

Instituto Politécnico Nacional
Escuela Superior de Cómputo



Paradigmas de Programación, 3CV1

"INTERFACES"

Profesor. García Floriano Andrés
Fecha de entrega: 4 de Junio 2024

Elaborado por:

- Elizalde Hernández Alan
- Reyes Ruíz Yoselyn Estefany
- Solares Velasco Arturo Misael
- Solis Lugo Mayra
- Toral Hernández Leonardo Javier

INTERFACES

El objetivo de esta tarea fue diseñar una jerarquía de clases utilizando interfaces para demostrar el uso del polimorfismo. Se implementaron las clases Triangle, Circle, Rectangle y Hexagon, cada una con sus atributos específicos para calcular el área y el perímetro, tanto en Python como en Java.

Diseño de la Jerarquía de Clases

En ambos lenguajes, se definió una interfaz Figure con métodos para obtener el color, calcular el perímetro y calcular el área. Las clases específicas (Triangle, Circle, Rectangle, Hexagon) implementan esta interfaz, proporcionando sus propias implementaciones de estos métodos.

Implementación en Python

Python no tiene interfaces nativas, pero podemos aproximarnos utilizando clases abstractas del módulo abc. Aquí se muestra la implementación de la interfaz Figure y las clases derivadas:

```
import matplotlib.pyplot as plt
import numpy as np
import math
from abc import ABC, abstractmethod

class Figure(ABC):
    @abstractmethod
    def getColor(self):
        pass

    @abstractmethod
    def perimetro(self):
        pass

    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def draw(self, ax):
        pass

class Triangle(Figure):
    def __init__(self, color, a, b, c):
        self.color = color
        self.a = a
        self.b = b
        self.c = c

    def getColor(self):
        return self.color

    def perimetro(self):
        return self.a + self.b + self.c

    def area(self):
        s = self.perimetro() / 2
        return math.sqrt(s * (s - self.a) * (s - self.b) * (s - self.c))

    def draw(self, ax):
        x = [0, self.a, self.b, 0]
        y = [0, 0, self.c, 0]
        ax.fill(x, y, facecolor=self.color, edgecolor="black")
        ax.plot(x, y, color="black")

class Circle(Figure):
    def __init__(self, color, radius):
        self.color = color
        self.radius = radius

    def getColor(self):
        return self.color

    def perimetro(self):
        return 2 * math.pi * self.radius

    def area(self):
        return math.pi * self.radius**2

    def draw(self, ax):
        circle = plt.Circle((0, 0), self.radius, facecolor=self.color,
                             edgecolor="black")
        ax.add_patch(circle)

class Rectangle(Figure):
    def __init__(self, color, width, height):
        self.color = color
        self.width = width
        self.height = height

    def getColor(self):
        return self.color

    def perimetro(self):
        return 2 * (self.width + self.height)

    def area(self):
        return self.width * self.height

    def draw(self, ax):
        rect = plt.Rectangle((0, 0), self.width, self.height,
                              facecolor=self.color, edgecolor="black")
        ax.add_patch(rect)

class Hexagon(Figure):
    def __init__(self, color, side):
        self.color = color
        self.side = side

    def getColor(self):
        return self.color

    def perimetro(self):
        return 6 * self.side

    def area(self):
        return (3 * math.sqrt(3) * self.side**2) / 2

    def draw(self, ax):
        angle = np.linspace(0, 2 * np.pi, 7)
        x = self.side * np.cos(angle)
        y = self.side * np.sin(angle)
        ax.fill(x, y, facecolor=self.color, edgecolor="black")
        ax.plot(x, y, color="black")

# Demostración de polimorfismo
figures = [
    Triangle("red", 3, 4, 5),
    Circle("blue", 10),
    Rectangle("green", 4, 7),
    Hexagon("yellow", 6)
]

fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs = axs.flatten()

for ax, figure in zip(axs, figures):
    ax.set_aspect('equal')
    figure.draw(ax)
    ax.set_title(f"Color: {figure.getColor()}\nPerimeter: {figure.perimetro():.2f}, Area: {figure.area():.2f}")

plt.tight_layout()
plt.show()
```

```

interface Figure {
    String getColor();
    double perimetro();
    double area();
}

class Triangle implements Figure {
    private String color;
    private double a, b, c;

    public Triangle(String color, double a, double b, double c) {
        this.color = color;
        this.a = a;
        this.b = b;
        this.c = c;
    }

    @Override
    public String getColor() {
        return color;
    }

    @Override
    public double perimetro() {
        return a + b + c;
    }

    @Override
    public double area() {
        double s = perimetro() / 2;
        return Math.Sqrt(s * (s - a) * (s - b) * (s - c));
    }
}

class Circle implements Figure {
    private String color;
    private double radius;

    public Circle(String color, double radius) {
        this.color = color;
        this.radius = radius;
    }

    @Override
    public String getColor() {
        return color;
    }

    @Override
    public double perimetro() {
        return 2 * Math.PI * radius;
    }

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }
}

class Rectangle implements Figure {
    private String color;
    private double width, height;

    public Rectangle(String color, double width, double height) {
        this.color = color;
        this.width = width;
        this.height = height;
    }

    @Override
    public String getColor() {
        return color;
    }

    @Override
    public double perimetro() {
        return 2 * (width + height);
    }

    @Override
    public double area() {
        return width * height;
    }
}

class Hexagon implements Figure {
    private String color;
    private double side;

    public Hexagon(String color, double side) {
        this.color = color;
        this.side = side;
    }

    @Override
    public String getColor() {
        return color;
    }

    @Override
    public double perimetro() {
        return 6 * side;
    }

    @Override
    public double area() {
        return (3 * Math.Sqrt(3) * side * side) / 2;
    }
}

public class Main {
    public static void main(String[] args) {
        Figure[] figures = {
            new Triangle("Red", 3, 4, 5),
            new Circle("Blue", 10),
            new Rectangle("Green", 4, 7),
            new Hexagon("Yellow", 6)
        };

        for (Figure figure : figures) {
            System.out.println("Color: " + figure.getColor() +
                ", Perimeter: " + figure.perimetro() +
                ", Area: " + figure.area());
        }
    }
}

```

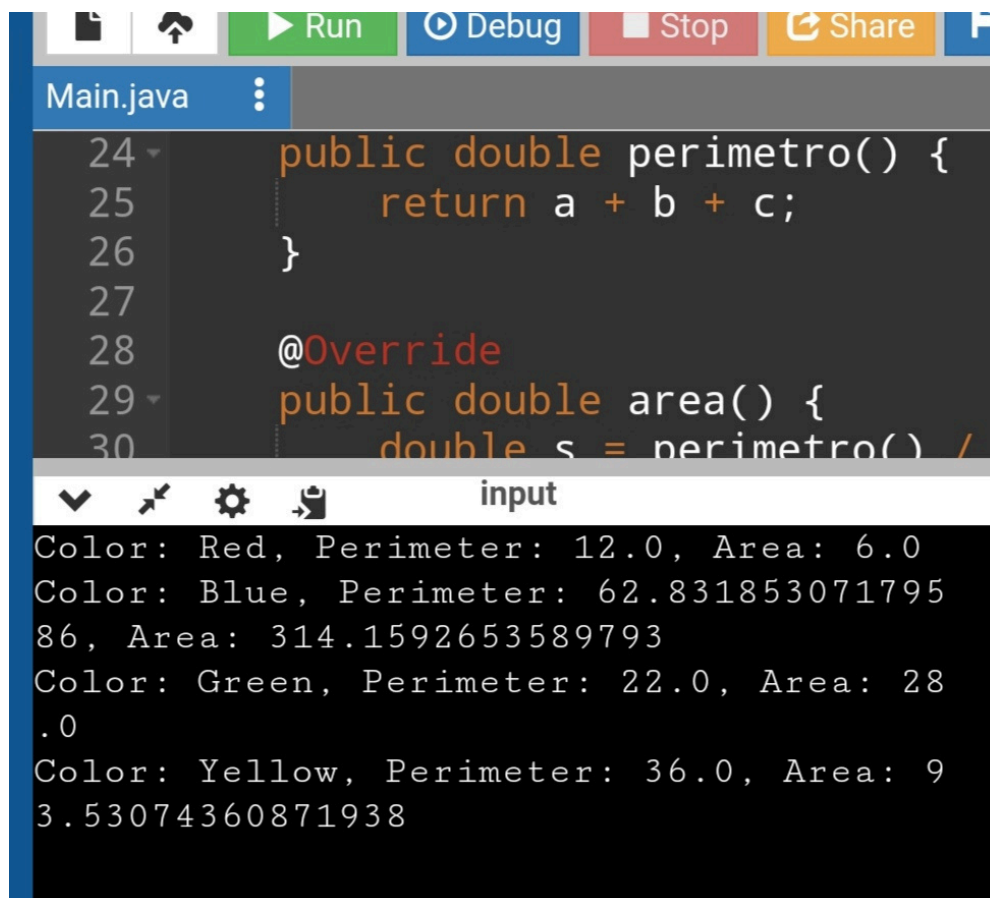
Resultados y Análisis

Polimorfismo: Las interfaces en ambos lenguajes permitieron demostrar polimorfismo. Pudimos llamar a los métodos `getColor()`, `perimetro()`, y `area()` en cada instancia de las clases `Triangle`, `Circle`, `Rectangle` y `Hexagon` sin importar su tipo específico.

Flexibilidad: La implementación de interfaces mejora la flexibilidad y la capacidad de mantenimiento del código. Cualquier nueva clase que implemente la interfaz `Figure` puede ser fácilmente integrada sin modificar el código existente.

Separación de Responsabilidades: Al usar interfaces, separamos claramente las definiciones de métodos comunes de sus implementaciones, siguiendo el principio de segregación de interfaces.

Consistencia: La implementación de interfaces en Java y la aproximación a interfaces en Python proporcionaron una estructura coherente y consistente en ambos lenguajes.



```
Run Debug Stop Share
Main.java
24 public double perimetro() {
25     return a + b + c;
26 }
27
28 @Override
29 public double area() {
30     double s = perimetro() /
input
Color: Red, Perimeter: 12.0, Area: 6.0
Color: Blue, Perimeter: 62.831853071795
86, Area: 314.1592653589793
Color: Green, Perimeter: 22.0, Area: 28
.0
Color: Yellow, Perimeter: 36.0, Area: 9
3.53074360871938
```

```

113 axs = axs.flatten()
114
115 for ax, figure in zip(axs, figures):
116     ax.set_aspect('equal')
117     figure.draw(ax)
118     ax.set_title(f"Color: {figure.getColor}")
119
120 plt.tight_layout()
121 plt.show()
122

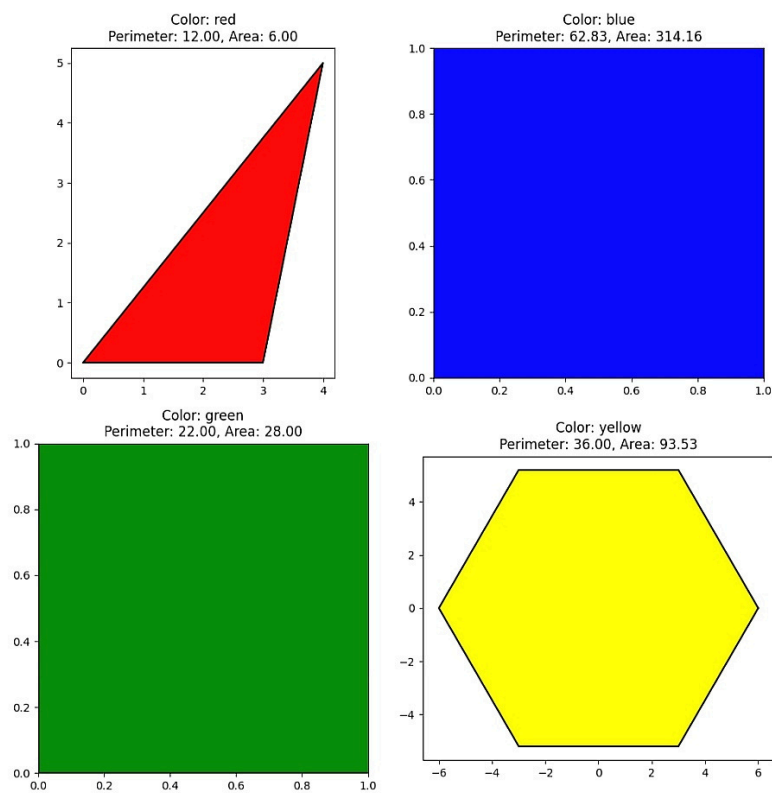
```

Entrada del programa

PYTHON

Salida del programa

[Execution complete with exit code 0]





En Python, se utilizó la librería matplotlib para dibujar las figuras en un gráfico. Se crearon las siguientes clases:

Figure: Una clase abstracta que define los métodos comunes para todas las figuras geométricas.

Triangle: Representa un triángulo con los métodos para calcular su área y perímetro.

Circle: Representa un círculo con los métodos para calcular su área y perímetro.

Rectangle: Representa un rectángulo con los métodos para calcular su área y perímetro.

Hexagon: Representa un hexágono con los métodos para calcular su área y perímetro.

Se demostró el polimorfismo creando instancias de cada clase y llamando a los métodos comunes getColor(), perimetro() y area().

La implementación de la jerarquía de clases utilizando interfaces en Python y Java demostró con éxito el uso del polimorfismo y mejoró la flexibilidad y mantenibilidad del código. Esta práctica es esencial para el desarrollo de software modular y escalable.

