Java

I. Object-Oriented Programming (OOP) Fundamentals

A. Constructor Chaining: Initialization Order

Concept: The process where one constructor calls another constructor, ensuring the object's inherited state is initialized completely, starting from the highest ancestor (Object) down to the current class.

Type of Chaining	Keyword	Direction	Purpose
Within Same Class	this(...)	Horizontal	Reuse code among constructors of the same class.
Across Classes	super(...)	Vertical	Initialize the immediate parent class's state.

The Universal Rule:

- Every constructor **must** have either an explicit this(...) or super(...) call as its very first statement.
- If neither is present, the compiler **automatically inserts** an **implicit super();**.

Abstract Class Constructors:

- Abstract classes **can have constructors**. They are called via the chaining mechanism when a concrete subclass is instantiated (you cannot call them directly with new).

B. Polymorphism & Binding

Concept: The ability for a single interface to represent multiple underlying forms/implementations ("performing the same action in different ways").

Type	Mechanism	Binding Time	Resolution
Compile-Time	**Method Overloading**	**Static (Early)**	Resolved by the **compiler** based on parameter list.
Runtime	**Method Overriding**	**Dynamic (Late)**	Resolved by the **JVM** based on the actual object's type.

C. The Static vs. Instance Rule

Concept: Access rules based on whether a member belongs to the class (static) or a specific object (instance).

Context	Access to Non-Static (Instance)	Access to Static (Class)
Static Method/Block	**Forbidden** (Cannot access without an object reference).	**Allowed** (Accessed directly via Class Name).
Instance Method/Block	**Allowed** (Implicitly uses the this object).	**Allowed** (Accessed directly).

D. Variable Hiding vs. Method Overriding

- **Instance Variables** (**Hiding**): Variable access is resolved at **compile time** based solely on the **reference type**. (e.g., Parent p = new Child(); p.variable accesses Parent's variable). Variables are **not polymorphic**.
- **Instance Methods** (**Overriding**): Method calls are resolved at **runtime** based on the **actual object type**. This is **polymorphism**.

E. Keywords, Exceptions & Design

- **this**: A reference variable pointing to the **current** object instance.
- **super**: A reference variable pointing to the **immediate superclass** object.
- **final Method**: Prevents a method from being **overridden** by a subclass.
- **ClassCastException**: A **runtime exception** that occurs during **downcasting** when an object is treated as a specific type it is not an instance of.
- **Reference Type Rule**: The **compiler** checks the methods defined in the **reference type** to deem a call legal. The object's actual capabilities are irrelevant to the compiler.

II. Java Collections (ArrayList) & Conversions

A. Safe Iteration and Removal

Scenario	Problem	Safe Solution
Removal during iteration	Throws **ConcurrentModificationException**.	Use the **Iterator's remove()** method.

B. Array and List Conversions

Conversion	Purpose	Syntax	Pitfall Avoided
Array $\rightarrow$ ArrayList	Create a mutable list from an array.	new ArrayList<>(Arrays.asList(arr))	Avoiding the fixed-size list returned by Arrays.asList().

Conversion	Purpose	Syntax	Pitfall Avoided
ArrayList $\rightarrow$ Array	Convert back to a strongly typed array.	list.toArray(new String[0])	**ClassCastException** (which occurs when casting the raw Object[] returned by list.toArray() to String[]).

C. Efficient List Operations

- **Intersection:** To find common elements and store them in a third list (intsec), use:

Java

```
ArrayList<Integer> intsec = new ArrayList<>(list1);

intsec.retainAll(list2);
```

- **Modification:** Use add(), remove(), and set() for basic manipulation.
- **Ordering:** Use Collections.sort() and Collections.reverse().