**Polymorphism Recap**

**Definition:**  
Polymorphism is the OOP ability for the **same method name** to perform **different actions** depending on the object or arguments.

**Types**

1. **Compile-time polymorphism** (*static binding*) → Method Overloading
2. **Run-time polymorphism** (*dynamic binding*) → Method Overriding

**Method Overloading**

* **Same method name**, different parameter lists (type, number, or order) in the **same class**.
* Decided at **compile time**.
* Example: add(int a, int b) and add(double a, double b).

**Method Overriding**

* **Same method name and parameters**, but implemented in a **subclass** to change behavior from the parent.
* Decided at **runtime** via dynamic dispatch.
* Example: Animal.speak() overridden in Dog.speak().

**Rules**

**Overloading:**

* Same name
* Different parameter lists
* Same class (can also happen between parent and child but no overriding rules apply)

**Overriding:**

* Same name and parameters
* Different class (parent-child relationship)
* Return type same or covariant
* Access level can be more permissive, but not more restrictive
* Cannot override final or static methods (static methods are **hidden**, not overridden)

**Why it’s Useful**

* Cleaner, more maintainable code.
* Removes branching logic in client code.
* Extensible without modifying existing code.

**Without Polymorphism**

* Every client needs explicit if/else or switch for each type.
* Adding a new type means changing all clients → high maintenance cost.

💡 **Interview Quick Answer to** *“When is the call resolved?”*:

* **Overloading** → Resolved at **compile-time**.
* **Overriding** → Resolved at **runtime**.
* **Static methods** → Always resolved at compile time, even if they look like they’re “overridden” (they’re actually hidden).