	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): M.C. RENE ADRIAN DAVILA PEREZ

Asignatura: PROGRAMACIÓN ORIENTADA A OBJETOS

Grupo: 01

No de Práctica(s): 7 y 8

Integrante(s): 322118311

322094028

322092842

322078673

322067738

No. de lista o brigada: 03

Semestre: 2026-1

Fecha de entrega: 17/10/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del problema.** Para esta practica buscaremos modificar el planteamiento de una aplicación que modela figuras geometricas basicas. El problema consiste en aplicar herencia y polimorifsmo para obtener una jerarquia de clases que representaran las figuras a modelar en la aplicación.
- **Motivación.**Aplicar la herencia y el polimorfismo (los cuales permiten reutilizar código, mantener una estructura clara y extender funcionalidades de manera eficiente), nos ayudara a mejorar nuestras practicas de programación orientada a objetos, lo cual nos servira para en un futuro desarollar aplicaciones mas complejas.
- **Objetivos.** Los obejtivos de esta práctica son: implementar correctamente la herencia y el polimorfismo en un programa que manipulara diferentes tipos de figuras geométricas. Asimismo, buscaremos comprender el funcionamiento de las clases abstractas, recalcar la importancia de los paquetes en la organización del código y en conjunto, como todos estos elementos nos ayudan a desarrollar programas de manera eficiente.

2. Marco Teórico

2.1. Herencia:

Es apartir de una clase a la que denominamos clase padre y apartir de esta desarrollar otra que posea las mismas caracteristicas, sin embargo, extendiendo nuevos metodos o funcionalidades. La clase nueva nos ahorra código y que reutiliza código ya definido de en una clase padre y que ademas lo extiende. Dicho de otra forma "La herencia es el mecanismo de basar un objeto o clase en otro objeto (herencia basada en prototipos) o clase (herencia basada en clases), conservando una implementación similar. También se define como derivar nuevas clases (subclases) de las existentes, como superclase o clase base, y luego formarlas en una **jerarquía de clases**.[1]

Listing 1: Sintaxis de herencia

```
class Figura {
    // Clase padre
}

class Circulo extends Figura {
    // Subclase que hereda de Figura
}
```

2.2. Polimorfismo:

De una clase padre podemos derivar varias subclases, en ese sentido el polimorfismo es que cada clase padre sera diferente con respecto a cada subclase. Denominamos polimorfismo al "mecanismo que nos permite tener un método en una clase padre como vimos en la herencia y sobrescribirlo en la clase hija".

Esto quiere decir que tendremos el mismo método en ambas clases, pero en la clase hija realizara diferentes acciones. Por lo que el polimorfismo es también denominado **sobreescritura de métodos.**

Hay dos formas de polimorfismo:

En tiempo de ejecución, que tiene que ver con las **interfaces**. La segunda llamada **polimorfismo estático**, la cual determina que método se va a ejecutar durante la compilación. [2]

Listing 2: Sintaxis de polimorfismo

```
Figura f;           // Referencia de tipo Figura
f = new Circulo(); // Puede apuntar a cualquier subclase de
   Figura
```

2.3. Clases abstractas:

Son clases en las que no se pueden crear objetos, su funcion principal es servir como plantilla para las clases que se heredaran de una clase padre, estas clases pueden contener metodos pero unicamente se les colocara la firma, pues estos metodos deberan definirse en las subclases. [3]

Listing 3: Sintaxis de clase abstracta

```
abstract class Figura {
    abstract void calcularArea(); // Metodo abstracto
}

class Circulo extends Figura {
    void calcularArea() {
        // Implementacion del metodo
    }
}
```

3. Desarrollo

3.1. Clase abstracta Figura:

Esta clase sirve como base para establecer los métodos abstractos area(), perimetro() y dibujar(Graphics2D g, Dimension size) que todas las subclases de tipo figura debían implementar.

Esta clase no se modifico

3.2. Clase Circulo:

En esta clase se realizaron las siguientes implementaciones: Lo primero que hicimos fue establecer que Circulo extienda de Figura, para despues implementar los métodos **area()**, **perimetro()** y **dibujar()**. Esto para que Circulo pueda ser tratada como un objeto de tipo Figura y sus metodos se invoquen correctamente.

Tambien añadimos el atributo radio y su constructor para posteriormente usarlo en los métodos. Los métodos **getRadio()** y **setRadio()** se implementaron para controlar el acceso al radio (encapsulación). Para los métodos heredados, **area()** y **perimetro()** se modificaron para realizar los calculos correctamente haciendo uso de respectivas sus respectivas formulas:

$$\text{Área: area}() = \pi \times \text{radio}^2$$
$$\text{Perímetro: perimetro}() = 2 \times \pi \times \text{radio}$$

3.3. Clase Rectangulo:

Primero modificamos la clase para que esta extienda de figura, posteriormente de la misma forma que se modifico la clase **Circulo**, implementamos los métodos **area()**, **perimetro()** y **dibujar()**.Para que al igual que la clase anterior, Rectangulo pueda ser tratada como un objeto de tipo Figura y sus metodos se invoquen de forma correcta. Posteriormente añadimos los atributos **Ancho** y **Alto** junto con su constructor para inicializar cada rectángulo con dimensiones agregadas por el usuario. Despues se agregaron los métodos getAncho(), setAncho(), getAlto() y setAlto() para controlar acceso controlado a estos valores (Encapsulamiento).Finalmente modificamos los metodos **area()** y **perimetro()** para realizar los calculos correspondientes.

$$\text{Área: area}() = \text{Ancho} \times \text{Alto}$$
$$\text{Perímetro: perimetro}() = 2 \times (\text{Ancho} + \text{Alto})$$

3.4. Clase TrianguloRectangulo:

De la misma forma que las clases anteriores, se modifico la clase para que TrianguloRectangulo extienda de la clase Figura, procedimos a crear los metodos **area()**, **perimetro()** y **dibujar()**, para que TrianguloRectangulo se convierta en una clase de tipo Figura, despues agregamos los atributos **Base** y **Altura** junto con su constructor para crear cada triangulo con medidas especificas. Creamos los métodos **getBase()**,**SetBase()**, **SetAltura()** y **getAltura()**manteniendo la encapsulación al igual que en las clases anteriores, finalmente se implementaron los metodos utilizando las formulas correspondientes para área y perimetro.

$$\text{Área: area}() = \frac{\text{Base} \times \text{Altura}}{2}$$

$$\text{Perímetro: perimetro}() = \text{Base} + \text{Altura} + \sqrt{\text{Base}^2 + \text{Altura}^2}$$

3.5. Clase NumericTextField:

La clase NumericTextField sirve para que el programa solo acepte números y punto decimal, y proporciona un método (safeParse) para convertir su contenido a double.

Esta clase no se modifico

3.6. Clase PanelDibujo:

Esta clase PanelDibujo crea un panel gráfico que dibuja la figura elegida por el usuario (usa Graphics2D) haciendo uso de proporciones de los valores ingresados por el usuario.

Esta clase no se modifico

3.7. Clase MainApp:

Esta clase crea la interfaz gráfica ; permite al usuario seleccionar una figura, ingresar sus dimensiones, calcular su área y perímetro, y la dibuja haciendo uso de la clase PanelDibujo

Esta clase no se modifico

4. Resultados

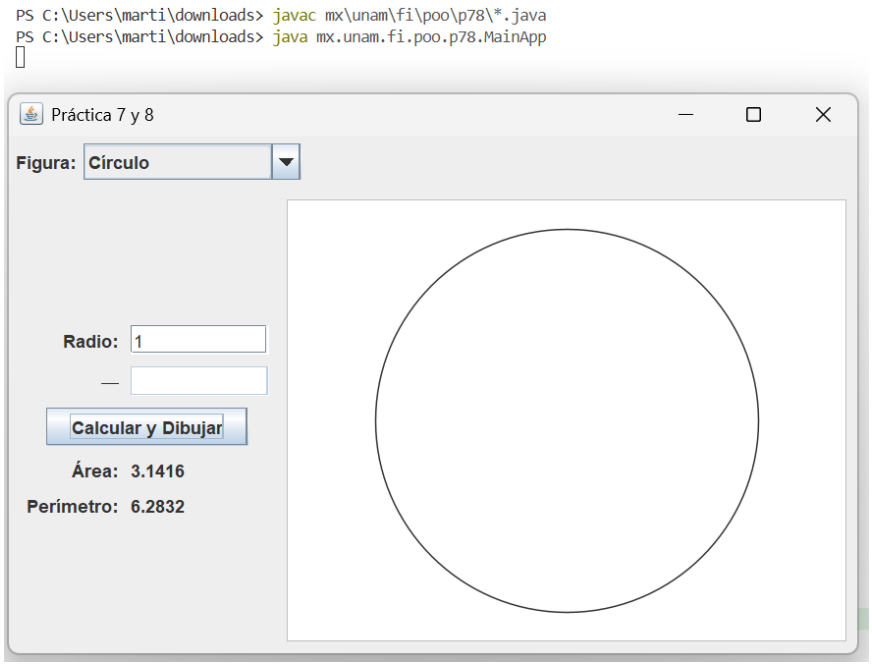

Figura 1: Ejecucion del programa haciendo uso del comando para ejecutar un paquete, ademas se muestra el resultado de el calculo y dibujo del circulo.

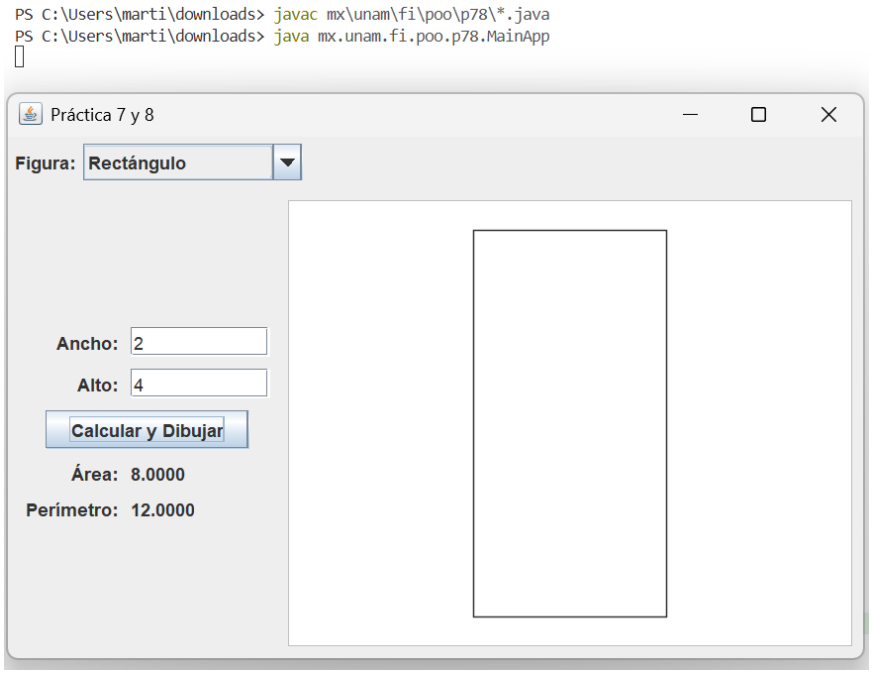

Figura 2: Resultado de el calculo de el área y perimetro de un rectangulo

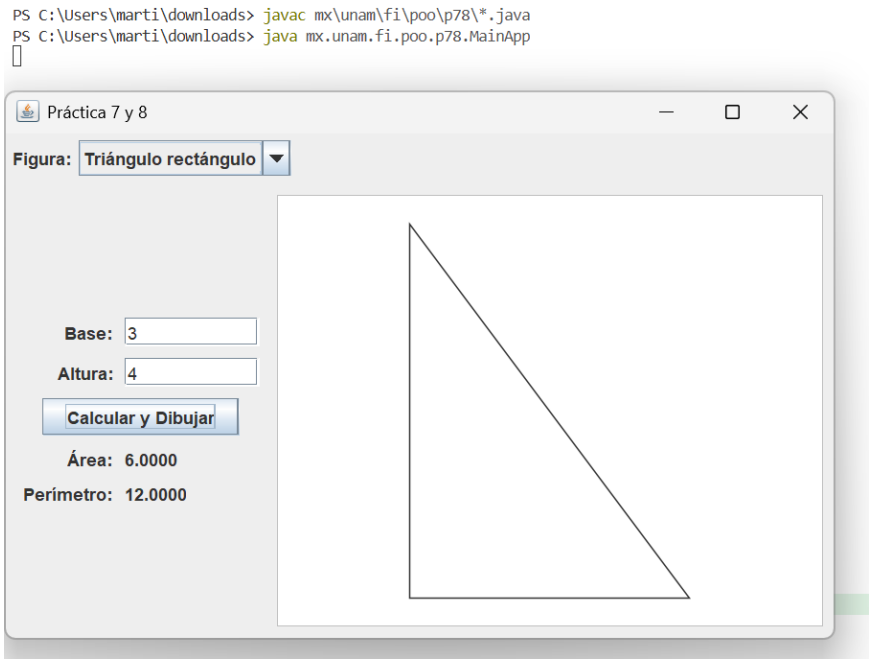

Figura 3: Resultado de el calculo de área y perimetro de un triangulo rectangulo.

5. Conclusiones

La práctica nos permitió comprender cómo la herencia y el polimorfismo son herramientas esenciales en la programación orientada a objetos. Logramos modificar el programa haciendo uso de estos conceptos pues se logró establecer una jerarquía de clases donde “Figura” funciona como clase abstracta y “Circulo”, “Rectangulo” y “TrianguloRectangulo” implementan correctamente sus métodos. Con ello se demostró cómo la herencia nos facilita la reutilización de código y la organización de las clases, mientras que el polimorfismo posibilita el tratamiento uniforme de objetos de diferentes tipos.

6. Bibliografía

- **[1]** AcademiaLab, "Herencia (programación orientada a objetos)", AcademiaLab, 2025. [Enlace]. Disponible en: `https://academia-lab.com/enciclopedia/herencia-programacion-orientada-a-objetos/\#google_vignette`
- **[2]** NetMentor, "Polimorfismo en programación orientada a objetos", NetMentor, 14 sep. 2019. [Enlace]. Disponible en: `https://www.netmentor.es/Entrada/polimorfismo-poo/\#mcetoc_1ehncnesj4`
- **[3]** Arturo Parra, .Entendiendo las Clases Abstractas en Java", ProgramandoJava, 23 ago. 2023. [Enlace]. Disponible en: `https://www.programandojava.com/blog/clases-abstractas-java/`