Práctica 2

Asignatura: Tecnologías avanzadas de programación

Desarrollo de una aplicación web de gestión de ascensores de un edificio

Integrantes del grupo:

Andrea María García
Paula Hípola Gómez
Ignacio Moll Amorós
Belén Ortega Pérez
Carlos Tendero Martínez-Algora
Borja García Lamas

Universidad Francisco de Vitoria
UFV Madrid

ÍNDICE

1.Desarrollo de las funcionalidades del software

En este apartado se desarrolla de forma breve las distintas funcionalidades que ofrece nuestro software.

El propósito del mismo es gestionar y controlar el funcionamiento de un grupo de ascensores en las distintas plantas de un edificio. Concretamente, el edificio tiene 7 plantas y 3 ascensores, que se desplazan a lo largo de todas estas. El usuario es capaz de gestionar y monitorizar los ascensores a través de una aplicación web, a la vez que puede monitorizar el estado de cada planta.

Desde la página web, el usuario puede ver 3 zonas: el edificio con el estado de cada ascensor, cada una de las plantas con la situación actual, y el interior de cada uno de los ascensores en una planta. A continuación se describen estas 3 zonas y las posibilidades de interacción que ofrece nuestro software:

1. **Con respecto al edificio,** el usuario podrá ver en todo momento la situación de cada ascensor, que incluye: la planta actual en la que se encuentra, el estado de la puerta(abierto, cerrado), el estado del ascensor (subiendo, bajando, parado) y si la alarma está activada.

2. **Con respecto a las plantas,** el usuario podrá acceder a una vista de cada una de ellas desde el panel de control del edificio. En cada planta podrá llamar a cada uno de los 3 ascensores para que vayan a esa planta. Podrá ver el estado actual de cada ascensor. Desde la planta actual del edificio se podrá acceder a cualquiera de las plantas.

3. **Con respecto a los ascensores**, el usuario podrá “entrar” en el ascensor, controlar a qué piso quiere que vaya con una botonera y activar la alarma. La alarma es un dispositivo de emergencia que lleva el ascensor a la planta más cercana que tenga y a continuación abre las puertas. Con este sistema, evitamos que el ascensor se mueva mucho mientras la emergencia sigue vigente y al mismo tiempo evitamos que se quede atascado entre dos plantas.

El ascensor tiene un sistema de guardado de destinos, de modo que si un ascensor es requerido en una planta mientras está de camino a otra, guarda el próximo destino para ir allí nada más haber acabado el anterior viaje. Esto lo hace de forma secuencial, es decir, uno detrás de otro en el orden de entrada.

2.Decisiones de diseño

A la hora de realizar los constructores, hemos decidido no diseñar constructores vacios para que a la hora de crear objetos de las distintas clases implementadas, estoss e creen con algo en su interior y no un objeto vacio incapacitado de posible funcionalidad.

Por ejemplo: a la hora de crear un Edificio, este nos devuelve un objeto con 3 ascensores y 7 plantas.

A la hora de generar el código hemos decidido estructurar mediante el uso de paquetes, teniendo así un paquete para las clases, un paquete para las Implementaciones (Impl -> Implements)

2.1 UML

En este apartado hay que meter el UML cuando esté terminado completamente y revisado. Explicarlo también un poco.

Esta herramienta nos permite representar las distintas clases e interfaces usadas en el programa, y cómo están relacionadas y que tipo de relación entre ellas.

El UML está adjunto al documento word debido a su tamaño.

2.2 Patrones definidos

En este apartado se enumeran y explican los patrones aplicados en el proyecto y su función.

1. Patrón State: el patrón state es un patrón de diseño de comportamiento que sirve para desarrollar correctamente un sistema que funcione como una máquina de estados, en la que según el estado en el que se está se realizan unas acciones u otras.

   En este proyecto implementamos el state para controlar los distintos estados en los que puede estar un ascensor (ej. paradoAbierto, paradoCerrando). Dentro de cada clase estado existen 3 acciones que dependen de los mismos, que son:

   - **cambiarEstadoPuerta**: con esta función controlamos el cambio de estados y cómo afecta esto a la puerta del ascensor

   - **moverAscensor**: con esta función controlamos el movimiento que debe realizar el ascensor en cada estado, incluyendo el no moverse.

   - **activarAlarma**: con esta función controlamos lo que hará el ascensor cuando la alarma esté activada en cada estado. En caso de que se esté moviendo, el ascensor se moverá a la planta más cercana y se abrirá. En caso de que esté quieto, dejará las puertas abiertas y se estará quieto hasta que se apague la alarma.

   Para la implementación del patrón State en nuestro software, hemos usado los siguientes .java:

   - Una interfaz llamada State que contiene todas las acciones que se realizan dentro de un estado y que dependen del mismo.

   - Un paquete de clases que se llama Clases.AscensorPuertaState que contiene todos los estados posibles del ascensor, cada uno en un fichero .java. Estas clases son:
     - MoviendoAscensor
     - ParadoAbierto
     - ParadoAbriendo
     - ParadoCerrado
     - ParadoCerrando

     Todas estas clases implementan la interfaz State

   - Una clase Ascensor que tiene un atributo tipo State y con la que ejecuta las funciones del estado en el que se encuentra.

2. Patrón Observer: el patrón Observer es un patrón de diseño de tipo comportamiento que se usa para definir dependencias entre varios objetos a uno sólo. De este modo, cuando las componentes dependientes varían, los componentes independientes son avisados y pueden actuar en consecuencia.

   En este proyecto implementamos el Observer para controlar y actualizar la aplicación web con los valores de los ascensores, plantas y el edificio. Los observadores observan los estados de los ascensores, e informan al edificio. Este es actualizado con una función en la clase que usamos para desarrollar la interfaz. Esta clase se llama myUI.java.

3. Patrón Singleton: este patrón facilita una única instancia global, declarando el constructor privado para que no sea accesible directamente. Permite el acceso global a dicha instancia mediante un método de clase.

   En este proyecto implementamos el patrón singleton a la hora de instanciar la clase edificio, ya que nuestro edificio no varía, varían los valores de los ascensores y las plantas dentro del edificio.
   Para la implementación del patrón Singleton en nuestro software,vamos usar el siguiente .java:

   - Edificio.

   Lo utilizaremos para instanciar una única vez la clase edificio que contendrá los tres ascensores y las 7 plantas, además de agrupar todo se ve mucho más claro y el código lleva más limpio.

2.3 Principios cubiertos

En este apartado se explican los principios de diseño que se cumplen en nuestro proyecto.

- **Principio de segregación de interfaces**: procuramos que todas las clases usen todos los métodos que contiene.

- **Principio de inversión de dependencia**s: en nuestro proyecto, las clases de más bajo nivel dependen de las de más alto nivel y no al revés. Por ejemplo, edificio depende de ascensor, y no ascensor de edificio.

- **Encapsular lo que varía:** todas las variables y clases que varían están encapsuladas de modo que minimizamos el impacto de los cambios.

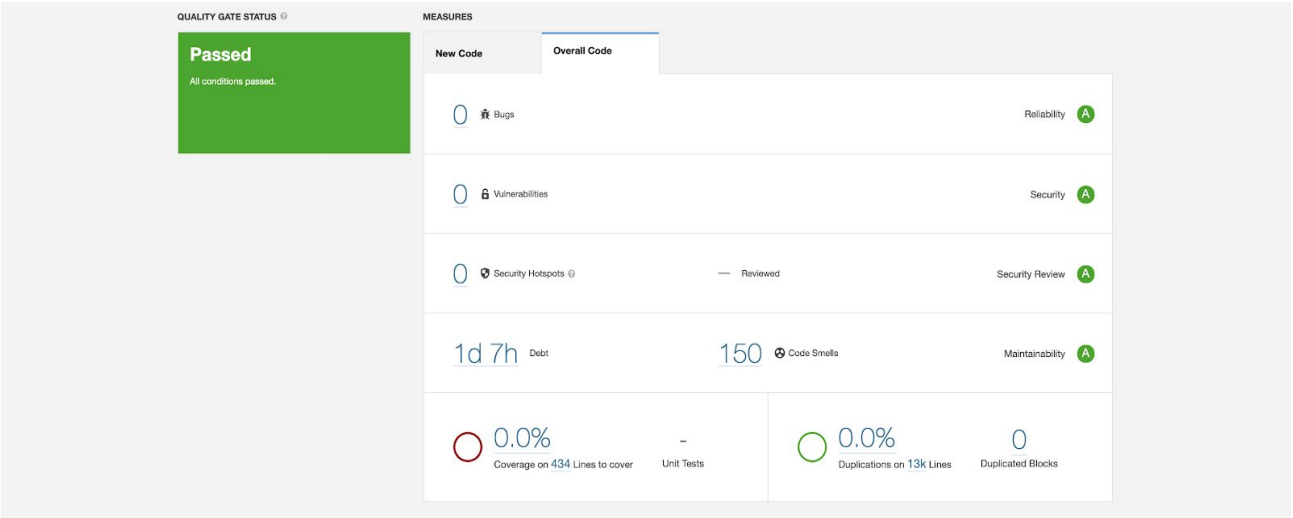
- **Programar a interfaces, no implementaciones**: todas las clases principales dependen de su interfaz, que define todos los métodos que contienen. Por ejemplo, la clase Ascensor con AscensorImpl o Planta con PlantaImpl.

- **Favorecer la composición sobre la herencia**: para este proyecto, usamos la menor cantidad de herencias posibles, y optamos por hacer instancias de los objetos en las clases que lo necesiten.

- **Do not repeat yourself**: en nuestro proyecto reutilizamos todo el código que tenemos, no hay dos métodos que hagan lo mismo en distintas clases.

- **Open/Close**: como nuestro software es bastante modular, y a lo largo del desarrollo hemos ido realizando muchas implementaciones de nuevos métodos, podemos decir que cumplimos este principio. Cuando debíamos añadir nuevas cosas al código, nunca hacía falta modificar el código anterior.

3.Reporte Sonarqube



Esta imagen muestra el reporte de Sonarqube que hemos obtenido sobre nuestro software. Como podemos ver, pasamos todas los aspectos con A, es decir, buena valoración. El único problema que hay en nuestro código es que tenemos varios code smells, los cuales no nos han dado tiempo a corregir.