**Polymorphism:**

The word *polymorphism* comes from Greek:

* **Poly** → many
* **Morphism** → forms

👉 In Java, **polymorphism means the ability of an object to take many forms**.  
It allows the **same method** to behave **differently** depending on the object that is calling it.

**🔹 Types of Polymorphism in Java**

**1. Compile-time Polymorphism (Static Binding / Method Overloading)**

* Achieved using **method overloading** or **operator overloading** (Java doesn’t support custom operator overloading).
* Decision is made **at compile time**.

✅ Example:

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

Calculator c = new Calculator();

System.out.println(c.add(5, 3)); // int version

System.out.println(c.add(5.5, 3.2)); // double version

}

}

**2. Runtime Polymorphism (Dynamic Binding / Method Overriding)**

* Achieved using **method overriding**.
* Decision is made **at runtime** (which method runs depends on the actual object, not reference).

✅ Example:

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

void sound() {

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

}

}

class Cat extends Animal {

void sound() {

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

}

}

public class Main {

public static void main(String[] args) {

Animal a;

a = new Dog();

a.sound(); // Woof! (Dog’s version)

a = new Cat();

a.sound(); // Meow! (Cat’s version)

}

}

**🔹 Why is Polymorphism Useful?**

**✅ 1. Code Reusability**

You can write **generalized code** once and reuse it for multiple object types.

void makeAnimalSound(Animal a) {

a.sound();

}

Now you can pass Dog, Cat, Lion etc.

**✅ 2. Flexibility and Extensibility**

Easier to add new classes without changing old code.  
Example: Add a new Horse class → existing code still works.

**✅ 3. Achieves Loose Coupling**

You can program to an **interface or abstract class**, not to specific implementations.  
This makes your code more flexible.

Example:

List<String> list = new ArrayList<>();

// Later can change to

list = new LinkedList<>();

**✅ 4. Frameworks & Libraries Use It Everywhere**

* **JDBC**: Connection con = DriverManager.getConnection(...)  
  → con could be MySQL, Oracle, PostgreSQL connection, but code remains same.
* **Servlets**: HttpServlet methods overridden by different frameworks.
* **Spring Framework**: Uses interfaces + polymorphism heavily.

**🔹 Simple Real-World Analogy**

Think of a **remote control**:

* Same button "play" works for TV, Music Player, or DVD Player.
* Behavior depends on the device (object), not on the remote (reference).

**✅ Summary**

* **Polymorphism = one interface, many implementations.**
* **Types:**
  1. Compile-time → Method Overloading.
  2. Runtime → Method Overriding.
* **Uses:**
  1. Code reusability
  2. Flexibility & extensibility
  3. Loose coupling
  4. Foundation for frameworks

**What is Upcasting?**

* **Upcasting** is when a **child class object** is assigned to a **parent class reference**.
* The reference “moves up” the inheritance hierarchy → hence called *upcasting*.
* It is **safe** and **implicit** (no explicit cast needed).