	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 01

No. de práctica(s): 07 y 08

Integrante(s):
322276824
322258516
425037384
320108116
322221415

No. de brigada: 01

Semestre: 2026-1

Fecha de entrega: 17 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del problema:**
  Desarrollar una aplicación en Java que dibuje figuras, calculando el área y perímetro de estas, recibiendo como entrada los datos ingresados por el usuario en una interfaz gráfica, mostrando como opciones las distintas figuras que son posibles dibujar, acoplando los conceptos de herencia y polimorfismo para la reutilización y organización del programa.

- **Motivación:**
  Los conceptos de herencia y polimorfismo son claves en el paradigma de la programación orientada a objetos, pues facilitan la creación de programas que abstraen conceptos más complejos, añadiendo consistencia, claridad y utilidad al posibilitar la reutilización de código, fomentando un diseño modular y con mayor flexibilidad para adaptarse a los requerimientos del usuario. Además, comprender cómo una clase abstracta puede servir de base para múltiples subclases y cómo pueden comportarse de forma diferente ante los mismos métodos es esencial para el diseño extensible de software.

- **Objetivos:**
  Aplicar los conceptos de herencia y polimorfismo en la implementación de clases derivadas, integrando de manera correcta la lógica de los conceptos y sobrescribir métodos abstractos para implementar comportamientos específicos, con el objetivo de lograr un programa funcional y entendible que permita reforzar los conceptos y su aplicación.

2. Marco Teórico

Herencia

La herencia es un mecanismo que permite que una clase derive de otra, adquiriendo sus atributos y métodos. Esto con el objetivo de fomentar la reutilización del código y la creación de jerarquías. De acuerdo con [3] “Una clase que se deriva de otra clase se denomina subclase (también clase derivada, clase extendida o clase hija). La clase de la que se deriva la subclase se denomina superclase (también clase base o clase padre).”.

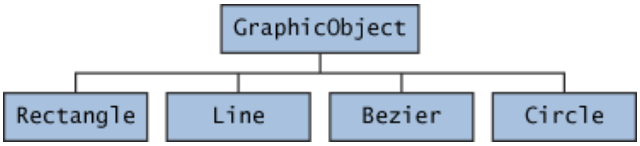

Figura 1: Diagrama demostrativo de herencia. [1]

Polimorfismo

El polimorfismo se refiere a la capacidad de un mismo método de comportarse de manera distinta según el objeto que lo invoque. Esto permite que una referencia de un tipo pueda comportarse como uno diferente, ejecutando el método adecuado en cada caso. El polimorfismo es la habilidad de un objeto para tomar distintas formas y responder adecuadamente según su tipo real. [4]

Clase abstracta

Una clase abstracta es aquella que no puede instanciarse directamente y que sirve como plantilla para sus subclases. Contiene métodos abstractos que deben ser implementados por las clases derivadas. Las clases abstractas son útiles cuando se espera que las clases que extienden de la clase abstracta tengan muchos métodos o campos en común o cuando se desea compartir código entre varias clases estrechamente relacionadas. [1]

Perímetros y Superficies

Cada figura comparte características comunes (área y perímetro), pero requiere implementar sus propios cálculos [2]:

Círculo:

- Perímetro:
$$P = 2\pi r$$
- Área:
$$A = \pi r^2$$

Rectángulo:

- Perímetro:
$$P = 2(a + b)$$
- Área:
$$A = ab$$

Triángulo rectángulo:

- Perímetro:
$$P = a + b + c$$
- Área:
$$A = \frac{bh}{2}$$

3. Desarrollo

La aplicación se estructuró dentro del paquete `mx.unam.fi.poo.p78`, siguiendo las buenas prácticas de modularidad y organización de proyectos en Java. Está compuesta por una clase abstracta base llamada `Figura`, tres subclases concretas: `Circulo`, `Rectangulo` y `TrianguloRectangulo`, y un conjunto de clases de apoyo para la interfaz gráfica: `MainApp`, `PanelDibujo` y `NumericTextField`. Estas clases permitieron dividir de una manera organizada el programa, cada una cumpliendo una labor específica.

Clase Figura

Esta clase funciona como la superclase abstracta de toda la jerarquía. Es una clase que define tres métodos: `area()` y `perimetro()` que son de tipo `double` y `dibujar()` que no regresa valor alguno. Esta clase, por su naturaleza abstracta, prepara sus métodos para heredarlos a clases más concretas que serán las encargadas de implementarlos.

Clase Circulo

Es una clase que extiende de la clase `Figura` e implementa los métodos para calcular el área y el perímetro usando las expresiones anteriormente mencionadas. Se redefine el método `dibujar()` haciendo uso de `@Override`, donde se utiliza el objeto `Graphics2D` y las dimensiones del panel para representar visualmente el círculo de manera proporcional además de realizar cálculos de escalado para que el radio se ajuste al tamaño de la ventana, finalmente usa el método `drawOval` para dibujar el círculo.

Clase Rectangulo

Esta clase hereda de `Figura` e implementa los métodos de cálculo de área y perímetro según las propiedades de un rectángulo. El método `dibujar()` calcula la escala en función del espacio disponible en el panel, conservando las proporciones originales del rectángulo, y lo representa centrado en la interfaz mediante el método `drawRect()` del objeto gráfico.

Clase TrianguloRectangulo

Es una clase que también extiende de `Figura` y define sus propios métodos para el área y el perímetro, considerando que la hipotenusa se obtiene mediante el método `Math.hypot()`. En el método `dibujar()`, se calcula una escala que mantiene las proporciones del triángulo en el panel. Luego, se trazan los tres vértices para formar el triángulo rectángulo y dibujarlo usando el método `drawPolygon`.

Clase PanelDibujo

Esta clase, que no fue modificada, extiende de `JPanel` y se encarga de representar gráficamente la figura que se le asigne. Recibe una referencia a un objeto de tipo `Figura`, invocando su método `dibujar()` dentro del método sobrescrito `paintComponent()`. Gracias al polimorfismo, el panel puede dibujar cualquier tipo de figura concreta sin necesidad de conocer su clase exacta.

Clase NumericTextField

Sirve para restringir la entrada del usuario a valores válidos mediante la captura de eventos de teclado. Su método estático `safeParse()` garantiza una conversión segura de texto a número, devolviendo un valor de control en caso de entrada inválida.

Clase MainApp

La clase principal implementa la interfaz gráfica y coordina toda la lógica de la aplicación, el usuario puede seleccionar el tipo de figura mediante un `JComboBox`, ingresar las dimensiones correspondientes en campos de texto numéricos que estarán activados o no dependiendo de la figura seleccionada y presionar el botón `Calcular y Dibujar` con el que dependiendo de la figura elegida, el programa crea un nuevo objeto de tipo `Figura` y, a través del polimorfismo, calcula e imprime el área y el perímetro utilizando las implementaciones específicas de cada clase. También se incluye el método `actualizarFormulario` que da valores por defecto a cada etiqueta de la interfaz y activa o desactiva los campos de texto según sea la figura escogida. Finalmente, el objeto se envía al `PanelDibujo` para su representación gráfica.

4. Resultados

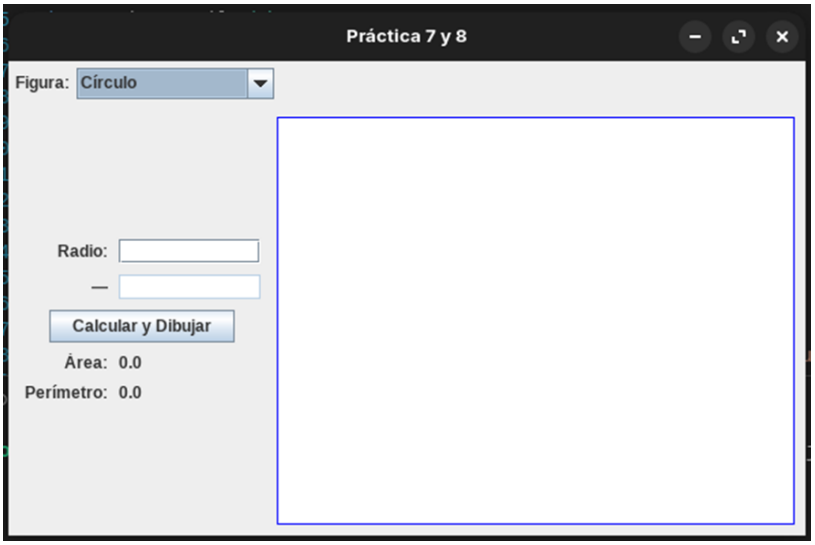

Figura 2: Interfaz principal de la aplicación.

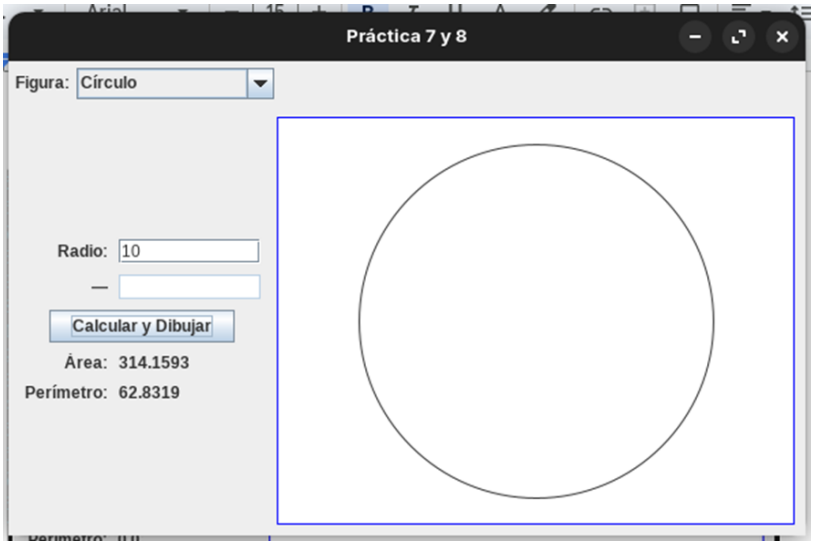

Figura 3: Representación y resultados del cálculo de un círculo.

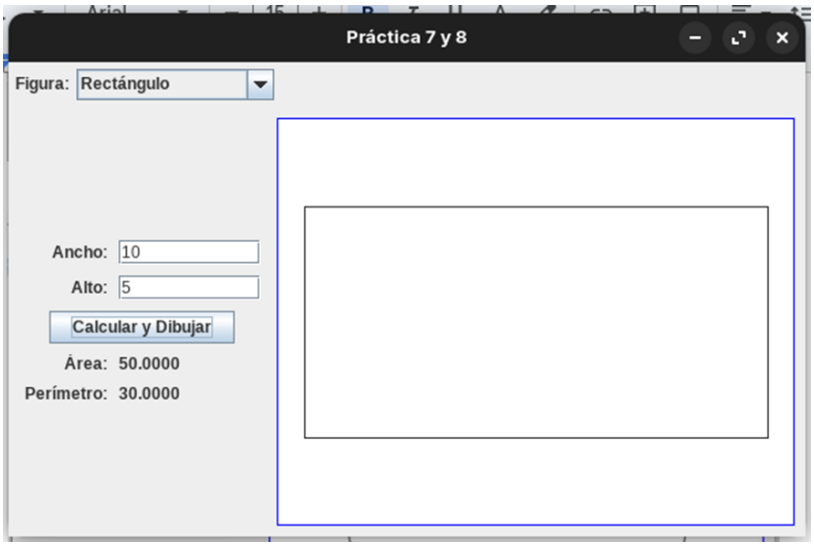

Figura 4: Representación y resultados del cálculo de un rectángulo.

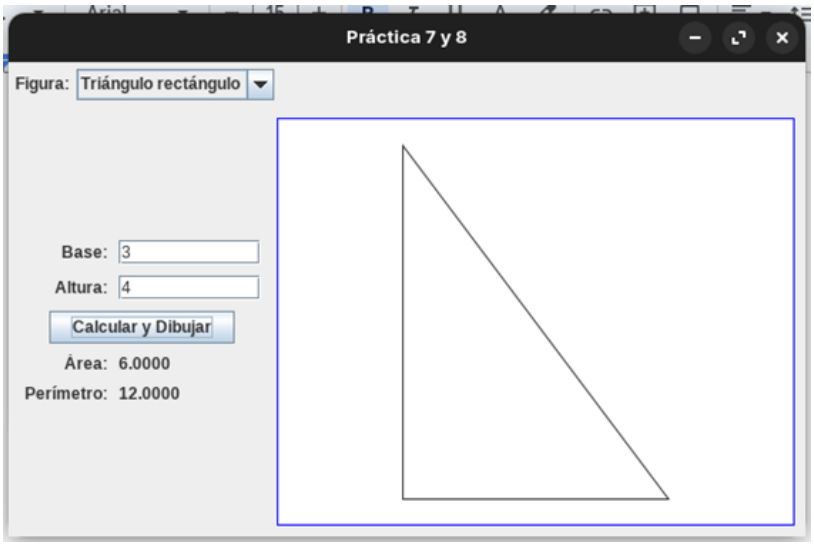

Figura 5: Representación y resultados del cálculo de un triángulo rectángulo.

5. Conclusiones

La práctica permitió consolidar los conocimientos sobre los principios de la herencia y el polimorfismo en Java, aplicados a un contexto gráfico e interactivo. Se demostró cómo una clase abstracta puede definir el comportamiento general de objetos, delegando a las subclases las implementaciones específicas de sus métodos, manteniendo una arquitectura limpia, extensible y mantenible. El uso del polimorfismo se evidenció al manipular objetos de distintas clases concretas a través de una referencia común de la clase padre, lo que permitió usar sus métodos y realizar los dibujos sin necesidad de conocer el tipo exacto del objeto. Asimismo, se fortaleció la comprensión del trabajo con interfaces gráficas, integrando los cálculos matemáticos con la representación visual de las figuras, y confirmando la importancia del diseño modular en el desarrollo de aplicaciones orientadas a objetos.

Referencias

[1] *Abstract Methods and Classes*. Oct. de 2025. URL: `https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html`.

[2] Arturo A. Márquez Fabián B. Bravo Herman A. Gallegos Miguel C. Villegas Ricardo R. Figueroa. *Geometría y Trigonometría*. Pearson, 2009.

[3] *Inheritance*. Oct. de 2025. URL: `https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html`.

[4] *Polymorphism*. Oct. de 2025. URL: `https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html`.