OBJECT ORIENTED PROGRAMMING : INHERITANCE AND POLYMORPHISM

1st EDITION

SOPHIAN BIN SOUT
MAZLIZA BINTI ROSLAN
Politeknik Mukah

OBJECT ORIENTED PROGRAMMING : INHERITANCE AND POLYMORPHISM

1st EDITION

SOPHIAN BIN SOUT
MAZLIZA BINTI ROSLAN
Politeknik Mukah

Perpustakaan Negara Malaysia

Published in Malaysia By:
Politeknik Mukah
KM 7.5 Jalan Oya
96400 Mukah,
Sarawak, Malaysia

No Tel : +6084-874001
Fax : +6084-874005
Website : https://www.pmu.edu.my

Object Oriented Programming : Inheritance and Polymorphism
1st Edition
Author : Sophian bin Sout
Mazliza binti Roslan

Cataloguing-in-Publication Data

Perpustakaan Negara Malaysia

A catalogue record for this book is available from the National Library of Malaysia

eISBN 978-629-7710-05-1

e ISBN 978-629-7710-05-1

Preface

This book offers a beginner-friendly introduction to inheritance and polymorphism in Object-Oriented Programming (OOP), focusing on Java. Designed for students new to programming, it explains these essential OOP concepts with clear, easy-to-understand examples and practical code snippets. The book starts with a gentle overview of OOP and inheritance, illustrating how classes can reuse and extend existing code, followed by a deep dive into polymorphism, showing how objects can take multiple forms at runtime. Each concept is accompanied by real-world examples, diagrams, and explanations to build confidence in writing Java programs. By the end, readers will have the foundational knowledge to write efficient, maintainable object-oriented code using inheritance and polymorphism.

TABLE OF CONTENTS

Synopsis ---------- 2
Learning Outcomes ---------- 3

Object Oriented Programming
- Introduction ---------- 4
- Features ---------- 6
- Approach ---------- 7
- Benefits ---------- 8
- Concepts ---------- 9
- Objects ---------- 10
- Attributes ---------- 11
- Behaviour ---------- 12

Inheritance
- Introduction ---------- 13
- Extends ---------- 15
- Base Class ---------- 16
- Derived Class ---------- 17
- Single ---------- 19
- Multilevel ---------- 21
- Multiple ---------- 23
- Protected Access Specifier ---------- 24
- Constructors ---------- 26
- Super ---------- 27
- Augmented Reality AR ---------- 29

Polymorphism
- Introduction ---------- 30
- Method Overriding ---------- 33
- Method Overloading ---------- 40
- Operator Overloading ---------- 45

Author ---------- 49

SYNOPSIS

Introduces students to the principles and concepts behind the paradigm of object oriented programming.

Introduces students to write, compile and run programs, make effective use of some of the standard packages, write object-oriented code using inheritance and polymorphism.

LEARNING OUTCOMES

Construct Object Oriented Programming concept and exception handling in Java Programming.

Display skills to use graphical/visual data to visualize the concept of OOP.

Follow the professional ethics in group to develop a solution for a given scenario.

OOP INTRODUCTION

Object Oriented Programming (OOP) is based on the concept of "objects".

It can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated.

In OOP, computer programs are designed by making them out of objects that interact with one another.

OOP INTRODUCTION

A programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability.

Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

OOP FEATURES

Bottom up approach in program design

Programs organized around objects, grouped in classes

Focus on data with methods to operate upon object's data

Interaction between objects through functions

Reusability of design through creation of new classes by adding features to existing classes

---

OOP APPROACH

An object in nature, process certain characteristics that are unique to it.

A real world object also exhibits unique behavior, described as operation.

In OOP, the unique characteristics are attributes of an object ,and the unique behaviors are the methods of an object.

OOP BENEFITS

Creating well-structured program

Easily reusable

Easily extended

Reduce maintenance cost

Encapsulation

Less flaws design

Inheritance - code reuse

Polymorphism - flexibility

OOP CONCEPTS

Object

Attribute

Behaviour

OOP
OBJECT

Object

Attribute

Behaviour

Something that is or is capable of being seen, touched, or other wise sensed, and about which users store data and associate behavior

OOP
ATTRIBUTE

Object

Attribute

Behaviour

The individual things that differentiate one object from another and determine the appearance, state, or other qualities of that object

The data that represent characteristics of interest about an object

OOP
BEHAVIOUR

Object

Attribute

Behaviour

A set of operations that it performs to meet the system goal

The only way objects can do anything to themselves or have anything done to them

OOP INHERITANCE

INTRODUCTION

Creating new classes based on existing ones.

A class that inherits from another class can reuse the attributes and methods of that class.

You can add new attributes and methods to your current class as well.

OOP INHERITANCE

INTRODUCTION

Parent class

Child class

```
class A {
    // attributes
    // methods
}
```

```
class B extends A {
    // class A attributes
    // class A methods
}
```

Child class inherits the attributes and methods from Parent class

OOP INHERITANCE

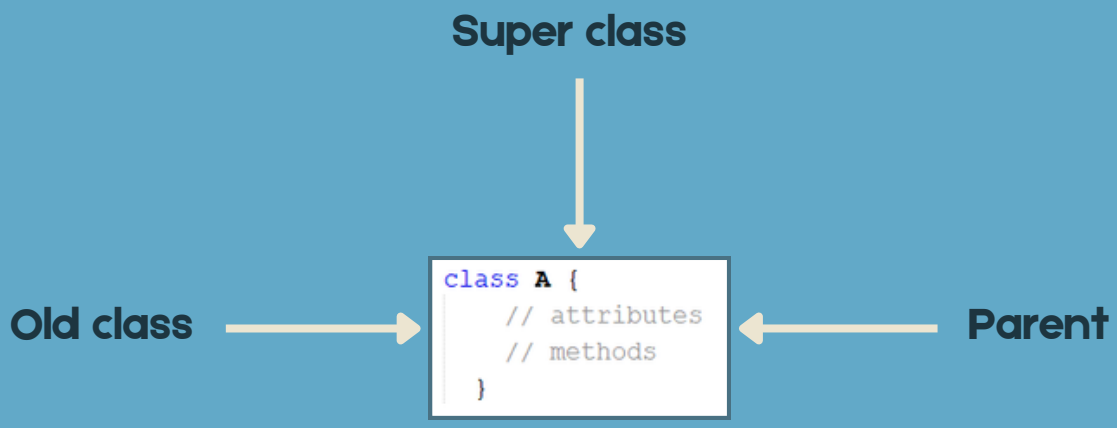
EXTENDS

Keyword extends is used to inherit a subclass from the superclass.

```
class A {
    // attributes
    // methods
}
class B extends A {
    // class A attributes
    // class A methods
}
```

OOP INHERITANCE

BASE CLASS



The class being inherited from

OOP INHERITANCE

DERIVED CLASS

Sub class

New class → `class B extends A { // class A attributes // class A methods }` ← Child

The class that inherits from another class

OOP INHERITANCE

DERIVED CLASS

The sub class inherits all of its attributes and methods defined by the super class

It also adds its own attributes and methods

The sub class can be said that it is specialized version of super class.

OOP INHERITANCE

SINGLE

One class extends one class only

A super class can have any number of sub classes but a sub class can have only one super class

```
class A {
    // attributes
    // methods
}
class B extends A {
    // class A attributes
    // class A methods
}
```

OOP
INHERITANCE

SINGLE

```
class A {
    // attributes
    // methods
}
class B extends A {
    // class A attributes
    // class A methods
}
class C extends A {
    // class A attributes
    // class A methods
}
class D extends A {
    // class A attributes
    // class A methods
}
```

OOP INHERITANCE

MULTILEVEL

Is the extension of Single Inheritance

When a class is inherited from another sub class, then it is Multilevel Inheritance

The sub class at the lowest level can access the member attributes and methods of all the super classes at the higher level

OOP
INHERITANCE

MULTILEVEL

```
class A {
    // attributes
    // methods
}
class B extends A {
    // class A attributes
    // class A methods
}
class C extends A {
    // class A attributes
    // class A methods
}
class D extends A {
    // class A attributes
    // class A methods
}
```

OOP INHERITANCE

MULTIPLE

Inherited from more than one existing class

The inherited class is a sub class and all the existing classes from which the sub class is created are super classes

Java supports Multiple Inheritance through a concept called Interface

```
interface A {
    // methods
}
interface B {
    // methods
}
class C implements A, B {
    // class A, B methods
}
```

OOP INHERITANCE

PROTECTED ACCESS SPECIFIER

The protected access modifier is accessible within package and outside the package but through inheritance only

Protected attributes and methods allow the class itself, classes inside and sub classes to access them

OOP INHERITANCE

PROTECTED ACCESS SPECIFIER

```
class A {
    protected String hairColour = "Black";
}

class B extends A {
    // class A haircolour attributes
}
```

The hairColour attribute in A class is set to a protected access modifier. B class can access the attribute

If it is set to private, B class would not be able to access the attribute

OOP INHERITANCE

CONSTRUCTORS

If we want to call parameterized constructor of base class, then we can call it using super( )

Base class constructor call must be the first line in derived class constructor

```
class A {
    A() // Constructor of super class
    {
        // Print statement
        System.out.println("A Class Constructor Called");
    }
}

class B extends A {
    B() // Constructor of sub class
    {
        // Print statement
        System.out.println("B Class Constructor Called");
    }
}
```

OOP INHERITANCE

SUPER

Super is a reference attributes that is used to refer immediate parent class object.

```
class A {
    protected String hairColour = "Black";
}

class B extends A {
    void display(){
        System.out.println(super.hairColour);
    }
```

OOP INHERITANCE

SUPER

USES

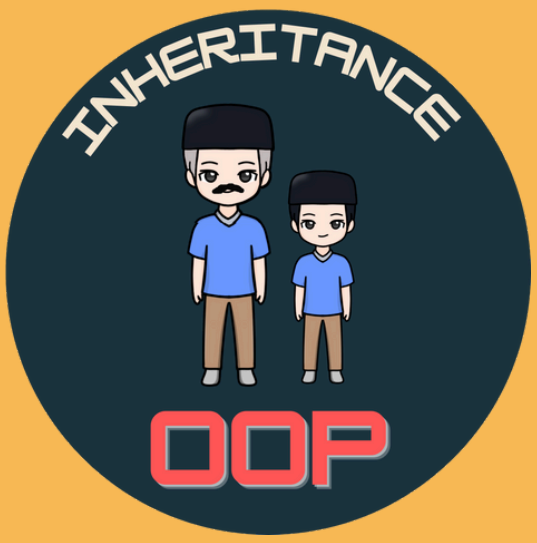
Super is used to invoke immediate parent class construction

Super is used to invoke immediate parent class methods

Super is used to refer immediate parent class attributes

OOP INHERITANCE

AUGMENTED REALITY (AR)



OOP
POLYMORPHISM

INTRODUCTION

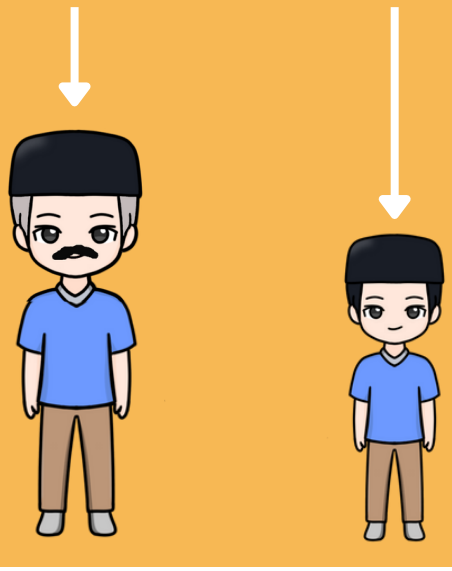
Polymorphism is an important concept of object-oriented programming. It simply means more than one form.

It occurs when we have many classes that are related to each other by inheritance.

OOP
POLYMORPHISM

INTRODUCTION

Parent class **Child class**



```
class A {
    // methods
}
```

```
class B extends A {
    // class A methods
}
```

Inheritance lets us inherit attributes and methods from another class.

Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

OOP
POLYMORPHISM

INTRODUCTION

We can achieve polymorphism in Java using the following ways.

METHOD OVERRIDING

METHOD OVERLOADING

OPERATOR OVERLOADING

OOP
POLYMORPHISM

METHOD OVERRIDING

When the same method is present in both a superclass and a subclass, and the subclass method overrides the superclass method, this is known as method overriding.

In object-oriented programming, method overriding allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

OOP POLYMORPHISM

METHOD OVERRIDING

KEY POINTS

Same Method Signature

The method in both the superclass and subclass must have the same name, return type, and parameters.

Dynamic Polymorphism

Method overriding is a form of dynamic polymorphism, meaning the method that gets called is determined at runtime based on the object type.

Super Keyword

In many languages, such as Java or Python, you can use the super keyword in the subclass to invoke the overridden method in the superclass.

Subclass Behavior

The method in the subclass takes precedence over the superclass method when invoked on an instance of the subclass.

OOP POLYMORPHISM

METHOD OVERRIDING

EXPLANATION

Superclass Method
The display() method in the Superclass outputs a specific message.

Subclass Override
The display() method in the Subclass overrides the one in the superclass and outputs a different message.

Polymorphism
Even when a superclass reference points to a subclass object (Superclass superClassRef = new Subclass();), the subclass version of the method is invoked.

OOP
POLYMORPHISM

METHOD OVERRIDING

INCONSISTENT CODE

We can also create different methods

- calculateSquareArea() to calculate area of Square
- calculateCircleArea() to calculate area of Circle
- calculateRectangleArea() to calculate area of Rectangle

This will work perfectly.
However, for every shape, we need to create different methods.

It will make our code inconsistent.

OOP
POLYMORPHISM

METHOD OVERRIDING

CONSISTENT CODE

To solve this, polymorphism in Java allows us to create a single method that will behave differently for different shapes.

The same method can perform different operations in different scenarios.

- calculateArea() to calculate area of Square
- calculateArea() to calculate area of Circle
- calculateArea() to calculate area of Rectangle

This method will work more perfectly.

Polymorphism will make our code consistent.

OOP POLYMORPHISM

METHOD OVERRIDING

PRACTICAL CODE

```
class Area {

    void calculateArea(){
        // calculate area
        }
    }

    class Square extends Area{
        void calculateArea(){
            // calculate area of Square
        }
    }

    class Circle extends Area{
        void calculateArea (){
            // calculate area of Circle
        }
    }

    class Rectangle extends Area{
        void calculateArea(){
            // calculate area of Rectangle
        }
    }
```

OOP
POLYMORPHISM

METHOD OVERRIDING

PRACTICAL CODE

```java
class Superclass {
    // Method in the superclass
    void display() {
        System.out.println("This is the method in the superclass.");
    }
}

class Subclass extends Superclass {
    // Overriding the method in the subclass
    @Override
    void display() {
        System.out.println("This is the method in the subclass.");
    }
}

public class MainMethod {
    public static void main(String[] args) {
        // Object of the superclass
        Superclass superClassObj = new Superclass();
        superClassObj.display();
        // Output: This is the method in the superclass.

        // Object of the subclass
        Subclass subClassObj = new Subclass();
        subClassObj.display();
        // Output: This is the method in the subclass.

        // Superclass reference, but subclass object
        Superclass superClassRef = new Subclass();
        superClassRef.display();
        // Output: This is the method in the subclass.
    }
}
```

OOP
POLYMORPHISM

METHOD OVERLOADING

When methods in the same class share the same name but differ in parameters (number, type, or both), it is called method overloading.

This feature enables a class to perform different operations based on the type or number of arguments passed.

Method overloading is a form of compile-time polymorphism in Java.

OOP POLYMORPHISM

METHOD OVERLOADING

KEY FEATURES

Same method name but different parameter lists (either in type, number, or order of parameters).

Return type can be different, but it doesn't affect method overloading.

The method that gets called is determined at compile-time based on the arguments provided.

Overloading happens in the same class (or in subclasses using inherited methods).

OOP
POLYMORPHISM

METHOD OVERLOADING

PRACTICAL CODE

```java
class Calculator {

    // Method to add two integers
    int add(int a, int b) {
        System.out.println("Adding two integers: " + a + " + " + b);
        return a + b;
    }

    // Method to add two double values
    double add(double a, double b) {
        System.out.println("Adding two doubles: " + a + " + " + b);
        return a + b;
    }
}

public class MainMethod {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        // Calls the method with two integers
        int result1 = calc.add(5, 10);
        System.out.println("Result: " + result1);

        // Calls the method with two doubles
        double result3 = calc.add(2.5, 3.7);
        System.out.println("Result: " + result3);
    }
}
```

OOP
POLYMORPHISM

METHOD OVERLOADING

EXPLANATION

Different Parameters

add(int a, int b) adds two integers.
add(int a, int b, int c) adds three integers.
add(double a, double b) adds two double values.

Compile-Time Polymorphism

When the code is compiled, the correct version of the add() method is determined based on the arguments passed.

Overloaded Methods

Although the methods share the same name (add), they are treated as different methods because their parameter lists differ.

OOP POLYMORPHISM

METHOD OVERLOADING

KEY DIFFERENCES BETWEEN OVERLOADING AND OVERRIDING

ASPECT	METHOD OVERLOADING	METHOD OVERRIDING
Location	Same class	Subclass overrides superclass method
Polymorphism	Compile-time polymorphism	Runtime polymorphism
Parameters	Must differ in type, number, or order	Same signature as superclass
Return Type	Can differ	Must be the same or covariant
Inheritance	Not required	Involves inheritance

OOP
POLYMORPHISM

OPERATOR OVERLOADING

In Java, the + operator serves a dual purpose:

Arithmetic Addition: When used with numeric operands (integers, floats, doubles, etc.)

String Concatenation: When used with strings.

OOP POLYMORPHISM

OPERATOR OVERLOADING

When + is used with numbers (integers and floating-point numbers), it performs mathematical addition.

```java
public class ArithmeticAddition {

    public static void main(String[] args) {
        int a = 5;
        int b = 10;
        int sum = a + b;

        System.out.println("Sum of integers: " + sum);
        // Output: Sum of integers: 15

        double x = 2.5;
        double y = 3.5;
        double result = x + y;

        System.out.println("Sum of doubles: " + result);
        // Output: Sum of doubles: 6.0
    }
}
```

OOP
POLYMORPHISM

OPERATOR OVERLOADING

When we use the + operator with strings, it will perform string concatenation (join two strings).

```
String firstName = "Mazliza";
String lastName = "Roslan";
String fullName = firstName + " " + lastName;

System.out.println("Full Name: " + fullName);
// Output: Full Name: Mazliza Roslan

int age = 40;
String message = "Age: " + age;

System.out.println(message);
// Output: Age: 40
```

When one operand is a string, the + operator converts the other operand to a string and concatenates them.

In the second example, age (an int) is converted to "40" before being concatenated with the "Age: " string.

OOP POLYMORPHISM

OPERATOR OVERLOADING

SUMMARY

OPERANDS	OPERATION	RESULT
Both numeric	Arithmetic addition	Numeric sum
At least one string	String concatenation	Concatenated string

The + operator is versatile in Java, making it easy to perform both arithmetic operations and string manipulations with minimal syntax.

Understanding how it behaves with different operand types is essential to avoid unintended results.

AUTHOR

SOPHIAN BIN SOUT

PENSYARAH
Politeknik Mukah

MAZLIZA BINTI ROSLAN

PENSYARAH
Politeknik Mukah

e ISBN 978-629-7710-05-1