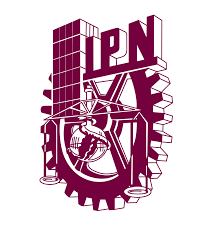
**Escuela Superior de Computo (ESCOM)**

**Instituto Politécnico Nacional**

**Paradigmas de Programación**

Olivares Cruz Victor Manuel

**Práctica 10**

Fecha de entrega: 28/05/2024

**Instrucciones**

## Diseño de una jerarquía de clases basada en la clase de figura

En esta tarea, diseñará una jerarquía de clases basada en la clase abstracta Figure. Cada clase de la jerarquía tendrá un atributo String llamado color y métodos llamados getColor(), perimetro() y area(). Implementará las clases: Triángulo, Círculo, Rectángulo y Hexágono. Cada una de estas clases tendrá sus propios atributos para calcular el área y el perímetro.

### Descripción general de la tarea

Su tarea es crear una jerarquía de clases que demuestren el uso del polimorfismo. Utilizará Python y Java para implementar las clases.

### Instrucciones detalladas

Siga estos pasos para completar la tarea:

1. **Entendiendo la clase de figura:**
   * La clase Figure sirve como clase base para la jerarquía. Tiene un atributo String llamado color y métodos para recuperar el color, calcular el perímetro y calcular el área.
2. **Implementación de la clase Triángulo:**
   * Cree una clase Triangle que herede de la clase Figure.
   * Definir atributos específicos de un triángulo para calcular su área y perímetro.
   * Asegúrese de que los métodos getColor(), perimetro() y area() se anulen correctamente en la clase Triangle.
3. **Implementación de la clase Círculo:**
   * Cree una clase Circle que herede de la clase Figure.
   * Defina atributos específicos de un círculo para calcular su área y perímetro.
   * Anula los métodos getColor(), perimetro() y area() en la clase Circle.
4. **Implementación de la clase Rectángulo:**
   * Cree una clase Rectangle que herede de la clase Figure.
   * Defina atributos específicos de un rectángulo para calcular su área y perímetro.
   * Anula los métodos getColor(), perimetro() y area() en la clase Rectangle.
5. **Implementación de la clase Hexágono:**
   * Cree una clase Hexagon que herede de la clase Figure.
   * Defina atributos específicos de un hexágono para calcular su área y perímetro.
   * Anula los métodos getColor(), perimetro() y area() en la clase Hexagon.
6. **Demostración de polimorfismo:**
   * Cree instancias de cada clase (Triángulo, Círculo, Rectángulo y Hexágono) y demuestre el polimorfismo llamando a los métodos getColor(), perimeter() y area() en cada instancia.
7. **Codificación en Python y Java:**
   * Escribe el código de las clases tanto en Python como en Java para demostrar tu comprensión de la implementación de la jerarquía en diferentes lenguajes.
8. **Entrega:**
   * Prepare un informe que resuma su enfoque para diseñar la jerarquía de clases.
   * Adjunte el código desarrollado tanto para Python como para Java, o proporcione un enlace a un repositorio donde se pueda acceder al código.

### **Envío**

Envíe su informe y archivos de código de acuerdo con las instrucciones de entrega proporcionadas.

Recuerde demostrar una comprensión clara de la herencia, el polimorfismo y el diseño de clases tanto en Python como en Java. ¡Buena suerte!

**Clase Figura**

La clase Figura es una clase abstracta que define una estructura común para todas las figuras geométricas en la jerarquía. Esta clase contiene un atributo de instancia llamado color y define métodos abstractos que las subclases deben implementar: perimetro() y area(). A continuación se detallan sus características:

* Atributo color: Define el color de la figura.
* Método getColor(): Devuelve el color de la figura.
* Métodos abstractos perimetro() y area(): Deben ser implementados por cualquier subclase que herede de Figure.

**Herencia**

La herencia es el mecanismo mediante el cual una clase (subclase o clase derivada) hereda atributos y métodos de otra clase (superclase o clase base). En esta jerarquía de clases, Figure es la clase base, y Triangle, Circle, Rectangle y Hexagon son las clases derivadas que heredan de Figure.

**Polimorfismo**

El polimorfismo permite que una operación se comporte de manera diferente en diferentes contextos. En este caso, se refiere a la capacidad de llamar a los métodos perimetro() y area() en objetos de diferentes subclases a través de una referencia de la clase base (Figure).

**Java:**

**Figura.java**

package Java;

public abstract class Figura {

    protected String color;

    public Figura(String color) {

        this.color = color;

    }

    public String getColor() {

        return color;

    }

    public abstract double perimetro();

    public abstract double area();

}

**Circulo.java**

package Java;

public class Circulo extends Figura {

    private double radius;

    public Circulo(String color, double radius) {

        super(color);

        this.radius = radius;

    }

    @Override

    public double perimetro() {

        return 2 \* Math.PI \* radius;

    }

    @Override

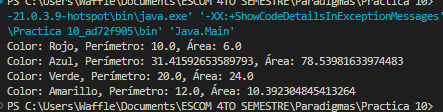
    public double area() {

        return Math.PI \* radius \* radius;

    }

}

**Prueba de ejecución:**

****

**Python:**

**Figura.py**

from abc import ABC, abstractmethod

class Figura(ABC):

    def \_\_init\_\_(self, color: str):

        self.color = color

    def getColor(self) -> str:

        return self.color

    @abstractmethod

    def perimetro(self) -> float:

        pass

    @abstractmethod

    def area(self) -> float:

        pass

**Circulo.py**

from Figura import Figura

import math

class Circulo(Figura):

    def \_\_init\_\_(self, color: str, radius: float):

        super().\_\_init\_\_(color)

        self.radius = radius

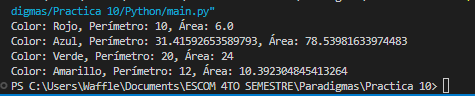
    def perimetro(self) -> float:

        return 2 \* math.pi \* self.radius

    def area(self) -> float:

        return math.pi \* self.radius\*\*2

**Prueba de ejecución:**

****