	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: POO

Grupo: 01

No de Práctica(s): Practica 7 y 8

Integrante(s): 322125337

322080869

322059179

323629814

322113536

No. de lista o brigada: 6

Semestre: 2025

Fecha de entrega: 17/10/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del problema**: La práctica se enfoca en completar una aplicación de escritorio desarrollada en Java con la biblioteca Swing. El programa debe ser capaz de calcular el área y el perímetro de tres figuras geométricas distintas: un círculo, un rectángulo y un triángulo rectángulo. La tarea principal consiste en realizar correctamente la lógica de las figuras, aplicando los conceptos de herencia a partir de una clase abstracta Figura y polimorfismo para los métodos de cálculo y dibujo.
- **Motivación**: El desarrollo de esta aplicación permite pasar del conocimiento teórico de la Programación Orientada a Objetos a una implementación más práctica. Utilizar herencia y polimorfismo es una técnica fundamental para construir software reutilizable y fácil de mantener.
- **Objetivos**: Los objetivos específicos a alcanzar con la resolución de esta práctica son los siguientes:
  - Aplicar la herencia mediante la creación de clases concretas (Circulo, Rectangulo, TrianguloRectangulo) que extiendan la clase abstracta Figura.
  - Implementar el polimorfismo al sobreescribir los métodos abstractos area(), perimetro() y dibujar() en cada subclase, proporcionando la lógica específica para cada figura.
  - Reforzar la comprensión de cómo las clases abstractas y el polimorfismo facilitan la creación de código robusto, organizado y escalable.

2. Marco Teórico

Herencia: Nos permite crear nuevas clases (subclases o clases hijas) a partir de clases existentes (superclases o clases padres). La clase hija hereda los atributos y métodos de la clase padre, lo que facilita la reutilización de código y la creación de una jerarquía de clases. Lo que significa que se pueden definir características y comportamientos comunes en una clase padre y luego crear clases hijas más especializadas que añaden o modifican esas características.[1]

Polimorfismo:Es la capacidad que tienen los objetos de diferentes clases de responder al mismo mensaje o método de manera específica para cada una. En Java, esto se logra comúnmente a través de la herencia y la sobrescritura de métodos (method overriding). La sobrescritura ocurre cuando una clase hija proporciona una implementación específica para un método que ya está definido en su clase padre. Para asegurar que la sobrescritura se realiza correctamente, es una buena práctica usar la anotación @Override. Esta anotación le indica al compilador que el método que sigue está destinado a anular un método de la superclase[2]

Biblioteca Swing: Swing es una biblioteca gráfica (GUI toolkit) para Java que forma parte de las Java Foundation Classes (JFC). Su principal función es proporcionar un conjunto de componentes, como botones, campos de texto, tablas y menús, para crear interfaces de usuario de escritorio, los componentes de Swing son "ligeros", lo que significa que están escritos completamente en Java y no dependen de los componentes nativos del sistema operativo. Esto garantiza que la apariencia y el comportamiento de la aplicación sean consistentes en diferentes plataformas.[3]

3. Desarrollo

Primero investigamos las herramientas que íbamos a utilizar en la práctica las cuales en este caso fueron: el manejo de interfaces gráficas en Java con la biblioteca Swing, la utilización de clases y herencia para modelar distintos tipos de figuras geométricas y el uso de la clase Graphics2D para dibujar las figuras en un panel. Una vez entendido esta parte, creímos necesario también comprender cómo funcionan los paneles, los eventos y los métodos de dibujo que ofrece Java así como la estructura de una aplicación con ventanas.

Una vez comprendido lo anterior empezamos a crear una clase abstracta llamada Figura la cual representa el modelo base para cualquier figura geométrica, esta clase define tres métodos abstractos: area(), perimetro() y dibujar(), esto permite que las clases hijas implementen sus propios cálculos y formas de representación visual según sus características.

Después definimos la clase Circulo la cual contiene el atributo radio.[en] esta clase se implementaron los métodos para calcular el área y el perímetro donde se usaron las fórmulas que ya conocemos (r^2 para el area y 2r para el perimetro) además se sobreescribió el método dibujar() donde utilizamos el objeto Graphics2D para trazar un círculo dentro del panel de dibujo aplicando un margen y una escala para que la figura se ajuste correctamente al tamaño del panel.

Despues desarrollamos la clase Rectangulo con los atributos ancho y alto, al igual que en el caso anterior se implementaron los métodos area() y perimetro() y en el método dibujar(), se calcularon las proporciones y coordenadas necesarias para centrar y escalar el rectángulo en pantalla en esta clase se muestra cómo adaptar la lógica del dibujo para diferentes tipos de figuras sin cambiar la estructura general del programa.

Luego creamos la clase TrianguloRectangulo que contiene los atributos base y altura, en ella el área se obtiene multiplicando la base por la altura y dividiendo entre dos, mientras que el perímetro se calcula sumando los tres lados, considerando el uso del teorema de Pitágoras para la hipotenusa, en su método dibujar() se representó un triángulo rectángulo calculando los tres puntos del polígono que forman la figura y también ajustando su tamaño a las dimensiones del panel.

Una parte importante fue la implementación de la clase PanelDibujo la cual hereda de JPanel y se encarga de mostrar visualmente la figura seleccionada, esta clase sobrescribe el método paintComponent() para dibujar con suavizado de bordes y recibir el objeto Figura que debe representarse, además el método setFigura() permite actualizar la figura actual y redibujarla automáticamente.

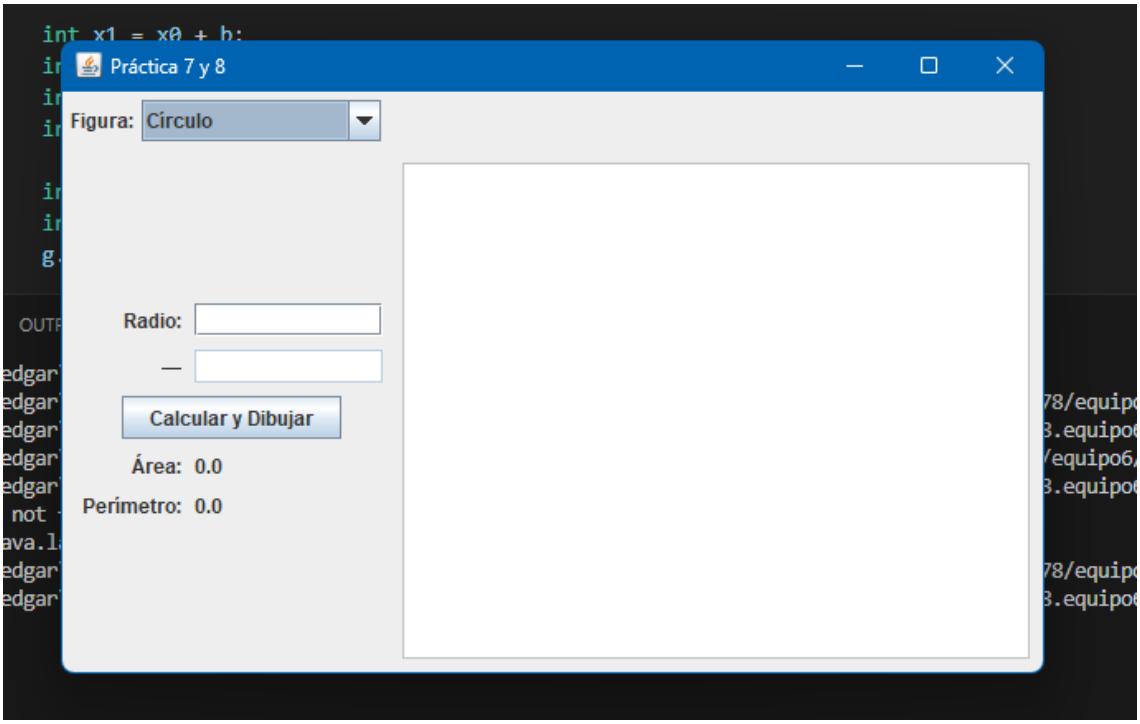
También implementamos la clase NumericTextField la cual es una mejora del campo de texto común que restringe la entrada únicamente a números y puntos decimales evitando que el usuario introduzca caracteres no válidos (como por ejemplo texto) esta clase incluye un método auxiliar safeParse() que convierte de forma segura el texto introducido a un valor numérico devolviendo 0.0 si el texto no es válido.

Por ultimo construimos la clase principal MainApp donde se diseña toda la interfaz gráfica en ella se crean los componentes como etiquetas, campos de texto, botones y un JComboBox para seleccionar el tipo de figura: Círculo, Rectángulo o Triángulo rectángulo, dependiendo de la figura seleccionada se actualizan los campos de entrada mediante el método actualizarFormulario().

4. Resultados

Se comprueba la correcta ejecución de cada uno de las figuras solicitadas, asi como tambien se grafica cada una y se calcula el area y el perimetro.

Estructura de la ventana



-En la parte superior hay un pequeño panel con una etiqueta y un menú desplegable, este sirve para elegir entre las figuras circulo, Rectangulo y Triangulo Rectangulo.

A la izquierda del centro de la ventana se muestra un formulario con etiquetas, campos y botones. Cuando está seleccionada cada figura, este formulario cambia dinámicamente:
-Si se elige circulo: Solo el campo “Radio” está habilitado. El segundo campo está desactivado porque no se necesita.
-Si se elige Rectangulo: Ahora ambos campos están habilitados (Alto y Ancho).
-Si se elige Triangulo Rectangulo: Nuevamente los dos campos están activos (Base y Altura)

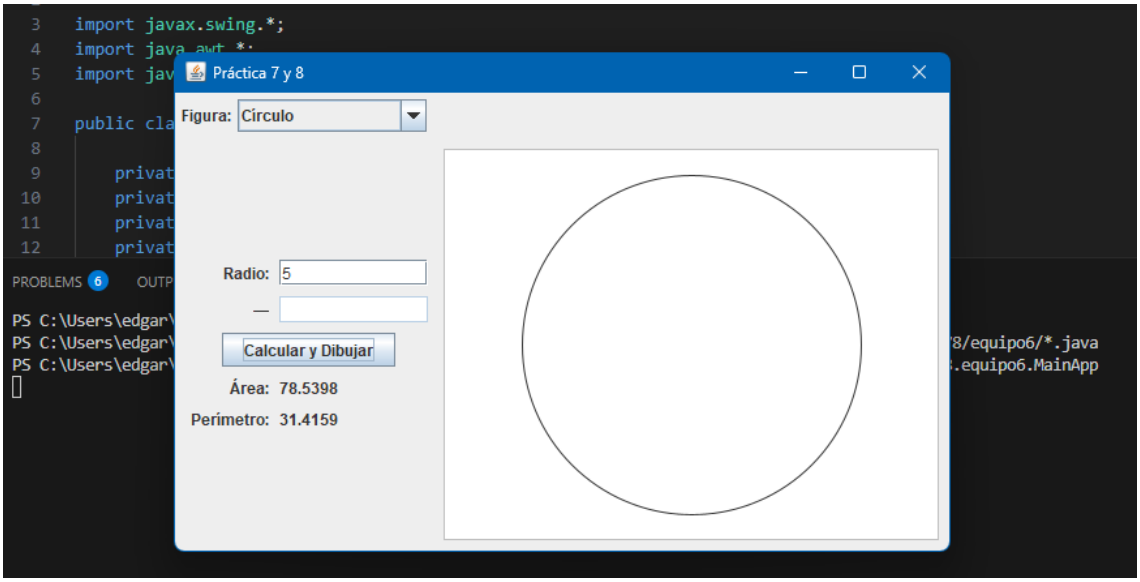
-A la derecha aparece un rectángulo blanco con borde gris (el PanelDibujo), de tamaño aproximado 380×300 píxeles ahí es donde se dibujará la figura elegida con el tamaño correspondiente.

-Si se escribe un valor no numérico (por ejemplo letras), NumericTextField no te dejará escribirlo, simplemente no aparecerán los caracteres.

-Si se deja un campo vacío o se escribe 0 el programa internamente reemplaza ese valor por 1.0 para evitar cálculos o dibujos de tamaño cero.

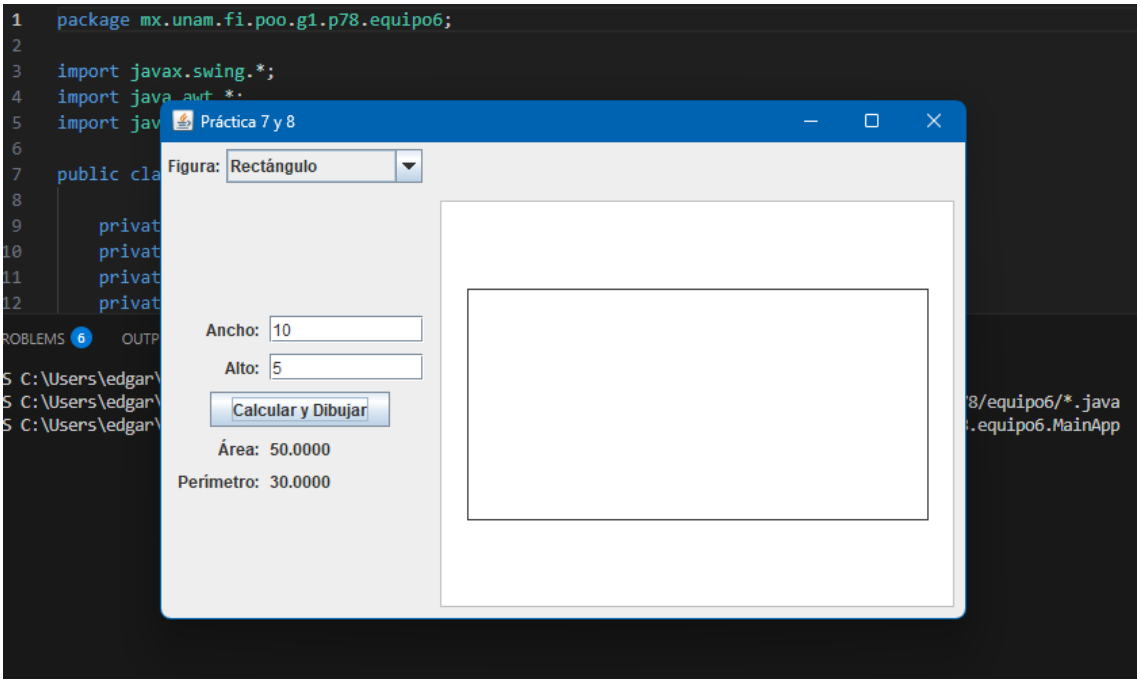
-Cuando se cambia de figura en el combo el dibujo anterior desaparece y se limpia el panel.

-Caso 1: Circulo



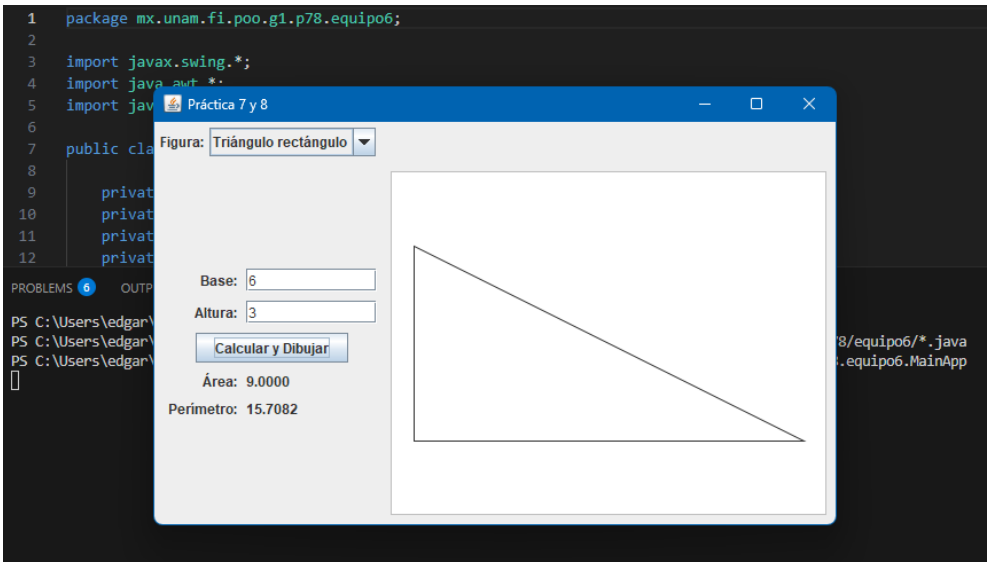
-En “Radio” en este caso ingresamos 5.
-Se presiona el botón “Calcular y Dibujar”.
-En la parte inferior del formulario, las etiquetas cambian a:
Área: 78.5398.
Perímetro: 31.4159.
-En el panel blanco se grafica la figura conforme los valores ingresados.

-Caso 2: Rectangulo:



- Se cambia la figura en el menu a “Rectángulo”.
-Los campos cambian automáticamente a ’Ancho y Alto’.
-En este caso ingresamos los valores Ancho: 10 y Alto: 5.
-Al presionar “Calcular y Dibujar”:
El área mostrará: 50.0000
El perímetro: 30.0000.
-En el panel aparece un rectángulo centrado, más ancho que alto, ajustado al espacio disponible del panel.

-Caso 3: Triangulo Rectangulo



- Se selecciona en el menu “Triángulo rectángulo”.

-Aparecen los campos “Base” y “Altura”.
-En este caso ingresamos los valores, Base: 6 y Altura:3.
-Se presiona el botón “Calcular y Dibujar”:
El área mostrará: 9.0000.
El perímetro: 15.7082.
-En el panel se dibuja un triángulo rectángulo centrado, con su base horizontal y la altura vertical.

5. Conclusiones

La correcta implementación de la aplicación de figuras geométricas demuestra de manera efectiva cómo los principios teóricos de la herencia y el polimorfismo son fundamentales para el desarrollo de software estructurado y escalable. La creación de la clase abstracta Figura sirvió para que todas las clases concretas (Circulo, Rectangulo, TrianguloRectangulo) implementaran una funcionalidad común para calcular su área y perímetro, así como para ser dibujadas. El polimorfismo fue el concepto central que permitió que la lógica principal de la aplicación, gestionada en la clase MainApp, interactuara con cualquier objeto de tipo Figura de manera genérica. Al invocar los métodos area(), perimetro() y dibujar(), el sistema ejecutaba la versión específica de la subclase correspondiente sin necesidad de comprobaciones de tipo explícitas.

Los resultados visuales y numéricos obtenidos en la interfaz gráfica son lo que demuestra que se cumplieron los objetivos principales de la práctica. El hecho de que cada figura se calcule y se dibuje correctamente no deja ver que la aplicación de la herencia para compartir una estructura común y del polimorfismo para definir comportamientos específicos resultó en un programa funcional, robusto y fácil de mantener.

6. Referencias bibliográficas

[1] Oracle. (s.f.). Inheritance (The Java™ Tutorials >Learning the Java Language >Interfaces and Inheritance). Oracle Help Center.
https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

[2] Oracle. (s.f.). Polymorphism (The Java™ Tutorials >Learning the Java Language >Interfaces and Inheritance). Oracle Help Center.
https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html

[3] Oracle. (s.f.). Lesson: Getting Started with Swing (The Java™ Tutorials >Creating a GUI With Swing). Oracle Help Center.
https://docs.oracle.com/javase/tutorial/uiswing/start/index.html