Informe del ejercicio 1 de la práctica de diseño

Principios SOLID en el ejercicio

1. **Principio de Responsabilidad Única (SRP):**
   Este principio nos dice que una clase debe de tener una sola responsabilidad, y que está debe estar encapsulada en la clase.
   - Las clases que heredan de Empleado, Limpiador y Supervisor tienen la única responsabilidad de mostrar lo que está permitido que hagan estos empleados.
   - La clase Hotel maneja el programa, pero las responsabilidades las delega en el resto de clases que implementan las diferentes acciones que puede hacer el hotel.

2. **Principio de Abierto/Cerrado (OCP):**
   Este principio nos dice que el software deberían ser abierto para permitir su extensión, pero cerradas frente a la modificación, a menos que especificamente busquemos , como en este caso, proteger a la clase de ser extendida ya que la abertura no es conveniente.
   - Cerramos la clase EstadoHabitación para no permitir que se añadan nuevos estados que puedan romper el flujo de los diferentes estados por los que puede pasar la habitación.

3. **Principio de Sustitución de Liskov (LSP):**
   Este principio nos dice que las clases derivadas deben ser sustituibles por sus clases base.
   - Las clases derivadas de EstadoHabitación responden de manera correcta para todos los métodos abstractos que la clase pide implementar. Cuando se nos pide un objeto EstadoHabitacion podemos pasar cualquiera de las clases que la implementen.

4. **Principio de Segregación de Interfaces (ISP):**
   Este principio nos dice que debemos segregar en varias interfaces diferentes y no depender de un único interfaz de propósito general. En este caso no utilizamos ninguna interfaz, pero tampoco estamos rompiendo este principio.

5. **Principio de Inversión de Dependencias (DIP):**
   Este principio nos dice que debemos depender de abstracciones y no de concreciones.
   - La clase Habitacion, que usa instancias de EstadoHabitacion, depende de clases abstractas que extienden EstadoHabitacion en vez de depender de implementaciones concretas para cada uno de los estados.

Uso del patrón estado

Este es un patrón que permite que un objeto altere su comportamiento cuando su estado interno cambia. Este patrón es idóneo para representar el estado de la habitación. En vez de tener multitud de estructuras condicionales, tenemos una clase abstracta EstadoHabitacion que tiene varias clases hijo que implementan los métodos abstractos de la clase. Dependiendo del tipo de estado que sea la propiedad estadoHabitacion estado de la clase Habitacion las operaciones que modifican el estado se comportarán diferentes.

Diagrama de clases

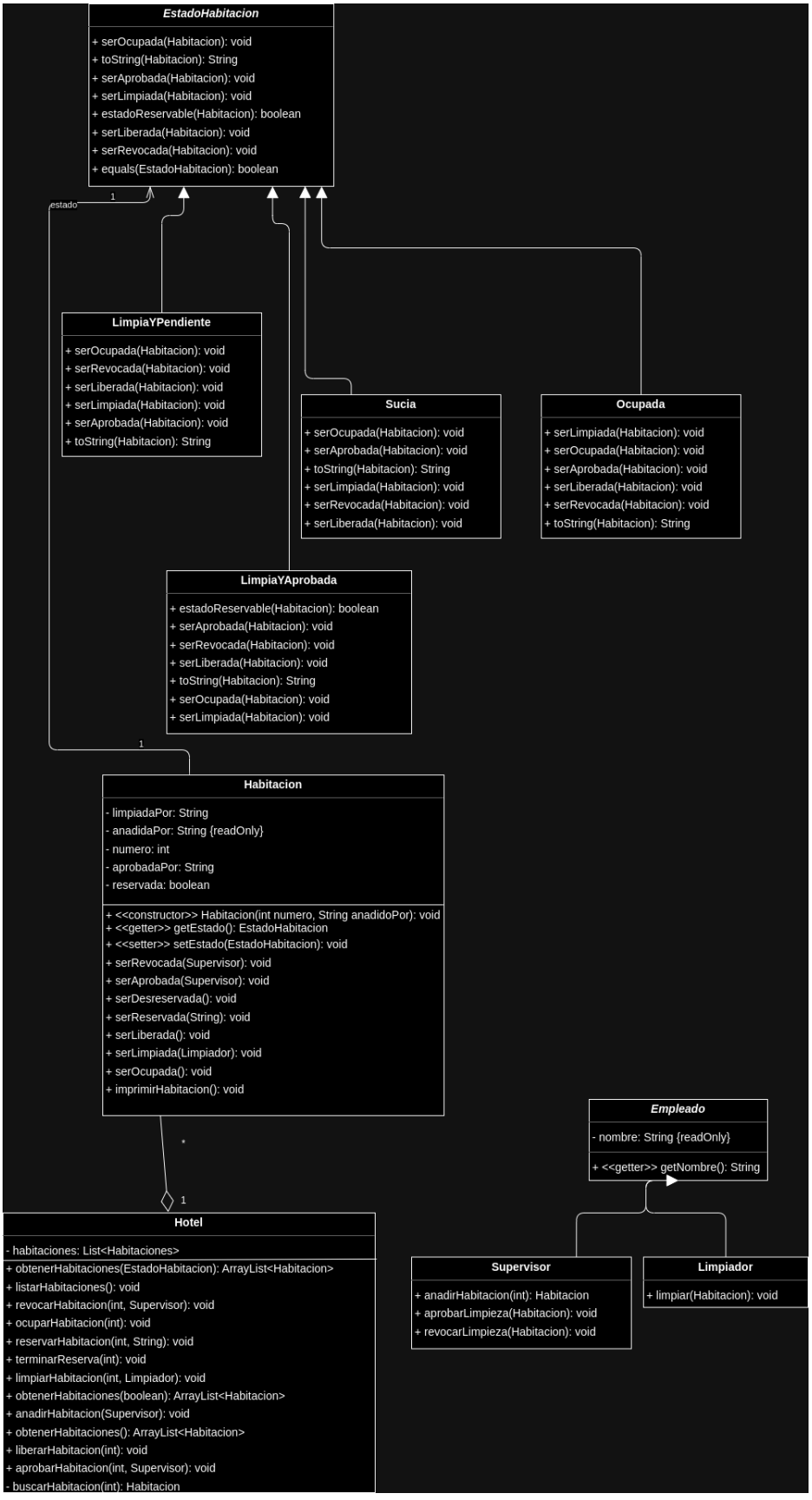

Diagrama de estados

El diagrama de estados es el adecuado para representar los diferentes estados por los que pasa una habitación y los eventos que le suceden y provoca estos cambios.