	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: 7

No de Práctica(s): 7 y 8

Integrante(s): 322044339

322094152

322114461

425093384

322150984

No. de brigada: 6

Semestre: 2026-1

Fecha de entrega: 15 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

Pensemos que estamos programando una aplicación en donde podemos calcular áreas y perímetros de diferentes figuras geométricas. Cada una de ellas (círculo, rectángulo, triángulo) necesita de ciertos datos independientes al resto; por ello, tendríamos que repetir código para acciones semejantes pero con ligeras variaciones para que logre su propósito, como el mostrar los resultados y dibujarlos en una interfaz. Veamos esto como una cocina, donde los cocineros usan los mismos espacios pero cada uno sigue su propia receta, usa sus propios utensilios y va a su propio ritmo, generando con ello ineficiencia y desorden. Sin un modelo que nos permita unir las características de las figuras, el código se volvería extensísimo, difícil de mantener y probablemente tienda a los errores si hacemos cambios pequeños.

Por ello, encontramos como herramientas a la herencia y polimorfismo, que nos permitirán crear una estructura organizada donde las figuras tendrán algo así como un tronco común pero donde cada una tendrá sus particularidades aún. Sería como tener una guía básica de instrucciones para las figuras geométricas en general, pero donde cada una aún conserve sus propios cálculos característicos. La herencia entra en juego al permitirnos crear una familia de figuras que comparten características fundamentales, mientras que el polimorfismo nos ayudaría a emplear un mismo método, digamos calcularArea(), pero adaptándolo a la figura con la que se trabaje. Aparte, el encapsulamiento protege los datos internos de las figuras permitiendo acceder a ellos únicamente por los getters y setters.

Con esta práctica se busca implementar una aplicación en Java que muestre el uso de herencia y polimorfismo, esto reflejado principalmente en la clase Figura, de la cual heredan las clases Círculo, Rectangulo y TrianguloRectangulo. Cada una de ellas debe tener su propia versión de los métodos, pero manteniendo la interfaz ya definida por la clase padre. Esta aplicación va a estar empaquetada bajo el nombre mx.unam.fi.poo.p78 y va a permitir la selección de distintas figuras, el ingreso de sus parámetros y visualizar la construcción de la figura a partir de ellos, esto con el fin de aprovechar las ventajas del polimorfismo para tratar de igual manera a objetos diferentes.

2. Marco Teórico

2.1. Herencia

La herencia es una parte fundamental dentro de la programación orientada a objetos, pues, permite construir nuevas clase a partir de otras previamente establecidas, pudiendo reciclar código de dicha clase; esta característica, nos permite compartir métodos y datos, entre clases u objetos, de forma directa.

Existen dos tipos de herencia definidos, "herencia simplez "herencia múltiple"; lo que las diferencia es que, en la herencia simple, solo puedes generar nuevas clases a partir de una sola superclase, mientras que en la herencia múltiple, las nuevas clases pueden ser definidas a partir de dos o más superclases.

Una forma de poder representar el concepto de "herencia simple"de forma visual, es por medio de una estructura de árbol donde la raíz representa la clase que heredará sus atributos y métodos a las demás clases y cada nodo del diagrama representa una clase creada a partir de su ”superclase”.

2.1.1. Superclase

Son aquellas clases que servirán como base para poder construir nuevas, clases; son aquellas que heredarán sus datos y métodos, además cada superclase puede tener un número indeterminado de subclases. Dentro de un diagrama de árbol, representan la raíz.

2.1.2. Subclase

Son las clases derivadas de las superclases, pueden utilizar los datos y métodos previamente establecidos, sin embargo, en la "herencia simple"solo pueden tener una superclase. Dentro del diagrama de árbol representan los nodos.[1]

2.2. Polimorfismo

Como ya se definió previamente, la herencia nos permite reutilizar métodos previamente establecidos en otras clases, sin embargo, si necesitamos utilizar un dato o método de la superclase, pero queremos que el comportamiento sea diferente, es donde entra en juego el polimorfismo.

El polimorfismo es la capacidad de un objeto para adquirir múltiples formas y comportamientos; en POO esto se traduce en la capacidad de utilizar métodos con el mismo nombre pero que el comportamiento sea diferente, pero no solo eso; también nos permite crear objetos y redefinirlos en tiempo de ejecución para que sean otro tipo de objeto, algo muy útil, si se necesita optimizar el manejo de memoria.[2]

3. Desarrollo

En esta práctica priorizamos los temas respecto a herencia y polimorfismo, aunque también abordamos conceptos de encapsulamiento y uso de GUI.

El objetivo de la práctica fue simple, mediante una figura seleccionada por el usuario junto con sus medidas, nosotros debíamos de devolver el área, perímetro y un dibujo representativo de la figura, por ello se decidió por usar cinco clases, una

principal, una con una clase abstracta Figura de la que heredar, tres clases para las figuras geométricas, una para el panel de dibujo y una última para el ingreso de datos.

3.1. MainApp.java

Esta es la clase principal, con ella enlazamos todas nuestras demás clases. Sus funciones son el uso de swing para dar lugar a la interfaz gráfica, permitir que el usuario seleccione una de las tres figuras (círculo, rectángulo o triangulo rectángulo), recibir parámetros numéricos pertenecientes a las medidas, calcular el área y perímetro, además de dibujar la figura seleccionada.

El método CrearGUI() está destinado a armar la interfaz gráfica, con Jframe creamos la ventana y con JPanel se crea un panel horizontal para elegir la figura de nuestra preferencia. Con ayuda de etiquetas y campos de texto el usuario puede ingresar las medidas de las figuras y se puede regresar el cálculo del área y perímetro con solo presionar un botón. También se crea un objeto del tipo PanelDibujo(), en el cual se plasmará gráficamente la figura.

El método actualizarFormulario(), tiene como finalidad cambiar los campos de entrada de acuerdo a la figura, porque el círculo pide solo el radio, a diferencia de las otras dos figuras que piden dos parámetros, esto mediante un booleano que con un setEnable desactiva o no el segundo campo. Además limpia los campos y inicia el panel de dibujo.

Método calcularYDibujar() con este se obtiene la figura elegida y se llenan los valores, si la entrada es inválida se ponen valores por defecto. Este método es importante porque aquí ya se maneja la herencia y el polimorfismo, porque la variable f de tipo Figura se convierte en un objeto de una subclase ya sea círculo, Rectangulo o TrianguloRectangulo. En esta también se llaman los métodos area() y perimetro() y se pasa el objeto f al panel de dibujo, para que posteriormente ilustre.

El método main() simplemente hace uso de la GUI y con ello de todo lo demás del programa.

3.2. Figura.java

Esta clase es abstracta por lo que no se pueden crear objetos, puede contener métodos abstractos y puede heredar, en nuestro caso en base a Figura podemos darle forma a nuestras figuras concretas como si se tratase de un molde al que solo lo adecuáramos para cumplir un propósito.

Los métodos abstractos area() y perimetro() solo son declarados puesto que en general, las tres figuras usan fórmulas distintas para sus cálculos. Por su lado el método abstracto dibujar recibe un objeto Graphics2D g y un tamaño Size, este también arrojará algo distinto de acuerdo a que figura fue seleccionada.

3.3. Circulo.java

Este método es una extensión de Figura al igual que Triangulo y TrianguloRectangulo. Primero declaramos el atributo radio y una constante pi con valor aproximado. Después tenemos el método constructor que inicializa al círculo con un valor de radio y usa un setter para hacerlo.

El método área simplemente retornará la multiplicación de PI * getRadio() * getRadio(), usando un get para acceder al radio. De manera similar el perímetro se calcula retornando PI * 2 * getRadio(), siendo las fórmulas comunes para obtener estos parámetros. El método dibujar simplemente dibujará el círculo de forma adecuada a las proporciones del panel y las indicaciones de los datos ingresados, esto haciendo escalas y redondeos. Estos tres métodos antes de declararse se valen de @Override puesto que indica que heredan de otra clase, en este caso Figura.

3.4. Rectangulo.java

Caso similar al del círculo, pero ahora el usuario necesita ingresar dos datos, la altura y el ancho, los cuales son accedidos por getters y setters en el método constructor inicializando al rectángulo, que precisamente recibe como parámetros ancho y alto.

El método área retorna la multiplicación de getAlto() * getAncho(), y el método perímetro retorna 2* getAlto() * getAncho(). Mientras que el método drawRect() vuelve a repetir el procedimiento de dibujar el rectángulo en el panel haciendo una escala, por lo que si la medida de la altura es mayor dibuja un rectángulo vertical, pero si es menor que el ancho dibuja un rectángulo horizontal. Estos métodos antes de ser declarados también usan @Override por lo que también heredan de la clase Figura.

3.5. TrianguloRectangulo.java

La última subclase de Figura, esta también recibe dos parámetros, base y altura, los cuales serán los catetos del triangulo. El método constructor recibe ambos datos e inicializa al triangulo haciendo uso de setBase y setAltura con valores por defecto.

El método area() retorna (getAltura() * getAncho())/2, mientras que el perímetro() primero calcula la hipotenusa haciendo uso del teorema de Pitágoras y a esto le suma las medidas de los catetos por lo que retorna el valor de los tres lados y con ello el perímetro. El método dibujar es similar en funcionamiento a los anteriores, usando escalación y centrando nuestra figura, este sitúa vértices en los puntos ”x’s” y ”y’s”, de una forma similar a prácticas pasadas.

3.6. NumericTextField.java

Esta clase es una subclase de JTextField, su función es solo permitir el ingreso de números y puntos flotantes como entrada. Su constructor es un super de colums y con un listener va filtrando caracteres no válidos anulándolos. Con el método safeParse convierte una cadena a un tipo de dato Double, y si no, retorna 0.0. Esta clase es simple, pues solo es un transformador, pero su función de filtro hace que el programa funcione correctamente.

3.7. PanelDibujo.java

Es un elemento importante del apartado visual que recibe la figura solicitada y la dibuja haciendo uso de las bondades del polimorfismo.

Esta clase primero guarda de forma privada la figura que se va a dibujar en Figura figura, pudiendo ser cualquiera de las tres porque son tipos de figuras, aprovechando el polimorfismo. El constructor define el tamaño y apariencia del panel de dibujo. El método setFigura recibe cualquier objeto Figura, los guarda y llama a repaint, que será el dibujante. El método paintComponent() usa @Override, llama a figura.dibujar que ejecuta el método correspondiente de acuerdo a la figura que fue seleccionada, pero si no hay figura seleccionada no dibuja nada.

4. Resultados

4.1. Pantalla inicial

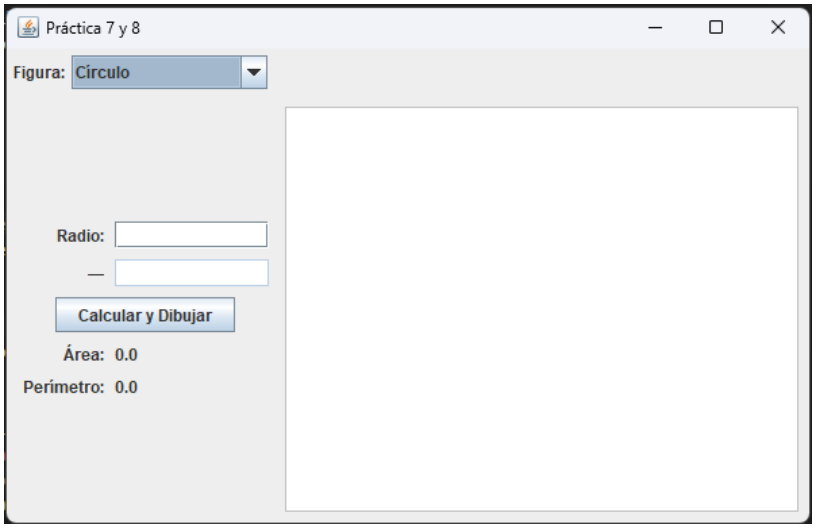

Figura 1: Programa recién ejecutado

La imagen muestra la interfaz que se muestra apenas se ejecuta el programa.

4.2. Círculo

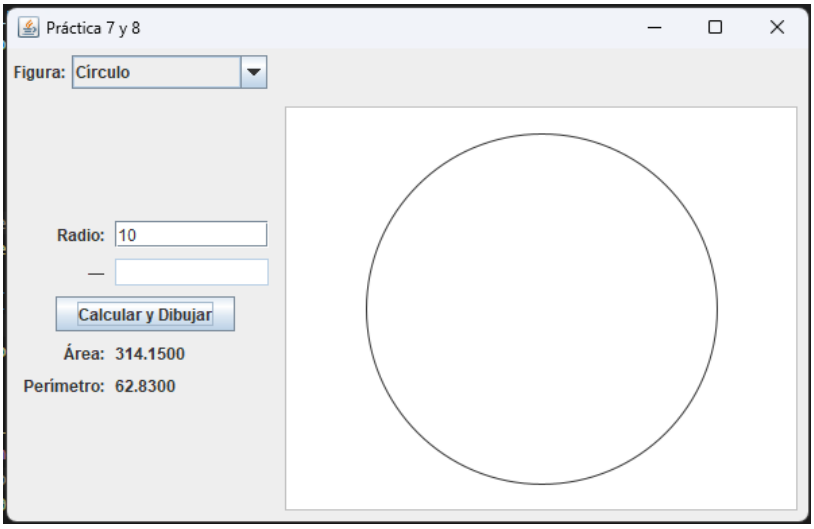

Figura 2: Círculo

Cuando se selecciona la opción de círculo y se introduce la medida de su radio, el programa dibuja la imagen y calcula e imprime el área y perímetro de la figura.

4.3. Rectángulo

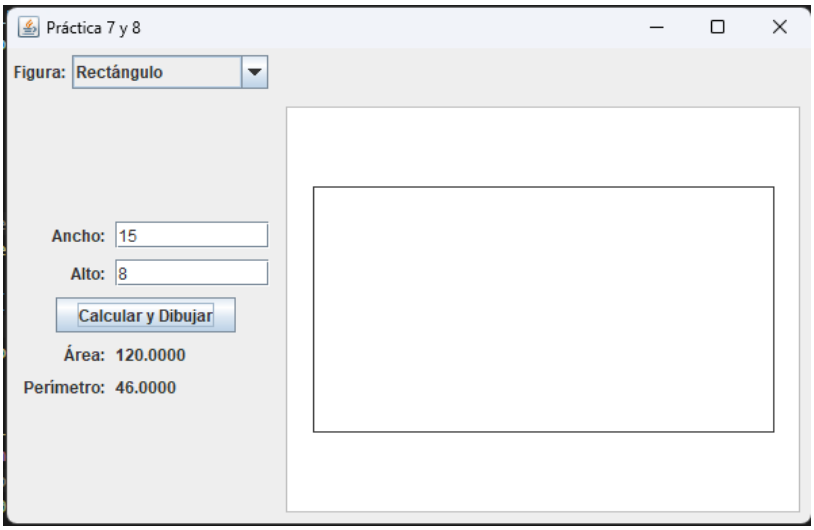

Figura 3: Rectángulo

Cuando se selecciona la opción de rectángulo y se introducen las medidas del alto y del ancho del mismo, se dibuja, se calcula el área y perímetro y se muestran en pantalla.

4.4. Triángulo rectángulo

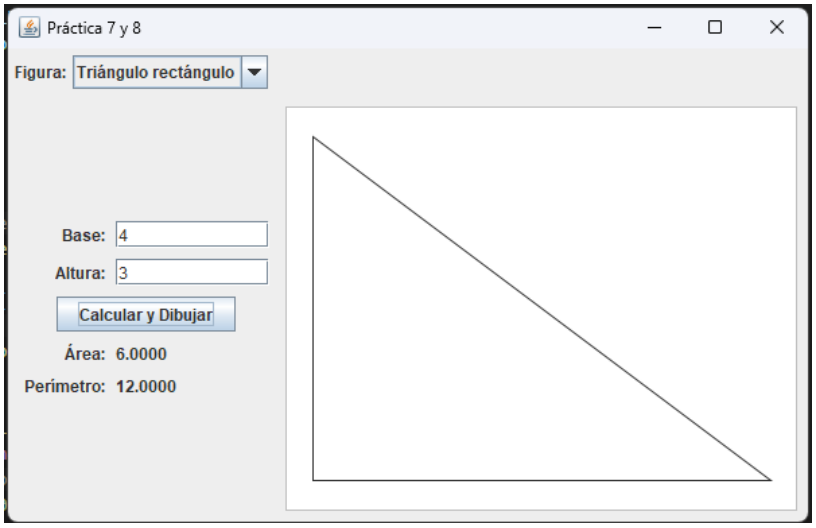

Figura 4: Trángulo rectángulo

Cuando se selecciona la opción de triángulo rectángulo y se introducen las medidas del alto y del ancho del mismo, se dibuja, se calcula la hipotenusa y con ella se calcula el área y perímetro y se muestran en pantalla.

5. Conclusiones

La práctica permitió comprobar los conceptos de herencia y polimorfismo dentro del paradigma de la programación orientada a objetos en Java. A través de la implementación de una jerarquía de clases bastante sencilla, fue posible observar cómo una clase abstracta puede fungir como "plantilla"para distintas figuras geométricas, las cuales heredaron sus atributos y métodos, adaptándolos a sus particularidades mediante el polimorfismo. El uso de la herencia facilita la reutilización de código, reduciendo redundancia y mejorando la organización del programa. Por otro lado, el polimorfismo permitió que cada clase hija redefiniera los métodos de acuerdo con su comportamiento específico.

En conjunto, la práctica permitió consolidar los conocimientos de herencia y polimorfismo, demostrando su relevancia en el desarrollo de aplicaciones eficientes a pequeña y gran escala.

6. Referencias bibliográficas

Referencias

[1] Facultad de Ingenieria UNAM. *Herencia*. Consultado el 15 de Octubre 2025. URL: http://profesores.fi-b.unam.mx/carlos/java/java_basico3_4.html.

[2] OpenWebinars. *Introducción a POO en Java: Herencia y Polimorfismo*. Consultado el 15 de Octubre 2025. URL: https://openwebinars.net/blog/introduccion-a-poo-en-java-herencia-y-polimorfismo/.