Patrones de Diseño

SalUMAble

ESTADO

Este patrón de diseño se adaptó correctamente a la idea que teníamos para la resolución del problema de tener un sistema que alteraría su funcionamiento de forma casi absoluta en función del usuario que hiciera uso de la aplicación.

Lo que este patrón sugiere es, en lugar del uso de una variable global que sea comprobada cada vez que una función que varíe dependiendo del estado sea ejecutada, se cree una clase general que se conoce como "Contexto" de la cual encontramos unas clases representativas de cada posible estado subordinadas al contexto, estos estados pueden o no ser intercambiables durante la ejecución.

Esta implementación del patrón de diseño en nuestro caso se realiza utilizando el Usuario (clase abstracta) como la clase contexto de la que heredan el administrador (clase UsuarioAdministrador), los pacientes (clase UsuarioPaciente) y los doctores (clase UsuarioDoctor). Estas tres clases son accedidas en el momento del login en el cual, en función de las credenciales utilizadas, se obtienen unas funciones distintas para interaccionar con el sistema. Estos tres posibles estados pueden intercambiarse con las funciones de logout y login que permiten acceder a todas las funciones del sistema.

También usamos una variante de este patrón a la hora de mostrar la información del paciente. Cada paciente tiene asociado un enumerado con su estado actual (EstadoPaciente). Dependiendo de cuál sea su estado, la vista de la página del paciente actuará de una forma u otra, mostrando una información diferente según su estado.

SINGLETON

Este patrón se utilizó a la hora de asegurar que solo pueda existir una instancia de la clase UsuarioAdministrador, ya que no tendría sentido crear varias instancias.

El patrón Singleton sugiere cambiar el constructor original de la clase a privado para evitar su uso por parte de clases externas con "new". Se crea además una función estática similar al constructor que llama al verdadero constructor, creando la clase deseada si aún no ha sido instanciada y si lo ha sido, devolviendo esta instancia.

Esta implementación del patrón de diseño en nuestro caso se realiza creando una variable estática de la clase llamada "instance", que inicialmente está a null, pero cuando se inicializa por primera vez, almacena la instancia que se ha creado. Al ser el constructor privado, para crear un objeto de dicha clase hay que llamar al método getInstance(). Dicha clase comprueba si la variable "instance" está a null, indicando que no se ha creado ninguna instancia todavía, por lo que se crea una nueva llamando al constructor de la clase. En caso de no estarlo, significa que ya ha sido creada anteriormente y la instancia está almacenada en dicha la variable, por lo que la devuelve en vez de crear otro objeto.