DAYANANDA SAGAR UNIVERSITY

Devarakaggalahalli, Harohalli Kanakapura Road, Dt, Ramanagara, Karnataka 562112

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

SKILL ENHANCEMENT COURSE-JAVA PROGRAMMING
(22AM2306)

“ Shape Hierarchy and Polymorphism in Java. ”

By

Tanya Gopal	ENG22AM0193
Vaishnavi	ENG22AM0198

Under the supervision of

Prof. Jeevaraj R
Assistant Professor,
CSE(AI & ML)

Dr. Vegi Fernando
Associate Professor,
CSE(AI & ML)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (AI&ML),
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY,BANGALORE
(2023-2024)

Dayananda Sagar University

School of Engineering

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Devarakaggalahalli, Harohalli Kanakapura Road, Dt, Ramanagara, Karnataka 562112

CERTIFICATE

This is to certify that the **SKILL ENHANCEMENT COURSE-JAVA PROGRAMMING (22AM2306)** work titled **“ Shape Hierarchy and Polymorphism in Java. ”** is carried out by **Tanya Gopal (USN : ENG22AM0193)** **and Vaishnavi (USN : ENG22AM0198)** Bonafede student of Bachelor of Technology in Computer Science and Engineering (AI&ML) at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering(AI & ML), during the year **2023-2024**.

Prof. Jeevaraj R
Assistant Professor
Dept. of CS&E (AI&ML),
School of Engineering
Dayananda Sagar University

Dr. Jayavrinda Vrindavanam V
Chairman CSE (AI&ML)
Dept. of CS&E (AI&ML),
School of Engineering
Dayananda Sagar University

Dr. Vegi Fernando
Associate Professor,
Dept. of CS&E(AI&ML)
School of Engineering
Dayananda Sagar University

DECLARATION

We, Tanya Gopal and Vaishnavi, students of third semester B. Tech in **Computer Science and Engineering with specialization in Artificial Intelligence and Machine Learning** , at School of Engineering, **Dayananda Sagar University**, hereby declare that the **SKILL ENHANCEMENT COURSE-JAVA PROGRAMMING (22AM2306)** titled **" Shape Hierarchy and Polymorphism in Java "** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2023 - 2024**.

Student 1	**Signature**
Name: Tanya Gopal	
USN: ENG22AM0193	

Student 2	**Signature**
Name: Vaishnavi	
USN: ENG22AM0198	

Place: Bangalore

Date:

ACKNOWLEDGEMENT

It is a great pleasure for me to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, We take this opportunity to express my sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

We would like to thank ***Dr. Udaya Kumar Reddy K R, Dean, School of Engineering & Technology, Dayananda Sagar University*** *for his constant encouragement and expert advice.*

It is a matter of immense pleasure to express our sincere thanks to ***Dr. Jayvrinda Vrindavanam v, Department Chairman****,* ***Computer Science and Engineering (AI AND ML)****,* ***Dayananda Sagar University,*** *for providing right academic guidance that made the task possible.*

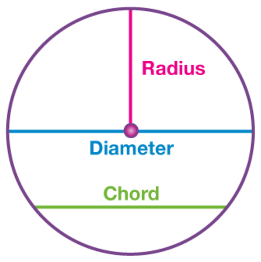
We would like to thank our guides ***Prof. Jeevaraj R, Assistant Professor****,* ***Dept. of Computer Science and Engineering (AI AND ML)****,* ***Dayananda Sagar University****, for sparing his valuable time to extend help in every step of the project work, which paved the way for smooth progress and fruitful culmination of the project.*

We would like to thank our professor ***Dr. Vegi Fernando, Associate Professor****,* ***Dept. of Computer Science and Engineering (AI AND ML)****,* ***Dayananda Sagar University****, for sparing her valuable time to extend help in every step of the project work, which paved the way for smooth progress and fruitful culmination of the project.*

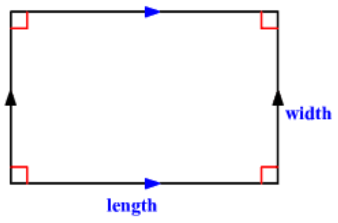
We am also grateful to my family and friends who provided me with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped me in the Project work.

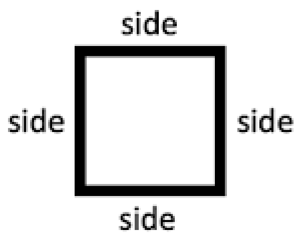
TABLE OF CONTENTS



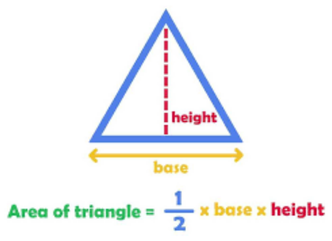
Circle



Rectangle



Square



Triangle

ABSTRACT

This report aims to demonstrate the concept of polymorphism in Java through the implementation of a hierarchy of shapes. The program showcases how polymorphism enables the treatment of different shapes uniformly, facilitating flexible and extensible code.

This project focuses on the practical implementation of polymorphism in Java through the creation of a hierarchy of shapes. The primary objective is to illustrate how polymorphism enables diverse shapes to be handled uniformly, fostering code flexibility and extensibility within an object-oriented program.

The Java programming language, renowned for its object-oriented features, serves as the foundation for this demonstration. Polymorphism, a core concept in object-oriented programming, allows objects of different classes to be treated as instances of a common superclass. In this context, a set of geometric shapes (e.g., circles, squares, triangles) embodies the diverse entities manipulated using polymorphic behaviour.

The program's methodology involves establishing a class hierarchy comprising a superclass named 'Shape' and various subclasses such as 'Circle,' 'Square', 'Rectangle'and 'Triangle.' Each subclass encapsulates specific shape attributes and behaviours, showcasing method overriding to enable polymorphic behaviour.

Through the utilization of overridden methods and common interfaces, the program illustrates how different shapes can be uniformly accessed and manipulated via their superclass. This facilitates the seamless addition of new shapes without necessitating alterations to existing code, emphasizing the adaptability and scalability achieved through polymorphism.

The outcomes of this project highlight the dynamic and versatile nature of polymorphism, as disparate shapes are treated uniformly, demonstrating the inherent flexibility and extensibility afforded by this program.

CHAPTER 1 : INTRODUCTION

In the realm of object-oriented programming (OOP), polymorphism is a powerful and essential concept that facilitates code organization, flexibility, and extensibility. It allows objects of different types to be treated uniformly through a common interface, enabling the creation of versatile and scalable systems. One notable application of polymorphism is evident in the representation of geometric shapes in software development.

This report delves into the implementation of a simple Java program that illustrates the concept of polymorphism through a hierarchy of shapes. By creating a foundation with a common base class and deriving specific shapes from it, we will showcase how polymorphism enables the uniform treatment of diverse shapes, fostering a code structure that is not only flexible but also easily extensible. Through a practical example, this report aims to highlight the inherent benefits of employing polymorphism in Java programming for the creation and maintenance of robust and adaptable code.

CHAPTER 2 : PROBLEM DEFINITION

Question :

You have been tasked with implementing a simple program to demonstrate the concept of Polymorphism in Java. You are required to create a hierarchy of shapes and showcase how polymorphism allows different shapes to be treated uniformly, enabling flexible and extensible code.

The task at hand involves the development of a Java program that vividly demonstrates the concept of polymorphism, a key aspect of object-oriented programming. The specific focus is on creating a hierarchy of geometric shapes, illustrating how polymorphism facilitates the uniform treatment of diverse shapes within the codebase. The objective is to design a program that not only calculates the areas of different shapes but does so in a manner that promotes flexibility and extensibility.

Key Requirements:

Shape Hierarchy: Develop a hierarchy of shapes, with a common base class and at least two derived classes representing specific shapes (e.g., circles, rectangles).

Common Interface: Implement a common interface or method within the base class, ensuring that all derived shapes provide a consistent way to calculate their
respective areas.

Polymorphic Usage: Create instances of different shapes and store them in a data structure (e.g., an array or a collection). Utilize polymorphism to iterate through the data structure and calculate the areas of each shape without explicitly knowing their specific types.

Flexibility and Extensibility: Design the program in a way that allows for easy addition of new shapes in the future without modifying existing code. Highlight how polymorphism contributes to the system's adaptability.

Expected Output:

The program should produce output showcasing the calculated areas of different shapes, emphasizing the uniform treatment of these shapes through polymorphism. Each shape should be processed using the same method or interface, demonstrating the inherent flexibility of the implemented code.

CHAPTER 3 : METHODOLOGY

This methodology demonstrates the use of polymorphism in Java to create a flexible and extensible system for calculating areas of different shapes through a common interface (Shape). The derived classes (Circle, Rectangle, Square and Triangle) provide their specific implementations, and the code demonstrates how polymorphism allows treating objects of different types uniformly.

Methodology for Implementing Polymorphism in a Java Shape Hierarchy:

Step 1: Define the Shape class-The Shape class is created as the base class for other shapes.

It includes a method calculateArea() with a default implementation that returns 0.0.

Step 2: Define the Circle class-the Circle class extends the Shape class.

It has a private field radius and a constructor to initialize it.

The calculateArea() method is overridden to provide the area calculation for a circle.

Step 3: Define the Rectangle class-The Rectangle class extends the Shape class.

It has private fields width and height and a constructor to initialize them.

The calculateArea() method is overridden to provide the area calculation for a rectangle.

Step 4: Define the Square class-The Square class extends the Shape class.

It has private field side and a constructor to initialize them.

The calculateArea() method is overridden to provide the area calculation for a square.

Step 5: Define the Triangle class-The Triangle class extends the Shape class.

It has private fields base and height2 and a constructor to initialize them.

The calculateArea() method is overridden to provide the area calculation for a triangle.

Step 6: Demonstrate Polymorphism-An array of Shape objects is created, and instances of Circle,

Rectangle, Square and Triangle are assigned to it. The for loop iterates through the array, calling the calculateArea() method on

each shape. Due to polymorphism, the appropriate calculateArea() method of the actual object type (circle, rectangle, square and triangle) is called at runtime.

Let's create a simple shape hierarchy consisting of a base Shape class and four derived

classes **Circle, Rectangle, Square and Triangle**. We will then demonstrate polymorphism by creating an array of shapes and performing operations on them without knowing their exact types.

CHAPTER 4 : PROGRAM CODE (JAVA)

```
//Define the Shape Class and its Derived Classes
class Shape {
    public double calculateArea() {
        // Default implementation for base class
        return 0.0;
    }
}
class Circle extends Shape {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}
class Rectangle extends Shape {
    double width;
    double height;
       Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
      @Override
    public double calculateArea() {

return width * height;
    }
}
class Square extends Shape {
    public double side;
    Square(double side) {
        this.side = side;
    }
    @Override
    public double calculateArea() {
        return side * side;
    }
}
class Triangle extends Shape {
    double base;
    double height2;

    Triangle(double base, double height2) {
        this.base = base;
        this.height2 = height2;
    }
```

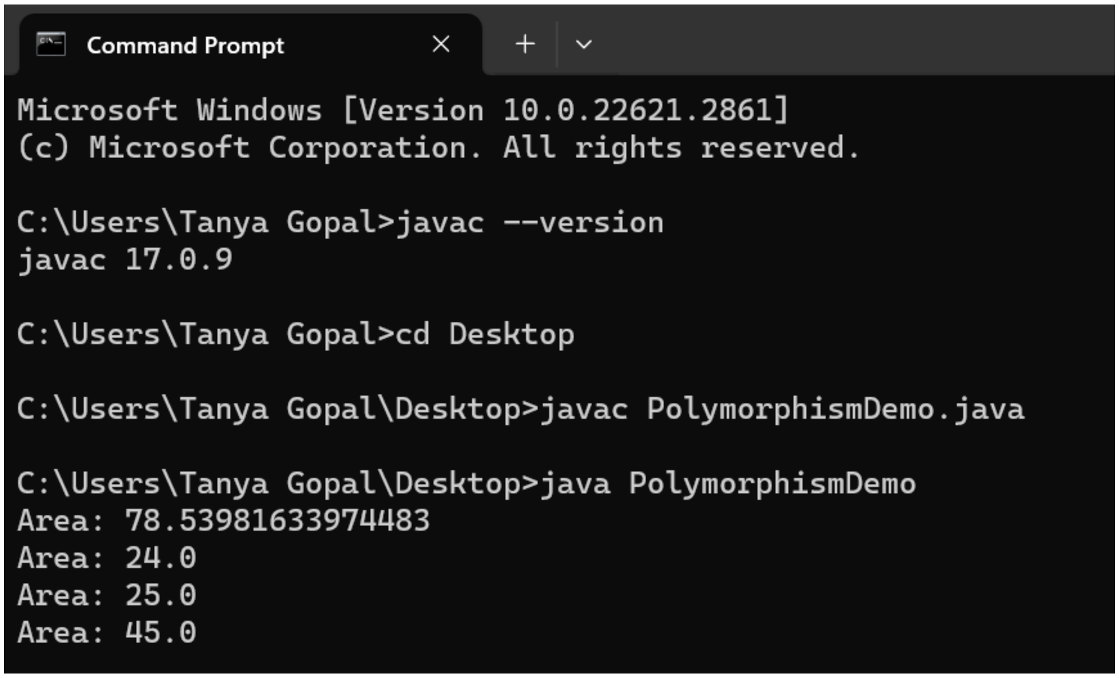
```
 @Override
    public double calculateArea() {
        return 0.5 * base * height2;
    }
}
class PolymorphismDemo {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[4];
        shapes[0] = new Circle(5.0);
        shapes[1] = new Rectangle(4.0, 6.0);
        shapes[2] = new Square(5);
        shapes[3] = new Triangle(6, 15);

        // Calculate and display areas of all shapes
        for (Shape shape : shapes) {
            System.out.println("Area: " + shape.calculateArea());
        }
    }
}
```

CHAPTER 5 : RESULT, OUTPUT AND TABULATION

Result :

```
Area: 78.53981633974483
Area: 24.0
Area: 25.0
Area: 45.0
```

Output :



```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Tanya Gopal>javac --version
javac 17.0.9

C:\Users\Tanya Gopal>cd Desktop

C:\Users\Tanya Gopal\Desktop>javac PolymorphismDemo.java

C:\Users\Tanya Gopal\Desktop>java PolymorphismDemo
Area: 78.53981633974483
Area: 24.0
Area: 25.0
Area: 45.0
```

Table :

SHAPES	**DIMENSIONS** (units)	**RESULT** **(AREA)** (sq. units)
1. Circle	radius = 5.0	78.53981633974483
2. Rectangle	width = 4.0, height = 6.0	24.0
3. Square	side = 5	25.0
4. Triangle	base = 6, height2 = 15	45.0

CHAPTER 6 : CONCLUSION

In conclusion, the implementation of a hierarchy of shapes in Java vividly illustrates the powerful concept of polymorphism. Polymorphism, a fundamental principle of object-oriented programming, enables diverse objects to be treated uniformly through a common interface, facilitating code flexibility and extensibility.

The hierarchical structure comprising the base class Shape and its subclasses (Circle, Rectangle, Square, Triangle) exemplifies polymorphism in action. By inheriting from the Shape class and overriding the calculateArea() method, each shape exhibits its unique behaviour while conforming to the common interface.

This demonstration showcases the versatility and adaptability of polymorphism. Despite their distinct implementations of the calculateArea() method, all shapes can be manipulated uniformly via the Shape superclass. This inherent flexibility allows for seamless addition of new shapes without altering existing code, underscoring the extensibility and maintainability offered by polymorphic design.

Moreover, the ability to treat objects of different types uniformly not only streamlines code implementation but also enhances readability and scalability. Through polymorphism, complex systems can be modelled and managed efficiently, enabling developers to build robust and adaptable software solutions.

Thus, the demonstration of polymorphism in Java using a hierarchy of shapes exemplifies the essence of object-oriented programming paradigms. The uniform treatment of diverse shapes highlights the elegance and efficiency achieved through polymorphic behaviour, thereby emphasizing its indispensable role in developing flexible, maintainable, and extensible software systems.

This highlights the significance of polymorphism in allowing different shapes to be handled uniformly, promoting code flexibility, and emphasizing its pivotal role in object-oriented programming principles.

CHAPTER 7 : REFERENCES

References from reputable websites and sources used to gather information about Java, polymorphism, and geometric shape formulas are listed here.

References:

Oracle Java Documentation - https://docs.oracle.com/en/java/

GeeksforGeeks - https://www.geeksforgeeks.org/

W3Schools - https://www.w3schools.com/

https://byjus.com/

https://www.cuemath.com/

Academic papers on polymorphism and object-oriented programming