Justificación de Principios POO + Diagrama UML

Datos Generales de la Evidencia

Proyecto	SmartHome
Evidencia	N°6 - Módulo Programador
Institución	ISPC - Córdoba
Equipo	LeonesDev
Fecha	19/10/2025
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 final 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 y 6.
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 (`__atributo`), controlando el acceso mediante `@property` y `@setter` .
- Las validaciones se realizan internamente antes de modificar el estado del objeto.
- Ejemplos:

- `Perfil` valida nombre, mail y teléfono antes de asignarlos.
- `DispositivoHogar` controla el estado con métodos `encender()`, `apagar()` y `toggle()`.
- `Usuario` y `Hogar` 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)

Clase	Rol principal
Usuario	Manejo de credenciales, rol (admin/estándar) y asociación con perfil y hogar.
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.
DispositivoControl	Estadísticas de dispositivos activos/apagados.
Automatizacion	Reglas de apagado/encendido automático.
DAO (UsuarioDAO, etc.)	Persistencia y consultas SQL.

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

---

3. Abstracción

- `Usuario` abstrae al actor principal del sistema.
- `Automatizacion` abstrae reglas complejas de eficiencia energética tras una interfaz simple.
- Los DAOs abstraen el acceso a la base de datos, evitando SQL repetido.
- `DatabaseConnection` 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**:
  - `Hogar` posee varios `DispositivoHogar`, que 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 `IUsuarioDAO`, `IPerfilDAO`, `IHogarDAO`, `IDispositivoDAO` e `IAutomatizacionDAO` definen contratos obligatorios para los DAOs.
- Cada DAO concreto implementa esas interfaces, garantizando homogeneidad en métodos CRUD.

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
│   ├── hogar_dao.py
│   ├── dispositivo_dao.py
│   ├── automatizacion_dao.py
│   └── interfaces/
│       ├── i_usuario_dao.py
│       ├── i_perfil_dao.py
│       ├── i_hogar_dao.py
│       ├── i_dispositivo_dao.py
│       └── i_automatizacion_dao.py
```

```
│
├── connection/
│   └── obtener_conexion.py
│
├── tests/
│   ├── test_usuario.py
│   └── test_perfil.py
│
├── BD-Evidencia-5/
│   ├── init.sql
│   ├── queries.sql
│   └── README.md
│
├── BD-Evidencia-6/
│   ├── init.sql
│   ├── queries.sql
│   └── README.md
│
├── DC-Evidencia-5/
│   └── EV5_Justificacion_POO_UML_Diagrama_Clases.pdf
│
└── DC-Evidencia-6/
    └── EV6_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_hogar`, `dispositivos_control`, `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(20)	str (“encendido” / “apagado”)

Atributo	Tipo en BD	Tipo en Clase
es_esencial	BOOLEAN	bool
rol	VARCHAR(20)	str
consumo_energetico	FLOAT	float

3. Conexión y DAO

- El archivo `obtener_conexion.py` centraliza la conexión con MySQL.
- 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 la Versión Final

1. Implementación del patrón DAO: separación total entre lógica de negocio y acceso a datos.
2. Normalización de roles: `"estandar"` se mantiene 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 de dominio no contienen SQL, toda la interacción se maneja en los DAOs.
5. Compatibilidad con TDD: se mantienen pruebas unitarias de EV5 y se amplían para cubrir nuevas validaciones.
6. `DatabaseConnection`: conexiones siempre nuevas, `autocommit=False`, configuración desde `db_connection.env`.
7. `UsuarioDAO`: crea primero el `Perfil` y luego el `Usuario`, garantizando integridad referencial.
8. `PerfilDAO`: usa `cargar_registro` para reconstruir historial de actividad.

Conclusiones

- El diseño final 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.

Diagrama UML de clases

Para visualizar el diagrama, copie el código `.puml` y péguelo en PlantText UML Editor. Luego haga clic en **"Save and Refresh"** para generar automáticamente el diagrama. Debajo del diagrama generado tendrá diferentes opciones de descarga, en diferentes formatos (.pdf, .png, etc.).

```
@startuml SmartHome_Final

title SmartHome - Diagrama UML técnico final

skinparam monochrome false
skinparam classAttributeIconSize 0
skinparam shadowing false

' ==== Clases de dominio ====
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
  + clave(): str
  + 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
  + id_perfil(nuevo_id: int): void
```

```
  + nombre(): str
  + nombre(nuevo_nombre: str): void
  + mail(): str
  + mail(nuevo_mail: str): void
  + telefono(): str
  + telefono(nuevo_tel: str): void
  + registrar_actividad(act: str): None
  + registro_actividad(): Tuple<str,...>
  + limpiar_registro(): void
  + cargar_registro(registro: str): void
  + to_dict(): dict
  + __repr__(): str
}

class Hogar {
  - __id_hogar: int
  - __ubicacion: str
  - __tipo_de_vivienda: str
  + id_hogar(): int
  + ubicacion(): str
  + ubicacion(nueva: str): void
  + tipo_de_vivienda(): str
  + tipo_de_vivienda(nuevo: str): void
  + listar_dispositivos_asociados(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
}

class DispositivoControl {
  - _id_dispositivo_control: int
  - _id_usuario: int
  - _hora_de_conexion: str
  - _dispositivos_activos: int
  - _dispositivos_apagados: int
  - _dispositivos_en_ahorro: int
  + id_dispositivo_control(): int
  + id_usuario(): int
  + hora_de_conexion(): str
  + dispositivos_activos(): int
  + dispositivos_apagados(): int
  + dispositivos_en_ahorro(): int
  + calcular_total_dispositivos(): int
  + __repr__(): str
}

class Automatizacion {
  - _id_automatizacion: int
  - _nombre: str
  - _dispositivos: List<DispositivoHogar>
  + id_automatizacion(): int
  + nombre(): str
  + dispositivos(): tuple<DispositivoHogar,...>
  + agregar_dispositivo(d: DispositivoHogar): void
  + quitar_dispositivo(d: DispositivoHogar): void
  + activar(): int
  + guardar(): void
  + cargar(id: int): Automatizacion
}

' ==== Interfaces DAO ====
interface IUsuarioDAO
interface IPerfilDAO
interface IHogarDAO
interface IDispositivoDAO
interface IAutomatizacionDAO

' ==== Implementaciones DAO ====
class UsuarioDAO implements IUsuarioDAO
class PerfilDAO implements IPerfilDAO
class HogarDAO implements IHogarDAO
class DispositivoDAO implements IDispositivoDAO
```

```
class AutomatizacionDAO implements IAutomatizacionDAO

' ==== Relaciones de dominio ====
Usuario "1" *-- "1" Perfil
Usuario "1" --> "1" Hogar
Hogar "1" o-- "0..*" DispositivoHogar
Automatizacion "1" o-- "0..*" DispositivoHogar
DispositivoControl "1" --> "1" Usuario

@enduml
```

SmartHome - Diagrama UML técnico final