Justificación de Principios POO - SmartHome

Introducción

La Programación Orientada a Objetos (POO) es un paradigma de desarrollo de software que organiza el código en torno a objetos, los cuales representan entidades del mundo real. Este enfoque permite una mejor organización, reutilización y mantenimiento del código, facilitando el desarrollo de sistemas complejos y escalables."

Descripción de las clases involucradas

- Usuario: Representa a la persona que interactúa con el sistema, gestionando dispositivos y automatizaciones.
- DispositivoHogar: Modela los dispositivos inteligentes del hogar que pueden ser controlados por el sistema.
- Automatizacion: Define reglas y condiciones para automatizar el comportamiento de los dispositivos según eventos o horarios.

1. Abstracción

Cada clase representa una entidad del mundo real con atributos y comportamientos relevantes:
- Usuario abstrae a una persona que interactúa con el sistema.
- DispositivoHogar representa un aparato del hogar con estado y funcionalidad.
- Automatizacion modela una rutina que gestiona dispositivos.

Esto permite trabajar con conceptos complejos de forma simplificada y centrada en lo esencial.

2. Encapsulamiento

Los atributos están definidos como privados (_atributo) y se accede a ellos mediante métodos públicos:
- Ejemplo: DispositivoHogar tiene _estado, pero se modifica con encender() y apagar().
- Usuario expone mostrar_info() sin revelar directamente sus atributos.

Esto protege el estado interno de los objetos y evita modificaciones indebidas.

3. Agregación

Relación entre Usuario y DispositivoHogar:
- Usuario "1" -- "0..*" DispositivoHogar : controla
- Usuario "1" -- "0..*" DispositivoHogar : posee
  - El usuario no es dueño exclusivo de los dispositivos; estos pueden existir independientemente.
  - Se modela como agregación porque los dispositivos pueden ser compartidos o gestionados por otros usuarios.

La agregación indica una relación débil: el ciclo de vida del objeto agregado no depende del contenedor.

4. Composición

Relación entre Automatizacion y DispositivoHogar:
- Automatizacion "1" -- "0..*" DispositivoHogar : gestiona
  - La automatización depende directamente de los dispositivos que gestiona.
  - Si se elimina la automatización, su lógica y configuración pierden sentido.

Aunque no se modela como composición estricta (*--), la dependencia funcional sugiere una relación fuerte.

5. Modularidad

Cada clase tiene una responsabilidad clara:

- Usuario: gestión de identidad y control.
- DispositivoHogar: estado y operación.
- Automatizacion: lógica de apagado inteligente.

Esto facilita el mantenimiento, la escalabilidad y la reutilización del código.

Justificación de Relaciones

Relación	Tipo	Justificación
Usuario controla DispositivoHogar	Agregación	El usuario interactúa con dispositivos sin poseerlos exclusivamente.
Usuario posee DispositivoHogar	Agregación	Puede tener dispositivos asignados, pero no los crea ni destruye.
Automatizacion gestiona DispositivoHogar	Composición funcional	La automatización depende de los dispositivos para ejecutar su lógica.

Diagrama de clases - (SmartHome) POO

```
@startuml
title Diagrama UML con Principios de POO

' Encapsulamiento: atributos privados, acceso controlado por métodos
class Usuario {
  - _id_usuario: int
  - _nombre: str
  - _clave: str
  - _rol: str
  - _tiempo_de_conexion: str
  - _edad: int
  - _mail: str
  - _telefono: str
  - _registro_actividad: str
  - _id_hogar: int
  - dispositivos_control: List<DispositivoHogar>
  - dispositivos_hogar: List<DispositivoHogar>
  + get_id_usuario(): int
  + verificar_clave(clave: str): bool
  + cambiar_rol(nuevo_rol: str): void
  + actualizar_datos(nombre: str, mail: str, telefono: str): void
  + mostrar_info(): str
  + cambiar_clave(nueva_clave: str): void
  + registrar_actividad(actividad: str): void
  + rol: str
}

' Abstracción: representa un dispositivo con estado y funcionalidad
class DispositivoHogar {
  - _id: str
  - _nombre: str
  - _estado: bool
  - _es_esencial: bool
  + encender(): void
  + apagar(): void
  + estado_actual(): bool
  + es_esencial(): bool
  + nombre: str
  + id: str
```

```
}

' Modularidad: Automatización como clase independiente con lógica propia
class Automatizacion {
  - _nombre: str
  - _dispositivos: List<DispositivoHogar>
  + activar(): int
}

' Asociación: relaciones entre clases
Usuario "1" -- "0..*" DispositivoHogar : controla
Usuario "1" -- "0..*" DispositivoHogar : posee
Automatizacion "1" -- "0..*" DispositivoHogar : gestiona

@enduml
```