UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES

TRABAJO FINAL
RED DE TRANSPORTE URBANO

DOCENTE: Lic. Rosalia Lopez Montalvo

INTEGRANTES:

Univ. Guzmán Luna Alessandro Kael

Univ. Zenteno Carrillo Abran

Univ. Pacchi Romero Bryan Alejandro

Univ. Rondo Pahe Bryan Roger

Univ. Mamani Clavijo Omar

CARRERA: INFORMÁTICA

MATERIA: INF-121 Programación 2

PARALELO: “A”

FECHA: 24-07-2025

La Paz - Bolivia

2025

RED DE TRANSPORTE URBANO

1. RESUMEN DEL PROYECTO

1.1. Descripción General

El sistema Red de Transporte Urbano es una aplicación desarrollada en Java que implementa una solución integral para la gestión y administración del transporte público urbano. El proyecto aplica principios fundamentales de la Programación Orientada a Objetos mediante un diseño arquitectural robusto que modela las entidades reales de una red de transporte: rutas, estaciones, buses y horarios.

1.2. ¿Qué problema soluciona?

El sistema aborda la problemática de la gestión desorganizada del transporte público urbano, donde tradicionalmente existe:

- Falta de centralización en la administración de rutas y recursos de transporte.
- Dificultades en la asignación eficiente de buses a rutas específicas según demanda y capacidad.
- Carencia de control integrado sobre horarios y frecuencias de servicio.
- Ausencia de herramientas para la planificación y optimización de la red de transporte.
- Información fragmentada que impide la toma de decisiones operativas efectivas.

1.3. ¿Qué funcionalidades tiene?

El sistema proporciona las siguientes funcionalidades principales:
Gestión de Red Principal:

- Creación y administración de la red de transporte urbano
- Control centralizado del número de rutas operativas
- Visualización integral de la estructura de la red

1.3.1. Administración de Rutas:

- Creación de rutas con identificación única.
- Gestión de estaciones asociadas a cada ruta.
- Asignación dinámica de buses según capacidad requerida.
- Control de recursos por ruta (estaciones y buses).

1.3.2. Gestión de Flota de Transporte:

- Registro de buses con especificaciones técnicas completas.
- Administración de horarios múltiples por unidad de transporte.

- Control de capacidad y año de fabricación de cada vehículo.

1.3.3. Control de Operaciones:

- Programación de horarios de salida y llegada.
- Seguimiento de rutas y estaciones.
- Gestión de la información operativa integral.

1.4. Tecnologías usadas:

1.4.1. Lenguaje de Programación:

Java: Lenguaje principal para el desarrollo completo del sistema

1.4.2. IDE y Herramientas de Desarrollo:

IDE Java (NetBeans, IntelliJ IDEA o Eclipse): Para desarrollo y compilación
StarUML/Draw.io: Para diseño y modelado de diagramas UML

1.4.3. Librerías y Frameworks:

Java Standard Library: Para estructuras de datos y operaciones básicas
JavaFX o Swing: Para desarrollo de interfaces gráficas funcionales
JDBC (Java Database Connectivity): Para conectividad con base de datos

1.4.4. Persistencia y Base de Datos:

MySQL: Sistema de gestión de bases de datos relacionales
Archivos .sql: Para respaldo y restauración de datos
JDBC Driver: Para establecer conexión Java-MySQL

1.4.5. Control de Versiones:

GitHub: Repositorio para control de versiones y colaboración del equipo

1.4.6. Herramientas de Modelado:

UML: Para diseño de diagramas de clases y arquitectura del sistema

2. OBJETIVOS

2.1. Objetivo General:

Diseñar e implementar un sistema orientado a objetos que aplique principios de programación, estructuras de datos, y patrones de diseño para la gestión eficiente de una red de transporte urbano.

2.2. Objetivos Específicos:

2.2.1. Aplicación de Conceptos Fundamentales de POO:

Implementar herencia a través de la jerarquía Auto-Bus, demostrando reutilización de código y especialización de comportamientos
Aplicar polimorfismo mediante la sobreescritura de métodos en las clases del sistema, permitiendo comportamientos específicos según el tipo de objeto
Gestionar excepciones personalizadas para casos específicos del dominio como RutaNoEncontradaException, CapacidadExcedidaException, HorarioInvalidoException
Utilizar estructuras genéricas a través de la clase Ruta<T1, T2> para proporcionar flexibilidad y reutilización de código type-safe

2.2.2. Implementación de Patrones de Diseño:

Aplicar el patrón Singleton para garantizar una única instancia de la conexión a base de datos o configuración del sistema
Utilizar el patrón Factory Method para la creación centralizada de objetos Bus, Estacion y Horario sin exponer la lógica de instanciación
Implementar el patrón Strategy para diferentes algoritmos de asignación de buses a rutas o cálculos de capacidad
Considerar el patrón Observer para notificaciones de cambios en el estado de rutas o disponibilidad de buses

2.2.3. Desarrollo de Persistencia y Funcionalidad:

Implementar persistencia de datos mediante conexión JDBC a base de datos MySQL para almacenamiento permanente de información
Desarrollar una interfaz gráfica funcional utilizando JavaFX o Swing que permita interacción intuitiva con el sistema
Crear operaciones CRUD completas para todas las entidades del sistema (RedTransporte, Ruta, Bus, Estacion, Horario)
Implementar validaciones robustas de integridad de datos y manejo de errores

2.2.4. Simulación de Interacción Funcional:

Desarrollar casos de uso reales que simulen la operación diaria de una red de transporte urbano
Crear escenarios de prueba con datos representativos de rutas, estaciones, buses y horarios
Implementar funcionalidades de consulta que permitan a los usuarios obtener información sobre rutas, horarios y disponibilidad
Simular operaciones administrativas como asignación de buses, creación de nuevas rutas y modificación de horarios.
}

3. ANÁLISIS DEL PROBLEMA

El sistema de transporte urbano en muchas ciudades enfrenta varios desafíos: información dispersa, horarios poco confiables, y rutas difíciles de seguir para nuevos usuarios o turistas. En la actualidad, la planificación del viaje suele depender del conocimiento previo del usuario, sin apoyo de un sistema automatizado que brinde asistencia en tiempo real o sugerencias claras.

Los operadores de transporte también requieren un sistema que les permita gestionar y planificar eficientemente sus recursos (buses, personal, recorridos) y evitar problemas como superposición de horarios o mala asignación de buses.

3.1. Descripción del contexto

El proyecto simula una ciudad con una red de transporte compuesta por múltiples estaciones distribuidas geográficamente, rutas que las conectan, y una flota de buses que recorre dichas rutas. Cada estación puede pertenecer a varias rutas y un bus puede ser asignado a una ruta en determinados horarios.

El contexto del desarrollo está centrado en la representación digital de esta red, desde una perspectiva administrativa y de consulta, utilizando programación orientada a objetos para lograr una estructura clara, mantenible y escalable.

3.1.1. Requisitos Funcionales.-

- El sistema debe permitir agregar nuevas rutas a la red de transporte.
- El sistema debe permitir registrar buses dentro de una ruta específica.
- El sistema debe permitir agregar estaciones a una ruta determinada.
- El sistema debe mostrar toda la información de una ruta, incluyendo buses, estaciones y horarios.
- El sistema debe permitir modificar o consultar el nombre de la red de transporte.

3.1.2. Requisitos no funcionales.-

- El sistema debe estar implementado utilizando el lenguaje de programación Java con orientación a objetos.
- La aplicación debe ser fácil de mantener y expandir (por ejemplo, agregando nuevas entidades como trenes o tranvías en el futuro).
- El sistema debe ser usable por un operador con conocimientos básicos en informática, sin necesidad de formación especializada.
-

3.1.3. Casos de uso.-

- **Caso de Uso: Agregar Ruta**

El usuario introduce el nombre de la nueva ruta y la cantidad de buses y estaciones. El sistema crea y almacena la ruta en la red de transporte.

- **Caso de Uso: Registrar Bus**

El usuario selecciona una ruta e ingresa los datos del bus (capacidad, año). El sistema lo añade a la lista de buses de la ruta.

- **Caso de Uso: Añadir Estación**

El usuario selecciona una ruta e ingresa el nombre de la estación. El sistema la agrega al recorrido de la ruta.

- **Caso de Uso: Mostrar Ruta**

El usuario solicita la información de una ruta específica. El sistema muestra su nombre, buses, estaciones y horarios asociados.

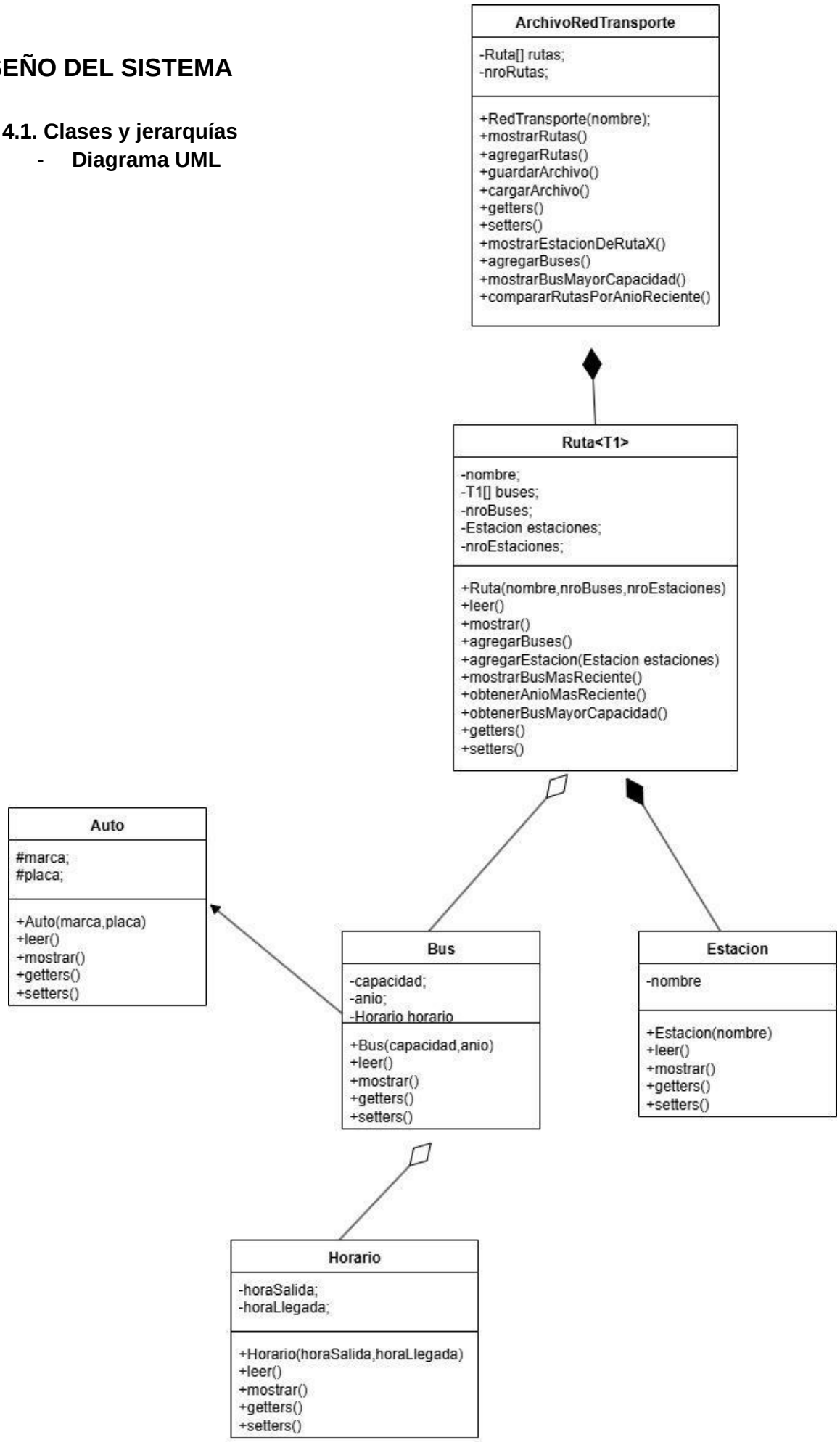
- **Caso de Uso: Modificar nombre de la red**

El usuario edita el nombre de la red de transporte urbana desde el menú o una función específica.

4. DISEÑO DEL SISTEMA

4.1. Clases y jerarquías

- **Diagrama UML**



}

- **Tabla de Clases**

Clase	Atributos Principales	Métodos Clave
ArchivoRedTransporte	nroRutas, Ruta rutas[ ]	mostrarRutas(), agregarRutas(), guardarArchivo(), cargarArchivo(), getters(), setters(), mostrarEstacionDeRutaX(), agregarBuses(), mostrarBusMayorCapacidad(), compararRutasPorAnioReciente()
Ruta <T1>	nombre, nroEstaciones, Estaciones estaciones, T1 buses[ ], nroBuses	leer(), mostrar(), agregarEstacion(Estacion estaciones), agregarBuses(Bus b), getters(), setters(), mostrarBusMasReciente(), obtenerAnioMasReciente(), obtenerBusMayorCapacidad()
Estacion	nombre	leer(), mostrar(), getters(), setters()
Bus	capacidad, anio, nroHorarios, Horario horario	leer(), mostrar(), agregarHorario(Horario h), getters(), setters()
Auto	marca, placa	leer(), mostrar(), getters(), setters()
Horario	horaSalida, horaLlegada	leer(), mostrar(), getters(), setters()

4.2. Relaciones (Herencia, Agregación, Composición)

- **Herencia**

  La clase Auto hereda de Bus.
  Ejemplo: Un Auto es un tipo específico de Bus.

- **Composición**

  La clase RedTransporte compone una lista de objetos Ruta <T1>.
  Es decir, si se elimina una red, también desaparecen sus rutas asociadas.

Igualmente, Ruta compone a Estación, ya que estas no existen fuera de una ruta.

- **Agregación**
  Ruta tiene una relación de agregación con Bus, ya que los buses pueden estar compartidos entre rutas y no dependen exclusivamente de una.
  Bus está compuesto por un objeto Horario. Si el bus deja de existir, también su horario asociado.

Interacción entre clases

- **ArchivoRedTransporte → Ruta<T1>**
  Tipo: Composición
  Cardinalidad: 1 a muchos (una red tiene muchas rutas)
  Descripción:
  La clase RedTransporte contiene un arreglo de objetos Ruta. Las rutas son parte esencial de la red; si se elimina la red, también desaparecen sus rutas.

- **Ruta<T1> → Estación**
  Tipo: Composición genérica
  Cardinalidad: 1 a muchos
  Descripción:
  La clase Ruta contiene un arreglo de Estación que representa las estaciones por donde pasa la ruta. Son parte de la ruta y no existen fuera de ella.

- **Ruta<T1> → Bus**
  Tipo: Agregación genérica
  Cardinalidad: 1 a muchos
  Descripción:
  La ruta agrega buses para su funcionamiento. Un mismo bus podría ser asignado a varias rutas. Por eso, no se destruyen junto con la ruta.

- **Bus → Horario**
  Tipo: Composición
  Cardinalidad: 1 a 1
  Descripción:
  Cada bus tiene exactamente un horario de salida y llegada. El horario depende totalmente del bus. Si se elimina el bus, también se elimina su horario.

- **Bus ← Auto**
  Tipo: Herencia

Descripción:
Auto hereda de Bus, extendiendo su comportamiento y atributos.
Todo Auto es un Bus, pero con atributos adicionales como marca y placa.

5. DESARROLLO

5.1. Estructura del proyecto

El proyecto está organizado de la siguiente manera:

- **IDE Utilizado:** Netbeans
- **Nombre del proyecto:** Red12
- **Paquete base:** com.mycompany.red12
- **Estructura de carpetas y archivos:**
    - Red12/
      └── Source Packages/
        └── com.mycompany.red12/
          ├── Auto.java
          ├── Bus.java
          ├── Estacion.java
          ├── Horario.java
          ├── Red12.java
          ├── RedTransporte.java
          └── Ruta.java
- **Clases principales:**
    - **RedTransporte.java**: Clase principal que contiene el conjunto de rutas. Maneja la lógica general del sistema.
    - **Ruta<T>**: Clase genérica que representa una ruta del sistema, asociada a buses y estaciones.
    - **Estacion.java:** Define las estaciones que forman parte de las rutas.
    - **Bus.java**: Clase que modela los vehículos asignados a las rutas.
    - **Auto.java**: Subclase de Bus que incluye atributos específicos como marca y placa.
    - **Horario.java**: Clase que gestiona la hora de salida y llegada de los buses.
    - **Main.java** (o ProyFeriaDeSalud.java, según el nombre final): Clase donde se encuentra el método main para ejecutar el sistema.
- **Características adicionales**
    - Se contempla la posibilidad de extender el sistema a nuevas funcionalidades, como interfaz gráfica con JavaFX/Swing y conexión con base de datos mediante JDBC.
    - Esta estructura refleja claramente los principios de encapsulamiento, herencia, polimorfismo y uso de clases genéricas, fundamentales en el desarrollo de sistemas orientados a objetos.

6. APLICACIÓN DE PATRONES DE DISEÑO

En el desarrollo del sistema de Red de Transporte Urbano se implementaron cuatro patrones de diseño fundamentales que resultaron especialmente adecuados para las características y necesidades específicas de este tipo de sistema.

¿Por qué estos patrones fueron adecuados para el sistema de transporte urbano?

Los patrones seleccionados responden a desafíos específicos del dominio de transporte público:

- **Gestión centralizada de datos:** Una red de transporte requiere un control unificado de todas las rutas, horarios y vehículos para evitar inconsistencias y garantizar la integridad de la información.
- **Diversidad de vehículos:** Los sistemas de transporte manejan diferentes tipos de vehículos (buses, microbuses, trolebuses) que comparten características básicas pero tienen particularidades específicas.
- **Algoritmos variables:** Las operaciones como búsqueda de vehículos, cálculo de tiempos y comparaciones requieren flexibilidad para adaptarse a diferentes criterios sin modificar el código base.
- **Interfaz reactiva:** La interfaz gráfica debe responder automáticamente a los cambios en el sistema para mantener la información actualizada en tiempo real.

Los patrones implementados proporcionan una arquitectura robusta, mantenible y extensible que facilita tanto el desarrollo actual como futuras ampliaciones del sistema.

6.1. Patrón(es) aplicados

Patrón	Rol en el sistema	Clases involucradas
Singleton	Garantiza una sola instancia de la gestión de la red de transporte y centraliza el acceso a los datos	**RedTrasnporte**
Factory Method	Crea objetos de transporte sin exponer la lógica de creación específica	**Auto**, **Bus** (constructor especializado)
Strategy	Permite intercambiar comportamientos como cálculo de duración de viaje y algoritmos de búsqueda	**Horario** (calcularDuracion), **Ruta** (obtenerBusMayorCapacidad, obtenerAnioBusMasReciente)
Observer	Para notificar cambios en el estado de las rutas cuando se agregan buses o se modifican horarios	**RedTransporte**, **Ruta**, **Menu** (interfaz gráfica)

6.2. Justificación

- Se eligió Singleton para centralizar la gestión de la red de transporte urbano. Esto garantiza que solo exista una instancia de RedTransporte que maneje todas las rutas del sistema, evitando inconsistencias en los datos y proporcionando un punto único de acceso.
- Se usó Strategy para separar los diferentes algoritmos de cálculo y búsqueda. Por ejemplo, el cálculo de duración en Horario.calcularDuracion() maneja diferentes escenarios (viajes que cruzan medianoche), y los métodos en Ruta implementan diferentes estrategias de búsqueda sin modificar la estructura principal.
- El patrón Factory Method permitió desacoplar la creación de objetos de transporte (Auto, Bus, Estacion) del código cliente. Los constructores especializados actúan como factories que crean instancias apropiadas según el contexto.
- Se aplicó Observer para manejar la comunicación entre la interfaz gráfica (Menu) y el modelo de datos (RedTransporte, Ruta). Cuando se realizan cambios en las rutas, la interfaz se actualiza automáticamente para reflejar estos cambios.

6.3. Diagrama y ejemplo de uso

Ejemplo de implementación del patrón Strategy:

```java
// Strategy para cálculo de duración de viaje
public class Horario implements Serializable {
    private double horaSalida;
    private double horaLlegada;

    // Strategy method - maneja diferentes escenarios de cálculo
    public double calcularDuracion() {
        double duracion = horaLlegada - horaSalida;
        if (duracion < 0) {
            duracion += 24; // Strategy para viajes que cruzan medianoche
        }
        return duracion;
    }
}
```

```java
// Strategy para diferentes tipos de búsqueda en Ruta
public class Ruta implements Serializable {
    // Strategy para búsqueda por capacidad máxima
    public Bus obtenerBusMayorCapacidad() {
        if (nroBuses == 0) return null;
        Bus mayor = buses[0];
        for (int i = 1; i < nroBuses; i++) {
            if (buses[i].getCapacidad() > mayor.getCapacidad()) {
                mayor = buses[i];
            }
        }
        return mayor;
    }


    // Strategy para búsqueda por año más reciente
    public int obtenerAnioBusMasReciente() {
        if (nroBuses == 0) return -1;
        int maxAnio = buses[0].getAnio();
        for (int i = 1; i < nroBuses; i++) {
            if (buses[i].getAnio() > maxAnio) {
                maxAnio = buses[i].getAnio();
            }
        }
        return maxAnio;
    }
}
```

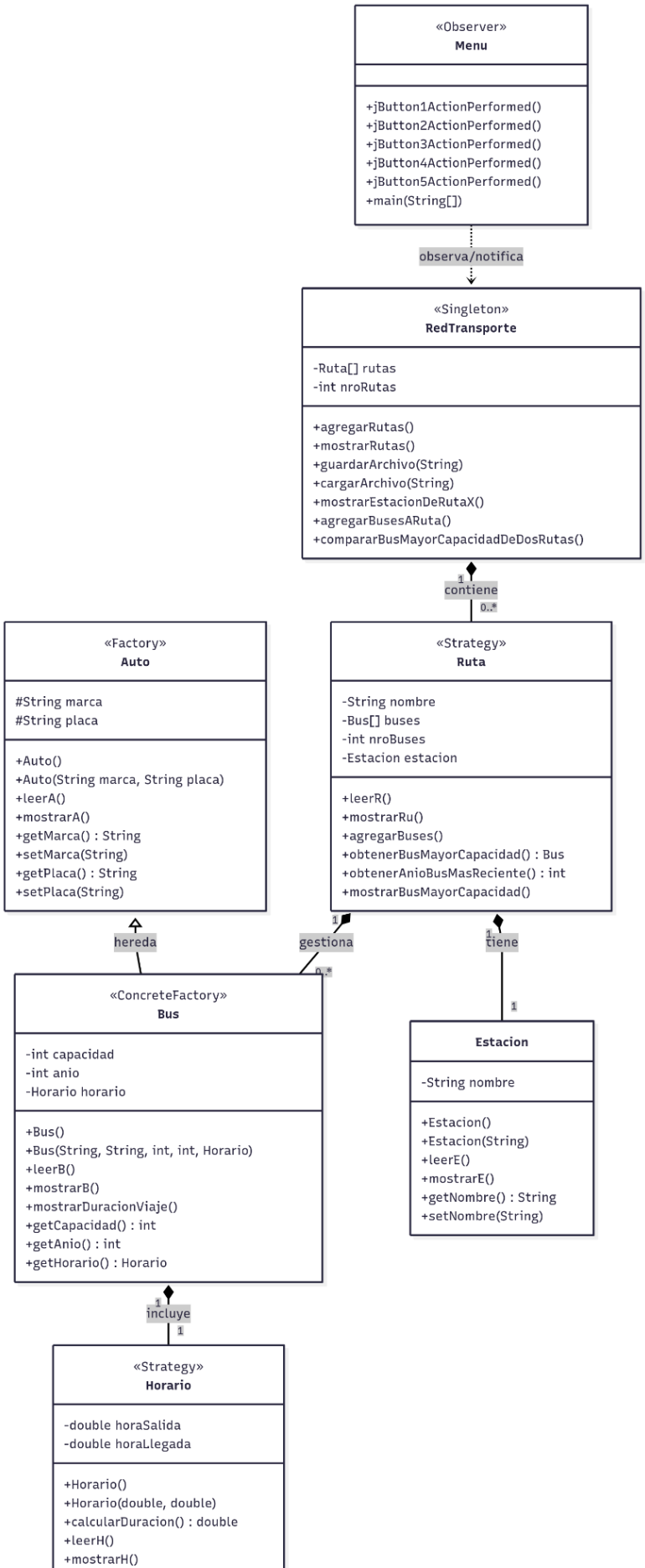
Ejemplo de implementación del patrón Factory Method:

```java
// Factory Method en la clase Auto
public class Auto implements Serializable {
    protected String marca;
    protected String placa;

    // Factory method - constructor base
    public Auto() {
        this.marca = "";
        this.placa = "";
    }

    // Factory method especializado
    public Auto(String marca, String placa) {
        this.marca = marca;
        this.placa = placa;
    }
}
```

Diagrama UML de Patrón de Diseño:



El diagrama ilustra cómo:

- **RedTransporte** implementa Singleton para gestión centralizada
- **Auto/Bus** implementan Factory Method para creación de vehículos
- **Ruta y Horario** implementan Strategy para algoritmos intercambiables
- **Menu** actúa como Observer de cambios en el sistema

Los patrones implementados facilitan la extensibilidad del sistema, permiten agregar nuevos tipos de vehículos sin modificar código existente, y proporcionan algoritmos intercambiables para diferentes operaciones de búsqueda y cálculo en el contexto de transporte urbano.

7. PERSISTENCIA DE DATOS

1.-Descripción del formato usado:

El proyecto utiliza el formato .dat, específicamente el archivo rutas.dat, para guardar la información de las rutas. Este archivo almacena objetos Java serializados en formato binario. Aunque no es legible directamente por humanos, permite guardar estructuras complejas como listas de objetos (ArrayList<Ruta>) y mantener los datos de forma persistente entre ejecuciones del programa.

2.-Clase encargada de lectura/escritura:

La clase que gestiona la lectura y escritura de datos es RedTransporte.java. Utiliza los flujos ObjectOutputStream para guardar datos en rutas.dat y ObjectInputStream para leerlos. Esto permite almacenar y recuperar automáticamente todas las rutas del sistema.

3.-Ejemplo de archivo generado:

El archivo generado se llama rutas.dat. Aunque su contenido es binario (no visible en texto plano), conceptualmente contiene una lista como esta:
Ruta{id="R1", nombre="Villa Fátima - Obrajes", estaciones=[...], horarios=[...]}
Ruta{id="R2", nombre="Pampahasi - Centro", estaciones=[...], horarios=[...]}

8. PRUEBAS Y VALIDACIÓN

8.1. Casos de Prueba Realizados

- **Casos de Prueba en Consola**

1) Agregar Rutas

```
 Men? del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci?n de la ruta por nombre
6. A?adir m?s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a?o m?s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci?n de viaje de cada bus
0. Salir
Opci?n: 1
Ingrese el numero de rutas que desea a?adir
2
Ingrese el nombre de la ruta:
LaPaz
Ingrese el numero de buses que desea insertar:
1
Ingrese los datos del bus
Ingrese la marca
Honda
Ingrese la placa
456BFG
Ingrese la capacidad
100
Ingrese el Anio
2000
Ingrese el horario de Salida
12,15
Ingrese la hora de llegada
16,00
Ingrese el nombre de la estacion:
ZonaZur
Ingrese el nombre de la ruta:
Ceja
Ingrese el numero de buses que desea insertar:
1
Ingrese los datos del bus
Ingrese la marca
Toyota
Ingrese la placa
789LPO
Ingrese la capacidad
25
Ingrese el Anio
2025
Ingrese el horario de Salida
6,00
Ingrese la hora de llegada
10,50
Ingrese el nombre de la estacion:
Azul
```

2) Mostrar Rutas

```
 Men  del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci n de la ruta por nombre
6. A adir m s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a o m s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci n de viaje de cada bus
0. Salir
Opci n: 2
PUMA KATARI
Las rutas son:
Nombre de la ruta: LaPaz nroBuses: 1
Mostrando el bus
Marca: Honda Placa: 456BFG
Capacidad: 100 Anio: 2000
Hora Salida: 12.15 Hora Llegada: 16.0
Estacion
Nombre: ZonaZur
Nombre de la ruta: Ceja nroBuses: 1
Mostrando el bus
Marca: Toyota Placa: 789LPO
Capacidad: 25 Anio: 2025
Hora Salida: 6.0 Hora Llegada: 10.5
Estacion
Nombre: Azul
```

3) Guardar en Archivo y Cargar desde Archivo

```
 Men  del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci n de la ruta por nombre
6. A adir m s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a o m s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci n de viaje de cada bus
0. Salir
Opci n: 3
Rutas guardadas correctamente

 Men  del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci n de la ruta por nombre
6. A adir m s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a o m s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci n de viaje de cada bus
0. Salir
Opci n: 4
Rutas cargados correctamente.
```

4) **Volver a Iniciar el programa, cargar desde archivo y mostrar las Rutas que fueron creadas anteriormente(Persistencia)**

```
Opci□n: 4
Rutas cargados correctamente.

 Men□ del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci□n de la ruta por nombre
6. A□adir m□s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a□o m□s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci□n de viaje de cada bus
0. Salir
Opci□n: 2
PUMA KATARI
Las rutas son:
Nombre de la ruta: LaPaz nroBuses: 1
Mostrando el bus
Marca: Honda Placa: 456BFG
Capacidad: 100 Anio: 2000
Hora Salida: 12.15 Hora Llegada: 16.0
Estacion
Nombre: ZonaZur
Nombre de la ruta: Ceja nroBuses: 1
Mostrando el bus
Marca: Toyota Placa: 789LPO
Capacidad: 25 Anio: 2025
Hora Salida: 6.0 Hora Llegada: 10.5
Estacion
Nombre: Azul
```

5) **Mostrar Estación de la ruta por nombre**

```
PUMA KATARI
Las rutas son:
Nombre de la ruta: LaPaz nroBuses: 2
Mostrando el bus
Marca: Honda Placa: 456BFG
Capacidad: 100 Anio: 2000
Hora Salida: 12.15 Hora Llegada: 16.0
Mostrando el bus
Marca: Ferrari Placa: 459QPR
Capacidad: 25 Anio: 2055
Hora Salida: 8.0 Hora Llegada: 9.5
Estacion
Nombre: ZonaZur
Nombre de la ruta: Ceja nroBuses: 1
Mostrando el bus
Marca: Toyota Placa: 789LPO
Capacidad: 25 Anio: 2025
Hora Salida: 6.0 Hora Llegada: 10.5
Estacion
Nombre: Azul

 Men□ del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci□n de la ruta por nombre
6. A□adir m□s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a□o m□s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci□n de viaje de cada bus
0. Salir
Opci□n: 5
Ingrese el nombre de la ruta a buscar: Ceja
Nombre de la estaci□n: Azul
```

6) Añadir más Buses a una Ruta

```
 Men  del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci n de la ruta por nombre
6. A adir m s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a o m s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci n de viaje de cada bus
0. Salir
Opci n: 6
Ingrese el nombre de la ruta a la que desea a adir buses: LaPaz
Cu ntos buses desea agregar?: 1
Ingrese los datos del bus
Ingrese la marca
Ferrari
Ingrese la placa
459QPR
Ingrese la capacidad
25
Ingrese el Anio
2055
Ingrese el horario de Salida
8,00
Ingrese la hora de llegada
9,50

 Men  del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci n de la ruta por nombre
6. A adir m s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a o m s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci n de viaje de cada bus
0. Salir
Opci n: 2
PUMA KATARI
Las rutas son:
Nombre de la ruta: LaPaz nroBuses: 2
Mostrando el bus
Marca: Honda Placa: 456BFG
Capacidad: 100 Anio: 2000
Hora Salida: 12.15 Hora Llegada: 16.0
Mostrando el bus
Marca: Ferrari Placa: 459QPR
Capacidad: 25 Anio: 2055
Hora Salida: 8.0 Hora Llegada: 9.5
```

7) Mostrar Bus con mayor capacidad de una ruta

```
 Men? del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci?n de la ruta por nombre
6. A?adir m?s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a?o m?s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci?n de viaje de cada bus
0. Salir
Opci?n: 7
Ingrese el nombre de la ruta: LaPaz
 Bus con mayor capacidad en la ruta 'LaPaz':
Mostrando el bus
Marca: Honda Placa: 456BFG
Capacidad: 100 Anio: 2000
Hora Salida: 12.15 Hora Llegada: 16.0
```

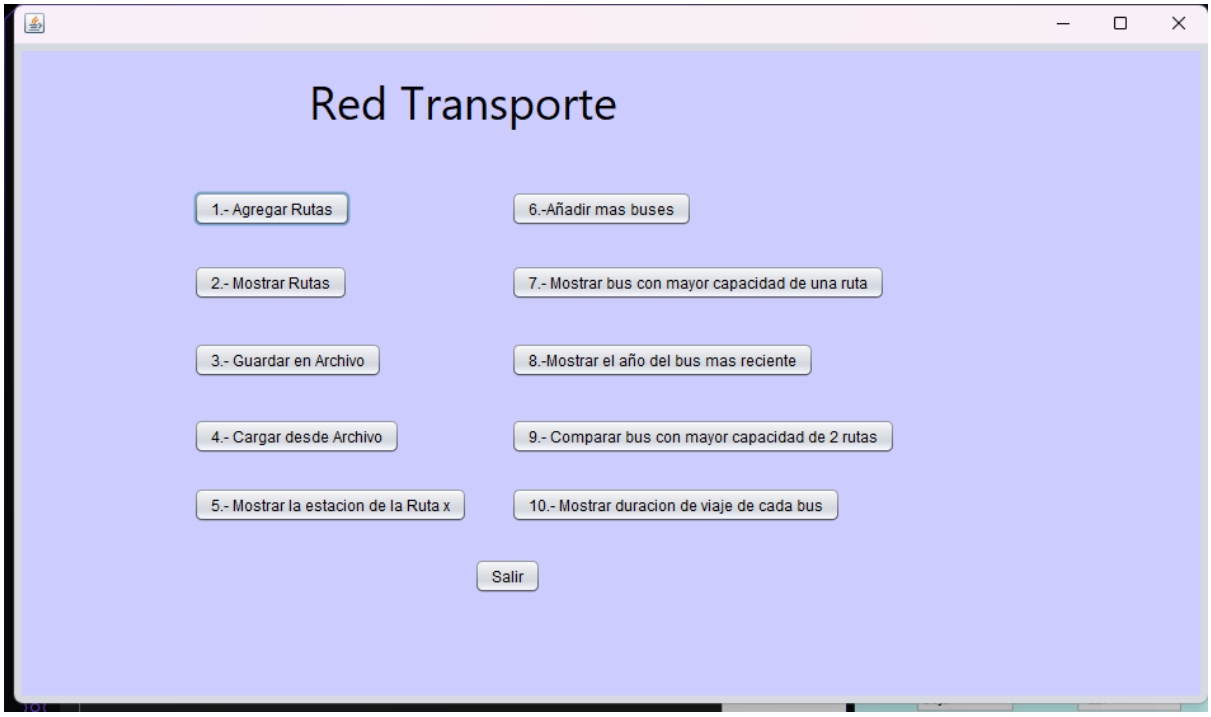
8) Comparar dos rutas por año más reciente de sus buses

```
 Men? del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci?n de la ruta por nombre
6. A?adir m?s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a?o m?s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci?n de viaje de cada bus
0. Salir
Opci?n: 8
Ingrese el nombre de la primera ruta: LaPaz
Ingrese el nombre de la segunda ruta: Ceja
? Comparaci?n de a?os m?s recientes:
LaPaz: A?o m?s reciente = 2055
Ceja: A?o m?s reciente = 2025
?? LaPaz tiene el bus m?s reciente.
```

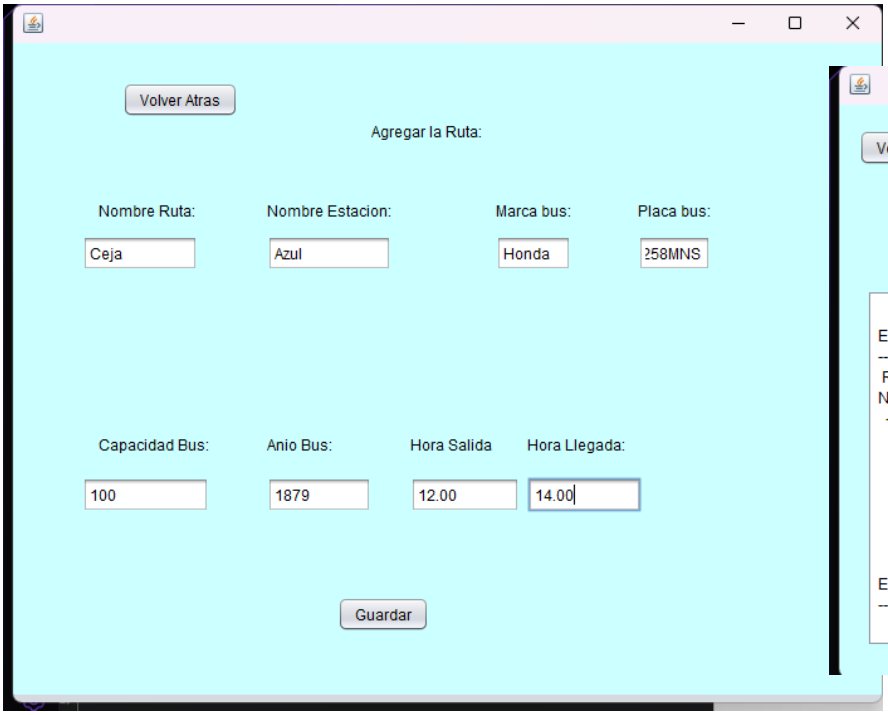
9) Comparar bus con mayor capacidad entre 2 rutas

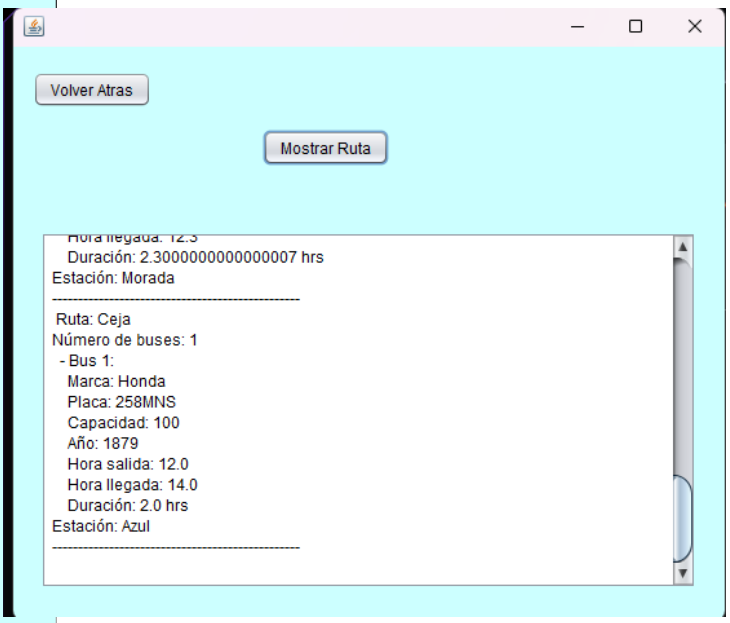
```
 Men? del Sistema de Registro
1. Agregar Ruta
2. Mostrar Rutas
3. Guardar en archivo
4. Cargar desde archivo
5. Mostrar estaci?n de la ruta por nombre
6. A?adir m?s buses a una ruta
7. Mostrar bus con mayor capacidad de una ruta
8. Comparar dos rutas por a?o m?s reciente de sus buses
9. Comparar bus con mayor capacidad entre 2 rutas
10. Mostrar duraci?n de viaje de cada bus
0. Salir
Opci?n: 9
Ingrese el nombre de la primera ruta: LaPaz
Ingrese el nombre de la segunda ruta: Ceja
 Bus con mayor capacidad en LaPaz:
Mostrando el bus
Marca: Honda Placa: 456BFG
Capacidad: 100 Anio: 2000
Hora Salida: 12.15 Hora Llegada: 16.0
Bus con mayor capacidad en Ceja:
Mostrando el bus
```

- **Casos de Prueba en Interfaz Gráfica**

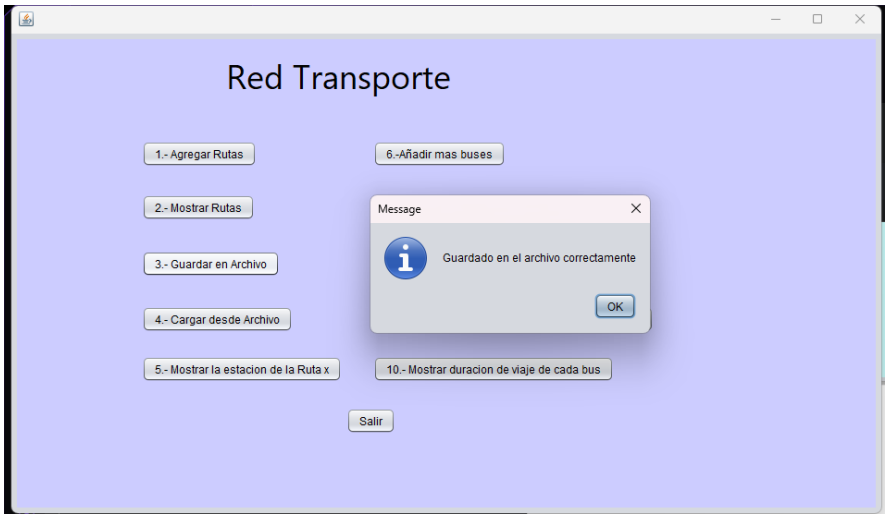


1) Agregar Rutas

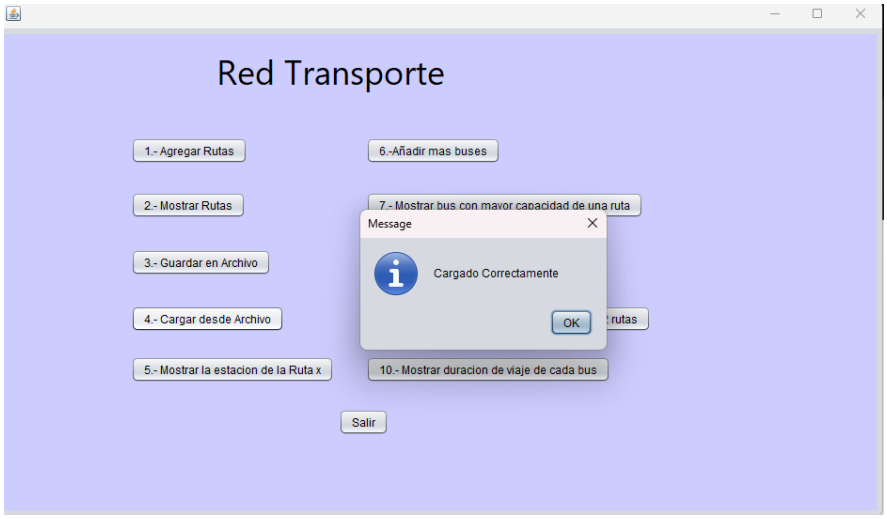




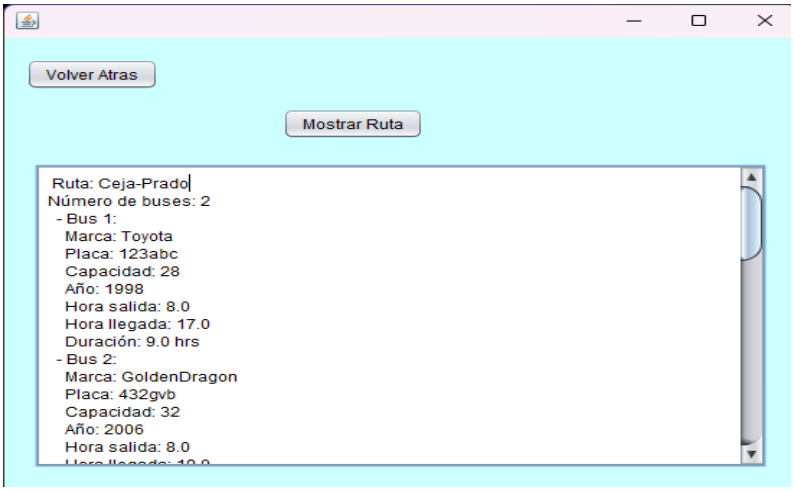
2) **Guardar en Archivo**



3) **Cargar desde Archivo**



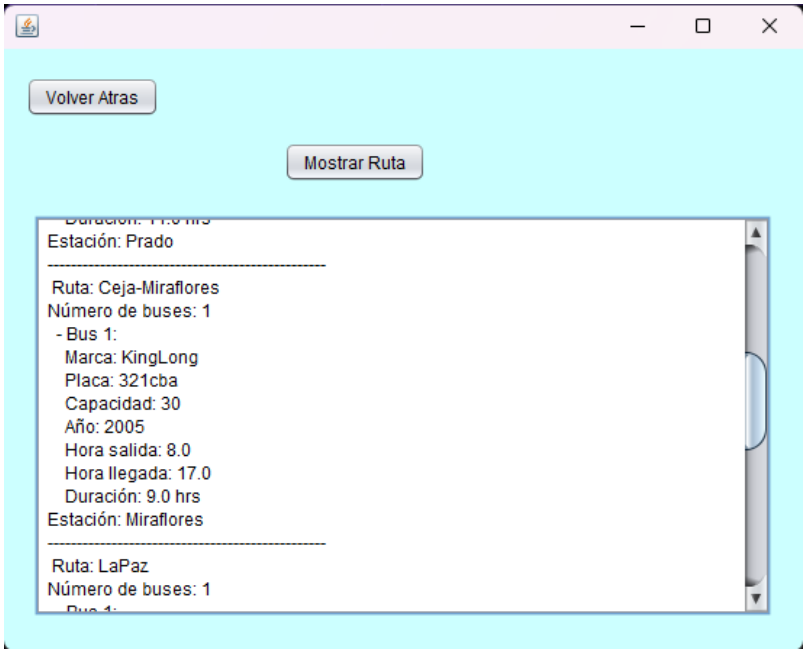
4) **Volver a Iniciar el programa, cargar desde archivo y mostrar las Rutas que fueron creadas anteriormente(Persistencia)**




Volver Atras
Mostrar Ruta
Estación: Miraflores

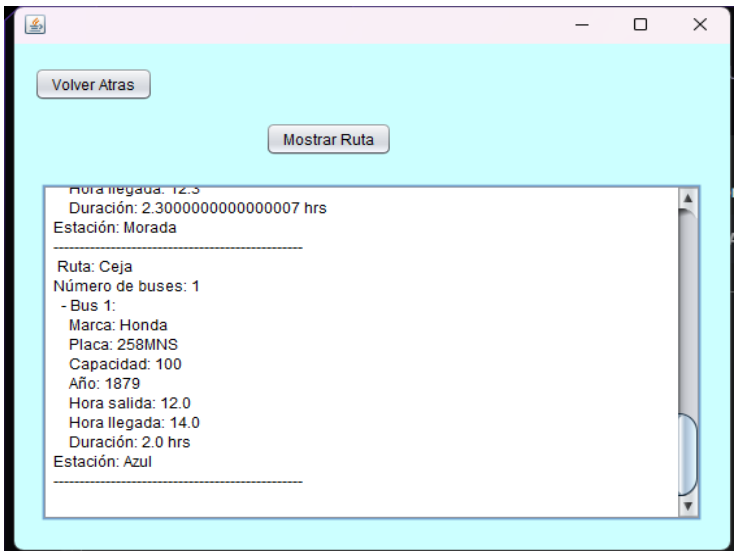
Ruta: LaPaz
Número de buses: 1
- Bus 1:
Marca: Toyota
Placa: 456ABC
Capacidad: 50
Año: 1999
Hora salida: 10.0
Hora llegada: 12.3
Duración: 2.3000000000000007 hrs
Estación: Morada

Ruta: Ceja
Número de buses: 1


Volver Atras
Mostrar Ruta
Estación: Prado

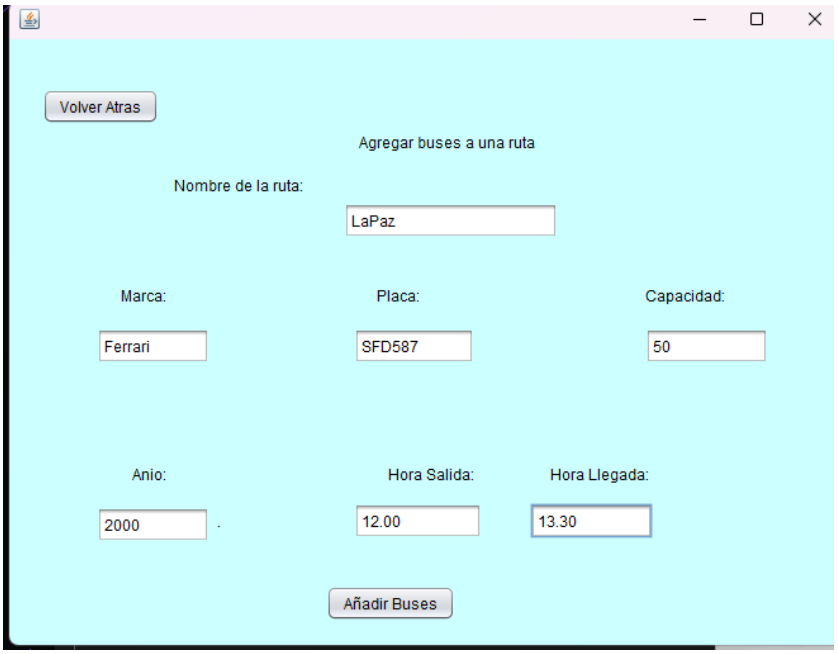
Ruta: Ceja-Miraflores
Número de buses: 1
- Bus 1:
Marca: KingLong
Placa: 321cba
Capacidad: 30
Año: 2005
Hora salida: 8.0
Hora llegada: 17.0
Duración: 9.0 hrs
Estación: Miraflores

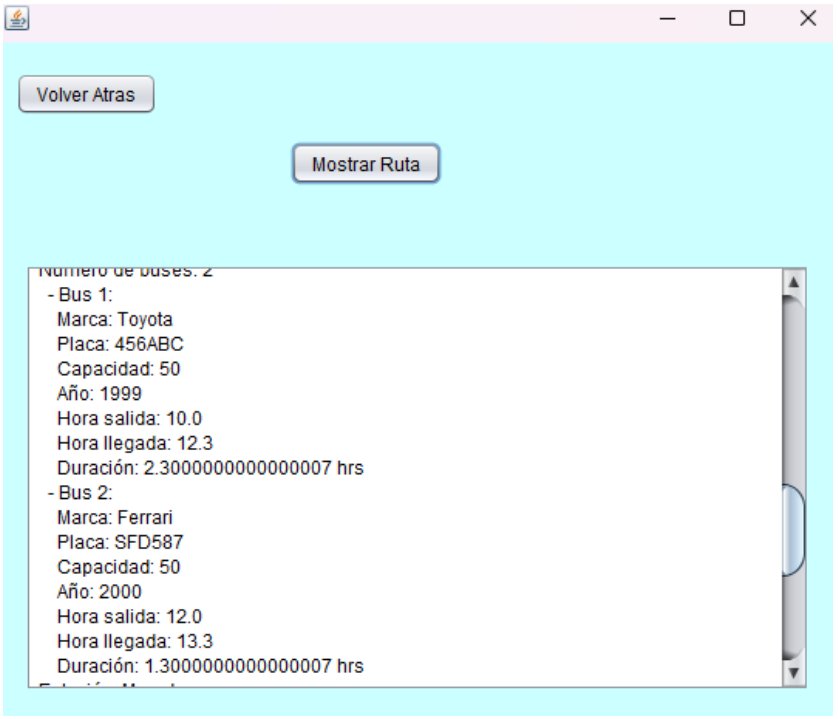
Ruta: LaPaz
Número de buses: 1


Volver Atras
Mostrar Ruta
Duración: 2.3000000000000007 hrs
Estación: Morada

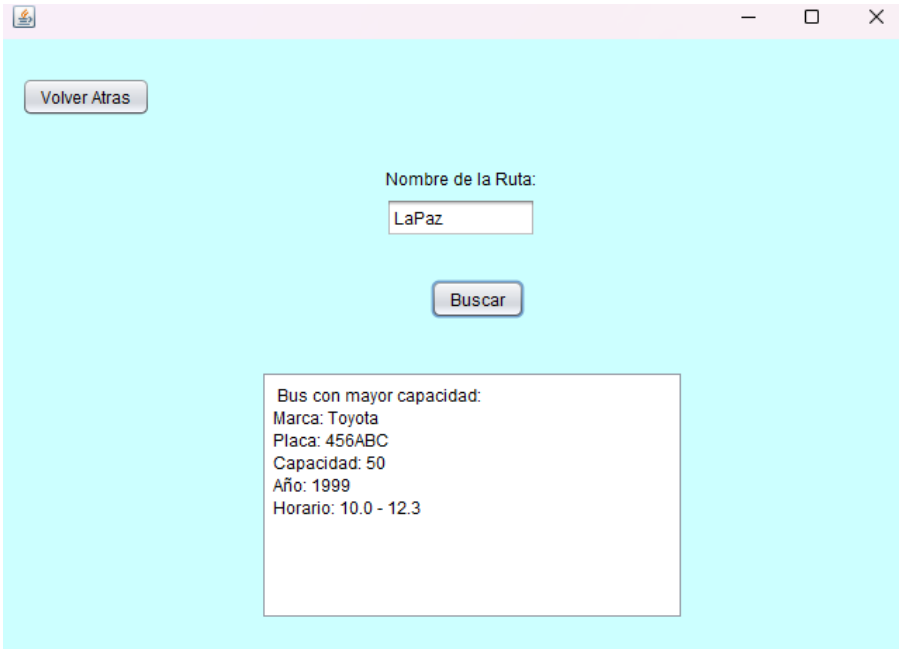
Ruta: Ceja
Número de buses: 1
- Bus 1:
Marca: Honda
Placa: 258MNS
Capacidad: 100
Año: 1879
Hora salida: 12.0
Hora llegada: 14.0
Duración: 2.0 hrs
Estación: Azul
---

5) Añadir Buses a una Ruta

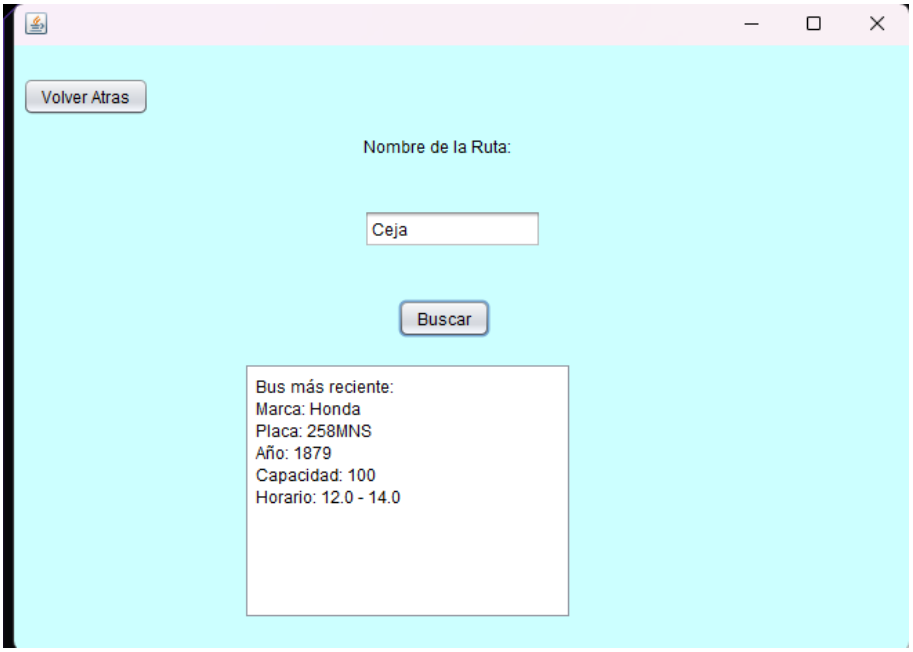




6) Mostrar Bus con mayor capacidad de una ruta



7) Mostrar el año del bus más reciente



9. CONCLUSIONES

El desarrollo del sistema “Red de Transporte Urbano” permitió aplicar de manera práctica los conceptos fundamentales de la Programación Orientada a Objetos, así como el diseño y modelado de software orientado a soluciones reales. Mediante la representación de entidades como rutas, estaciones, buses y horarios, se logró simular una red de transporte urbana funcional, con capacidades de administración, consulta y asignación de recursos.
Durante el proceso, se implementaron estructuras genéricas, manejo de excepciones, jerarquías de clases y patrones de diseño como Singleton, Factory Method y Strategy, lo que contribuyó a una arquitectura robusta, escalable y mantenible. La interacción entre clases fue cuidadosamente diseñada para reflejar relaciones reales como herencia, agregación y composición, consolidando así un modelo de datos coherente y lógico.

- **Reflexión sobre el uso de patrones de diseño**

El uso de patrones de diseño facilitó notablemente el desarrollo del sistema. El patrón Singleton permitió asegurar una única instancia controlada de la red de transporte; Factory Method brindó flexibilidad en la creación de objetos como estaciones o buses; y Strategy permitió implementar comportamientos intercambiables, como distintas formas de calcular rutas o asignar horarios. Estos patrones no solo mejoraron la reutilización de código, sino que promovieron una separación clara de responsabilidades, haciendo el sistema más mantenible y extensible.

- **Ventajas observadas en la estructura del sistema**

Entre las principales ventajas de la estructura del sistema destacan su modularidad, la claridad de las relaciones entre clases, y la facilidad para añadir nuevas funcionalidades. El uso de genéricos permitió que las rutas puedan ser definidas para distintos tipos de transporte. Además, la integración con bases de datos mediante JDBC y la interfaz gráfica mediante JavaFX o Swing aportaron un alto nivel de funcionalidad y usabilidad.

- **Aplicación de contenidos de la materia**

El proyecto abarcó una amplia gama de contenidos de la materia: encapsulamiento, herencia, polimorfismo, composición y agregación, estructuras genéricas, excepciones, patrones de diseño, persistencia con archivos y bases de datos, y desarrollo de interfaces gráficas. Cada uno de estos conceptos fue puesto en práctica en contextos reales, consolidando el aprendizaje teórico.

- **Aspectos a mejorar o ampliar**

Como mejora futura, se podría integrar un sistema de geolocalización para simular ubicaciones reales de buses y estaciones, así como optimizar el algoritmo de asignación de rutas en función del tráfico o la demanda. También se podría implementar un módulo para usuarios finales que

les permita consultar en tiempo real el estado del transporte. A nivel técnico, sería recomendable aplicar pruebas unitarias y de integración para garantizar aún más la calidad del sistema.
En resumen, el proyecto cumplió sus objetivos propuestos al ofrecer una herramienta de gestión orientada a la mejora de la red de transporte urbano, simulando escenarios reales y brindando una solución técnica alineada a las necesidades de ciudades modernas como La Paz.

10. DISTRIBUCIÓN DE ROLES EN EL EQUIPO

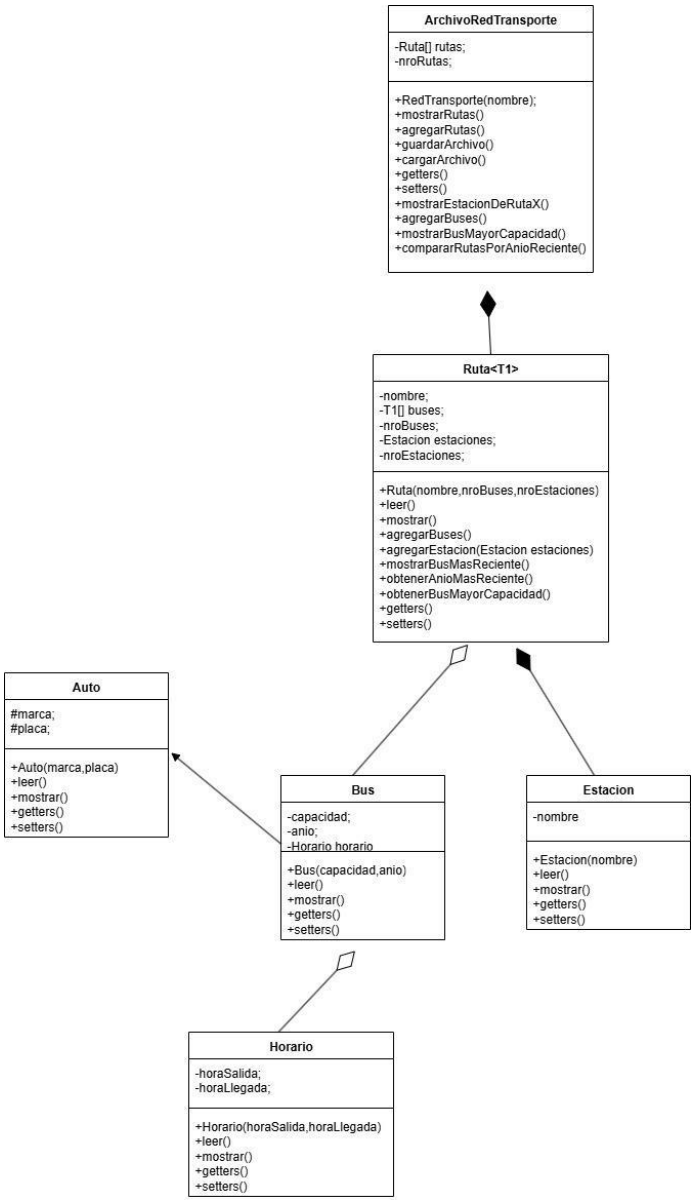
Rol / Integrante	Responsabilidad Principal
Guzmán Luna Alessandro Kael	Diseño del UML, implementación en Java, desarrollo de la interfaz gráfica (JavaFX/Swing) e implementación de la lógica del sistema, así como la elaboración del informe final.
Zenteno Carrillo Abran	Diseño del UML, implementación en Java, desarrollo de la interfaz gráfica (JavaFX/Swing) e implementación de la lógica del sistema, así como la elaboración del informe final.
Pacchi Romero Bryan Alejandro	Diseño del UML, implementación en Java, desarrollo de la interfaz gráfica (JavaFX/Swing) e implementación de la lógica del sistema, así como la elaboración del informe final.
Rondo Pahe Bryan Roger	Diseño del UML, implementación en Java, desarrollo de la interfaz gráfica (JavaFX/Swing) e implementación de la lógica del sistema, así como la elaboración del informe final.
Mamani Clavijo Omar	Diseño del UML, implementación en Java, desarrollo de la interfaz gráfica (JavaFX/Swing) e implementación de la lógica del sistema, así como la elaboración del informe final.

11. ANEXOS

Organización del archivo:

Red12/
└── Source Packages/
    └── com.mycompany.red12/
        ├── Auto.java
        ├── Bus.java
        ├── Estacion.java
        ├── Horario.java
        ├── Red12.java
        ├── RedTransporte.java
        └── Ruta.java

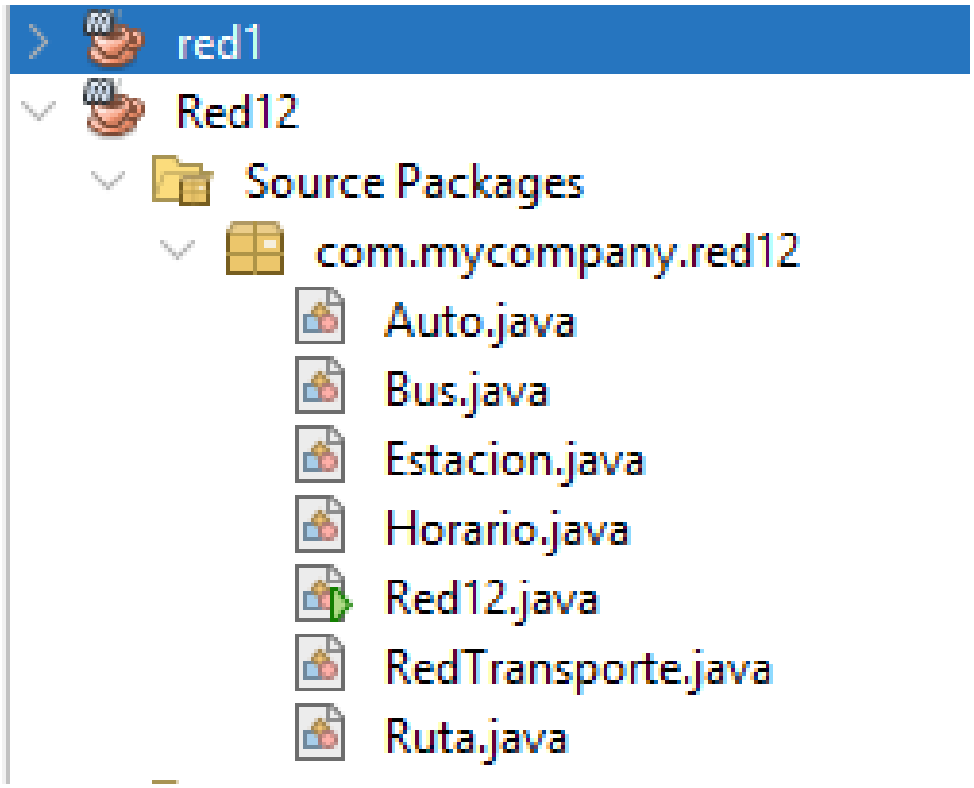
UML:

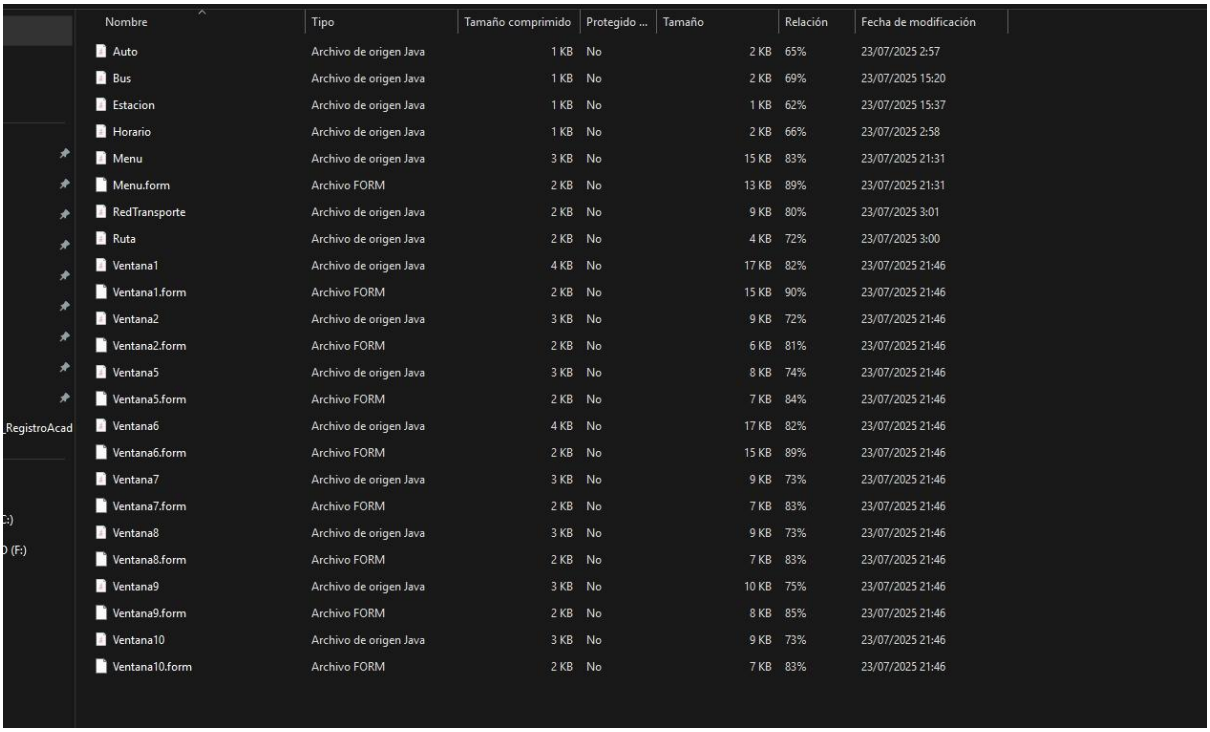


Link de UML:
https://drive.google.com/file/d/1xh1Czg61fX3ywEZdys257PEca2QAg9i5/view?usp=drivesdk

Repositorio de archivos de prueba:
https://github.com/MapachiXD/ProyINF-121-Pruebas.git

Repositorio de archivos oficial:
https://github.com/MapachiXD/ProyInf121.git





Nombre	Tipo	Tamaño comprimido	Protegido ...	Tamaño	Relación	Fecha de modificación
Auto	Archivo de origen Java	1 KB	No	2 KB	65%	23/07/2025 2:57
Bus	Archivo de origen Java	1 KB	No	2 KB	69%	23/07/2025 15:20
Estacion	Archivo de origen Java	1 KB	No	1 KB	62%	23/07/2025 15:37
Horario	Archivo de origen Java	1 KB	No	2 KB	66%	23/07/2025 2:58
Menu	Archivo de origen Java	3 KB	No	15 KB	83%	23/07/2025 21:31
Menu.form	Archivo FORM	2 KB	No	13 KB	89%	23/07/2025 21:31
RedTransporte	Archivo de origen Java	2 KB	No	9 KB	80%	23/07/2025 3:01
Ruta	Archivo de origen Java	2 KB	No	4 KB	72%	23/07/2025 3:00
Ventana1	Archivo de origen Java	4 KB	No	17 KB	82%	23/07/2025 21:46
Ventana1.form	Archivo FORM	2 KB	No	15 KB	90%	23/07/2025 21:46
Ventana2	Archivo de origen Java	3 KB	No	9 KB	72%	23/07/2025 21:46
Ventana2.form	Archivo FORM	2 KB	No	6 KB	81%	23/07/2025 21:46
Ventana5	Archivo de origen Java	3 KB	No	8 KB	74%	23/07/2025 21:46
Ventana5.form	Archivo FORM	2 KB	No	7 KB	84%	23/07/2025 21:46
Ventana6	Archivo de origen Java	4 KB	No	17 KB	82%	23/07/2025 21:46
Ventana6.form	Archivo FORM	2 KB	No	15 KB	89%	23/07/2025 21:46
Ventana7	Archivo de origen Java	3 KB	No	9 KB	73%	23/07/2025 21:46
Ventana7.form	Archivo FORM	2 KB	No	7 KB	83%	23/07/2025 21:46
Ventana8	Archivo de origen Java	3 KB	No	9 KB	73%	23/07/2025 21:46
Ventana8.form	Archivo FORM	2 KB	No	7 KB	83%	23/07/2025 21:46
Ventana9	Archivo de origen Java	3 KB	No	10 KB	75%	23/07/2025 21:46
Ventana9.form	Archivo FORM	2 KB	No	8 KB	85%	23/07/2025 21:46
Ventana10	Archivo de origen Java	3 KB	No	9 KB	73%	23/07/2025 21:46
Ventana10.form	Archivo FORM	2 KB	No	7 KB	83%	23/07/2025 21:46