Polymorphism in Java

What is Polymorphism?

- Polymorphism means "many forms."
- In Java, polymorphism allows an object to behave differently based on its data type or context.

- Types of Polymorphism:
  - Compile-Time Polymorphism (Method Overloading)
  - Run-Time Polymorphism (Method Overriding)

Why Use Polymorphism?

- - Increases code reusability
- - Enhances flexibility and scalability
- - Allows one interface, many implementations
- - Helps implement clean and maintainable OOP designs

Compile-Time Polymorphism

- Also Known As: Method Overloading
- Multiple methods in the same class with the same name but different parameters.

- Example:

```
class MathUtils {
    int add(int a, int b) {
        return a + b;
    }
    double add(double a, double
b) {
        return a + b;
    }
}
```

Run-Time Polymorphism

- Also Known As: Method Overriding
- Definition: A subclass provides a specific implementation of a method already defined in its superclass.
- Example:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
        void sound() {
        System.out.println("Dog barks");
    }
}
```

Run-Time Polymorphism in Action

```
Animal a;
a = new Dog();
a.sound();  // Output: Dog barks
```

- Explanation:
- Even though 'a' is of type Animal, the Dog's overridden method is called at runtime.

Handling Subclass Objects via Superclass Reference

- You can use a superclass reference to refer to objects of any of its subclasses.
- Let’s have a Cat class:

```
class Cat extends Animal {
    @Override
        void sound() {
         System.out.println("Cat meows");
    }
}
```

```
Animal a;
a = new Dog();
a.sound();  // Dog's sound()
a = new Cat();
a.sound();  // Cat's sound()
```

- Benefit:
- Allows general-purpose code for various subclass objects.

Upcasting and Downcasting

- Upcasting is the process of converting a subclass reference into a superclass reference.
  - It is **implicit**, safe, and doesn't require an explicit cast.

```
Animal a = new Dog();
a.sound();  // Dog's sound()
```

- Downcasting is the process of converting a superclass reference back into a subclass reference.
  - It is **explicit**, and can be **unsafe** if the actual object isn't of the target subclass.

```
Animal a = new Dog();
Dog d = (Dog) a;
d.sound();
```

- Note:
  - Downcasting can cause ClassCastException if not used carefully.

Using instanceof Operator

```
Animal a = new Dog();

if (a instanceof Dog) {
    Dog d = (Dog) a;
    d.sound();
} else {
    System.out.println("Not a Dog");
}
```

- Why Use It?
  - Ensures safe casting
  - Prevents runtime exceptions
  - Useful in polymorphic behavior checks

Polymorphism with Superclass Arrays

```
Animal[] animals = new Animal[3];
animals[0] = new Dog();
animals[1] = new Cat();
animals[2] = new Horse();

for (Animal a : animals) {
    a.sound();
}
```

- Explanation:
  - Each object calls its own overridden method due to dynamic method dispatch.
  - Usage: Managing diverse objects in a unified way.