Programación 2 // Práctica 1

Álvaro Recolons - Pepe Ferrer-Bonsoms

10 de Marzo 2024

Indice

1 Introducción

En este nuevo proyecto de Java, hemos tenido que crear la interfaz de la gestión de una estación de tren con vías, accesos de distintos tipos, incidencias y otras características. Para ello, hemos creado distintas clases de Java que representan cada elemento (Via, Incidencies, Ascensor, Passadis...), y otras que nos ayudan a mantener y usar de forma correcta la información (LlistaAccessos, LlistaIncidencies...). Todo esto para más tarde inicializar una estación de tren con unos datos proporcionados por el enunciado y proponer al usuario una interfaz con un menu y 13 opciones de acciones a hacer.

2 Respuestas

2.1 Clases Implementadas

Para hacer este trabajo, hemos tenido que crear y usar las siguietnes clases de Java:

Classe Via

Implementa la interfaz InVia. La clase Via representa una vía de la estación de tren con los siguientes atributos:

nom (String): Nombre de la vía.

ampladaVia (float): Amplitud de la vía.

numTunels (int): Número de túneles en la vía.

estatVia (boolean): Estado de la vía (abierta/cerrada).

estatIlluminacio (String): Estado de la iluminación de la vía.

Por otro lado, los métodos definidos en esta clase permiten interactuar con los objetos de vía. Por ejemplo, el método tancarVia(Incidencia in) se encarga de cerrar la vía en respuesta a una incidencia (una clase que veremos más tarde) recibida como parámetro. Esto implica cambiar el estado de la vía a "cerrada" y ajustar la iluminación según la situación de la incidencia. Por otro lado, el método obrirVia() abre la vía, restableciendo su estado a "abierta" y configurando la iluminación al máximo.

Además, la clase incluye getters y setters para acceder y modificar los atributos de la vía. El método toString() devuelve una representación en formato de cadena de la instancia de la clase, mostrando los valores de sus atributos.

Clase Access

Implementa la interfaz InAccess. La clase Access es una clase abstracta que define atributos comunes para representar distintos tipos de accesos a las vías. Sus argumentos son:

nom (String): Nombre del acceso.

accesibilidad (String): Si/No según si pueden ir las personas discapacitadas

estatAccess (boolean): Estado del acceso

accesVies (arrayList¡Via¿): Lista de vías a las que da acceso

Como la clase Access es abstracta, no puede ser instanciada directamente, sino que debe ser extendida por otras clases que se pueden instanciar y que implementen los atributos y métodos de Access. En este caso,

las clases AccessDesnivell y AccessNivell extienden la clase Access, aunque estás dos también son clases abstractas que seran extendidas por más clases (lo veremos a continuación).

La clase Access no tiene métodos, pero si tiene Constructor, Getters y Setters

Clase AccessDesnivell

La clase abstracta AccessDesnivell extiende la clase Access y representa un tipo específico de acceso que incluye información adicional sobre la altura. Su único argumento no hererdado es:

height (float): altura del acceso

Su constructor incializa los atributos heredados de Access y la altura que es un atributo propio. Además contiene los Getters y Setters de height. Por último el método toString() proporciona una representación en formato de cadena de texto de un objeto AccessDesnivell, mostrando la altura junto con otros atributos heredados como el nombre, la accesibilidad, el estado y el número de vías de acceso.

Será extendida por las siguientes clases: Ascensor, Escala, EscalaMecanica.

Clase AccesNivell

Al igual que la clase AccessDesnivell, la clase abstracta AccessNivell extiende la clase Access y representa un tipo específico de acceso que incluye información adicional sobre la longitud. Su único argumento no hererdado es:

longitud (float): longitud del acceso

Su constructor incializa los atributos heredados de Access y la longitud que es un atributo propio. Además contiene los Getters y Setters de longitud. Por último el método toString() proporciona una representación en formato de cadena de texto de un objeto AccessNivell, mostrando la longitud junto con otros atributos heredados como el nombre, la accesibilidad, el estado y el número de vías de acceso.

Será extendida por las siguientes clases: CintaTransportadora, Passadis.

Clase Ascensor

La clase Ascensor extiende la clase AccessDesnivell y representa un tipo específico de acceso que es un ascensor. Aparte de los atributos heredados de la clase AccessDesnivell, como el nombre, la accesibilidad, el estado, el número de vías de acceso y la altura, Ascensor también incluye un atributo propio:

carrega (float): carga máxima que puede soportar el ascensor

La clase implementa los siguientes métodos implementando la interfaz InAccess: afegirVia permite agregar una vía a la lista de vías de acceso del ascensor; tancarAcces cambia el estado de acceso del ascensor a cerrado; obrirAcces cambia el estado de acceso del ascensor a abierto y isAccessibilitat devuelve true, indicando que el ascensor es accesible para gente con discapacidad. Además, la clase contiene el método toString que proporciona una representación en cadena de texto del ascensor, mostrando su carga máxima y sus otros atributos heredados.

Por último, la clase también contiene el Constructor que inicializa los atributos (incluido el no heredado carrega), los Getter y los Setters de carrega.

Classe Escala

La clase Escala es una subclase de AccessDesnivell y representa un tipo específico de acceso, en este caso, una escalera. Al igual que la clase Ascensor, Escala hereda los atributos y métodos de la clase AccessDesnivell, pero a diferencia de Ascensor, Escala no tiene ningún atributo propio.

Los métodos de Escala son los mismos que el de Passadis, ya que ambas implementan la interfaz InAccess. La única diferencia es que Escala no es accesible por personas con dispacacidad y por eso el método isAccessibilitat devuelve false. El método toString es llamado como super., ya que al no tiene ningún atributo propio.

El constructor de Escala inicializa los atributos heredados de AccessDesnivell, como el nombre, la accesibilidad, el estado, las vías de acceso y la altura.

Clase EscalaMecánica

La clase EscalaMecanica sigue el mismo estilo que Ascensor y Escala, extendiendo la clase AccessDesnivell y añadiendo el siguiente atributo:

marca (String): la marca de la escalera mecánica.

Los métodos son exactamente iguales que los métodos de Escala, aunque el método toString, el Constructor y los Getters y Setters es como el de Passadís ya que tenemos el atributo marca.

Clase CintaTransportadora

La clase CintaTransportadora es una clase que extiende a la clase AccessNivell, por tanto tiene sus argumentos y además, contiene el siguiente argumento propio:

velocitat (float): velocidad a la que va la cinta

Además, tiene los mismos métodos que las anteriores clases: afegirVia, tancarAcces, obrirAccess y isAccessibilitat (que nos devolverá siempre false). Además tiene el método toString.

Por último, tiene el Constructor, el Getter y Setter del atributo velocidad.

Clase Passadis

La clase Passadis es similar a la clase CintaTransportadora ya que extiende a la clase AccessNivell. La diferencia entre estas dos clases es que Passadis no tiene argumentos propios. Al no tener argumentos propios, el Constructor unicamente tiene los atributos heredados de la clase AccessNivell. Además, no tenemos ni Getters ni Setters. Por último, los métodos son los mismos que en las anteriores clases, con el método isAccessibilitat devolviendo true ya que si que es apta para gente con discapacidad.

Clase LlistaVies

La clase LlistaVies es una clase que implementa la interfaz InLlistaVies, que contiene unos métodos que tendremos que implementar. Tiene un único argumento:

llistaVies (ArrayList¡Via¿): contiene el conjunto de Vias

La clase contiene los siguientes métodos: afegirVia: dada una vía la añade a la lista, si ya está en la lista lanza una Excepción; buidar: vacia la lista de vías; llistarVies: devuelve un String con la información de todas las vías de la lista; containsViesObertes: comprueba si la lista contiene vías abiertas; contais: dada una vía comprueba si ya está en la lista, y por último, getVia: busca la vía con el nombre recibido por parámetro y la devuelve.

Además, la clase contiene el Constructor, el Getter y Setter del atributo llistaVies

Clase LlistaIncidencies

Muy similiar a la clase LlistaVies. Implementa la interfaz InLlistaIncidencies y tiene un único argumento que es una lista de Incidencias. Los métodos que tiene son también muy similares a los de la clase anterior: afegirIncidencia, eliminarIncidencia, llistarIncidencies y getIncidencia.

Clase LlistaAccessos

Análoga a las dos anteriores. Implmeneta la interfaz InLlistaAccessos con un único argumetno que es una lista de accessos. Implementa algunos métodos que ya hemos visto y algunos métodos nuevos: afegirAccess, buidar, llistarAccessos, actualitzarEstatAccess (cierra todos los accesos excepto los que dan acceso a una vía abierta), calculaAccessosAccessibles (Itera sobre la lista de accesos y devuelve el número de accesos con accesibilidad) y calculaLongitudAccessosNivell (Itera sobre la lista de accesos, y por los accesos a nivell suma la longitud y la devuelve)

Clase EstacioTren

La clase EstacioTren define una estación de tren con los siguientes atributos:

nomEstacio (String): nombre de la estación

llistaAccessos (LlistaAccessos): lista de accesos de la estación

llistaVies (LlistaVies): lista de vías de la estación

llistaIncidencies (LlistaIncidencies): lista de incidencias de la estación

Además, la clase cuenta con los métodos , lista de vías y lista de incidencias afegirIncidencia y eliminarIncidencia que manejan la gestión de incidencias, actualizando el estado de los accesos correspondientes. Tenemos también el método inicialitzaDadesEstacioTren que inicializa la estación con unos datos del enunciado.

Además de estos métodos, tenemos los métodos carregar (para cargar una estación de un archivo) y guardar (para guardar nuestra estación en un archivo). Para poder utilizar estos métodos en la clases VistaEstacioTren, hemos creado también dos metodos más: recuperarEstacio (que directamente nos recupera una estación de un archivo (nombre del archivo es pedido por pantalla)) y guardarEstacio (nos guarda nuestra estación en un archivo (nombre del archivo pedido por pantalla))

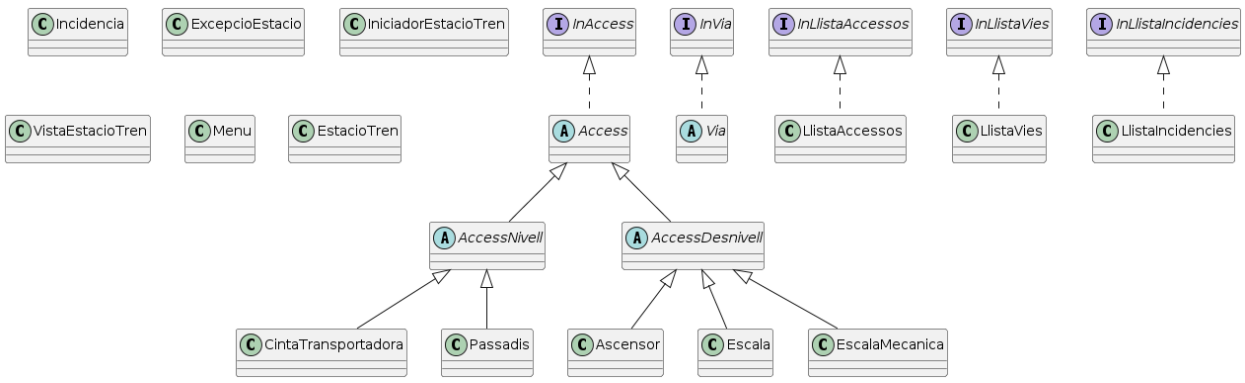
Clase VistaEstacioTren

La clase VistaEstacioTren se encarga de hacer la interacción del usuario con las distintas herramientas de la Estacion de Tren. Para ello presenta un menu al usuario, que debe escoger alguna de las opciones que más tarde la clase hará. Para todo esto, la clase unicamente tiene 1 atributo:

estacioTren (EstacioTren): es la estación que tratamos en el ejercicio

2.2 Diagrama de clases

Ponemos adjunto el diagrama de las clases. Lo hemos hecho teniendo en cuenta las clases que se llaman partiendo de la clase 'main': GestorServeisGimnas.



2.3 Atributos clase EstacioTren

La clase tiene los siguientes atributos:

nomEstacio (String): nombre de la estación

llistaAccessos (LlistaAccessos): lista de accesos de la estación

llistaVies (LlistaVies): lista de vías de la estación

llistaIncidencies (LlistaIncidencies): lista de incidencias de la estación

El nombre es simplemente para identificarla, las 3 listas son para gestionar todos los servicios de la estación, que son los Accesos, las Vías y las Incidencias. Con estas 3 listas nos es suficiente gestionar toda la estación.

2.4 Método isAccessibilitat

El método isAccessibilitat de la clase Access es abstracto porque lo que devuelve depende de cada acceso. La clase Escala devovlerá false mientras que Passadis devolvera true, y como un objecto de la clase Objecto si o si ha de ser de un tipo concreto de Acceso, la clase Acceso no puede devolver un boolean concreto.

2.5 Creieu que l'opció 8 del menú per eliminar una incidència es podria implementar demanant el nom de la via de la incidència, en lloc del número de la incidència que es vol eliminar?

Si, ya que se ha limitado el número de Incidencias por Vía a 1, por tanto si una Vía tiene alguna incidencia, a través de esta podríamos saber exactamente que incidencia tiene.

2.6 Método carregar static

El método carregar es estático para que pueda ser invocado sin la necesidad de crear una instancia de la clase EstacioTren, es decir, es static para poder cargar los datos de una estación de tren sin tener que instanciar un objeto de la clase

3 Pruebas realizadas

Sería difícil poner ahora por palabras todos los intentos y problemas que hemos tenido para llegar hasta el final de la práctica. El primero que no ha venido a la cabeza ha sido el hecho de estar cambiando constantemente el tipo de atributos de las clases, hemos aprendido que quizá lo mejor es mirar como se incializa la estación y sacar de ahí los tipos de los atributos de Access, Via...

Otro problema que nos hemos encontrado ha sido la clase VistaEstacioTren, que tiene un formato muy diferente a las demás (o al menos de la manera en la que lo hemos hecho).

4 Observaciones

Excepto la última parte de cargar y recuperar la estación, la práctica nos ha salido dentro de lo que cabe bastante fluido, a excepción de lo que comentabamos antes, los problemas de tipos de atributos en las clases y la última clase VistaEstacioTren. Ha sido interesante ver como todo lo que haces en un principio de crear los accesos, la clase via, las clases de las listas y todos los métodos más tarde se junta para utilizarse en una misma clase y mostrarselo al usuario.

5 Reparto de trabajo

Álvaro Recolons -¿ Todas las clases de AccessDesnivell (3), la clase LlistaIncidencia y la clase EstacioTren

Pepe Ferrer-Bonsoms -¿ Todas las clases de AccessNivell (2), la clase Via, las clases LlistaAccess y LlistaVies y la clase VistaEstacioTren

El informe lo hemos hecho juntos