Ex 01

public interface MyFirstInterface

{

int x = 10;

void display(); // Declaring an abstract method 'display()'

}

1. public interface MyFirstInterface {

int x = 10; // Declaring an integer type variable 'x'

public int y = 20;

static int z = 30;

final int w = 40;

void display();

}

In Java interfaces, all variables are implicitly public, static, and final. So, whether you explicitly use the keywords or not, the variables are treated as public, static, and final. Therefore, there's no difference between declaring "x" with or without the public static final keywords.

2. public interface MyFirstInterface

{

void display();

abstract void show();

}

In Java, when you declare a method within an interface, it is implicitly abstract, even if you don't use the "abstract" keyword. So, again, there is no difference between declaring the method with or without the "abstract" keyword.

Ex 04

public abstract class Shape {

public abstract double calculateArea();

public void display() {

System.out.println("This is a shape.");

}

}

public class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

public class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

public double calculateArea() {

return length \* width;

}

}

public class Main {

public static void main(String[] args) {

double circleRadius = 5.0;

Circle circle = new Circle(circleRadius);

double circleArea = circle.calculateArea();

System.out.println("Circle Area: " + circleArea);

double rectangleLength = 4.0;

double rectangleWidth = 3.0;

Rectangle rectangle = new Rectangle(rectangleLength, rectangleWidth);

double rectangleArea = rectangle.calculateArea();

System.out.println("Rectangle Area: " + rectangleArea);

}

}