Justificación

Clases y responsabilidades

- **Usuario:** Representa a cada persona registrada en el sistema. Sus atributos (_nombre, _contraseña, _es_admin) permiten diferenciar administradores de usuarios estándar. Sus métodos (iniciar_sesion(), registrarse(), ver_datos()) encapsulan la lógica de autenticación y gestión de datos personales.

- **Dispositivo:** Modela los aparatos del hogar inteligente. Incluye atributos (_nombre, _estado, _esencial) y métodos (encender(), apagar(), modificar_estado()), que representan acciones reales del dominio.

- **Automatizacion:** Representa reglas o acciones automáticas aplicadas sobre dispositivos. Contiene atributos (_nombre, _tipo, _activa) y método principal ejecutar().

- **SmartHome:** Clase central que administra colecciones de usuarios, dispositivos y automatizaciones, implementando el patrón de capas de presentación. Sus métodos (agregar_usuario(), agregar_dispositivo(), ejecutar_automatizacion()) permiten gestionar y coordinar las demás entidades.

RELACIONES Y CARDINALIDADES

- **Agregación** (SmartHome —○→ Usuario/Dispositivo):
  SmartHome tiene usuarios y dispositivos (cardinalidad 1..*), pero pueden existir independientemente.
- **Composición** (SmartHome —◆→ Automatizacion):
  SmartHome está compuesto por automatizaciones (cardinalidad 1..*). Si se destruye SmartHome, se destruyen sus automatizaciones.
- **Asociación** (Automatizacion —— Dispositivo):
  Una automatización puede involucrar varios dispositivos y un dispositivo puede pertenecer a varias automatizaciones (relación N:M). En la base de datos relacional esto se implementa mediante la tabla intermedia automatizacion_dispositivo.

Principios de POO aplicados

- **Abstracción:** Cada clase representa un concepto clave del sistema domótico, ocultando detalles innecesarios. Por ejemplo, Usuario oculta detalles de autenticación al implementar iniciar_sesion().
- **Encapsulamiento:** Los datos están protegidos (atributos privados _) y solo se manipulan mediante métodos públicos (+), asegurando un mayor control y coherencia. Por ejemplo, el estado de un Dispositivo no se puede acceder directamente mediante su atributo privado, sino con métodos como escender() y apagar().

- **Arquitectura de Capas**: El sistema implementa separación de responsabilidades con capa de dominio (clases), capa de acceso a datos (DAO) y capa de presentación (main.py)
- **Herencia** (a futuro): se podría especializar Usuario en UsuarioAdmin o UsuarioEstandar, reutilizando atributos y comportamientos. Así como diferentes tipos de dispositivos podrían heredar características generales de Dispositivo.

Otros fundamentos teóricos

- **Patrón DAO**: Separación entre lógica de dominio y acceso a datos, facilitando mantenimiento y testing.

- **Arquitectura de 3 capas**: Presentación → Lógica de Negocio → Acceso a Datos, siguiendo principios de separación de responsabilidades.

- **Modelado del dominio**: Las clases reflejan entidades del mundo real (Usuario, Dispositivo, Automatizacion).