INFORME PRÁCTICA 3 DISEÑO SOFTWARE

-Hugo Viqueira Ans

-Antón Vilas Pazos

Practica 1, Gestión de hotel:

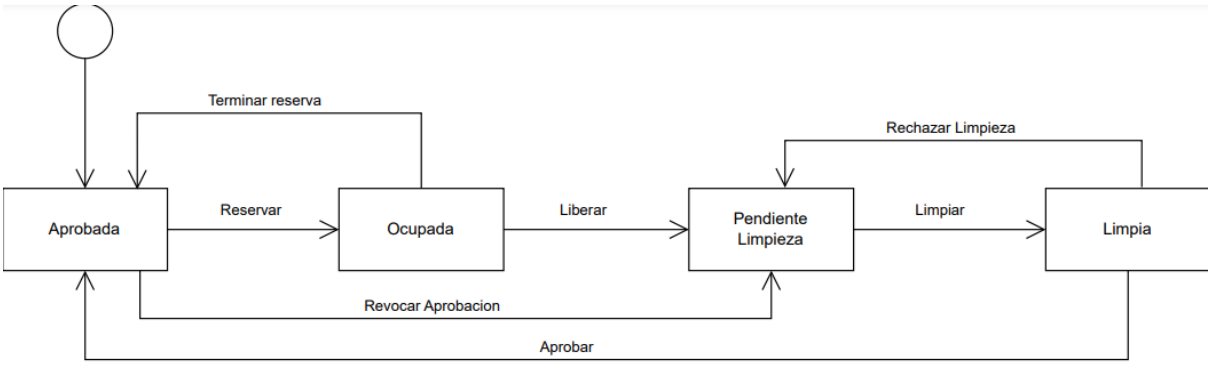
Para este primer ejercicio hemos decidido seguir el patrón estado, este patrón nos pareció el más correcto para seguir pues nos permite modificar la conducta de un objeto al cambiar su estado interno en el resto del código, así también permitiría añadir un nuevo estado cualquiera a la habitación cambiando a un nuevo comportamiento simplemente añadiendo una nueva clase estado.

Tal y como hemos definido la clase Habitación y la dependencia con la clase EstadoHabitacion (siendo un atributo de Habitación), al tratarse esta última de una abstracción y no una clase concreta, permite flexibilidad a la hora del cambio de estado de la habitación sin tener que modificar las funciones ya existentes de Habitación añadiendo que se encarguen de la evaluación de cada estado nuevo que se quiera añadir, siendo un factor favorable al mantenimiento y extensibilidad del código, por lo que cumple el principio de inversión de dependencias al separar la abstracción de la clase concreta.

En cuanto a los principios, podemos ver que se cumple parcialmente el principio de responsabilidad única, ya que una habitación es capaz de autogestionarse y cambiar de estado por si sola, aunque no se llega a cumplir al 100% debido a las funciones que gestionan los estados.

Y por último, el código cumple el principio de Sustitución de Liskov, esto se debe a que el teorema enunciado en este principio se cumple a la perfección con los estados definidos, cada estado se trata de una clase independiente la cual puede figurar como estado de la Habitación y cambiar sus propiedades y funciones dependientes del estado, sin afectar a propiedades y funciones independientes del estado.

El mejor diagrama dinámico para este ejercicio se trata del diagrama de estados, ya que al seguir este patrón, lo más obvio, eficiente y visual es realizar un diagrama de estados.



Practica 2, Incursiones navales:

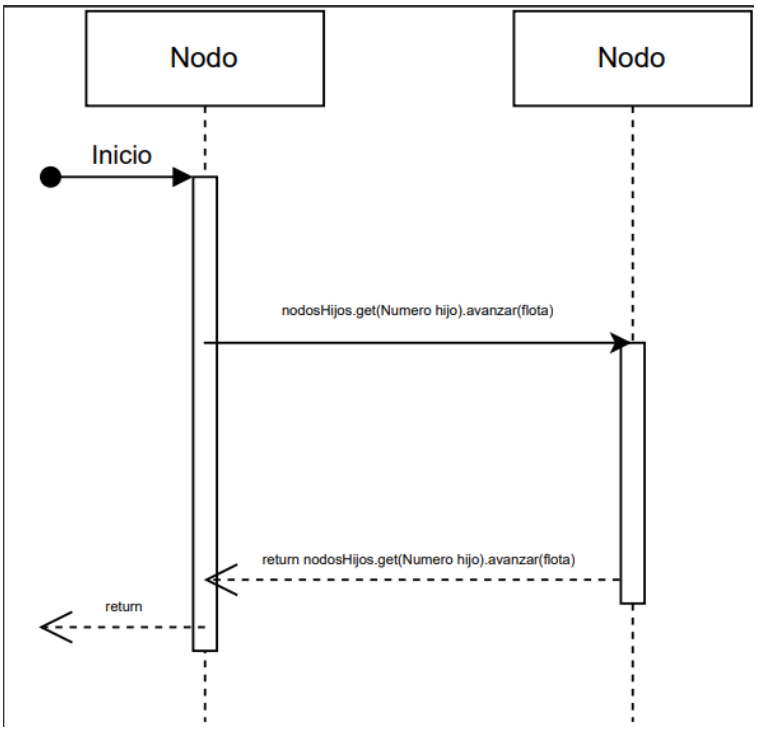
En este caso para este ejercicio hemos decidido usar 2 patrones, el primero es el patrón composición, este es usado debido a la necesidad de crear una estructura jerárquica de nodos donde cada nodo puede tener 0 o más nodos hijos, y tener la posibilidad de incorporar nuevos tipos de nodos con más hijos o variantes de nodos existentes con funcionalidad diferente y operaciones nuevas, también hemos usado debido a la facilidad que proporciona este patrón a la hora de realizar operaciones recursivas sobre los nodos.

El segundo es el patrón estrategia, debido a que en el diseño de la simulación de incursiones navales, mostrar información del nodo actual y el cálculo de la ruta mínima. Diferentes nodos tienen comportamientos específicos y complejos que deben ser encapsulados, y este patrón proporciona una manera eficaz de manejar esta variabilidad sin modificar la clase flota.

En cuanto a los principios de diseño, se cumple el principio de abierto cerrado, ya que este permite extender el sistema agregando nuevos tipos de nodos, con funcionalidades diferentes y distinto número de hijos, pero a su vez es cerrado en cuanto a la modificación del tipo nodo.

También cumple el principio de inversión de dependencia debido a la dependencia a las abstracciones e interfaces, permitiendo flexibilidad y facilidad a la hora de expandir el código mediante la herencia de estas últimas, por ejemplo podríamos añadir más tipos de nodos con operaciones diferentes a las establecidas gracias a este principio.

En cuanto a los diagramas para este ejercicio, consideramos más adecuado un diagrama UML de secuencia ya que se puede observar visualmente el comportamiento de las clases y sus relaciones a la hora de ejecutarse la incursión (método avanzar).



Para probar que funcione este ejercicio, hemos definido en el test, el siguiente mapa: las estadísticas de cada nodo, están definidas en el test y la incursión es denominada avanzar.