BITÁCORA TP ANUAL: GRUPO 9

Integrantes:

- **Dellafiore, Matias**
- **Hernandez Canetti, Diego**
- **Martina Lopez, Avril**
- **Navarro, Ramiro**
- **Paoli, Juan Francisco**

ÍNDICE:

Primera Entrega - 25/04

CASOS DE USO:

- En esta primera entrega, no se hace hincapié en la diferencia en las funcionalidades, dentro del sistema, entre los usuarios de la plataforma, usuarios administradores[1] y miembros de la comunidad, por lo tanto, se graficó como **usuario** con las funcionalidades básicas explicitadas en los requerimientos. En un futuro esto puede cambiar y se implementarán las modificaciones necesarias.
- El ente regulador de servicios consideramos que al ser un organismo estatal sus actividades ocurren por fuera del sistema. En cuanto al reglamento del estado de los servicios dentro del sistema, consideramos que es el propio sistema quien se encarga de ello.

DIAGRAMA DE CLASES:

MÓDULO SERVICIOS Y USUARIO

CLASE SERVICIO PÚBLICO

Decidimos modelar como una simple clase para esta entrega, ya que no conocemos el comportamiento que tendrán estos servicios, ni tampoco si todos tendrán el mismo comportamiento o no. Por lo tanto, se utilizó un atributo tipoDeServicioPublico de tipo enumerado para identificar el servicio que se quiere inicializar(por el momento, todos tendrán el mismo comportamiento).

INTERFAZ SERVICIOS

En la consigna quedó especificado que cada uno de los servicios va a tener un mismo comportamiento, pero estos van a tratar los métodos de manera diferente.

La última consideración, es que en un futuro, podremos agregar más servicios de manera polimórfica a nuestro sistema, sin necesidad de romper las estructuras ya predispuestas.

SERVICIOS DE ELEVACIÓN

Se usa un enumerado para los distintos tipos de servicios de elevación: ascensores y escaleras mecánicas. Decidimos hacer esto ya que a futuras entregas se pueden ir

[1] Asimismo, no se integró la idea de conflictos de intereses ya que esto va por afuera del sistema

agregando más servicios de elevación y por el momento no hay ninguna diferencia entre la funcionalidad de cada uno dentro del sistema (todos usan los mismo métodos).

Para el tramo, se optó por una solución donde el tramo no sea una clase con métodos, esto es porque el enunciado nunca menciona: de que se tenga que calcular el tramo, que el tramo afecte servicio, que tenga básicamente algún comportamiento. Por lo anteriormente mencionado, decidimos diagramarlo como una clase de tipo Enum que diga qué parte se está recorriendo, y de esta manera lograr una mejor abstracción.

PERSONA Y USUARIO

En nuestro diseño hemos decidido considerar una persona (física) y un usuario como la misma entidad, en donde el usuario toma los atributos de la persona que lo crea. El motivo de esta decisión es que en el sistema los que interactúan con los servicios son usuarios de la plataforma, no “personas”.

CLASE USUARIO

En el presente diseño no se tomó la clase usuario como una interfaz o una abstracción ya que, por lo mencionado anteriormente, no se han hecho diferenciaciones en las funcionalidades de los diferentes tipos de usuarios. Esto se puede implementar a futuro de ser necesario ya que nuestro diseño es adaptable.

REPOSITORIO DE USUARIOS

Se agregó una clase llamada repositorio de usuarios para el manejo de los mismos a través de una lista con todos los usuarios que se van creando en el sistema. Además, esta clase se encarga de validar los nombres de usuario, es decir, controla que no exista un usuario ya registrado con el mismo usuario (en caso de ser así, lanza una excepción evitando que el usuario pueda registrarse con dicho usuario).

MÉTODO REGISTRAR USUARIO

El método es void en lugar de bool, ya que no es necesario preguntarle al validador ed contraseñas, si la contraseña es válida o no, sino que es más eficiente decirle que la valide, y, en caso de ser inválida, cortar la ejecución con una excepción y un mensaje indicando que requisitos de seguridad incumple. Por otro lado, si se valida la contraseña, simplemente continúa ejecutándose el programa.

SERVICIOS PÚBLICOS Y LÍNEAS:

Consideramos que los ferrocarriles y los subterráneos tienen como atributo la línea que recorren. Una vez que completan su viaje, otra instancia es la que realiza el tramo de vuelta siguiendo una nueva línea (en la dirección contraria).

MÓDULO SEGURIDAD

Hemos decidido trabajar toda la parte de Seguridad del Sistema tiene un comportamiento totalmente diferente al de Usuario y Servicios. La modularidad ayuda además a la mantenibilidad del sistema a futuro.

CLASE VALIDADOR_CONTRASENIAS

Esta clase implementa una serie de reglas que toda contraseña debe cumplir para poder ser validada. Esto proporcionará:

- Mayor facilidad a la hora de agregar nuevas funcionalidades
- Mayor facilidad para corregir comportamiento ya integrado
- Mayor facilidad a la hora de entender el código(y el diagrama)
- Mayor facilidad para eliminar comportamientos obsoletos o ya no usados.

CONTRASEÑA

- Las contraseñas se van a almacenar como un string, y no como una clase, ya que estas no presentan inconsistencia de datos. Además, no consideramos necesario contar con una clase contraseña, ya que no necesitan atributos ni métodos propios.
- Tienen integradas todos los requerimientos pedidos para esta entrega, además de algunos extra para aumentar la seguridad de los usuarios dentro de la plataforma(en caso de que se quiera modificar esto, por la manera en que fue diseñado el código, es muy sencillo eliminarlo).
- Las especificaciones que deben cumplir las contraseñas se encuentran delegadas en clases que implementan una misma interfaz “ReglaContrasenia”. De esta manera, nos aseguramos todas tengan el mismo comportamiento, pero implementado a su determinada manera. Algunas de las reglas necesitan comparar la entrada con unos archivos en busca de símbolos, palabras de diccionario y contraseñas comunes. Para ello, esta información se almacena en archivos.txt, los cuales se encuentran en una configuración a parte. Esto permite una mayor facilidad a la hora de modificar y/o realizar un mantenimiento de estas teniendo en cuenta los requerimientos del usuario.

EXCEPCIÓN REGISTRO DE USUARIO

Para lograr una ejecución del programa más fluida y fácil de entender para los usuarios. En casos en los que la contraseña/nombre de usuario que quisieron utilizar no fuese válida (según los requerimientos definidos para este sistema), la excepción a lanzar permite que estos sepan que fue un error durante su registro, y deberán realizar uno o más cambios para lograr registrarse. Debido a que no consideramos necesario “catchear” la excepción, esta clase hereda de RuntimeException