	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Dávila Pérez René Adrián

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): 7 y 8

Integrante(s): 322089020

322089817

322151194

425091586

322085390

No. de lista o brigada: Equipo 4

Semestre: 2026-1

Fecha de entrega: 17 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

1.1. Planteamiento del Problema

La presente práctica tiene como propósito aplicar los fundamentos de la Programación Orientada a Objetos mediante el desarrollo de un programa en Java que permita representar y calcular las propiedades geométricas de distintas figuras básicas. El problema consiste en crear una aplicación gráfica interactiva donde el usuario pueda seleccionar una figura geométrica, ingresar sus dimensiones y obtener automáticamente su área y perímetro. Para ello, se deben implementar correctamente los principios de herencia, polimorfismo, abstracción y encapsulamiento, garantizando una estructura de clases organizada, reutilizable y funcional.

1.2. Motivación

El siguiente trabajo busca fortalecer los conocimientos previos sobre interfaz gráfica: a partir de la creación de figuras geométricas básicas en una ventana; herencia: al determinar cómo heredar atributos o métodos de una clase padre a una clase hija y polimorfismo: buscar una forma para que los objetos puedan tener más de una forma; y con lo anterior entender la extensión de funcionalidades en el programa. También, se busca que el alumno pueda proteger la integridad de los datos a partir del encapsulamiento.

1.3. Objetivos

Los objetivos de la práctica son:

- Diseñar una interfaz gráfica clara y funcional que permita visualizar y calcular propiedades de figuras geométricas básicas.
- Implementar adecuadamente la herencia y el polimorfismo para optimizar la reutilización de código y la extensión de comportamientos.
- Aplicar el encapsulamiento para proteger los datos internos de las clases.

- Validar el funcionamiento del programa mediante pruebas que confirmen la correcta aplicación de los conceptos teóricos.

2. Marco Teórico

2.1. Herencia

La herencia es un pilar de la programación orientada a objetos que permite crear una clase nueva (denominada subclase o clase derivada) a partir de una clase ya existente (conocida como superclase o clase base) [2]. La subclase adquiere los atributos y métodos de la superclase, lo que facilita la reutilización de código y establece una jerarquía lógica entre las clases [3]. La herencia es el proceso que implica la creación de clases ya existentes, permitiendo agregar más funcionalidades.

En Java, la herencia se implementa utilizando la palabra clave extends.

2.2. Polimorfismo

La palabra polimorfismo, de origen griego, significa "muchas formas". En este contexto, es la capacidad que permite que objetos de distintas clases respondan a un mismo mensaje (una llamada a un método) de maneras diferentes y específicas para cada clase [4].

El polimorfismo permite que una variable de referencia declarada con el tipo de una superclase pueda apuntar a objetos de cualquiera de sus subclases, cuando se invoca un método a través de esta referencia, el sistema decide en tiempo de ejecución cuál es la versión correcta del método que debe ejecutarse, basándose en el tipo real del objeto [6]. Esto se logra comúnmente a través de la sobrescritura de métodos (overriding). Es la habilidad de redefinir el comportamiento de un método específico en una subclase.

2.3. Ventajas y Riesgos de la Herencia y el Polimorfismo

Ventajas:

- Promueven la **reutilización de código**, reduciendo duplicidades.
- Favorecen la **extensibilidad**, permitiendo agregar nuevas clases sin modificar las existentes.
- Aumentan la **claridad y coherencia** del diseño mediante jerarquías lógicas.

Riesgos:

- Una dependencia excesiva entre clases puede dificultar el mantenimiento.
- La herencia mal aplicada puede generar jerarquías complejas o poco intuitivas.
- Se debe evitar el abuso de herencia cuando la composición (*"tiene-un"*) es más apropiada [1].

2.4. Abstracción

La abstracción consiste en identificar las características esenciales de un objeto y ocultar los detalles innecesarios. En programación, permite centrarse en lo que un objeto hace y no en cómo lo hace. Una clase es una abstracción que define atributos y métodos comunes a un conjunto de objetos similares [5].

3. Desarrollo

3.1. Implementación del Código

El programa implementa una aplicación gráfica que permite al usuario calcular y visualizar el área y perímetro de tres figuras geométricas: círculo, rectángulo y triángulo rectángulo. Está desarrollado con el lenguaje Java y hace uso del paradigma de Programación Orientada a Objetos (POO), aplicando conceptos de herencia, polimorfismo, abstracción y encapsulamiento. Se utilizan las siguientes librerías:

- `javax.swing`: para construir la interfaz gráfica con botones, etiquetas, campos de texto y paneles.
- `java.awt`: para manejar los componentes gráficos, la disposición (layouts) y el dibujo de las figuras.
- `java.lang.Math`: para operaciones matemáticas como potencias y raíces cuadradas.

3.1.1. Figura

Esta clase es **abstracta** y representa el modelo general de una figura geométrica. Define tres métodos abstractos: `area()`, `perimetro()` y `dibujar(Graphics2D g, Dimension size)`. Las subclases están obligadas a sobrescribir estos métodos, aplicando el **polimorfismo**. Así, desde una referencia del tipo `Figura`, se puede invocar el mismo método y obtener un comportamiento distinto según la figura seleccionada.

3.1.2. Circulo

Hereda de `Figura` y representa una figura con radio definido. Sobrescribe los métodos:

- `area()`: devuelve πr^2
- `perimetro()`: devuelve $2\pi r$

- `dibujar()`: dibuja un círculo centrado en el panel con `drawOval()`.

Demuestra herencia y polimorfismo porque redefine los métodos de cálculo y dibujo según su propia forma.

3.1.3. Rectangulo

Hereda de `Figura` y tiene los atributos `ancho` y `alto`. Sobrescribe:

- `area() = ancho × alto`
- `perimetro() = 2 × (ancho + alto)`
- `dibujar()`: calcula la escala proporcional y usa `drawRect()` para representar el rectángulo.

El polimorfismo permite ejecutar el método correspondiente al tipo de figura sin conocer su clase específica.

3.1.4. TrianguloRectangulo

Hereda de `Figura` y contiene los atributos `base` y `altura`. Sobrescribe:

- `area() = (base × altura) / 2`
- `perimetro() = base + altura + sqrt(base2̂ + altura2̂)`
- `dibujar()`: traza los tres vértices del triángulo usando `drawPolygon()`.

3.1.5. PanelDibujo

Extiende de `JPanel` y funciona como área de dibujo. Contiene un atributo `Figura figura`, que puede referirse a cualquier subclase. Cuando se llama a `setFigura(f)`, el panel repinta y ejecuta `figura.dibujar()`, mostrando el **polimorfismo dinámico**.

3.1.6. NumericTextField

Extiende de `JTextField` y restringe la entrada del usuario a números decimales, evitando caracteres inválidos. El método estático `safeParse()` convierte de forma segura el texto ingresado en un valor `double`.

3.1.7. MainApp

Contiene el `main()` y la interfaz gráfica. El usuario selecciona la figura desde un `JComboBox`, ingresa sus dimensiones y presiona el botón `Calcular y Dibujar`. Dependiendo de la selección, se crea una instancia de la figura correspondiente (`Circulo`, `Rectangulo` o `TrianguloRectangulo`). Luego se muestran el área y el perímetro, y la figura se dibuja en el panel.

Gracias a la herencia, el código no necesita saber el tipo exacto de figura: todas comparten la interfaz común definida en `Figura`.

3.2. Pruebas

3.2.1. Ejemplo 1

Figura 3: Círculo **Radio:** 4

- Área = 50.2655
- Perímetro = 25.1327
- En el panel se dibuja un círculo centrado con su area y perimetro calculada automáticamente.

3.2.2. Ejemplo 2

Figura 4: Rectángulo **Ancho:** 5 **Alto:** 3

- Área = 15.0000
- Perímetro = 16.0000
- En el panel se muestra un rectángulo proporcional con su area y perimetro calculada automáticamente.

3.2.3. Ejemplo 3

Figura 5: Triángulo rectángulo **Base:** 6 **Altura:** 8

- Área = 24.0000
- Perímetro = 24.0000
- Se dibuja el triángulo con su area y perimetro calculada automáticamente.

4. Resultados

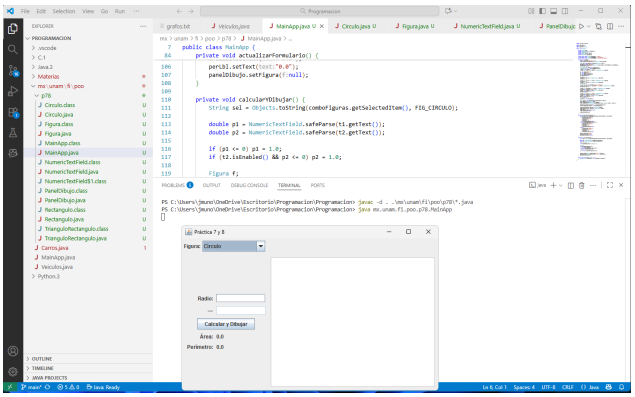

Figura 1: Se compila y ejecuta el código principal

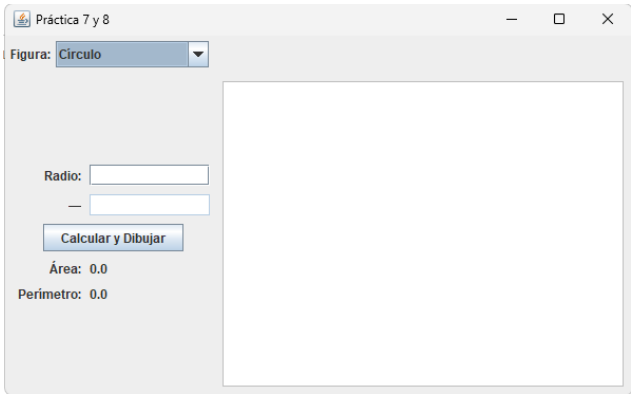

Figura 2: Saldrá una ventana con todas las posibles acciones

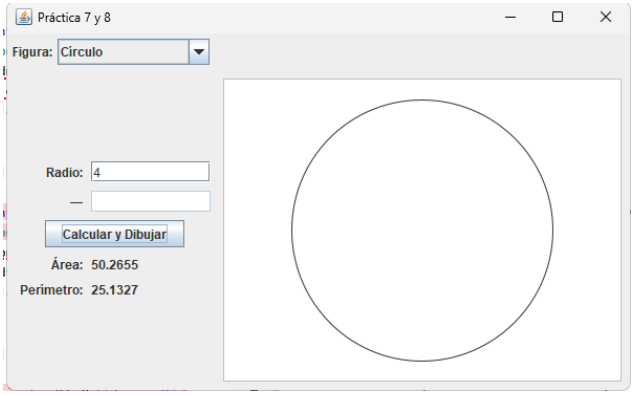

Figura 3: Se prueba calcular el area de un Cirulo de radio 4

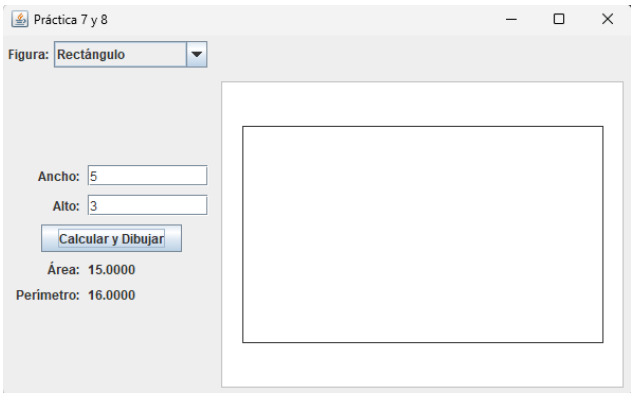

Figura 4: Se prueba calcular el area de un Rectangulo de 5x3

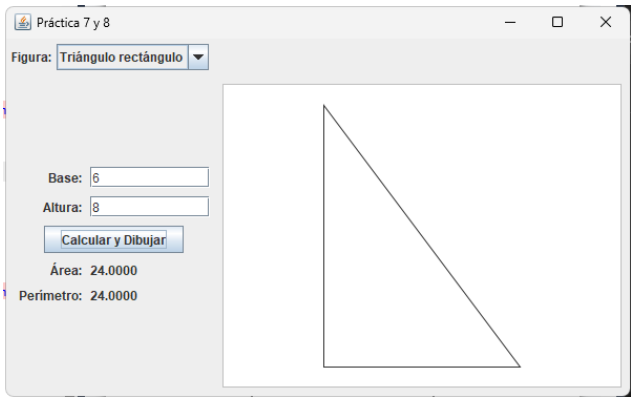

Figura 5: Se prueba calcular el area de un Rectangulo de 6x8

5. Conclusiones

La práctica permitió comprobar la efectividad de los principios de la Programación Orientada a Objetos en la resolución de un problema concreto: la representación y cálculo de propiedades de figuras geométricas. A través del uso de herencia, se logró estructurar las clases de manera jerárquica y coherente; mediante el polimorfismo, se implementaron comportamientos específicos para cada figura sin modificar el código base; y con el encapsulamiento, se mantuvo la integridad de los datos dentro de cada clase.

Los resultados obtenidos evidencian que el diseño orientado a objetos facilita la extensibilidad del programa, ya que nuevas figuras pueden incorporarse fácilmente sin alterar la estructura existente. Además, la implementación de una interfaz gráfica intuitiva demuestra la integración exitosa entre la lógica de programación y la presentación visual.

En conclusión, esta práctica refuerza la importancia de los conceptos teóricos aplicados en clase, mostrando cómo la herencia, el polimorfismo y la abstracción son herramientas esenciales para el desarrollo de aplicaciones robustas, legibles y mantenibles en el lenguaje Java.

Referencias

[1] IBM Developer. *Introduction to Object-Oriented Programming Concepts*. [En línea]. Disponible en: https://developer.ibm.com/articles/intro-to-oop/. 2023. URL: https://developer.ibm.com/articles/intro-to-oop/ (visitado 17-10-2025).

[2] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3.ª ed. Boston, MA: Addison-Wesley, 2004.

[3] Lenovo. *Herencia en Programación: ¿qué es?* [En línea]. Disponible en: https://www.lenovo.com/mx/es/glosario/herencia/. 2025. URL: https://www.lenovo.com/mx/es/glosario/herencia/ (visitado 14-10-2025).

[4] Lenovo. *Polimorfismo: ¿Qué es y cómo funciona?* [En línea]. Disponible en: https://www.lenovo.com/mx/es/glosario/polimorfismo/. 2025. URL: https://www.lenovo.com/mx/es/glosario/polimorfismo/ (visitado 14-10-2025).

[5] Oracle. *Abstraction in Java*. [En línea]. Disponible en: https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html. 2024. URL: https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html (visitado 17-10-2025).

[6] UNAM. *Polimorfismo*. [En línea]. Disponible en: https://repositorio-uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/3062/mod_resource/content/1/UAPA-Polimorfismo/index.html. 2025. URL: https://repositorio-uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/3062/mod_resource/content/1/UAPA-Polimorfismo/index.html (visitado 14-10-2025).