**WARDA YOUSAF**

**SP23-BSE-042**

**What is Polymorphism?**

**Polymorphism** is one of the core concepts of **Object-Oriented Programming (OOP)**. The term "polymorphism" comes from the Greek words *poly* (meaning many) and *morph* (meaning form or shape), so it literally means "many shapes."

In programming, **polymorphism** allows objects of different classes to be treated as objects of a common superclass. The most common use of polymorphism is when a child class overrides a method from its parent class, allowing the child class to provide a specific implementation of that method.

**Polymorphism with Respect to GRAPS Principle**

The **GRAPS Principle** is an acronym for key Object-Oriented Programming concepts: **Generalization**, **Reusability**, **Abstraction**, **Polymorphism**, and **Specialization**. Polymorphism plays an important role in this principle by enabling flexibility and adaptability in a system. Here's how it fits:

1. **Generalization**: Polymorphism allows general methods to be used across different types of objects. For instance, a method can be applied to both a Dog class and a Cat class if both inherit from a common Animal class. This allows for **generalization** of behavior across different types.
2. **Reusability**: With polymorphism, you can write code that works with many different types of objects, making your code more reusable. You don't need to rewrite the logic for each specific object type; the same code works for multiple objects, enhancing **reusability**.
3. **Abstraction**: Polymorphism helps with **abstraction** by allowing the programmer to interact with objects through their common interface, without needing to know the specific details of the object. The actual implementation details are hidden.
4. **Specialization**: In polymorphism, **specialization** comes into play when a subclass provides a specific implementation for a method, overriding the behavior defined in a more general superclass.

**Why Do We Use Polymorphism?**

We use **polymorphism** for several reasons:

1. **Flexibility and Extensibility**: Polymorphism allows your code to be flexible. You can introduce new classes that are automatically compatible with existing code, provided they implement the common methods or interfaces.
2. **Code Reusability**: Instead of writing multiple versions of the same code to handle different types of objects, polymorphism lets you write a single method that works for any object of a class hierarchy.
3. **Improved Maintainability**: Polymorphism leads to less redundant code, making maintenance easier. Changes made to a method in a parent class will automatically propagate to all subclasses that override it.
4. **Cleaner Code**: Polymorphism helps to keep code concise and clear by using shared interfaces or methods, reducing the need for conditionals and repeated logic.

**Two Types of Polymorphism**

Polymorphism can be broadly classified into two types:

1. **Compile-time Polymorphism (Static Polymorphism)**:
   * Occurs when the method or function call is resolved at compile time.
   * This is typically achieved via **method overloading** (where multiple methods have the same name but different parameters) or **operator overloading** (defining custom behavior for operators like +, -, etc.).

**Example**: A function add(int a, int b) and add(double a, double b) in the same class is an example of method overloading.

1. **Runtime Polymorphism (Dynamic Polymorphism)**:
   * Occurs when the method or function call is resolved at runtime.
   * This is typically achieved through **method overriding** (when a subclass provides a specific implementation of a method that is already defined in its superclass).

**Example**: A base class Shape might have a method draw(), and subclasses like Circle and Square provide their own versions of the draw() method.

**Benefits of Polymorphism**

Polymorphism offers several important benefits:

1. **Flexibility**: Polymorphism allows you to write more flexible and adaptable code. Methods or functions can be written that will work with any object that conforms to a certain interface or class, reducing the need for explicit type checks.
2. **Code Reusability**: By allowing one function to work with many types of objects, polymorphism encourages reusability. You can reuse the same code with different objects without modification.
3. **Ease of Maintenance**: Polymorphism makes systems easier to maintain. Once a method is defined in a superclass, it can be used by all subclasses, and if changes are needed, they can be made in the superclass, automatically affecting all child classes.
4. **Cleaner and Simpler Code**: It reduces the complexity of the code by allowing you to use a common interface for different object types, avoiding the need for complex conditionals and type checking.
5. **Extensibility**: Adding new functionality to a system is easier with polymorphism. New classes or behaviors can be introduced with minimal changes to the existing codebase.

**Code Example Without Polymorphism:**

class Manager {

public void work() {

System.out.println("Managing the team and overseeing projects.");

}

}

class Developer {

public void work() {

System.out.println("Writing and testing code.");

}

}

public class Main {

public static void main(String[] args) {

Manager manager = new Manager();

Developer developer = new Developer();

manager.work();

developer.work();

}

}

**Code Example With Polymorphism:**

abstract class Employee {

abstract void work();

}

class Manager extends Employee {

@Override

public void work() {

System.out.println("Managing the team and overseeing projects.");

}

}

class Developer extends Employee {

@Override

public void work() {

System.out.println("Writing and testing code.");

}

}

public class Main {

public static void main(String[] args) {

Employee manager = new Manager();

Employee developer = new Developer();

// Call work() method on different employee objects

manager.work();

developer.work();

// We can add more types of employees later without changing this code

}

}

**Conclusion**

In conclusion, **polymorphism** is a fundamental concept in object-oriented programming, allowing for flexible, reusable, and maintainable code. When applied within the **GRAPS principle**, polymorphism enhances generalization, reusability, abstraction, and specialization. It provides the flexibility to interact with different object types uniformly and reduces the complexity of code by leveraging a common interface. Polymorphism comes in two types—compile-time and runtime—each offering different advantages in different scenarios. Ultimately, polymorphism enables developers to write cleaner, more modular, and scalable software systems.