	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): 7 y 8

Integrante(s): 322184022

322238682

321570789

322069093

322249723

No. de lista o brigada: 5

Semestre: 2026-1

Fecha de entrega: 17/10/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción **2**
1.1. Planteamiento del problema 2
1.2. Motivación 2
1.3. Objetivos 2

2. Marco Teórico **2**
2.1. Herencia 2
2.2. Polimorfismo 3
2.3. Sobrescritura 3

3. Desarrollo **5**

4. Resultados **7**

5. Conclusiones **8**

6. Referencias **9**

1. Introducción

Esta práctica se centra en aplicar los conceptos de polimorfismo y herencia de POO mediante el lenguaje Java. Se busca reforzar el conocimiento y manejo de los conceptos escenciales que se ha visto a lo largo del curso mediante una implementación de estos.

1.1. Planteamiento del problema

El ejercicio se basa en una implementación de un sistema capaz de dibujar distintas figuras geométricas (círculo, rectágulo, triángulo rectángulo) y el cálculo de su respectiva área y perímetro a partir de ciertos parametros (radio, altura, base). Adicionalmente se utilizan librerías para el desarrollo en la interfaz gráfica (GUI).

1.2. Motivación

El dominio de la implementación de conceptos fundamentales para la programación orientada a objetos. Entender a la herencia a partir de como una clase abstracta puede tener elementos en común que obliguen a que sus clases derivadas desarrollen su propia lógica. Demostrar que el polimorfismo es capaz de tratar objetos de manera uniforme de forma que sea mas sencilla la implementación de estos en la clase principal.

1.3. Objetivos

1. Comprender los conceptos de herencia y polimorfismo en POO.
2. Implementar correctamente la herencia de la clase abstracta hacia sus subclases.
3. Demostrar que el polimorfismo puede dar diferentes implementaciones a un mismo método segun la clase.
4. Uso de constrores para la inicializacion de los atributos de cada objeto.

2. Marco Teórico

2.1. Herencia

[1] En términos simples, se puede definir como contruir "algo" a partir de cimientos ya existentes. Permite definir nuevas clases basadas en clases existentes, aprovechando y extendiendo su funcionalidad.

- **Super Clase**
  Se define una sola clase base, la cual contendrá los atributos y métodos comunes que deseamos compartir con las clases derivadas.

- **Clases derivadas**
  Se crea la clase hija, la cual como su nombre lo indica, será la que hereda de la clase padre.
  Con la ”herencia” ahora se pueden crear objetos de la clase derivada y acceder a los atributos y métodos tanto de la clase derivada como de la clase base.

- Sintaxis
  modificador class nombreSubClase
  extends nombreSuperClase

Ventajas

- Reutilización del código
- Abstracción y generalización
- Uso de **polimorfismo**

2.2. Polimorfismo

[1] Es la habilidad de una clase para implementar métodos con el mismo nombre, pero con comportamientos distintos dependiendo de la clase con la que se este interactuando.

- Crear clase base/interfaz
  De igual forma que con la herencia, se definen los métodos que deamos que las clases derivadas implementen.

- Clase derivada
  Se crean las clases que implementarán la interfaz o hereden de la clase padre. Cada clase debe proporcionar su propia implementación del método definido en la interfaz/clase padre.
  Ahora se pueden crear objetos de las clases derivadas y tratarlos como objetos de la interfaz/clase base.

2.3. Sobrescritura

[2] Permite que una sub-clase proporcione su propia implementación del método para adaptarlo a su comportamiento.
Ocurre cuando una sub-clase redefine un método heredado de su clase padre con:

- Mismo nombre
- Mismo tipo y número de parámetros.
- Mismo nivel de acceso.
- Mismo tipo de valor de retorno.

@Override *(se indica que se está sobrescribiendo un método)*

Para acceder:
La palabra clave **super** se utiliza para comunicar métodos y atributos de la clase base a una clase derivada, permitiendo llamar a los contructores, métodos o acceder a los atributos de la clase padre desde la clase hija.
Esto es útil si deseamos agregar funcionalidad adicional en la sub-clase, pero también queremos preservar el comportamiento original del método de la clase base.

3. Desarrollo

1. **Clase Figura:**
   Este programa comienza con la clase **Figura**, declarada como abstracta para definir la estructura que tendrán en común las figuras. Dentro de la clase se encuentran declarados tres métodos abstractos, los cuales son:

   - **area()**- Método que devuelve el área de la figura
   - **perimetro()**- Método que devuelve el perímetro de la figura
   - **dibujar()**- Método dado por el profesor que devuelve la figura gráficamente

   Las figuras (Círculo, Rectángulo y Triángulo Rectángulo) sobreescribirán estos métodos para hacer de ellos su propia implementación.

2. **Clase Círculo:**
   Para la figrua círculo se creo la clase **Círculo** en la cual se trabajo la implementación de los métodos aplicando el concepto de herencian de la clase **Figura**. También se declararon los atributos con **private**, para trabajar con encapsulamiento y para esta figura solo fue un atributo:

   - **radio** - De tipo double

   Luego se añadieron los métodos **getRadio()** y **setRadio()** para acceder y modificar el valor del radio. Seguido de la sobreesccritura de los métodos heredados de la clase **Figura**:

   - **area()**- Devuelve el área del círculo. $\pi * r^2$
   - **perimetro()**- Devuelve el perímetro del círculo.$2\pi r$
   - **dibujar()** Hace los calculos y devuelve el dibujo de la figura.

3. **Clase Rectángulo:**
   La clase **Rectángulo** siguiendo el procedimiento de la clase **Círculo** también hereda de **Figura**. Y se implementan los atributos:

   - **ancho**- De tipo double.
   - **alto**- De tipo double.

   Para ambos también se utilizó **private** para seguir trabajando con el encapsulamiento y los métodos **get** y **set** junto con un constructor que recibe los valores.

   - **area()**- Devuelve ancho*alto.
   - **perimetro()**- Devuelve la suma de todos los lados.
   - **dibujar()**- Hace el gráfico de la figura.

4. **Clase Triángulo Rectángulo:**
   En la clase **TriánguloRectángulo**, que también hereda de Figura, se implementaron los atributos base y altura, ambos con textbfprivate y con sus métodos de acceso y mutadores, junto con el constructor para inicializarlos.

   - **area()**- Devuelve (base*altura)/2.
   - **perimetro()**- Devuelve la suma de todos los lados, para esto se agrega la hipotenusa con la función Math.hypot(base, altura).
   - **dibujar()**- Hace el gráfico de la figura.

5. **Clases PanelDibujo y NumericTextField:**
   Estas clases fueron proporcionadas por el profesor. La clase PanelDibujo tiene que ver con todo lo de la representación gráfica, mientras que NumericTextField es para validar que solo se introduzcan números en los campos de texto. Se utilizaron únicamente con las clases desarrolladas, sin realizar modificación alguna.

6. **Clase MainApp:**
   La clase **MainApp** fue proporcionada por el profesor y su función principal fue todo lo de la interfaz gráfica, y la creación de los objetos de tipo Figura (Círculo, Rectángulo o TriánguloRectángulo).
   El codigo desarrollado se logró integrar correctamente, permitiendo que compilara correctamente.

4. Resultados

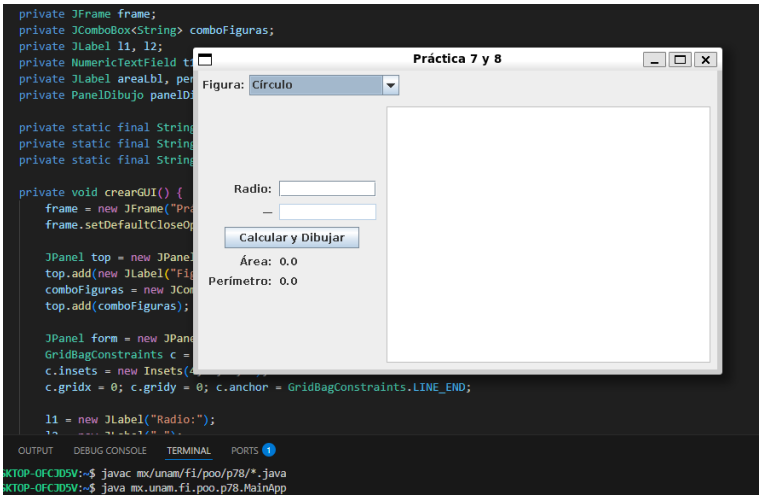

Figura 1: Ventana principal de la aplicación

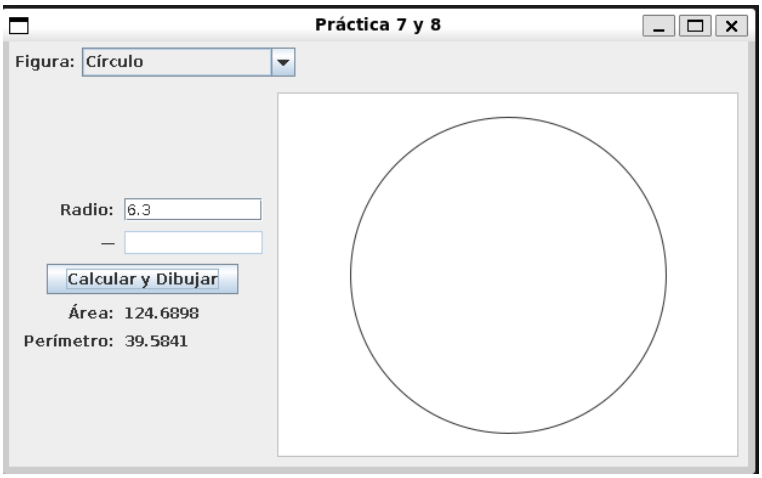

Figura 2: Figura del círculo mostrando el cálculo de su área y perímetro a partir del radio

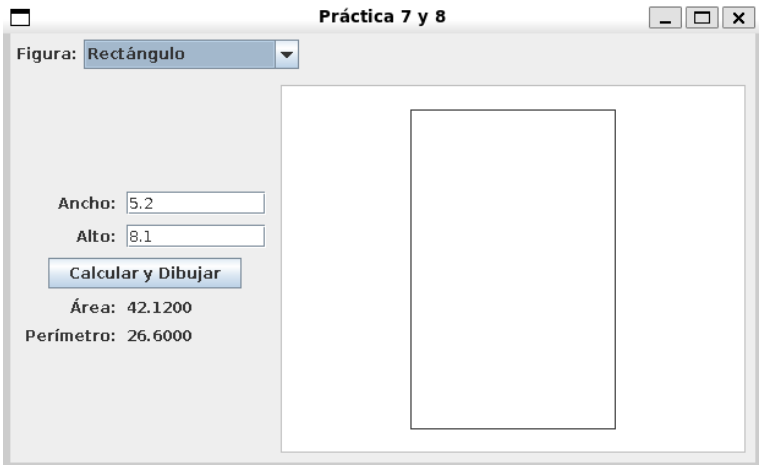

Figura 3: Figura del rectángulo mostrando el cálculo de su área y perímetro a partir de la base y altura

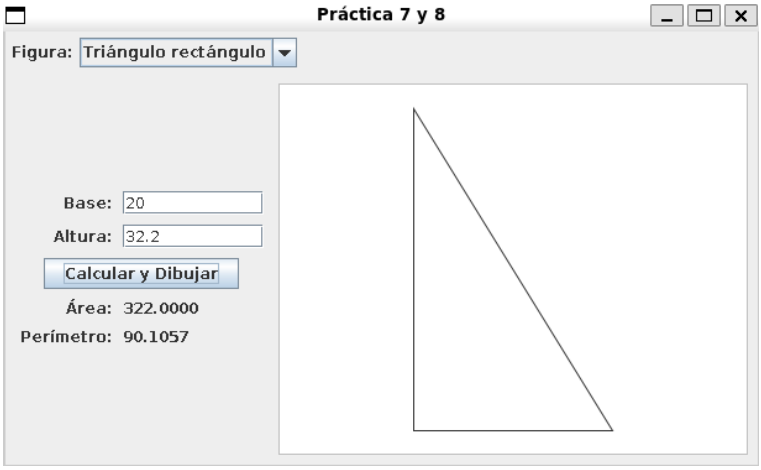

Figura 4: Figura del triángulo mostrando el cálculo de su área y perímetro a partir de la base y altura

5. Conclusiones

Esta práctica permitió aplicar exitosamente los conceptos fudamentales de herencia y poliformismo en java. Se utilizo una clase abstracta "Figura"para definir una estructura común obligango a las clases derivadas (Circulo, Rectangulo, Triangulorectangulo) al implementar sus propios metodos area(),perimetro() y dibujar() mediante la sobresritura. El polimorfismo fue clave, ya que permitio la clase Mainapp tratar a todos los objetos simplemente como figura, simplificando la lógica de la interfaz gráfica. Los resultados visuales y los calculos correctos demuestran que los objetivos se cumplieron, consolidando la comprensión de estos pilares de la Programación Orientada a objetos y su aplicacion práctica.

6. Referencias

[1] J.L. Blasco. Introducción a poo en java: Herencia y polimorfismo. (https://openwebinars.net/blog/introduccion-a-poo-en-java-herencia-y-polimorfismo/), 2023. Consultado el 16-10-2025.

[2] Nivardo. Sobrescritura de métodos en java. (https://oregoom.com/java/sobrescritura-de-metodos/), 2024. Consultado el 16-10-2025.