Documentación del taller: Sistema de Gestión Rutas S.A.

Integrantes: González María Paz, Retamoso Trotta Máximo, Zanandrea Martin.

Institución: Universidad Nacional de Entre Ríos (FCAD).

Asignatura: Programación II (Licenciatura en Sistemas).

Profesores: Aguirre Juan José, Balbuena Patricia, Silva Elizabeth.

Fecha de entrega: 23 de junio de 2025.

Link del repositorio: https://github.com/maxiretamoso/P2_Taller

Índice

1. Resumen del proyecto

- Este proyecto fue desarrollado en el marco de la asignatura Programación II y consiste en la creación de un sistema de gestión para una empresa llamada “Rutas S.A.”.
- El sistema permite registrar choferes, distintos tipos de vehículos (colectivos y minibuses), ciudades, planificar viajes entre ellas, y mostrar informes de los viajes.
- El proyecto se realizó en Java, utilizando POO (programación orientada a objetos) y funciona por consola.
- La lógica del sistema se organiza en clases interrelacionadas, simulando un flujo real de una empresa de transporte.
- Los datos se gestionan en memoria utilizando estructuras de tipo lista.

2. Objetivo principal

El objetivo principal del sistema es dar solución a una empresa que quiere administrar la planificación de viajes.

En este se deben registrar choferes, vehículos y ciudades, así como permitir la asignación de recursos a los viajes, evitando errores como la superposición de horarios, falta de datos, e ingresos no debidos, entre otros.

3. Diseño del sistema

Para cumplir con los objetivos antes nombrados, se diseñaron las siguientes clases:

- **Persona**: Clase abstracta base que contiene los atributos básicos que tiene una persona. Tiene DNI (long), nombre (String) y apellido (String).

  Sirve como clase base para heredar sus características:

  - **Chofer**: Clase que hereda de Persona y representa a los conductores. Cada chofer tiene asociado una o más categorías y puede tener varios viajes. Además de los atributos heredados, se agregan licencia y categoría.

- **Vehiculo**: Clase abstracta que contiene los atributos generales de un vehículo. Tiene como atributos patente y capacidad (de pasajeros). Cada vehículo puede tener asociado varios viajes.

  Tiene 2 tipos:

  - **Colectivo**: Clase que hereda de Vehículo y representa un vehículo con gran capacidad. Además de los atributos heredados, se agrega que puede tener o no piso doble.
  - **Minibus**: Clase que hereda de Vehículo y representa un vehículo con menor capacidad. Además de los atributos heredados, se agrega que puede tener o no bodega y/o aire acondicionado.

- **Ciudad**: Clase que representa las ciudades. Cada ciudad puede tener un origen y destino asociado a varios viajes. Tiene como atributos nombre (de la ciudad) y provincia.

- **Provincia**: Enumeración que representa las provincias de Argentina. Tiene como valores el nombre de cada provincia.

- **Viaje**: Clase que representa un viaje entre 2 ciudades en una determinada fecha. Cada viaje tiene asociado un chofer y vehículo. Tiene como atributos fecha (del viaje), horario de salida, y horario de llegada.
- **Categoria**: Clase que representa la categoría del chofer. Tiene como atributo el tipo de categoría.
    - **CategoriaTipo**: Enumeración de los tipos de categoría que puede tener un chofer. Tiene como valores MINIBUS y COLECTIVO.
    - **ChoferCategoria**: Clase que representa la relación entre el chofer y categoría. Permite que cada chofer pueda tener más de una categoría. Tiene como atributo la fecha de vencimiento (de la categoría).
- **GestionSistema**: Clase en la cual se centra toda la lógica del sistema de gestión de viajes. Cumple con la función de coordinar la interacción entre las anteriores clases y controlar el flujo del programa desde un menú por consola.

  Sus funciones principales son:
    - **Menú:** Mostrar un menú con opciones para el usuario para realizar distintas acciones como registrar, eliminar, planificar y mostrar informes.
    - **Almacenar:** Los datos se guardan en listas (choferes, vehículos, ciudades, categorías y viajes).
    - **Validar:** Verificar los ingresos del usuario.

  Esta clase simula el uso del sistema por parte del usuario en una situación real.

3.1. Herencia y Polimorfismo

El diseño del sistema hace un uso extensivo de la herencia para modelar relaciones "es un/a":

- Chofer **es una** Persona.
- Colectivo **es un** Vehículo.
- Minibús **es un** Vehículo.

Esto permite reutilizar código y definir comportamientos comunes en las clases base (Persona y Vehículo) mientras se especializan en las subclases.

3.2. Composición y Asociaciones

El sistema emplea la composición para modelar relaciones "tiene un/a" o "contiene un/a":

- Un Viaje **tiene un** Chofer y **tiene un** Vehículo.
- Un Viaje **tiene una** Ciudad de origen y **tiene una** Ciudad de destino.
- Un Chofer **tiene** una lista de ChoferCategorias.
- Una Chofer Categoría **tiene una** Categoría.
- Una Ciudad **tiene una** Provincia.

Estas relaciones permiten construir objetos complejos a partir de objetos más simples y gestionar la lógica de negocio de manera estructurada.

4. Validaciones generales

Para garantizar que los datos ingresados sean válidos y evitar errores durante el uso del sistema, usamos:

- Para las entradas del usuario se utilizan bucles do-while junto con Scanner para reintentar la solicitud hasta que se ingrese un valor válido. También se utilizan estructuras if-else para gestionar escenarios como listas vacías, listas con datos o elementos duplicados, entre otros.
- Para cada ingreso que el usuario quiera hacer ya sea a través del menú o mediante los métodos como registrar, planificar, asociar o informe, se debe validar que el dato ingresado sea del tipo correcto, evitando ingresos inválidos.
- Antes de trabajar en los métodos con las listas (por ejemplo, listas de choferes, vehículos, etc.), se verifica si están vacías o no, mostrando mensajes apropiados según corresponda (Ej: “No hay choferes registrados” / “Ya hay choferes registrados”).
- Al registrar nuevos choferes, vehículos, ciudades o planificar viajes, se verifica si ya existen registros previos. En caso de que sea así, se ofrece al usuario la opción de limpiar la lista (borrar los registros anteriores), de añadir el nuevo registro sin eliminar los anteriores, o volver al menú saliendo de alguno de esos métodos.
- Validación para evitar que los viajes se superpongan en fecha y horario, garantizando la coherencia en la planificación.
- Se verifica si las fechas que se ingresan son válidas y si son fechas pasadas o no, ya sea para el vencimiento de la categoría o para el viaje, comparándola con una fecha que definimos como ‘actual’.

5. Arquitectura del sistema

El sistema fue desarrollado utilizando las siguientes herramientas y recursos:

- **Lenguaje de programación:** Java SE 24.0.1. (compilador y máquina virtual). https://docs.oracle.com/en/java/javase/24/
- **Entorno de desarrollo:** Visual Studio Code (VS Code) con extensiones para Java.
- **Clases y paquetes utilizados del JDK**
    - java.util.Scanner: Entrada de datos desde consola.
    - java.util.ArrayList, java.util.List: Estructuras dinámicas de datos.
    - java.util.InputMismatchException: Manejo de errores de entradas.
    - java.time.LocalDate, java.time.LocalDateTime, java.time.LocalTime: Manejo de fechas y horarios.
    - java.util.HashMap y java.util.Map: para estructuras de datos con clave-valor.
    - java.time.format.DateTimeFormatter y java.time.format.DateTimeParseException: para el manejo de errores en ingreso de fechas y horas.
- **Paquetes y clases propios del proyecto**
    - com.rutassa.Categoria.*: Se encuentra la clase Categoría y CategoriaTipo.
    - com.rutassa.Chofer.*: Se encuentra la clase Persona, Chofer y ChoferCategoria.
    - com.rutassa.Ubicacion.*: Se encuentra la clase Ciudad y Provincia.

- com.rutassa.Vehiculo.* y com.rutassa.Vehiculo.tipoVehiculo.*: Se encuentra la clase Vehículo y los tipoVehiculo.
- com.rutassa.Viaje.*: Se encuentra la clase Viaje.
- com.rutassa.Sistema: Se encuentra la clase GestionSistema con la lógica central del sistema y la clase ObjetosPrueba.

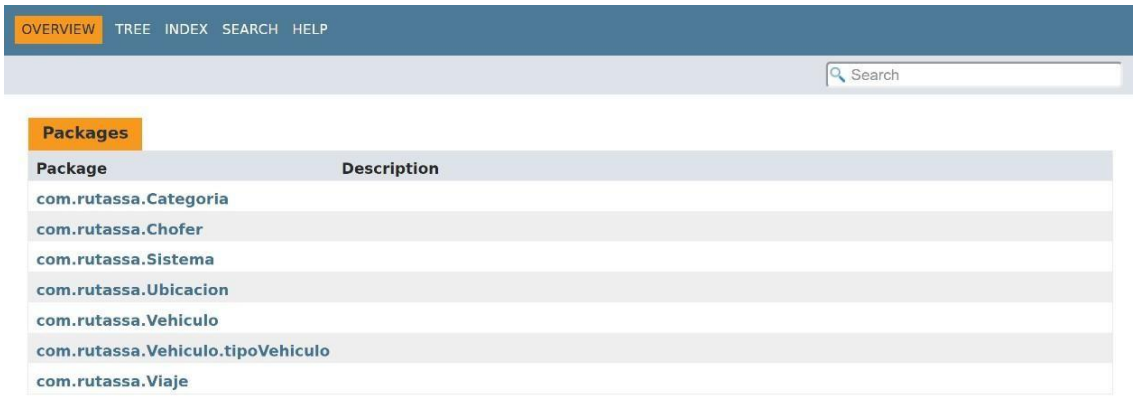
- **Persistencia de datos:** Los datos se mantienen en memoria mediante estructuras de tipo lista. Esto significa que los datos se pierden al finalizar la ejecución del programa.

6. Documentación JavaDoc

La documentación del sistema fue generada mediante la herramienta JavaDoc, a partir de los comentarios que están en el código. Esto nos permite generar una versión en formato HTML, que tiene una descripción ordenada y detallada de las clases, atributos y métodos, ayudando a la comprensión del funcionamiento interno del sistema de Rutas S.A.

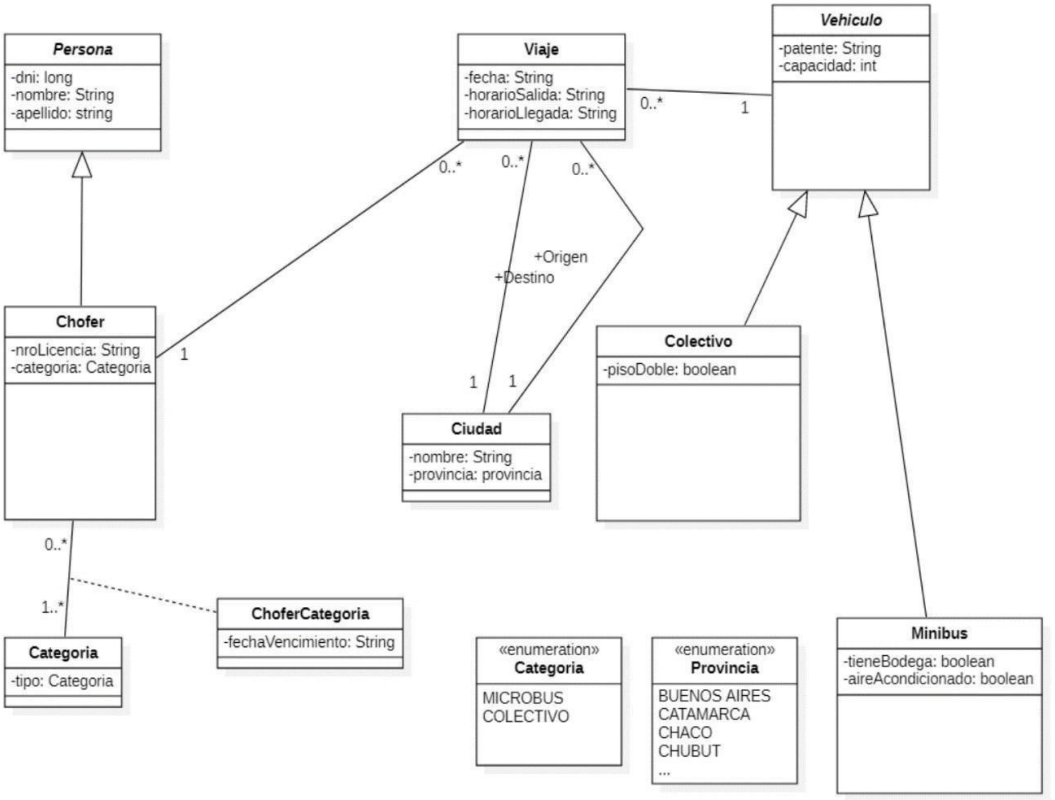
La documentación generada en HTML se encuentra en la carpeta '/docs' del proyecto, y puede abrirse desde el navegador web.

Index.html generado con JavaDoc (presente en la carpeta '/docs'):



7. Diagrama de clases

A continuación, se presenta el diagrama de clases del sistema, que muestra la estructura general y las relaciones entre los objetos:



8. menú del sistema

A continuación, se presenta el menú principal del sistema, mediante el cual el usuario puede interactuar por consola. Dependiendo de la operación que seleccione, podrá realizar distintas operaciones como se muestra en la imagen:

```
------------------------------------------
       Sistema de gestion Rutas S.A.
------------------------------------------
0. Para salir ingrese 0
1. Registrar choferes
2. Registrar vehiculos
3. Registrar ciudades
4. Planificar viajes entre ciudades
5. Asociar un vehiculo y un chofer a cada viaje
6. Mostrar los viajes programados con información detallada
7. Informe detallado de viajes que tiene para realizar un colectivo determinado
8. Informe de cantidad de viajes ya realizados por cada chofer de colectivos
------------------------------------------------------------------------------
Seleccione una opcion:
```