

First Task: Write a Java program to create a class called "Shape" with abstract methods for calculating area and perimeter, and subclasses for "Rectangle".

Source Code:

```
Source History
1
2 package com.mycompany.main;
3
4 abstract class Shape {
5     abstract double calculateArea();
6     abstract double calculatePerimeter();
7 }
8 class Rectangle extends Shape {
9     private double length;
10    private double width;
11
12    public Rectangle(double length, double width) {
13        this.length = length;
14        this.width = width;
15    }
16    @Override
17    double calculateArea() {
18        return length * width;
19    }
20    @Override
21    double calculatePerimeter() {
22        return 2 * (length + width);
23    }
24 }
25 public class Main {
26     public static void main(String[] args) {
27         Rectangle rectangle = new Rectangle(length: 5.0, width: 4.0);
28
29         System.out.println("Rectangle Area: " + rectangle.calculateArea());
30         System.out.println("Rectangle Perimeter: " + rectangle.calculatePerimeter());
31     }
32 }
33
34 com.mycompany.main.Rectangle >
```

Output:

```
Output - Run (Main) x
cd C:\Users\HP\OneDrive\Documents\NetBeansProjects\Main; "JAVA_HOME=C:\Program Files\Java\jdk1.8.0_101" cmd /c
Scanning for projects...

-----< com.mycompany:Main >-----
Building Main 1.0-SNAPSHOT
from pom.xml
-----[ jar ]-----

--- resources:3.3.0:resources (default-resources) @ Main ---
skip non existing resourceDirectory C:\Users\HP\OneDrive\Documents\NetBeansProjects\Main\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ Main ---
Changes detected - recompiling the module!
Compiling 1 source file to C:\Users\HP\OneDrive\Documents\NetBeansProjects\Main\target\classes

--- exec:3.1.0:exec (default-cli) @ Main ---
Rectangle Area: 20.0
Rectangle Perimeter: 18.0

BUILD SUCCESS

Total time: 1.418 s
Finished at: 2023-09-11T00:14:27+06:00
```

Discussion:

Here is a Java program that defines a Shape class with abstract methods for calculating area and perimeter, and a subclass Rectangle. In this program, the Shape class is an abstract class with abstract methods calculateArea and calculatePerimeter. The Rectangle class extends Shape and provides concrete implementations for the calculateArea and calculatePerimeter methods based on the properties of a rectangle. In the main method, we create an instance of Rectangle, calculate its area and perimeter, and then print the results. This program demonstrates the use of an abstract class Shape and a concrete subclass Rectangle to calculate the area and perimeter of a rectangle.

Second Task:

Q1. What is an interface in Java and how is it different from a class?

Answer:

In Java, an interface is a blueprint for a class that defines a set of abstract methods (methods without a body) that a class implementing the interface must provide concrete implementations for. Interfaces can also include constant fields (variables), but they are implicitly public, static, and final.

Here are some key characteristics and differences between interfaces and classes in Java:

Abstract Methods: In an interface, all methods are implicitly abstract and must be implemented by any class that implements the interface. In contrast, a class can have both abstract and concrete (implemented) methods.

Multiple Inheritance: A class in Java can inherit from only one superclass (single inheritance), but a class can implement multiple interfaces (multiple inheritance). This allows a class to inherit behavior from multiple sources.

Constructor: An interface cannot have constructors because interfaces cannot be instantiated. In contrast, a class can have constructors and be instantiated to create objects.

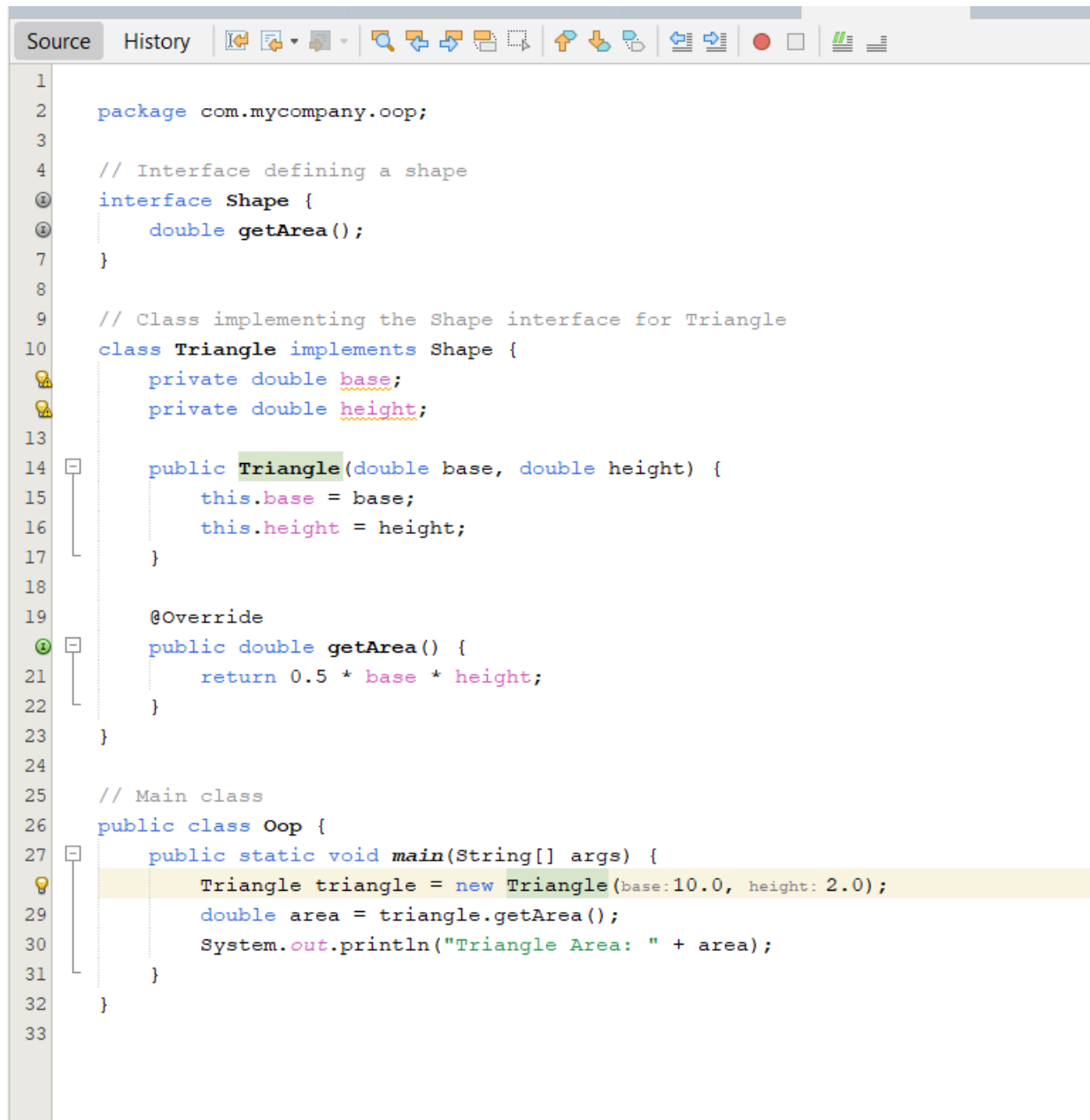
Fields: Fields (variables) declared in an interface are implicitly public, static, and final, meaning they are constants. In a class, fields can have various access modifiers and are not necessarily constants.

Methods: In an interface, methods do not have implementations (no method body), and implementing classes must provide concrete implementations. In a class, methods can have concrete implementations.

.

Q2. Write a Java program to create an interface Shape with the `getArea()` method. Create a class Triangle that implements the Shape interface. Implement the `getArea()` method for the class.

Source Code:



```
1
2  package com.mycompany.oop;
3
4  // Interface defining a shape
5  interface Shape {
6      double getArea();
7  }
8
9  // Class implementing the Shape interface for Triangle
10 class Triangle implements Shape {
11     private double base;
12     private double height;
13
14     public Triangle(double base, double height) {
15         this.base = base;
16         this.height = height;
17     }
18
19     @Override
20     public double getArea() {
21         return 0.5 * base * height;
22     }
23 }
24
25 // Main class
26 public class Oop {
27     public static void main(String[] args) {
28         Triangle triangle = new Triangle(base:10.0, height: 2.0);
29         double area = triangle.getArea();
30         System.out.println("Triangle Area: " + area);
31     }
32 }
33
```

Output:

```
Output - Run (oop) X
cd C:\Users\HP\OneDrive\Documents\NetBeansProjects\oop; "JAVA_HOME=C:\\Program Files\\Java\\jdk1.8.0_101" cmd /c "
Scanning for projects...

-----< com.mycompany:oop >-----
Building oop 1.0-SNAPSHOT
  from pom.xml
-----[ jar ]-----

--- resources:3.3.0:resources (default-resources) @ oop ---
- skip non existing resourceDirectory C:\Users\HP\OneDrive\Documents\NetBeansProjects\oop\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ oop ---
Changes detected - recompiling the module!
Compiling 1 source file to C:\Users\HP\OneDrive\Documents\NetBeansProjects\oop\target\classes

--- exec:3.1.0:exec (default-cli) @ oop ---
Triangle Area: 10.0

-----
BUILD SUCCESS
-----

Total time:  1.331 s
Finished at: 2023-09-11T00:28:04+06:00
-----
```

Discussion:

Here's a Java program that defines an interface Shape with a `getArea()` method and a class Triangle that implements the Shape interface and provides an implementation for the `getArea()` method. In this program, the Shape interface defines a single method `getArea()` that returns a double. This method is a contract that any class implementing the Shape interface must provide an implementation for. The Triangle class implements the Shape interface and provides a concrete implementation for the `getArea()` method, which calculates the area of a triangle using the formula $0.5 * \text{base} * \text{height}$. In the main method, we create a Triangle object with specified base and height values and then calculate and print its area.