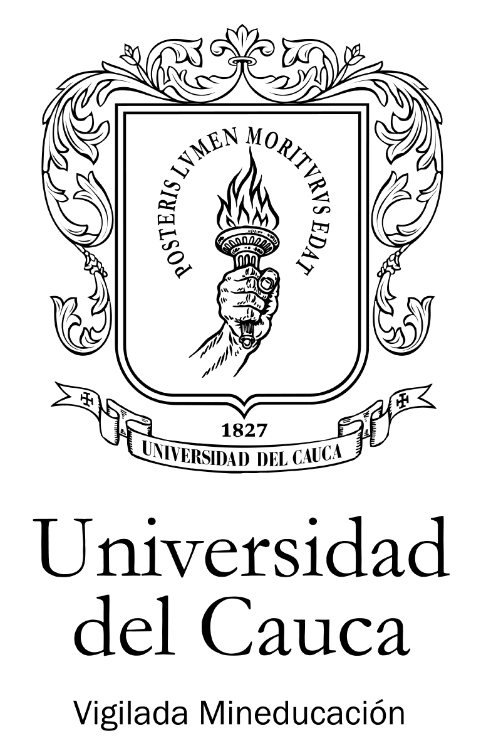
**TALLER DE POLIMORFISMO**



Laboratorio ingeniería de software II

Presentado por:

Cristian Javier Ortega

José Alejandro Rodríguez

Profesor:

W. Libardo Pantoja Y.

Universidad del Cauca

Ingeniería Electrónica y Telecomunicaciones

Ingeniería de sistemas

Popayán, Agosto de 2025

Contenido

[Introducción 4](#_Toc205741702)

[Capítulo 1 1](#_Toc205741703)

[1. Planteamiento del problema 1](#_Toc205741704)

[2. Estructura del proyecto 1](#_Toc205741705)

[3. Pruebas 2](#_Toc205741706)

[3.1. CircleTest.java 2](#_Toc205741707)

[3.2. SquareTest.java 2](#_Toc205741708)

[3.3. TriangleTest.java 3](#_Toc205741709)

[3.4. Encabezados, imágenes y tablas 3](#_Toc205741710)

[3.4.1. Imágenes 3](#_Toc205741711)

[3.4.2. Tablas 4](#_Toc205741712)

[Referencias 5](#_Toc205741713)

[Anexos 1](#_Toc205741714)

**Tabla de ilustraciones**

[Imagen 3.1.1. valores de prueba Main. 2](#_Toc205743948)

[Imagen 3.2.1. resultado CircleTest. 3](#_Toc205743949)

[Imagen 3.3.1. resultado SquareTest. 4](#_Toc205743950)

[Imagen 3.4.1. resultado TriangleTest. 4](#_Toc205743951)

**Índice de tablas**

[Tabla 1. Formato de una tabla. Elaboración propia. 2](#_Toc66445086)

Introducción

El presente documento reporta la implementación del taller de polimorfismo. El taller consiste en desarrollar una jerarquía de clases que representan figuras geométricas (círculo, cuadrado y triángulo) aplicando los conceptos de herencia, abstracción y polimorfismo.

La implementación se realizó en Java utilizando NetBeans como entorno de desarrollo integrado, incorporando pruebas unitarias para verificar la correcta funcionalidad del código y utilizando Git para el control de versiones del proyecto.

Capítulo 1

# Planteamiento del problema

* Crear una jerarquía de clases donde una clase abstracta Figura define métodos abstractos (calcularArea(), calcularPerimetro()).
* Implementar clases concretas (Círculo, Cuadrado, Triángulo) que sobrescriben estos métodos según sus propiedades.
* Demostrar el polimorfismo almacenando objetos de distintas figuras en una lista de tipo Figura y ejecutando sus métodos.
* Validar el código con pruebas unitarias (JUnit) y gestionar el proyecto en GitHub usando comandos de Git.

# Estructura del proyecto

La estructura del proyecto implementado en Java:

Estructura de Paquetes:

* co.unicauca.figures.figures: Contiene las clases del modelo de figuras
* co.unicauca.figures.figures.domain: Clases de dominio (Figure, Circle, Square, Triangle)

Clases de Prueba:

* co.unicauca.figures.figures.domain: Contiene las pruebas unitarias

CircleTest.java

SquareTest.java

TriangleTest.java

# Pruebas

## Main.java

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen 3.1.. valores de prueba Main.

Tabla

El contenido generado por IA puede ser incorrecto.

Imagen 3.1.2. salida ejecución del Main.

## CircleTest.java

Test of calculatePerimeter method, of class Circle.

* Low value (radius = 3)
* Medium value (radius = 10)
* High value (radius = 1000)

Test of calculateArea method, of class Circle.

* Low value (radius = 3)
* Medium value (radius = 333)
* High value (radius = 12000)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen 3.2.. resultado CircleTest.

## SquareTest.java

Test of calculatePerimeter method, of class Square.

* Low value (side = 2)
* Medium value (side = 50)
* High value (side = 1000)

Test of calculateArea method, of class Square.

* Low value (side = 4)
* Medium value (side = 70)
* High value (side = 1200)

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen 3.3.. resultado SquareTest.

## TriangleTest.java

Test of calculatePerimeter method, of class Triangle.

\* Low value (sideA = 2, sideB = 2, sideC = 2)

\* Medium value (sideA = 30, sideB = 40, sideC = 50)

\* High value (sideA = 2501, sideB = 2501, sideC = 4998)

Test of calculateArea method, of class Triangle.

\* Low value (sideA = 2, sideB = 2, sideC = 2)

\* Medium value (sideA = 30, sideB = 40, sideC = 50)

\* High value (sideA = 2501, sideB = 2501, sideC = 4998)

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen 3.4.. resultado TriangleTest.

Referencias

* Hurtado, J. A., & Pantoja, W. L. (2025). *Práctica de Laboratorio: Polimorfismo*. Universidad del Cauca.
* Jarroba.com. "Polimorfismo en Java Parte I con ejemplos". Recuperado de: <https://jarroba.com/polimorfismo-en-java-parte-i-con-ejemplos/>
* Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Anexos

En esta sección se deben colocar los anexos al trabajo principal.