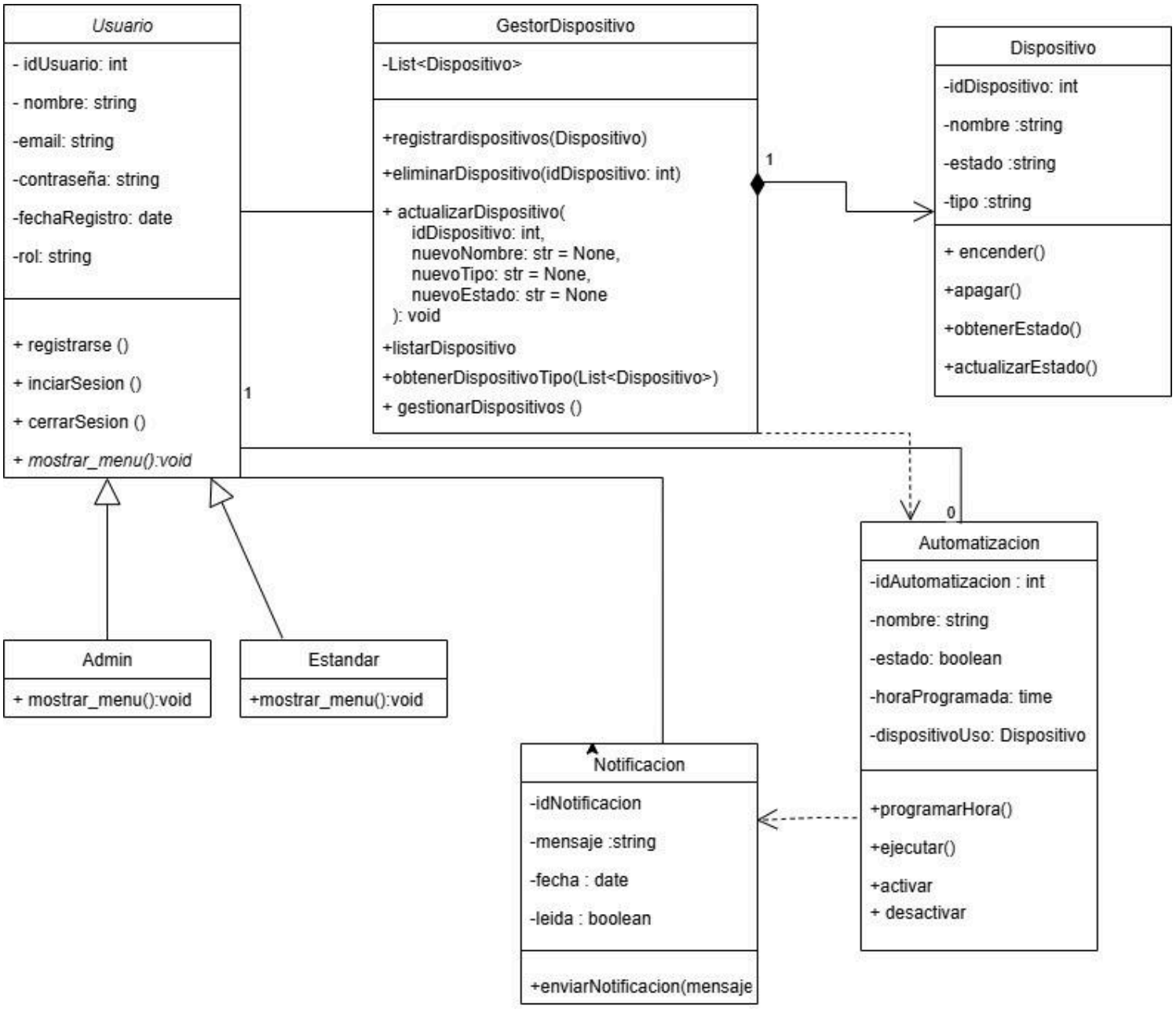
## DIAGRAMA DE CLASES




# Relaciones entre clases

1. **Usuario ↔ GestorDispositivo:** Es una relación de Asociación de 1 a 1 , porque cada usuario puede gestionar sus dispositivos mediante la clase GestorDispoistivo, que mantiene la lista de dispositivos.

2. **GestorDispositivo** ↔ **Dispositivo** Es una relación de Composición, porque GestorDispositivo contiene y controla completamente los objetos Dispositivo.

3. **Usuario** ↔ **Notificacion** es una relación de Asociación, porque cada usuario puede recibir notificaciones. La relación es débil: el objeto Notificación puede existir independiente del Usuario, pero tiene referencia a él.

4. **Automatizacion** ↔ **Dispositivo** es una relación de Asociación (Automatización "usa" un Dispositivo).porque Automatizacion actúa sobre un Dispositivo específico (dispositivoUso).

5. **Automatizacion** ↔ **Notificacion** es una relación de Dependencia porque Automatizacion puede generar notificaciones, pero no posee la clase Notificación permanentemente.

**Principios fundamentales de Poo en el diagrama**

La **abstracción** es separar la esencia de un objeto de los detalles específicos. En este diagrama lo podemos ver en las clases como GestorDispositivo y Automatizacion abstraen la lógica de gestión de dispositivos y automatización, la clase Usuario no necesita conocer cómo GestorDispositivo maneja la lista de dispositivos, solo interactúa mediante sus métodos (gestionarDispositivos()).

El **Encapsulamiento** es ocultar datos internos y exponer sólo lo necesario mediante métodos. En el proyecto lo podemos ver reflejado en que todos los atributos tienen **privacidad** como -idUsuario, -nombre, -estado.
Métodos públicos (+) permiten interactuar con los atributos, por ejemplo:Usuario.registrarse(), Dispositivo.encender().
Esto evita acceso directo a atributos, controlando cómo se leen y modifican.

Se accede a ellos solo mediante getters y setters.

La **Herencia** es un mecanismo que permite que una clase herede atributos y métodos de otra clase**,** en el proyecto podemos ver Admin y Estandar heredan de Usuario
Reutilizan los atributos y métodos comunes sin duplicar código.

El **Polimorfismo** es la capacidad de objetos de diferentes subclases de responder a un mismo método de manera distinta. En el proyecto, mostrar_menu() se llama de la misma forma en Admin y Estandar, pero se comporta diferente según la subclase.