پوهنتون کاردان
KARDAN UNIVERSITY

Object Oriented Programming (Java)

Polymorphism

Polymorphism

- The word polymorphism means having many forms. In simple words, we can define Java Polymorphism as the ability of a message to be displayed in more than one form.
- **Real-life Illustration of Polymorphism in Java**: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So the same person possesses different behaviors in different situations. This is called polymorphism.

What is Polymorphism in Java?

- Polymorphism is considered one of the important features of Object-Oriented Programming.

- Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.
- The word “poly” means many and “morphs” means forms, So it means many forms.

Types of Java Polymorphism

- In Java Polymorphism is mainly divided into two types:

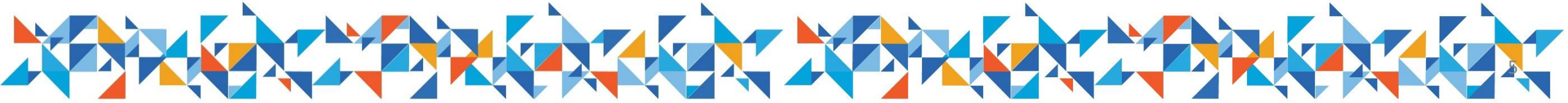
- **Compile-time Polymorphism**

- **Runtime Polymorphism**

1.Compile-Time Polymorphism in Java

- It is also known as static polymorphism.

- This type of polymorphism is achieved by Method overloading.



Method Overloading

- When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**.

- Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

```java
// Java Program for Method overloading
// By using Different Types of Arguments
public class Maths {
        // Method 1
        static int Multiply(int a, int b)
    {
        return a * b;
    }

    // Method 2
    // With same name but with 2 double parameters
    static double Multiply(double a, double b)
    {
        return a * b;
    }}

// Class 2
// Test class
class Testmath {
    // Main driver method
    public static void main(String[] args)
    {
        // Calling method by passing
        // input as in arguments
        System.out.println(Maths.Multiply(10,3 ));
        System.out.println(Maths.Multiply(8.5, 7.5));
    }}
```

```java
// Java program for Method Overloading
// by Using Different Numbers of Arguments
class Maths2 {
    // Method 1
    // Multiplication of 2 numbers
    static int Multiply(int a, int b)
    {
        return a * b;
    }

    // Method 2
    // // Multiplication of 3 numbers
    static int Multiply(int a, int b, int c)
    {
        return a * b * c;
    }}

// Class 2
// Test class
class TestMaths2 {
 public static void main(String[] args)
    {
        // Calling method by passing
        // input as in arguments
        System.out.println(Maths2.Multiply(50, 2));
        System.out.println(Maths2.Multiply(5, 5, 5));
    }}
```

```java
class Printer {
    // Method to print an integer
    public void print(int num) {
        System.out.println("Printing integer: " + num);
    }

    // Overloaded method to print a string
    public void print(String text) {
        System.out.println("Printing string: " + text);
    }

    // Overloaded method to print a float
    public void print(float num) {
        System.out.println("Printing float: " + num);
    }
}

class TestPrinter {
    public static void main(String[] args) {
        Printer printer = new Printer();

        printer.print(5);             // Calls the print(int) method
        printer.print("Hello!");      // Calls the print(String) method
        printer.print(5.5f);          // Calls the print(float) method
    }
}
```

Runtime Polymorphism in Java

- It is also known as Dynamic Method Dispatch.
- It is a process in which a function call to the overridden method is resolved at Runtime.

- This type of polymorphism is achieved by Method Overriding.

Method overriding

- **Method overriding**, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class.
- That base function is said to be **overridden**.

```java
class Human{
    //Overridden method
    public void eat()
    {
        System.out.println("Human is eating");
    }
}
class Boy extends Human{
    //Overriding method
    public void eat(){
        System.out.println("Boy is eating");
    }
    public static void main( String args[]) {
        Boy obj = new Boy();
        //This will call the child class version of eat()
        obj.eat();
    }}
```

```java
class Animal {
    // Method that is overridden in subclasses
    public void makeSound() {
        System.out.println("Some animal sound");
    }}

class Dog extends Animal {
    // Overriding the makeSound method
    public void makeSound() {
        System.out.println("Bark");
    }}

class Cat extends Animal {
    // Overriding the makeSound method
    public void makeSound() {
        System.out.println("Meow");
    }}

 class TestAnimal{
    public static void main(String[] args) {
        // Runtime polymorphism (Dynamic method dispatch)
        Animal animal1 = new Dog();  // Reference of Animal pointing to a Dog object
        Animal animal2 = new Cat();  // Reference of Animal pointing to a Cat object

        animal1.makeSound();  // Calls the Dog's makeSound method
        animal2.makeSound();  // Calls the Cat's makeSound method
    }}
```

Key Points:

- **Method Overloading (Compile-time Polymorphism):** Achieved by defining multiple methods with the same name but different parameters or different data types.

- **Method Overriding (Runtime Polymorphism):** Achieved by defining a method in a subclass with the same name and parameters as a method in the superclass.

پوهنتون کاردان
KARDAN UNIVERSITY

Thank You…!