Programación II: Herencia y Polimorfismo

Pilares fundamentales de la Programación Orientada a Objetos que permiten construir sistemas flexibles y escalables.

Agenda del Módulo

Herencia: Fundamentos y Aplicación

Comprender cómo la herencia permite reutilizar código y establecer relaciones "es un" entre clases.

Modificadores de Acceso y Constructores

Gestionar la visibilidad de miembros y la inicialización de objetos en jerarquías de herencia.

Conversión de Tipos y Abstracción

Dominar el upcasting, downcasting y el diseño con clases y métodos abstractos para flexibilidad.

Polimorfismo: Tipos y Aplicaciones

Implementar la sobrescritura de métodos y aplicar diferentes formas de polimorfismo para diseños robustos.

Comprendiendo la Herencia en POO

Principio "Es un" (Is-A)

La herencia establece una relación jerárquica donde una subclase **(hija)** extiende las funcionalidades de una superclase **(padre)**. Por ejemplo, un "Auto **es un** Vehículo" o un "Perro **es un** Animal". Esto promueve la reutilización de código y la creación de una taxonomía lógica.

A través de la herencia, las subclases pueden acceder a los atributos y métodos públicos y protegidos de sus superclases, evitando la duplicación y facilitando el mantenimiento.

Modificadores de Acceso y Constructores

La gestión de la visibilidad es crucial para el encapsulamiento y la seguridad de las clases heredadas.

Control de Visibilidad

- **private:** Solo accesible dentro de la propia clase.
- **protected:** Accesible dentro de la clase, sus subclases y clases del mismo paquete.
- **public:** Accesible desde cualquier lugar.

Uso de Constructores

- **super(...):** Se utiliza para invocar el constructor de la superclase, asegurando la correcta inicialización de los componentes heredados antes que los propios de la subclase.
- Es vital para una construcción adecuada de objetos en una jerarquía.

Conversión de Tipos y Abstracción

Upcasting y Downcasting

- **Upcasting:** Convertir una instancia de subclase a su tipo de superclase. Es seguro y automático, útil para trabajar con colecciones heterogéneas.
- **Downcasting:** Convertir una instancia de superclase a su tipo de subclase. Requiere un **cast** explícito y debe usarse con precaución, utilizando **instanceof** para verificar el tipo en tiempo de ejecución y evitar errores.

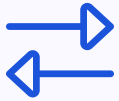
Clases y Métodos Abstractos

Las clases abstractas definen plantillas base con métodos que pueden ser implementados por sus subclases. Un método abstracto no tiene implementación en la superclase, forzando a las subclases a proporcionar su propia versión. Esto impone una estructura y garantiza que las funcionalidades esenciales sean implementadas, mientras que la herencia se encarga de los detalles.

Polimorfismo: La Flexibilidad del Código

Sobrescritura de Métodos

Permite a una subclase proporcionar una implementación específica para un método que ya está definido en su superclase, usando **@Override**.



Tipos de Polimorfismo

- **Por Sobrecarga:** Múltiples métodos con el mismo nombre pero diferentes parámetros.
- **Por Coerción:** Conversión implícita de tipos.
- **Paramétrico:** Métodos genéricos que operan con diferentes tipos de datos.
- **Por Herencia:** Comportamiento diferente de objetos de distintas clases pero misma interfaz común.

Optimización y Diseño

El polimorfismo facilita la creación de software modular y fácil de extender, permitiendo que el mismo código interactúe con objetos de diferentes tipos de manera uniforme.