## ****Polymorphism in Java****

**Polymorphism** is one of the key concepts in Object-Oriented Programming (OOP), derived from Greek meaning "many forms." In Java, polymorphism allows objects of different classes to be treated as objects of a common superclass. It is mainly classified into two types:

* **Compile-time Polymorphism (Method Overloading)**
* **Runtime Polymorphism (Method Overriding)**

### ****1. Compile-time Polymorphism (Method Overloading)****

Compile-time polymorphism occurs when multiple methods have the same name but differ in the number of parameters or the type of parameters. The correct method is chosen at compile-time based on the method signature.

#### ****Example: Compile-time Polymorphism (Method Overloading)****

In this example, we will demonstrate a system for managing student and employee details, where we have overloaded methods for displaying details based on the number of parameters passed.

class Person {

String name;

int age;

// Constructor for Person class

Person(String name, int age) {

this.name = name;

this.age = age;

}

// Method to display basic details

void displayDetails() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

}

// Overloaded method to display additional details for Employee

void displayDetails(String department) {

displayDetails(); // Calls the original displayDetails method

System.out.println("Department: " + department);

}

// Overloaded method to display additional details for Student

void displayDetails(String major, double GPA) {

displayDetails(); // Calls the original displayDetails method

System.out.println("Major: " + major);

System.out.println("GPA: " + GPA);

}

}

public class PolymorphismExample {

public static void main(String[] args) {

Person student = new Person("John", 20);

student.displayDetails("Computer Science", 3.8); // Overloaded method for Student

System.out.println(); // Blank line

Person employee = new Person("Alice", 30);

employee.displayDetails("HR"); // Overloaded method for Employee

}

}

**Explanation:**

* The displayDetails() method is overloaded to show basic details as well as additional information based on whether the object is a Student or an Employee.
* Method overloading allows us to have methods with the same name but different parameter lists. This decision is made at compile-time.

### ****2. Runtime Polymorphism (Method Overriding)****

Runtime polymorphism, also known as method overriding, occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. The method in the subclass is invoked at runtime, depending on the object type (not the reference type).

#### ****Example: Runtime Polymorphism (Method Overriding)****

In this example, we will demonstrate a system where both Student and Employee inherit from a common superclass Person. We will override a displayDetails method in both subclasses to provide specific details for each type of object.

// Base class

class Person {

String name;

int age;

// Constructor for Person class

Person(String name, int age) {

this.name = name;

this.age = age;

}

// Method to display details (This will be overridden)

void displayDetails() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

}

}

// Subclass Employee

class Employee extends Person {

String department;

// Constructor for Employee

Employee(String name, int age, String department) {

super(name, age); // Call the superclass constructor

this.department = department;

}

// Overriding displayDetails method for Employee

@Override

void displayDetails() {

super.displayDetails(); // Call the superclass method

System.out.println("Department: " + department);

}

}

// Subclass Student

class Student extends Person {

String major;

double GPA;

// Constructor for Student

Student(String name, int age, String major, double GPA) {

super(name, age); // Call the superclass constructor

this.major = major;

this.GPA = GPA;

}

// Overriding displayDetails method for Student

@Override

void displayDetails() {

super.displayDetails(); // Call the superclass method

System.out.println("Major: " + major);

System.out.println("GPA: " + GPA);

}

}

public class PolymorphismExample {

public static void main(String[] args) {

// Creating Student and Employee objects

Person student = new Student("John", 20, "Computer Science", 3.9);

Person employee = new Employee("Alice", 30, "Marketing");

// Calling the overridden methods

System.out.println("Student Details:");

student.displayDetails(); // Calls Student's displayDetails

System.out.println();

System.out.println("Employee Details:");

employee.displayDetails(); // Calls Employee's displayDetails

}

}

**Explanation:**

* Person is the superclass, with common attributes like name and age.
* Student and Employee are subclasses that override the displayDetails() method to provide additional details.
* Even though both student and employee are references of type Person, the correct displayDetails() method is called at runtime depending on the actual object type (Student or Employee).

**Output:**

Student Details:

Name: John

Age: 20

Major: Computer Science

GPA: 3.9

Employee Details:

Name: Alice

Age: 30

Department: Marketing

### ****Key Points about Polymorphism in Java****

1. **Method Overloading (Compile-time Polymorphism):**
   * Multiple methods with the same name but different parameters.
   * The correct method is selected at compile time based on the method signature.
2. **Method Overriding (Runtime Polymorphism):**
   * A subclass provides a specific implementation of a method that is already defined in its superclass.
   * The method to call is determined at runtime based on the actual object type.
3. **Advantages of Polymorphism:**
   * **Code Reusability**: You can use the same method name for different purposes (overloading) and write a common interface (overriding).
   * **Maintainability**: Polymorphism helps in writing cleaner and more maintainable code as behavior can be extended without modifying the existing code.
4. **Usage of super with Polymorphism:**
   * The super keyword is used to refer to the parent class's methods or variables. It is particularly useful when you want to call a method in the superclass and then add additional behavior in the subclass.