INFORME EJERCICIO 1 PRÁCTICA PD: GESTIÓN HOTEL

En este ejercicio nos encargamos de la creación de un hotel que dispone de habitaciones que pueden cambiar de estado según nos beneficie. Para ello usaremos una serie de principios de diseño y un patrón, que en dicho ejercicio suponemos que es evidente cual es, el patrón Estado. Antes de centrarse en la explicación de los principios y del patrón, es interesante hacer una breve descripción del problema presentado.

El problema presentado es la gestión de un hotel. Para ello necesitaremos una clase habitación y una clase hotel. El hotel estará compuesto por habitaciones (inicialmente no tendrá, las tendremos que añadir nosotros. Mencionar también que inicialmente estás estarán disponibles) y tendremos métodos que nos permitan reservarlas, limpiarlas, ver el estado de las mismas, etc. Además necesitaremos crear una interfaz, que le daremos el nombre de Estado para representar los distintos estados que pueden tener las habitaciones. Y con esa interfaz después haremos clases que implementen sus métodos y nos permitan cambiar el estado de las habitaciones del hotel.

Con esto claro, nos disponemos a explicar los principios de diseño y el patrón empleado en el desarrollo del ejercicio:

PRINCIPIOS DE DISEÑO:

- **PRINCIPIO DE INVERSIÓN DE LA DEPENDENCIA:** Lo que buscamos es que las dependencias apunten hacia abstracciones. La clase Habitación no depende directamente de las clases concretas de los estados, pero si del interfaz Estado. Esto significa que la clase Habitación no necesita saber los detalles específicos de como cada estado maneja sus operaciones, solo necesita saber que puede delegar estas operaciones a su estado actual.
- **PRINCIPIO DE RESPONSABILIDAD ÚNICA:** Cada objeto debe tener una responsabilidad única, permitiéndonos que los servicios de dicho objeto estén alineados con dicha responsabilidad. En cuanto a la clase Habitación, esta se centra en representar y gestionar el estado y los datos de una habitación, es decir, se enfoca exclusivamente en comportamientos de una habitación individual o, dicho de otra manera, no se encarga de gestionar varias habitaciones. La clase hotel, por lo contrario, es justo lo que hace, permitiéndonos realizar acciones de reserva, cancelación o mostrar información sobre el estado en el que se encuentran las habitaciones, siendo esta su única responsabilidad. Finalmente, con respecto a las distintas clases de los estados, tienen la responsabilidad de definir el comportamiento de una habitación, es decir, dependiendo en el estado en el que se encuentren, si pueden o no reservar una habitación, limpiarla, etc.

- **PRINCIPIO ABIERTO-CERRADO:** Este principio lo usamos en la clase Habitación y en las implementaciones del interfaz Estado en cada una de las clases. Esto nos permite extender nuestras clases, pero sin necesidad de modificar nuestro código.

PATRÓN ESTADO:

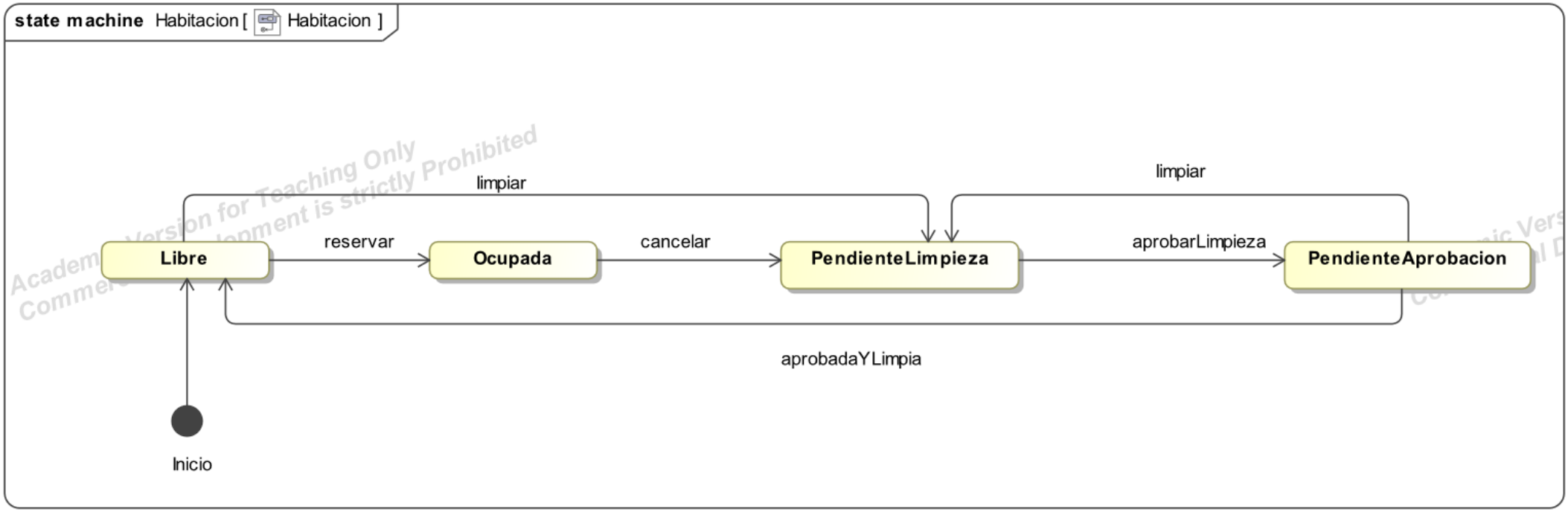
Es claramente necesario su uso en esta práctica. Dicho patrón nos permite cambiar el estado interno de las habitaciones. Es una buena solución para el problema ya que tenemos un número pequeño de estados y todos son estables. Los estados en nuestro ejercicio son:

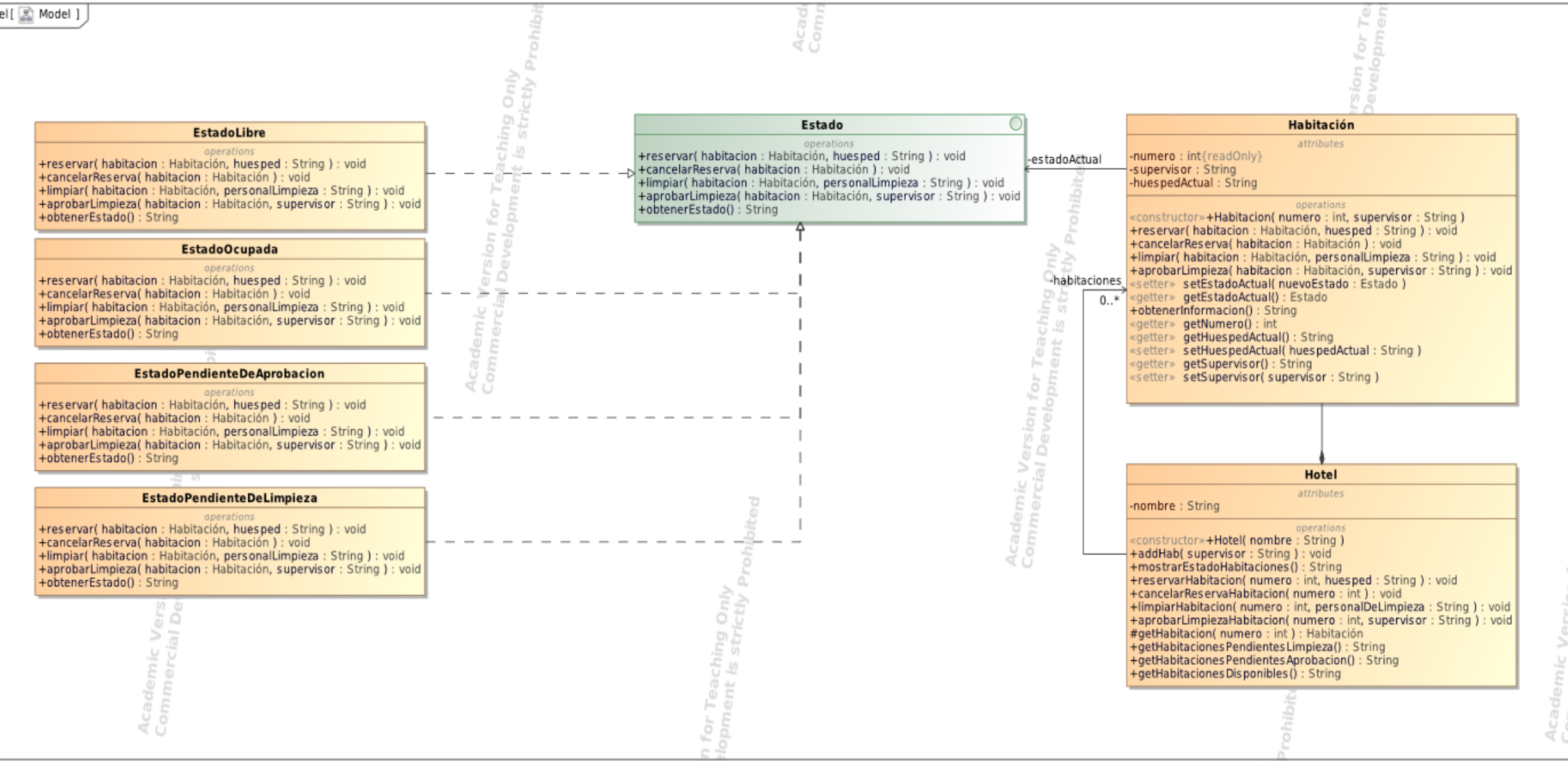
- **Ocupada**: la habitación se encuentra ocupada por un huésped
- **Pendiente de Limpieza**: la habitación esta libre y se manda limpiar o se canceló una reserva y pasa estar libre la habitación
- **Pendiente de aprobación**: dicha habitación fue limpiada y está pendiente de aprobación por un supervisor
- **Libre**: consideramos que una habitación está libre cuando esté limpia y aprobada por un supervisor (*Nuestro **libre** se refiera al **estado disponible** del enunciado*).

Cosas a tener en cuenta con los estados: los estados pueden cambiar entre sí, mas bien no es posible pasar de un determinado estado a otro por razones obvias. Por ejemplo, si una habitación está libre (disponible), no vamos a poder pasar a estado pendiente de aprobación porque una sencilla razón, la habitación ya está limpia y aprobada por un supervisor. Otro posible caso sería pasar de ocupada a libre. Para esto, primero tendrá que pasar un proceso de limpieza y ser aprobada por un supervisor.

Con respecto a la aplicación del patrón, se realiza en la clase Habitación, en la interfaz estado y en cada una de sus implementaciones (clases: EstadoLibre, EstadoPendienteDeAprobacionLimpieza, etc). Finalmente, estos son algunos casos, pero lo veremos mejor en nuestro diagrama de estados y en nuestro diagrama de clases:

Comentar de todas formas que si no se puede visualizar de forma adecuada las imágenes de los diagramas en este informe, dichas imágenes están subidas en el repositorio de github en formato vectorial.


state machine Habitacion [Habitacion]
limpiar
limpiar
reservar
cancelar
aprobarLimpieza
Libre
Ocupada
PendienteLimpieza
PendienteAprobacion
aprobadaYLimpia
Inicio


Model
EstadoLibre
operations
+reservar(habitacion : Habitación, huesped : String) : void
+cancelarReserva(habitacion : Habitación) : void
+limpiar(habitacion : Habitación, personalLimpieza : String) : void
+aprobarLimpieza(habitacion : Habitación, supervisor : String) : void
+obtenerEstado() : String
EstadoOcupada
operations
+reservar(habitacion : Habitación, huesped : String) : void
+cancelarReserva(habitacion : Habitación) : void
+limpiar(habitacion : Habitación, personalLimpieza : String) : void
+aprobarLimpieza(habitacion : Habitación, supervisor : String) : void
+obtenerEstado() : String
EstadoPendienteDeAprobacion
operations
+reservar(habitacion : Habitación, huesped : String) : void
+cancelarReserva(habitacion : Habitación) : void
+limpiar(habitacion : Habitación, personalLimpieza : String) : void
+aprobarLimpieza(habitacion : Habitación, supervisor : String) : void
+obtenerEstado() : String
EstadoPendienteDeLimpieza
operations
+reservar(habitacion : Habitación, huesped : String) : void
+cancelarReserva(habitacion : Habitación) : void
+limpiar(habitacion : Habitación, personalLimpieza : String) : void
+aprobarLimpieza(habitacion : Habitación, supervisor : String) : void
+obtenerEstado() : String
Estado
operations
+reservar(habitacion : Habitación, huesped : String) : void
+cancelarReserva(habitacion : Habitación) : void
+limpiar(habitacion : Habitación, personalLimpieza : String) : void
+aprobarLimpieza(habitacion : Habitación, supervisor : String) : void
+obtenerEstado() : String
-estadoActual
Habitación
attributes
-numero : int{readOnly}
-supervisor : String
-huespedActual : String
operations
«constructor»+Habitacion(numero : int, supervisor : String)
+reservar(habitacion : Habitación, huesped : String) : void
+cancelarReserva(habitacion : Habitación) : void
+limpiar(habitacion : Habitación, personalLimpieza : String) : void
+aprobarLimpieza(habitacion : Habitación, supervisor : String) : void
«setter» setEstadoActual(nuevoEstado : Estado)
«getter» getEstadoActual() : Estado
+obtenerInformacion() : String
«getter» getNumero() : int
«getter» getHuespedActual() : String
«setter» setHuespedActual(huespedActual : String)
«getter» getSupervisor() : String
«setter» setSupervisor(supervisor : String)
-habitaciones
0..*
Hotel
attributes
-nombre : String
operations
«constructor»+Hotel(nombre : String)
+addHab(supervisor : String) : void
+mostrarEstadoHabitaciones() : String
+reservarHabitacion(numero : int, huesped : String) : void
+cancelarReservaHabitacion(numero : int) : void
+limpiarHabitacion(numero : int, personalDeLimpieza : String) : void
+aprobarLimpiezaHabitacion(numero : int, supervisor : String) : void
#getHabitacion(numero : int) : Habitación
+getHabitacionesPendientesLimpieza() : String
+getHabitacionesPendientesAprobacion() : String
+getHabitacionesDisponibles() : String