Justificación de Principio POO + Diagrama UML

Datos Generales de la Evidencia

Proyecto	**SmartHome**
Evidencia	N°6 - Módulo Programador
Institución	ISPC - Córdoba
Equipo	LeonesDev
Fecha	13/10/2025
Repositorio	`Leones-Dev-Team/proyecto_smarthome_poo`

Objetivos del Documento

Este documento presenta la justificación técnica del diseño orientado a objetos implementado en la versión EV6 del proyecto SmartHome, incluyendo:

1. Aplicación de principios fundamentales de POO.
2. Diagrama UML de clases actualizado.
3. Justificación de decisiones de diseño aplicadas durante la corrección de la Evidencia 5.
4. Relación con la base de datos y coherencia con las estructuras persistentes.

Principios de POO Aplicados

1. Encapsulamiento

- Todos los atributos de dominio se definen como **privados o protegidos** (`_atributo`), controlando el acceso mediante properties (`@property` , `@setter`).
- Las **validaciones se realizan internamente** antes de modificar el estado del objeto.
- **Ejemplos del proyecto**:
  - `Perfil` valida nombre, mail y teléfono antes de asignarlos.
  - `DispositivoHogar` controla el estado con métodos `encender()` , `apagar()` y `toggle()` .

    - `Hogar` y `Usuario` verifican datos obligatorios antes de ser persistidos por su DAO.
- **Beneficio**: garantiza la integridad de los datos y evita estados inválidos en memoria o en la base de datos.

---

2. Principio de Responsabilidad Única (SRP)

Cada clase tiene una sola responsabilidad, como se muestra en la siguiente tabla:

Clase	Rol principal
Usuario	Manejo de credenciales, rol (admin/estándar) y asociación con perfil.
Perfil	Gestión de datos personales y registro de actividad.
Hogar	Identificación de vivienda y relación con dispositivos.
DispositivoHogar	Estado, tipo, consumo y métodos de control.
Automatizacion	Reglas de apagado/encendido automático.
DispositivoControl	Estadísticas de dispositivos activos/apagados.
DAO (UsuarioDAO, etc.)	Persistencia y consultas SQL.

- **Beneficio**: separación clara de responsabilidades y facilidad para depurar o extender el sistema.

---

3. Abstracción

El sistema representa conceptos del dominio de forma simplificada:

- `Usuario` **abstrae** al actor principal del sistema.
- `Automatizacion` **abstrae** reglas complejas de eficiencia energética tras una interfaz simple.
- `DispositivoDAO` y otros DAO **abstraen** el acceso a la base de datos, evitando SQL repetido.
- `obtener_conexion.py` **abstrae** el motor de base de datos, permitiendo cambiar de DBMS sin alterar la lógica de dominio.
- **Beneficio**: simplifica el mantenimiento y reduce el acoplamiento entre capas.

---

4. Agregación y Composición

- **Agregación**:
    - `Usuario` posee varios `DispositivoHogar`, pero estos pueden existir de forma independiente.
    - `Automatizacion` utiliza dispositivos ya existentes (no los crea ni destruye).
- **Composición**:

    - `Usuario` contiene un `Perfil` (si se elimina el usuario, el perfil deja de tener sentido).
- **Beneficio**: modelado coherente del ciclo de vida de los objetos.

---

5. Herencia e Interfaces

- Las interfaces `i_usuario_dao.py`, `i_perfil_dao.py`, `i_dispositivo_dao.py` y `i_automatizacion_dao.py` definen **contratos obligatorios** para los DAO.
- Cada DAO concreto implementa esas interfaces, garantizando **homogeneidad en métodos** (`insertar`, `obtener`, `modificar`, `eliminar`).
- **Beneficio**: refuerza el principio de sustitución de Liskov y permite intercambiar implementaciones sin romper el flujo del sistema.

---

6. Modularidad

Estructura del Proyecto SmartHome

```
proyecto_smarthome_poo/
├── main.py
├── dominio/
│   ├── usuario.py
│   ├── perfil.py
│   ├── hogar.py
│   ├── dispositivo_hogar.py
│   ├── dispositivo_control.py
│   └── automatizacion.py
├── dao/
│   ├── usuario_dao.py
│   ├── perfil_dao.py
│   ├── dispositivo_dao.py
│   ├── automatizacion_dao.py
│   └── interfaces/
│       ├── i_usuario_dao.py
│       ├── i_perfil_dao.py
│       ├── i_dispositivo_dao.py
│       └── i_automatizacion_dao.py
├── connection/
│   └── obtener_conexion.py
├── BD-Evidencia-5/
│   ├── init.sql
│   ├── queries.sql
│   └── README.md
```

```
├── BD-Evidencia-6/
│   ├── init.sql
│   ├── queries.sql
│   └── README.md
├── DC-Evidencia-5/
│   └── Justificacion_POO_UML_Diagrama_Clases.pdf
└── DC-Evidencia-6/
    ├── init.sql
    ├── queries.sql
    ├── README.md
    └── Justificacion_POO_UML_Diagrama_Clases.pdf
```

- **Beneficio**: permite mantener y testear cada módulo de forma independiente.

Relación con la Base de Datos

1. Estructura y coherencia

- El script `init.sql` define las tablas: `usuarios`, `perfiles`, `hogares`, `dispositivos`, `automatizaciones`.
- Las claves primarias son `INT AUTO_INCREMENT`.
- Las relaciones mediante `FOREIGN KEY` son coherentes con las asociaciones de clases.

2. Consistencia de tipos

Atributo	Tipo en BD	Tipo en Clase
`id_usuario`, `id_hogar`	INT	`int`
`estado_dispositivo`	VARCHAR(10)	`str` ('encendido' / 'apagado')
`es_esencial`	BOOLEAN	`bool`
`rol`	VARCHAR(15)	`str`
`consumo_energetico`	FLOAT	`float`

3. Conexión y DAO

- El archivo `obtener_conexion.py` centraliza la conexión con el DBMS (`mysql.connector` o SQLite).
- Cada DAO la reutiliza, evitando duplicación y favoreciendo la escalabilidad.
- **Beneficio**: asegura correspondencia 1:1 entre las entidades del dominio y las tablas persistentes.

Decisiones Técnicas en EV6

1. **Implementación del patrón DAO**: Separación total entre lógica de negocio y acceso a datos, lo que permite futuras migraciones de DBMS (p. ej. MySQL → PostgreSQL).
2. **Normalización de roles**: El rol `estandar` se mantuvo en minúsculas para compatibilidad entre código y datos.
3. **Validaciones reforzadas**: `Perfil` y `Hogar` incluyen validaciones de campos vacíos y formatos de email.
4. **Persistencia delegada**: Las clases `UsuarioDAO`, `DispositivoDAO`, etc., manejan toda la interacción SQL. Las clases de dominio solo contienen la lógica de negocio.
5. **Compatibilidad con TDD**: Se conservan las pruebas unitarias de la EV5 (todas en verde). Los "archivos-dummy" se mantienen para preservar cobertura sobre módulos críticos.

Conclusiones

- El diseño EV6 cumple plenamente los principios de **POO y SRP**, garantizando modularidad, mantenibilidad y consistencia con la base de datos.
- La introducción del **patrón DAO** mejora la arquitectura, separando claramente las capas de dominio, persistencia y presentación.
- La base de datos se encuentra **normalizada y alineada** con las clases de Python.
- El sistema está preparado para **escalar**, integrar nuevas automatizaciones y conectar microservicios en versiones futuras.
- Adicionalmente, el diseño modular y desacoplado sienta las bases para cumplir con los pilares del **AWS Well-Architected Framework**, promoviendo la seguridad, fiabilidad, eficiencia en el rendimiento y sostenibilidad operativa a futuro.

Diagrama UML de Clases

> *Para visualizar el diagrama:*
>
> 1. *Copie el siguiente bloque de código PlantUML.*
> 2. *Visite **planttext.com**.*
> 3. *Pegue el código en el editor, haga click en Save & Refresh y se generará la imagen del diagrama lista para descargar en varios formatos.*

```
@startuml SmartHome_EV6

title SmartHome EV6 - Diagrama UML técnico

```

```
skinparam monochrome false
skinparam classAttributeIconSize 0
skinparam shadowing false

' ========================
' Clases de dominio (POO)
' ========================

class Usuario {
  - _id_usuario: int
  - _clave: str
  - _rol: str
  - _perfil: Perfil
  - _id_hogar: int
  - _edad: int
  + id_usuario(): int
  + rol(): str
  + rol(nuevo_rol: str): void
  + perfil(): Perfil
  + id_hogar(): int
  + edad(): int
  + verificar_clave(clave: str): bool
  + cambiar_clave(nueva_clave: str): void
  + mostrar_info(): str
  + __repr__(): str
}

class Perfil {
  - _id_perfil: int
  - _nombre: str
  - _mail: str
  - _telefono: str
  - _registro_actividad: List<str>
  + id_perfil(): int
  + nombre(): str
  + nombre(nuevo_nombre: str): void
  + mail(): str
  + mail(nuevo_mail: str): void
  + telefono(): str
  + telefono(nuevo_telefono: str): void
  + registrar_actividad(actividad: str): None
  + registro_actividad(): Tuple<str,...>
  + limpiar_registro(): void
```

```
  + to_dict(): dict
  + __repr__(): str
  - _validar_nombre(nombre: str): void
  - _validar_mail(mail: str): void
}

class Hogar {
  - _id_hogar: int
  - _ubicacion: str
  - _tipo_de_vivienda: str
  + id_hogar(): int
  + ubicacion(): str
  + ubicacion(nueva_ubicacion: str): void
  + tipo_de_vivienda(): str
  + tipo_de_vivienda(nuevo_tipo: str): void
  + listar_dispositivos_asociados(dispositivo_dao: DispositivoDAO):
List<DispositivoHogar>
  + __str__(): str
  + __repr__(): str
}

class DispositivoHogar {
  - _id_dispositivo: int
  - _id_hogar: int
  - _nombre: str
  - _tipo: str
  - _marca: str
  - _estado_dispositivo: str
  - _consumo_energetico: float
  - _es_esencial: bool
  + id_dispositivo(): int
  + id_hogar(): int
  + nombre(): str
  + tipo(): str
  + marca(): str
  + estado_dispositivo(): str
  + consumo_energetico(): float
  + es_esencial(): bool
  + encender(): void
  + apagar(): void
  + toggle(): void
  + __repr__(): str
}
```

```
90.
91. class DispositivoControl {
92.   - _id_dispositivo_control: int
93.   - _id_usuario: int
94.   - _hora_de_conexion: str
95.   - _dispositivos_activos: int
96.   - _dispositivos_apagados: int
97.   - _dispositivos_en_ahorro: int
98.   + id_dispositivo_control(): int
99.   + id_usuario(): int
100.  + hora_de_conexion(): str
101.  + dispositivos_activos(): int
102.  + dispositivos_apagados(): int
103.  + dispositivos_en_ahorro(): int
104.  + calcular_total_dispositivos(): int
105.  + __repr__(): str
106. }
107.
108. class Automatizacion {
109.   - _id_automatizacion: int
110.   - _nombre: str
111.   - _dispositivos: List<DispositivoHogar>
112.   + id_automatizacion(): int
113.   + nombre(): str
114.   + dispositivos(): tuple<DispositivoHogar,...>
115.   + agregar_dispositivo(dispositivo: DispositivoHogar): void
116.   + quitar_dispositivo(dispositivo: DispositivoHogar): void
117.   + activar(): int
118.   + guardar(): void
119.   + cargar(id_automatizacion: int): Automatizacion
120. }
121.
122. ' ========================
123. ' Interfaces DAO
124. ' ========================
125.
126. interface IUsuarioDAO {
127.   + crear(usuario: Usuario): bool
128.   + leer(id_usuario: int): Usuario
129.   + actualizar(usuario: Usuario): bool
130.   + eliminar(id_usuario: int): bool
131.   + obtener_todos(): List<Usuario>
132. }
```

```
133.
134. interface IPerfilDAO {
135.   + crear(perfil: Perfil): int
136.   + leer(id_perfil: int): Perfil
137.   + actualizar(perfil: Perfil, id_perfil: int): bool
138.   + eliminar(id_perfil: int): bool
139.   + obtener_todos(): List<Perfil>
140. }
141.
142. interface IDispositivoDAO {
143.   + crear(dispositivo: DispositivoHogar): bool
144.   + leer(id_dispositivo: int): DispositivoHogar
145.   + actualizar(dispositivo: DispositivoHogar): bool
146.   + eliminar(id_dispositivo: int): bool
147.   + obtener_todos(): List<DispositivoHogar>
148.   + listar_por_hogar(id_hogar: int): List<DispositivoHogar>
149. }
150.
151. interface IAutomatizacionDAO {
152.   + crear(automatizacion: Automatizacion): int
153.   + leer(id_automatizacion: int): Automatizacion
154.   + actualizar(automatizacion: Automatizacion): bool
155.   + eliminar(id_automatizacion: int): bool
156.   + obtener_todos(): List<Automatizacion>
157.   + agregar_dispositivo(id_automatizacion: int, id_dispositivo: int): bool
158.   + quitar_dispositivo(id_automatizacion: int, id_dispositivo: int): bool
159.   + obtener_dispositivos(id_automatizacion: int): List<DispositivoHogar>
160. }
161.
162. ' ========================
163. ' Implementaciones DAO
164. ' ========================
165.
166. class UsuarioDAO implements IUsuarioDAO
167. class PerfilDAO implements IPerfilDAO
168. class DispositivoDAO implements IDispositivoDAO
169. class AutomatizacionDAO implements IAutomatizacionDAO
170.
171. ' ========================
172. ' Relaciones (dominio)
173. ' ========================
174.
175. Usuario "1" *-- "1" Perfil
```

```
Usuario "1" --> "1" Hogar
Hogar "1" o-- "0..*" DispositivoHogar
Automatizacion "1" o-- "0..*" DispositivoHogar
DispositivoControl "1" --> "1" Usuario

' ========================
' Relaciones (DAO y Lógica)
' ========================

Hogar ..> DispositivoDAO : usa
Automatizacion ..> AutomatizacionDAO : usa para persistencia
Automatizacion ..> DispositivoDAO : usa para actualizar

@enduml
`
```