Output: 10. Note that a blank final variable marked with a static keyword can only be assigned in the static block.

**13. Will the following code compile fine?**

import java.util.ArrayList;

import java.util.List;

public class Myclass {

public static void main(String[] args)

{

final List<Integer> obj = new ArrayList<Integer>();

obj.add(24);

obj.add(10);

obj = new ArrayList<Integer>();

System.out.println(obj);

}

}

JAVACopy code

Ans: No, there will be a compile-time error because a reference variable declared with final keyword cannot refer to another object.

**14. Will the below code snippet compile successfully? If yes, what will be the output of the program?**

public class Myclass

{

final void m1() {

System.out.println("One");

}

}

public class Test extends Myclass

{

void m2() {

System.out.println("Two");

}

public static void main(String[] args)

{

Test t = new Test();

t.m1();

t.m2();

}

}

JAVACopy code

Ans: Yes, the above code will compile successfully because a final method can be inherited in the subclass but we cannot override it.

**15. What is the difference between an abstract method and final method in Java?**

Ans: The key difference between abstract method and final method is that abstract method must be overridden in the subclass but final method cannot be overridden in the subclass.

**16. Is there any error in the following code snippet? If not, what will be the output of the following program?**

public class Myclass

{

final void m1() {

System.out.println("One");

}

}

public class Test extends Myclass

{

void m1(Test t) {

System.out.println("Two");

}

public static void main(String[] args)

{

Test t = new Test();

t.m1();

t.m1(null);

}

}

JAVACopy code

Ans: No, there is no error in the above code snippet. Output: One, Two.

**17. What is final class in Java?**

Ans: A class declared with a final keyword is called final class. It does not allow itself to be inherited by another class.

**18. Which is the most common predefined final class object in Java?**

Ans: String

**19. What are the two ways to make a class final?**

Ans: The first way to make a final class is to declare a class with final keyword. Another way is to declare all of its constructors as private. If a class has only private constructors, it cannot be subclassed.

**20. Can we create an instance of final class in another class?**

Ans: Yes, we can create an instance of final class in another class but cannot be inherited.

**21. Is it possible to change the value of a final variable in Java?**

Ans: No, Java does not allow changing the value of a final variable. Once the value is set, it cannot be modified.

**22. How can we restrict inheritance in Java?**

Ans: If we declare a class final, it cannot be extended. This will restrict the inheritance of that class in Java.

**23. Why Integer class has been defined final in Java?**

Ans: Integer class is a wrapper for int. If it is not declared final, any other class can extend it and modify the behavior of Integer  
operations. To avoid it, Integer wrapper class is declared as final.

**24. Can we declare main method as final?**

Ans: Yes, we can declare the main method as final.

**25. Can we mark a block final in Java?**

Ans: No, a block cannot be marked final in java.