	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación
Salas A y B

Profesor(a):

Asignatura:

Grupo:

No de Práctica(s): Practica 7-8

Integrante(s): 322246320

322267567

322012051

322236688

322330872

No. de lista o brigada:

Semestre

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del Problema:** Se busca implementar una aplicación en Java que permita representar distintas figuras geométricas, utilizando los principios de herencia y polimorfismo. La práctica consiste en crear una clase abstracta 'Figura' que defina las características y comportamientos generales, y clases derivadas que sobrescriban los métodos necesarios para calcular las áreas o perímetros, demostrando así el uso del polimorfismo en tiempo de ejecución.
- **Motivación:** En esta práctica aprenderemos a utilizar la herencia para reutilizar código y establecer jerarquía entre clases, así como aprender a aplicar el polimorfismo para modificar comportamientos de manera dinámica según el tipo de objeto.
- **Objetivos:** Implementar un programa que utilice una clase abstracta y varias clases derivadas de ella para representar figuras geométricas aplicando herencia y polimorfismo. Comprender cómo las clases derivadas pueden sobrescribir métodos de la clase padre y cómo el uso de referencias polimórficas permite invocar diferentes comportamientos con una misma interfaz.

2. Marco Teórico

- **Herencia:** Es un concepto que permite definir nuevas clases basadas en clases ya existentes, aprovechando y extendiendo su funcionalidad, ahorrando tiempo y esfuerzo, creando una clase en base a otra con el uso de la palabra reservada 'extends' al declarar la clase. Se puede obtener herencia de dos tipos de clases: clases normales, lo que permite acceder a sus métodos; clases abstractas, lo que hace que la clase que heredó, solo tenga una guía de que métodos debe tener y completar. [1]
- **Polimorfismo:** Es la capacidad de un objeto para adquirir diferentes comportamientos, mas especificamente, es el concepto de implementar métodos con el mismo nombre pero distinto comportamiento en distintas clases. [1]
- **Clases abstractas:** Son un tipo de clases de las cuales no se puede crear un objeto a partir de ellas, pero sirven para definir una especie de 'esqueleto' que otras clases deben seguir al momento de heredar de la clase; una clase abstracta puede contener métodos abstractos, que indican que se deben implementar en la clase que herede o metodos con comportamiento ya definido; la forma en la que se crea una clase abstracta es usando la palabra reservada 'abstract' en vez de public al momento de crear la clase. [2]

3. Desarrollo

Librerías, clases, paso de mensajes y empaquetado de clases: Para esta práctica se ha implementado un graficador de diferentes figuras geométricas con interfaz gráfica de usuario (GUI). Para ello, las principales librerías que se han utilizados son:

1. **javax.swing.*** : En este paquete se incluyen las clases necesarias para crear una interfaz gráfica de usuario, permite además crear ventanas, botones, listas y campos de texto.

2. **java.awt.*** : Proporcionan clases para modificar el diseño de las ventanas (dimensiones, colores, etc).

Todas las clases han sido organizadas o empaquetadas dentro de la carpeta:
/mx/unam/fi/poo/p78/, al agregar: **package mx.unam.fi.poo.p78;** al inicio de cada clase.

La clase abstracta **Figura** representa un objeto genérico dentro de las figuras que se pueden crear, contiene los métodos abstractos **area()**, **perimetro()** y **dibujar()**, esta última nos ayudara a crear el dibujo de la figura seleccionada.

En la subclase **Circulo** heredada de **Figura**, se ha inicializado un constructor con un atributo privado llamado **radio** para el que se le han generado sus respectivos **set** y **get**. Implementamos también los métodos sobreescritos **area()**, **perimetro()** y **dibujar**.

Para la subclase **Rectangulo** heredada de la misma clase **Figura**, inicializamos el constructor con los atributos privados **alto** y **ancho**, con sus respectivos **setters y getters**. Igualmente incluye los métodos sobreescritos **area()**, **perimetro()** y **dibujar**.

Por último desarrollamos la subclase **TrianguloRectangulo** heredada de **Figura**, con su constructor, atributos privados **base** y **altura**. Nuevamente esta subclase presenta los métodos **area()**, **perimetro()** y **dibujar**.

Finalmente para **MainApp** se inicializaron diversos atributos, que sirven para crear la ventana (frame), lista desplegable para crear figuras (comboFiguras), pies de página para los campos de texto (l1, l2), campos de texto(t1, t2), muestra del área y perímetro calculados (areaLbl, perLbl), panel donde se dibuja la figura (panelDibujo) y cadenas de texto fijas para identificar a las figuras(FIGCIRCULO, FIGRECTANGULO, FIGTRIANGULO).
Los métodos implementados son:

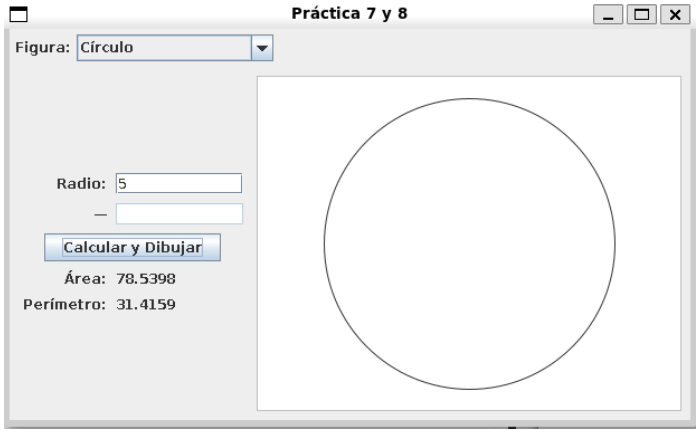
1. **crearGUI()**: Crea la ventana principal de la aplicación.

2. **actualizarFormulario()**: Sirve para actualizar las etiquetas (l1, l2), los campos (t1, t2) y el estado de la interfaz según la figura seleccionada.

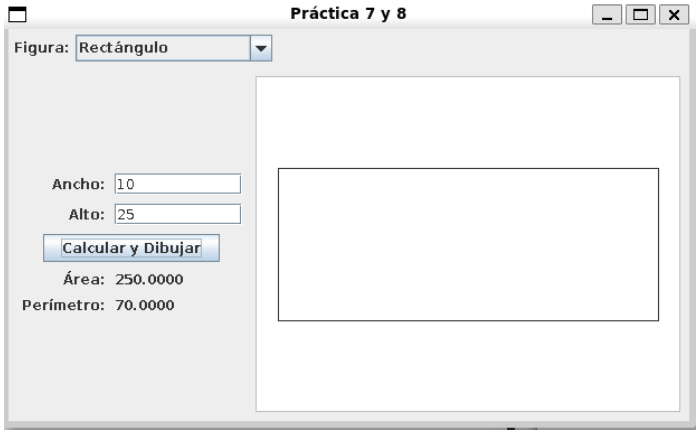
3. **calcularYDibujar()**: Lee los valores numéricos, crea el objeto de la figura, calcula su área y perímetro, y lo manda a dibujar.

4. **main()**: Inicia la interfaz gráfica

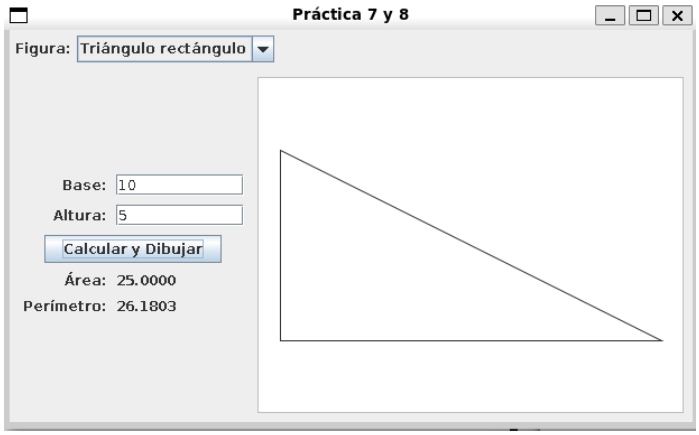
4. Resultados



Pruebas de circulo



Pruebas de rectángulo



Pruebas de triangulo

5. Conclusiones

La práctica nos permitió comprender y aplicar de manera efectiva los conceptos de herencia y polimorfismo en Java, observando cómo es que una clase abstracta puede servirnos como una base para definir comportamientos generales que las clases derivadas adaptan según sus necesidades. A través de la creación de distintas figuras geométricas, se demostró cómo el polimorfismo en tiempo de ejecución posibilita invocar métodos sobrescritos de forma dinámica, facilitando la extensibilidad y reutilización del código. En conjunto, la práctica nos permitió fortalecer la comprensión de la programación orientada a objetos, destacando la importancia de diseñar jerarquías de clases bien estructuradas para lograr programas más flexibles, organizados y eficientes.

Referencias

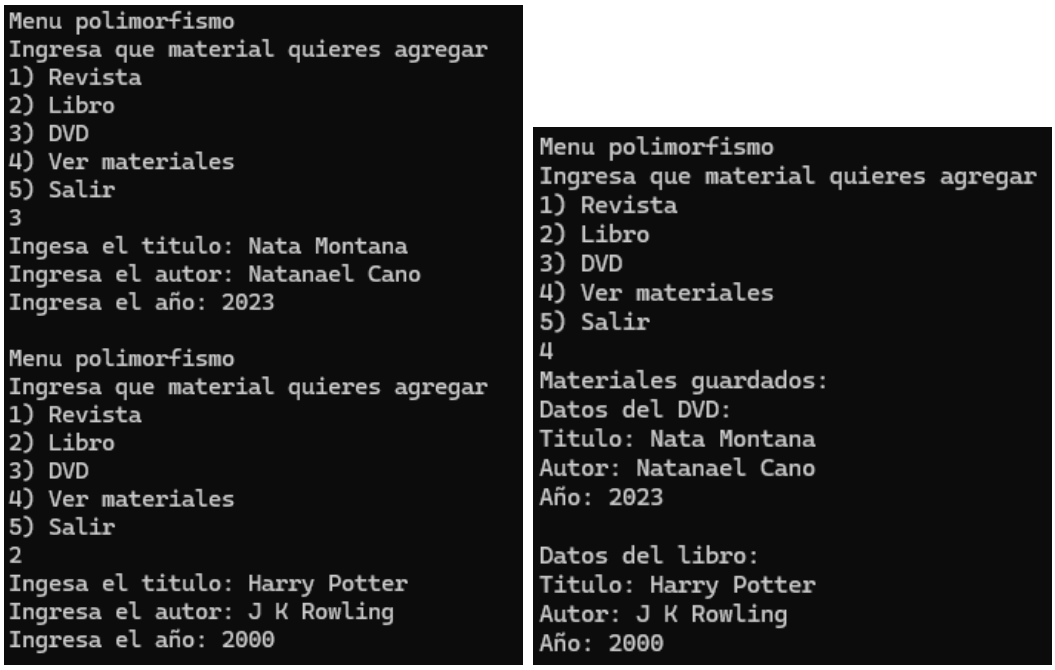
[1] *Introducción a POO en Java: Herencia y polimorfismo*. 2023-11-17. URL: https://openwebinars.net/blog/introduccion-a-poo-en-java-herencia-y-polimorfismo/ (visitado 17-10-2025).

[2] *Java Clases Abstractas y el polimorfismo*. 2024-09-28. URL: https://www.arquitecturajava.com/java-clases-abstractas-y-el-polimorfismo/ (visitado 17-10-2025).

6. Reto para token

Para el desarrollo de este token, realizamos la implementación de la clase 'Material', que es la clase padre de las clases 'Revista', 'DVD' y 'Libro'. Cada clase sobrescribe un método llamado 'mostrarInformacion', que se sobrescribe para cada material y se utiliza para imprimir su información correspondiente

Dentro del 'main', creamos un menú donde se le da a elegir al usuario que tipo de material quiere agregar, y mediante polimorfismo se convierte un objeto de tipo 'Material' en el seleccionado por el usuario. Cada material es guardad en un 'ArrayList' utilizado para imprimir los materiales ingresados.



Ejecución