Manual de Desarrollo del Proyecto - ISFPP Colectivos Urbanos

Versión: 1.3

Fecha: 05/11

Autores: Grupo 5

Índice

Introducción

Este documento está dirigido a los profesores de la cátedra con el objetivo de facilitar el entendimiento del programa.

Prerrequisitos de Software

Antes de empezar a manipular el programa se recomienda encarecidamente tener las siguientes herramientas.

- Sistema Operativo: Windows 10/11
- JavaFX: paquete correspondiente a la versión 21.0.8
- Log4j: es necesario el uso de los archivos core y api de la biblioteca
- Base de datos: uso de la biblioteca PostgresSQL; postgresql-42.2.17.jar
- Argumentos de la VM:
    - ➔ Para la implementación de JavaFX: --module-path <dirección del archivo> --add-modules javafx.controls,javafx.fxml
      Ejemplo: --module-path C:\Users\Usuario\Downloads\openjfx-21.0.8_windows-x64_bin-sdk\javafx-sdk-21.0.8\lib --add-modules javafx.controls,javafx.fxml
    - ➔ Para la implementación de librerías específicas de JavaFX: --add-modules javafx.controls,javafx.fxml,javafx.graphics,javafx.web

Instalación y Arranque rápido

Clonar el repositorio

git clone https://github.com/moises-kutnich/Colectivo_POO_2025

Importar el proyecto en tu IDE

Importa el proyecto en tu IDE preferido. El IDE cargará automáticamente el código fuente, las librerías y los recursos de configuración.

Patrones de diseño utilizados

Facade (Fachada)

El patrón Facade proporciona una interfaz simplificada y unificada a un conjunto de interfaces en un subsistema complejo. Se utiliza para ocultar la complejidad interna y facilitar la interacción con un sistema más grande

Singleton

El patrón Singleton asegura que una clase solo tenga una instancia y proporciona un punto de acceso global a ella. Es útil para componentes que deben ser únicos en toda la aplicación, como la gestión de configuraciones o conexiones.

DAO (Data Access Object)

El patrón DAO abstrae y encapsula el acceso a la fuente de datos. Se utiliza para separar la lógica de negocio de la lógica de persistencia (cómo se leen o guardan los datos en una base de datos o un archivo).

Factory

El patrón Factory permite la creación de objetos sin especificar la clase exacta del objeto que se va a crear. Delega la lógica de instanciación a una clase "fábrica", permitiendo que la aplicación sea flexible a la hora de decidir qué implementación concreta utilizar.

Model-View-Controller (MVC)

El patrón MVC separa la aplicación en tres componentes interconectados: el Modelo (los datos y la lógica de negocio), la Vista (la interfaz de usuario) y el Controlador (que maneja la entrada del usuario y actualiza la Vista y el Modelo).

Estructuras de datos utilizadas

- List (Listas): Para almacenar colecciones ordenadas. Se usa para guardar la secuencia de paradas en un recorrido, para los resultados de un cálculo y para poblar los menús desplegables de la interfaz.
- Map (Mapas): Para asociar identificadores únicos (claves) a objetos (valores). Es la estructura principal para la caché de datos de la aplicación, permitiendo un acceso instantáneo a las paradas, líneas y tramos por su ID.
- Graph (Grafo): Esta es una estructura conceptual. El proyecto no usa una clase Graph, sino que representa la topología de la red de colectivos implícitamente. Las Paradas son los nodos y los Tramos son las aristas. El algoritmo BFS opera sobre esta estructura de grafo.
- Queue (Colas): Para gestionar elementos en un orden FIFO (Primero en Entrar, Primero en Salir). Se utiliza de forma específica en el algoritmo BFS (Breadth-First Search) “Búsqueda por anchura” para explorar la red de paradas de forma ordenada.

Persistencia de datos

Se emplearon patrones de diseño Factory y DAO para intercambiar entre dos tipos de persistencia de datos:

- Archivos de texto: Para almacenar los datos de la aplicación (paradas, líneas, tramos) como alternativa a la base de datos. Las rutas y el delimitador se configuran desde application.properties.
- Base de datos PostgreSQL: Para almacenar los datos de manera estructurada y realizar consultas SQL. La conexión a esta base de datos se configura en jdbc.properties.

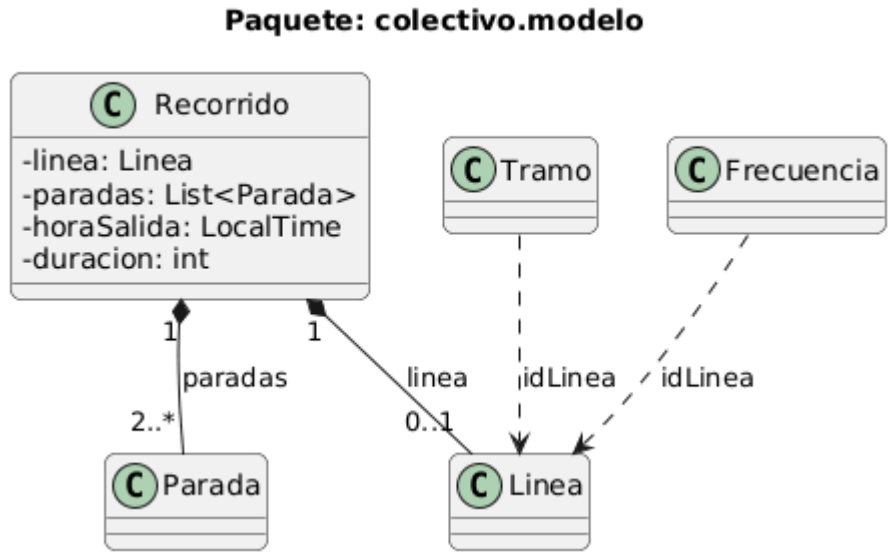
Conclusiones

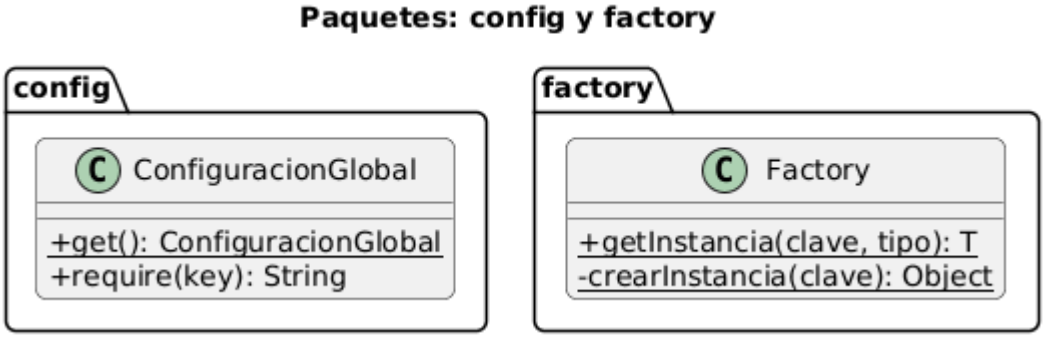
El proyecto tiene una arquitectura en capas muy limpia, que separa la Interfaz (interfaz), la Aplicación (aplicación), el Negocio (negocio) y los Datos (dao).

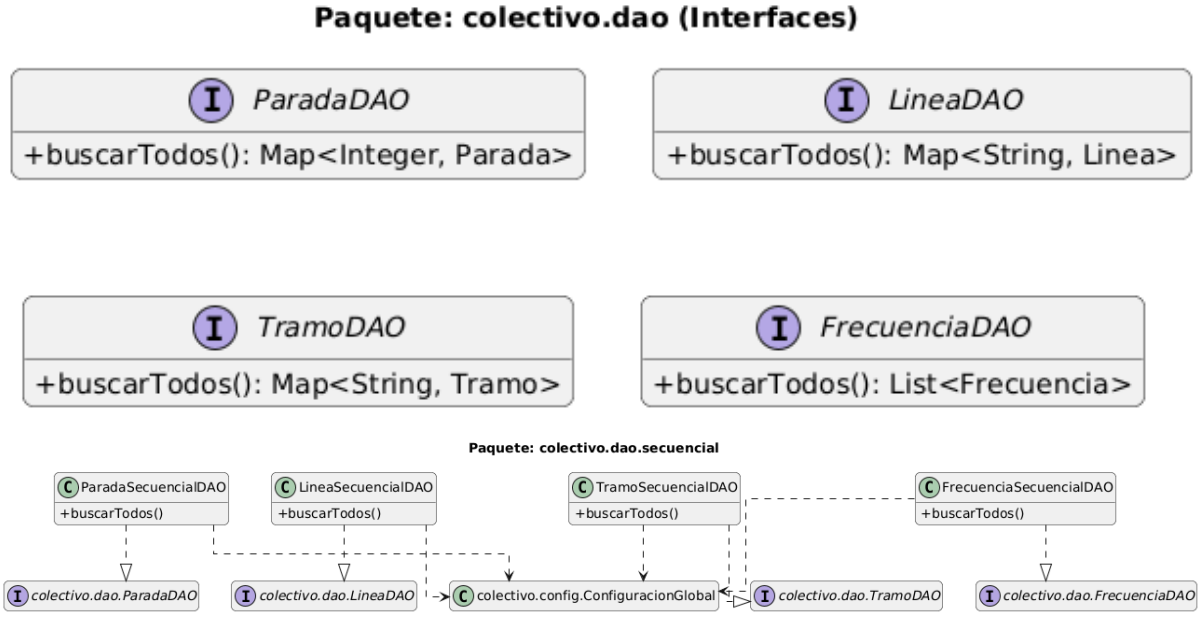
El uso de patrones como DAO y Factory es un gran acierto, ya que permite cambiar la fuente de datos (de archivos TXT a una base de datos) sin tocar el resto del código.

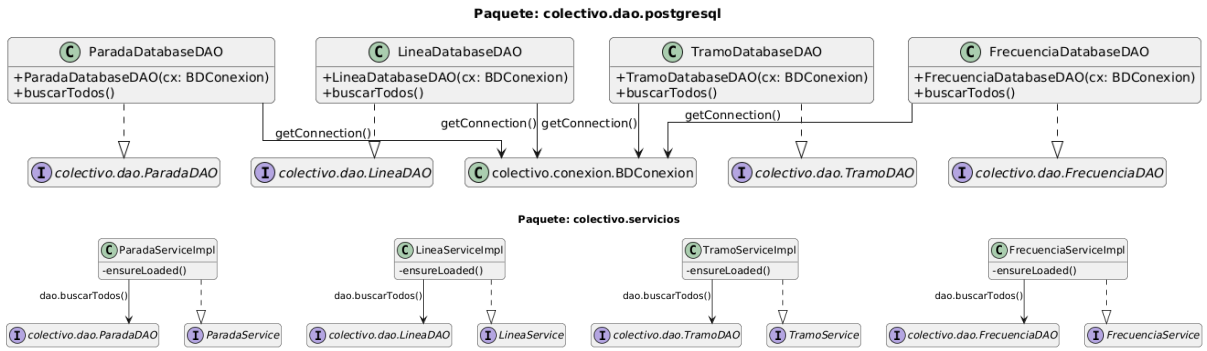
Se usan patrones clave como Facade (Coordinador App) para simplificar la lógica, y Singleton (ConfiguracionGlobal) para la configuración.

UML

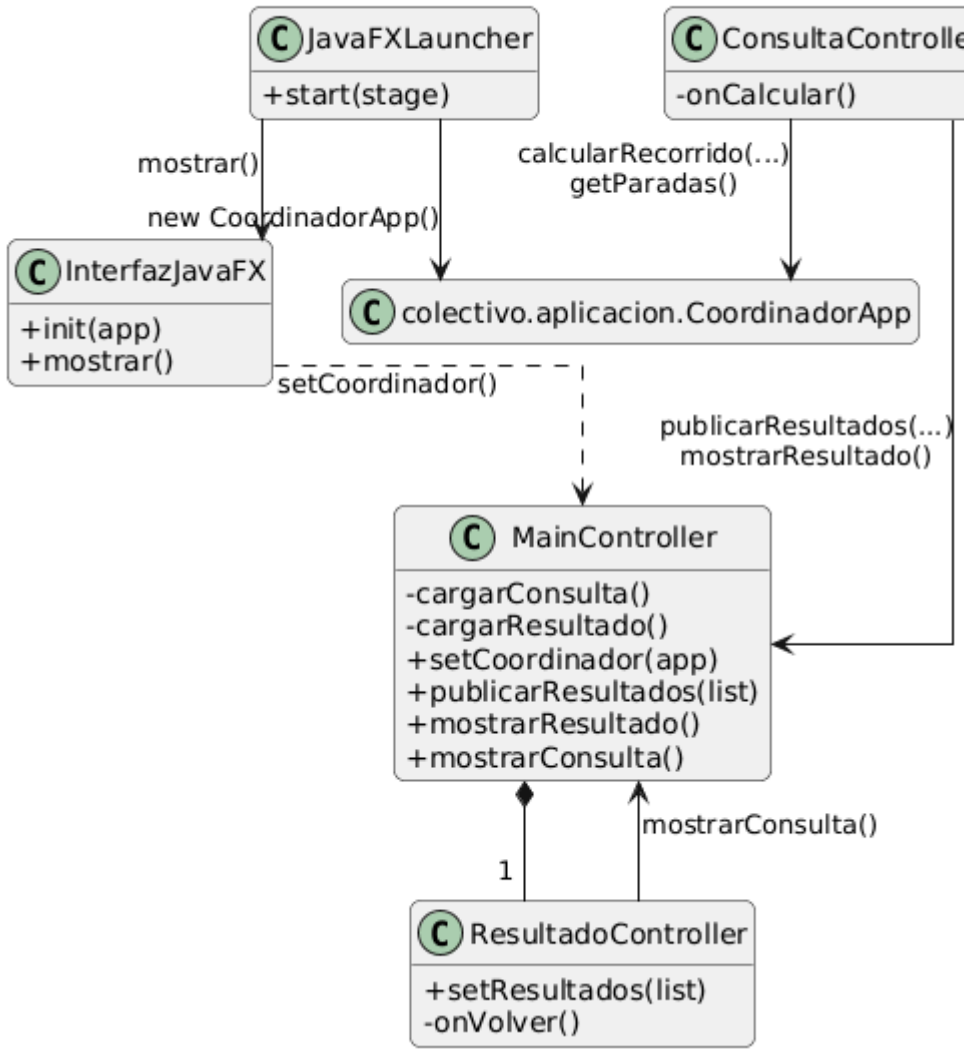

Paquete: colectivo.modelo
Recorrido
-linea: Linea
-paradas: List<Parada>
-horaSalida: LocalTime
-duracion: int
Tramo
Frecuencia
1
1
paradas
linea
idLinea
idLinea
2..*
0..1
Parada
Linea

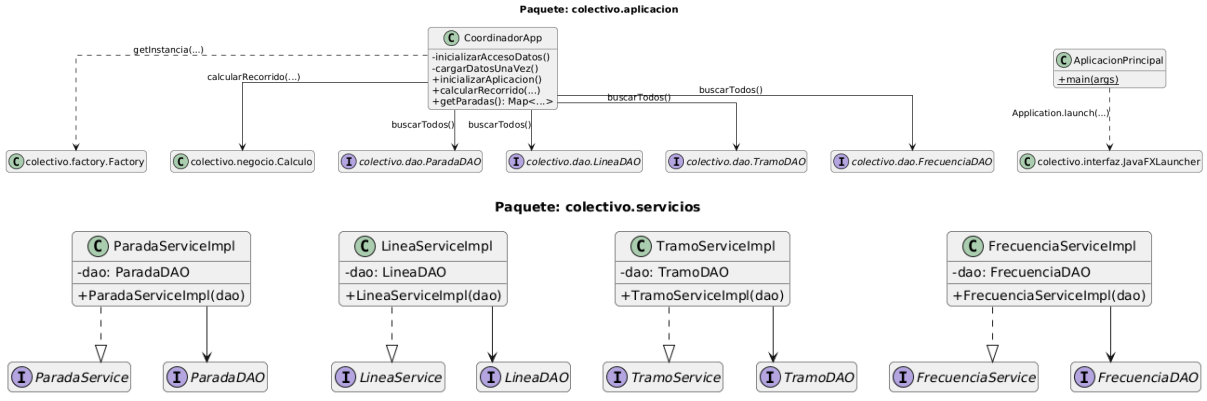

Paquetes: config y factory
config
ConfiguracionGlobal
+get(): ConfiguracionGlobal
+require(key): String
factory
Factory
+getInstancia(clave, tipo): T
-crearInstancia(clave): Object


Paquete: colectivo.dao (Interfaces)
ParadaDAO
+buscarTodos(): Map<Integer, Parada>
LineaDAO
+buscarTodos(): Map<String, Linea>
TramoDAO
+buscarTodos(): Map<String, Tramo>
FrecuenciaDAO
+buscarTodos(): List<Frecuencia>
Paquete: colectivo.dao.secuencial
ParadaSecuencialDAO
+buscarTodos()
LineaSecuencialDAO
+buscarTodos()
TramoSecuencialDAO
+buscarTodos()
FrecuenciaSecuencialDAO
+buscarTodos()
colectivo.dao.ParadaDAO
colectivo.dao.LineaDAO
colectivo.config.ConfiguracionGlobal
colectivo.dao.TramoDAO
colectivo.dao.FrecuenciaDAO



Paquete: colectivo.interfaz




Paquete: colectivo.aplicacion
CoordinadorApp
-inicializarAccesoDatos()
-cargarDatosUnaVez()
+inicializarAplicacion()
+calcularRecorrido(...)
+getParadas(): Map<...>
AplicacionPrincipal
+main(args)
getInstancia(...)
calcularRecorrido(...)
buscarTodos()
buscarTodos()
buscarTodos()
buscarTodos()
Application.launch(...)
colectivo.factory.Factory
colectivo.negocio.Calculo
colectivo.dao.ParadaDAO
colectivo.dao.LineaDAO
colectivo.dao.TramoDAO
colectivo.dao.FrecuenciaDAO
colectivo.interfaz.JavaFXLauncher
Paquete: colectivo.servicios
ParadaServiceImpl
-dao: ParadaDAO
+ParadaServiceImpl(dao)
LineaServiceImpl
-dao: LineaDAO
+LineaServiceImpl(dao)
TramoServiceImpl
-dao: TramoDAO
+TramoServiceImpl(dao)
FrecuenciaServiceImpl
-dao: FrecuenciaDAO
+FrecuenciaServiceImpl(dao)
ParadaService
ParadaDAO
LineaService
LineaDAO
TramoService
TramoDAO
FrecuenciaService
FrecuenciaDAO