**Informe**

**Gestión de hotel**

El programa implementa un conjunto de funcionalidades que permiten realizar operaciones como añadir habitaciones, reservar, limpiar, aprobar limpieza, y otras relacionadas con la gestión de un hotel. El diseño del programa sigue varios principios SOLID y hace uso del patrón de diseño Estado para gestionar los diferentes estados de las habitaciones (Disponible, Ocupada, Pendiente de Limpieza y Pendiente de Aprobación)

En este programa utilizamos todos los principios SOLID:

El primero de ellos es el Principio de Responsabilidad Única (SRP), la interfaz EstadoHabitacion tiene una única responsabilidad, definir los métodos relacionados con los diferentes estados de una habitación y cada implementación de esta tiene una única razón para cambiar, cambios en el comportamiento asociado al estado.

El siguiente principio es el Principio Abierto-Cerrado (OCP), el patrón Estado permite la extensión del comportamiento sin modificar las clases existentes y Cada nuevo estado puede agregarse fácilmente sin modificar el código existente.

Después, se utiliza el Principio de Sustitución de Liskov (LSP), las clases concretas que implementan EstadoHabitacion son sustituibles por su interfaz común, esto se cumple ya que se puede utilizar cualquier implementación de EstadoHabitacion en lugar de otra sin cambiar el comportamiento esperado.

Otro de los principios es el Principio de Inversión de la Dependencia (DIP), la clase Habitacion depende de la abstracción EstadoHabitacion, no de implementaciones concretas y se invierte la dependencia, ya que Habitacion no depende directamente de las clases de estado concretas, permitiendo una mayor flexibilidad.

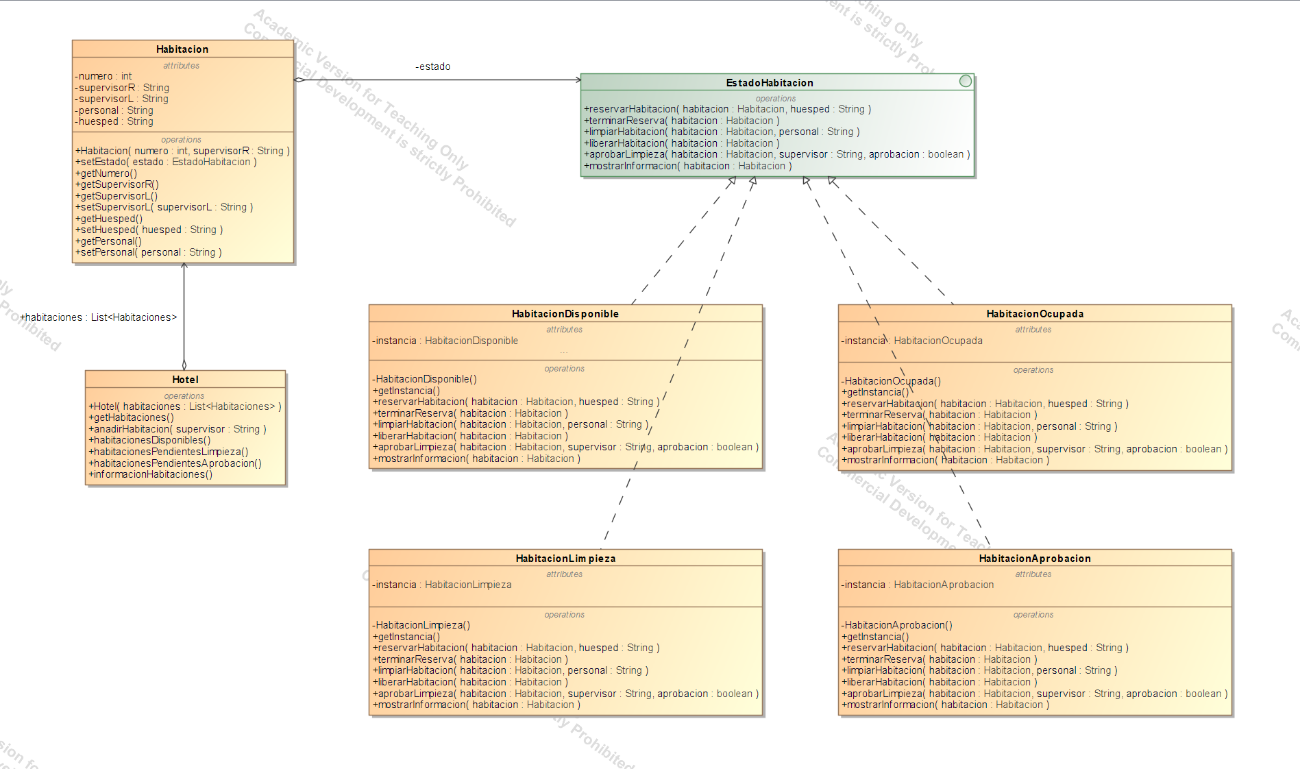
El ultimo es el Principio de Segregación de Interfaces (ISP), la interfaz EstadoHabitacion es específica y contiene solo los métodos necesarios para los diferentes estados y cada clase concreta implementa solo los métodos relevantes para su estado.

En este programa se utiliza el patrón Estado, este permite a un objeto cambiar su comportamiento cuando su estado interno cambia. Cada estado es modelado como una clase separada, y el objeto tiene una referencia a una instancia de estado que representa su estado actual.

Para este ejercicio se utiliza una clase Habitación que juega el rol de Contexto y representa una habitación del hotel que delega las operaciones al estado actual. Una interfaz, EstadoHabitacion, que define los métodos asociados a los diferentes estados y por último, Implementaciones concretas de esta interfaz, HabitacionDisponible, HabitacionOcupada, HabitacionLimpieza, HabitacionAprobacion

En conclusión, el diseño del programa sigue los principios SOLID y utiliza el patrón Estado para gestionar los diferentes estados de las habitaciones de manera clara y extensible. La aplicación de estos principios facilita la comprensión, mantenimiento y expansión del código en el futuro, promoviendo un diseño sólido y flexible.

**Diagrama de Clases**



**Diagrama de Estados**

Diagrama

Descripción generada automáticamente

**Incursiones Navales**

Este ejercicio de simulación de la incursión de flotas en un mapa sigue el principio SOLID de "Responsabilidad Única" y el patrón de diseño "Composición" para estructurar y separar las responsabilidades de las clases.

En este ejercicio el principio SOLID más claro que se utiliza es el Principio de Responsabilidad Única (SRP). En el código, cada clase tiene una responsabilidad específica. Por ejemplo, la clase Flota se ocupa de gestionar los atributos y el estado de una flota. Los nodos (NodoAtaqueAereo, NodoTormentaMarina, etc.) implementan comportamientos específicos de la simulación. La clase Mapa coordina la simulación, el seguimiento de los nodos visitados y la representación de la incursión. El principio de Responsabilidad Única se refleja claramente en la distribución de responsabilidades entre las clases. Cada nodo tiene la responsabilidad de simular su comportamiento específico, asegurando que cada clase tenga una única razón para cambiar. Esto permite que cada nodo se centre en su propia lógica sin interferir con las otras partes del sistema.

Utilizamos también el Principio de Abierto/Cerrado (OCP). Este principio apunta a que las entidades deben estar abiertas para la extensión pero cerradas para la modificación. El código está diseñado para ser extendido sin alterar su comportamiento original. Por ejemplo, se pueden agregar nuevos tipos de nodos simplemente creando nuevas clases que implementen la interfaz NodoFlota sin necesidad de modificar la lógica existente en Mapa. Del mismo modo, se puede extender la lógica de Flota sin afectar la interacción con los nodos en Mapa.

En cuanto a los patrones de diseño, nos hemos basado en el “Patrón de Composición”, que es un patrón estructural que se utiliza para componer objetos en estructuras de árbol que representan jerarquías todo-parte. El código emplea una estructura jerárquica de nodos. Cada nodo puede tener uno o dos hijos, lo que crea una composición jerárquica que el Mapa utiliza para simular la incursión. Los nodos pueden bifurcarse (NodoBifurcacion), tener un único hijo (NodoRutaFija), o representar el punto final de la incursión (NodoFinal). Esta estructura en árbol permite una simulación modular y flexible, ya que Mapa navega a través de estos nodos durante la incursión. Esta composición permite organizar y representar la incursión espacial, dando flexibilidad para extender o modificar la simulación sin alterar la estructura general.

En conjunto, el código muestra cómo los principios de diseño SOLID, en especial el de Responsabilidad Única, junto con el patrón de Composición, permiten crear un sistema modular y flexible para simular las incursiones de flotas marítimas

**Diagrama de Clases**

**Diagrama

Descripción generada automáticamente**

**Diagrama de Estados**

**Diagrama

Descripción generada automáticamente**