Pontificia Universidad Católica de Valparaíso

Facultad de Ingeniería

Escuela de Ingeniería informática

Sistema de Gestión de Buses -Avance

Joshua Villavicencio Mesías

German Oses Vargas

Profesor: Claudio Cubillo Figueroa

Clave/Asignatura: INF2236 – Programación Avanzada

Domingo 8 de septiembre de 2024

Índice

Pontificia Universidad Católica de Valparaíso 1

1. Introducción 3

1.1. Objetivos del Sistema 3

2. Descripción General del Sistema 3

2.1. Características Clave 3

3. Diseño del Sistema 4

3.1. Clase Bus 4

3.2. Clase Pasajero 4

4. Lógica de Negocio 8

4.1. Ejemplo de Flujo de Trabajo 8

5. Relación con los Requisitos del SIA 8

5.1 SIA 1.2 EC: Inclusión de Atributos y Métodos en las Clases 8

5.2 SIA 1.3 EC: Asociación entre Clases (Relaciones de Agregación y Composición) 9

5.3 SIA 1.4 EC: Métodos para Modificación y Consulta 10

5.4 SIA 1.5 EC: Gestión de Horarios en Ruta 10

5.5 SIA 1.7 EC: Inicialización de Datos Predeterminados 10

5.6 SIA 1.8 EC: Validación y Manejo de Errores 11

6. Implementación Técnica 11

6.1. Entorno de Desarrollo 11

6.2. Estructura del Código 11

7. Conclusiones 12

7.1. Próximo Avance 12

1. Introducción

Este informe describe el desarrollo de un sistema de gestión de buses que permite la asignación de pasajeros a buses, así como el seguimiento de la capacidad de cada vehículo y los costos de mantenimiento asociados. El sistema está diseñado para un ambiente de transporte de pasajeros, facilitando la administración eficiente de los recursos y proporcionando un control claro sobre el uso de los buses.

1.1. Objetivos del Sistema

- Asignar pasajeros a los buses respetando la capacidad máxima.
- Registrar y gestionar los datos relacionados con los buses, como patente, capacidad, pasajeros actuales y costos de mantención.
- Proveer una interfaz para consultar y administrar los pasajeros asignados a cada bus.

2. Descripción General del Sistema

El sistema implementa una clase Bus que representa los buses de la flota. Cada bus tiene atributos como la patente, la capacidad, los pasajeros actuales, el costo de mantención y un mapa de pasajeros que permite acceder a los detalles de cada uno mediante su identificador único (RUT).

2.1. Características Clave

- **Capacidad Máxima de los Buses**: Cada bus tiene una capacidad fija de 44 pasajeros.
- **Asignación de Pasajeros**: Los pasajeros son asignados a los buses hasta completar la capacidad máxima.
- **Seguimiento de Costos de Mantención**: El sistema permite registrar los costos de mantención de cada bus para un mejor control financiero.
- **Mapa de Pasajeros**: Un mapa (HashMap) asocia el RUT de cada pasajero con sus datos, lo que facilita la búsqueda y gestión.

3. Diseño del Sistema

3.1. Clase Bus

La clase Bus es el componente central del sistema, diseñada para manejar las operaciones de asignación de pasajeros, gestión de la capacidad y almacenamiento de datos relacionados con el mantenimiento del vehículo.

Atributos:

- **String patente**: Identificador del bus.
- **int capacidad**: Capacidad máxima de pasajeros (44 por defecto).
- **int pasajerosActuales**: Cantidad de pasajeros actualmente asignados al bus.
- **double costoMantencion**: Costo de mantención asociado al bus.
- **Map<String, Pasajero> mapaBus**: Mapa que almacena los pasajeros asignados al bus, donde la clave es el RUT del pasajero y el valor es un objeto Pasajero.

Métodos:

- **asignarPasajero(Pasajero pasajero)**: Asigna un pasajero al bus si la capacidad lo permite. Retorna true si la asignación fue exitosa y false si el bus está lleno.
- **getMapaBus(String rut)**: Retorna el objeto Pasajero asociado a un RUT específico.
- **setPatente(String patente)**: Modifica la patente del bus.
- **getCostoMantencion() y setCostoMantencion(double costo)**: Permiten obtener y establecer el costo de mantención.

3.2. Clase Pasajero

El sistema también cuenta con la clase Pasajero, que incluye información relevante de los pasajeros, como su RUT y nombre.

Atributos de Pasajero:

- **String rut**: El RUT (identificación) del pasajero.
- **String nombre**: El nombre del pasajero.
- **List<Tickect>tickets** : (lista de tickets adquiridos por el pasajero).

3.3 Clase Ticket

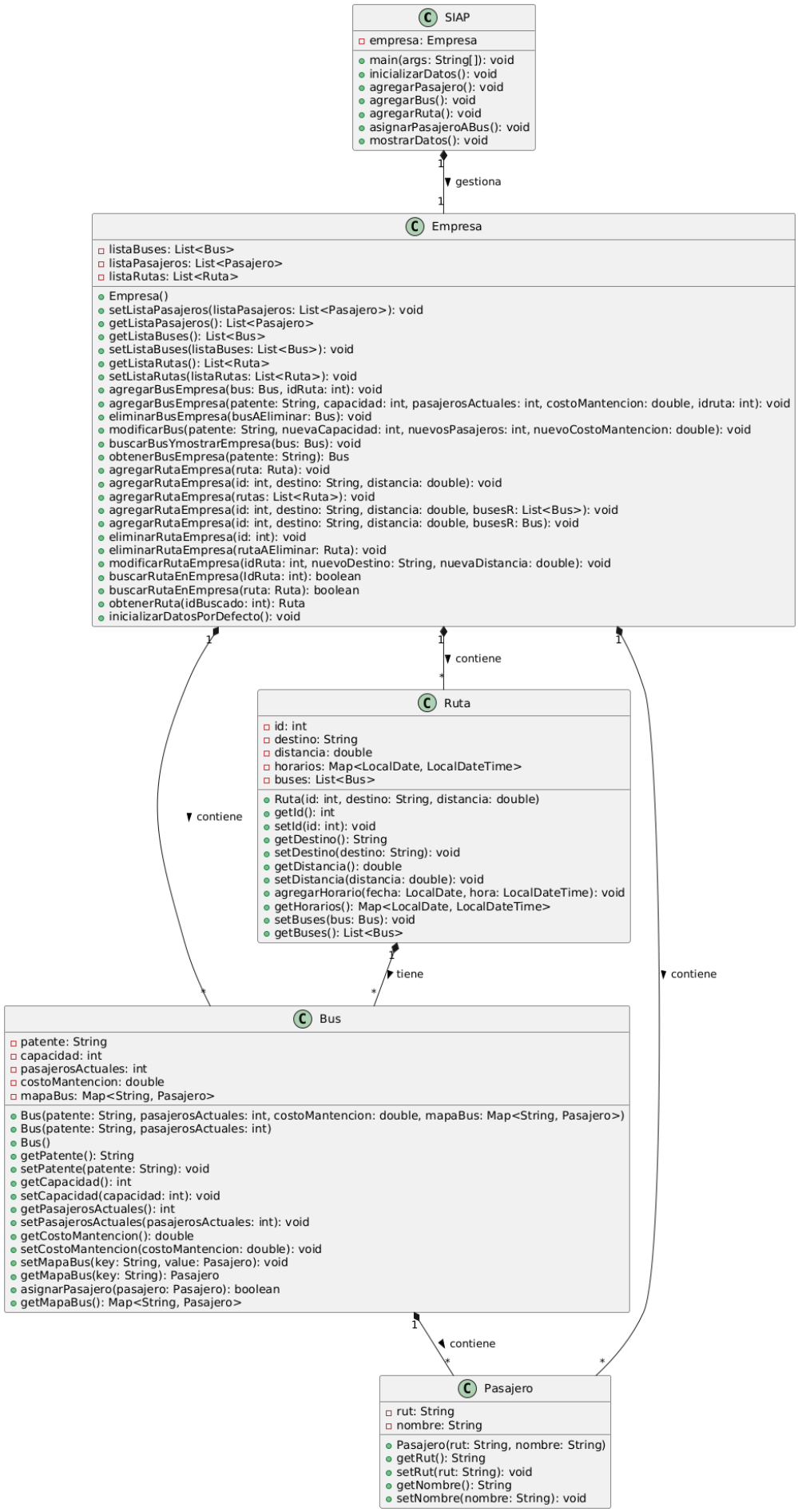
- **Descripción**: Representa un ticket que un pasajero adquiere para un viaje en un bus en una ruta específica.
- **Atributos**:
    - **horaVenta**: La fecha y hora en que se emitió el ticket.
    - **ruta**: La ruta en la que el ticket es válido.
    - **bus**: El bus asignado para el ticket.
    - **pasajero**: El pasajero que adquirió el ticket.
    - **precio**: El precio del ticket.
- **Métodos**:
    - **getPrecio()**: Devuelve el precio del ticket.
    - **getFecha()**: Devuelve la fecha de venta del ticket.
    - **toString()**: Método para representar el ticket como cadena de texto con todos sus detalles.
    - **Métodos**: getters y setters para cada atributo.

3.4 Clase Ruta

- **Descripción**: Representa una ruta en la que operan buses, con destino y distancia especificados, además de una lista de buses y un calendario de horarios.
- **Atributos**:
    - **id**: Identificador único de la ruta.
    - **destino**: El destino final de la ruta.
    - **distancia**: La distancia en kilómetros de la ruta.
    - **buses**: Lista de buses asignados a la ruta.
    - **horariosPorFecha**: Mapa que asocia fechas con listas de horarios de salida.
- **Métodos**:
    - **agregarBusARuta()**: Asigna un bus a la ruta.
    - **agregarHorario()**: Añade un horario a la ruta en una fecha específica.
    - **getBuses()**: Devuelve la lista de buses asignados a la ruta.
    - **Métodos**: getters y setters para todos los atributos.

3.5 Clase Empresa

- **Descripción**: Representa la empresa de transporte que gestiona los buses, rutas, pasajeros y tickets. Esta clase coordina todas las operaciones del sistema.
- **Atributos**:
  - **listaBuses**: Lista de todos los buses que pertenecen a la empresa.
  - **listaPasajeros**: Lista de todos los pasajeros registrados en la empresa.
  - **listaRutas**: Lista de todas las rutas que gestiona la empresa.
- **Métodos**:
  - **agregarBusEmpresa()**: Añade un nuevo bus a la empresa.
  - **eliminarBusEmpresa()**: Elimina un bus de la empresa.
  - **modificarBus()**: Modifica los detalles de un bus existente.
  - **agregarRutaEmpresa()**: Añade una nueva ruta a la empresa.
  - **eliminarRutaEmpresa()**: Elimina una ruta de la empresa.
  - **modificarRutaEmpresa()**: Modifica los detalles de una ruta existente.
  - **calcularViabilidadViaje()**: Evalúa si un viaje es viable económicamente en función de los ingresos y costos.


SIAP
- empresa: Empresa
+ main(args: String[]): void
+ inicializarDatos(): void
+ agregarPasajero(): void
+ agregarBus(): void
+ agregarRuta(): void
+ asignarPasajeroABus(): void
+ mostrarDatos(): void
1
gestiona
1
Empresa
- listaBuses: List<Bus>
- listaPasajeros: List<Pasajero>
- listaRutas: List<Ruta>
+ Empresa()
+ setListaPasajeros(listaPasajeros: List<Pasajero>): void
+ getListaPasajeros(): List<Pasajero>
+ getListaBuses(): List<Bus>
+ setListaBuses(listaBuses: List<Bus>): void
+ getListaRutas(): List<Ruta>
+ setListaRutas(listaRutas: List<Ruta>): void
+ agregarBusEmpresa(bus: Bus, idRuta: int): void
+ agregarBusEmpresa(patente: String, capacidad: int, pasajerosActuales: int, costoMantencion: double, idruta: int): void
+ eliminarBusEmpresa(busAEliminar: Bus): void
+ modificarBus(patente: String, nuevaCapacidad: int, nuevosPasajeros: int, nuevoCostoMantencion: double): void
+ buscarBusYmostrarEmpresa(bus: Bus): void
+ obtenerBusEmpresa(patente: String): Bus
+ agregarRutaEmpresa(ruta: Ruta): void
+ agregarRutaEmpresa(id: int, destino: String, distancia: double): void
+ agregarRutaEmpresa(rutas: List<Ruta>): void
+ agregarRutaEmpresa(id: int, destino: String, distancia: double, busesR: List<Bus>): void
+ agregarRutaEmpresa(id: int, destino: String, distancia: double, busesR: Bus): void
+ eliminarRutaEmpresa(id: int): void
+ eliminarRutaEmpresa(rutaAEliminar: Ruta): void
+ modificarRutaEmpresa(idRuta: int, nuevoDestino: String, nuevaDistancia: double): void
+ buscarRutaEnEmpresa(IdRuta: int): boolean
+ buscarRutaEnEmpresa(ruta: Ruta): boolean
+ obtenerRuta(idBuscado: int): Ruta
+ inicializarDatosPorDefecto(): void
1
1
contiene
*
1
contiene
Ruta
- id: int
- destino: String
- distancia: double
- horarios: Map<LocalDate, LocalDateTime>
- buses: List<Bus>
+ Ruta(id: int, destino: String, distancia: double)
+ getId(): int
+ setId(id: int): void
+ getDestino(): String
+ setDestino(destino: String): void
+ getDistancia(): double
+ setDistancia(distancia: double): void
+ agregarHorario(fecha: LocalDate, hora: LocalDateTime): void
+ getHorarios(): Map<LocalDate, LocalDateTime>
+ setBuses(bus: Bus): void
+ getBuses(): List<Bus>
1
tiene
*
*
contiene
contiene
Bus
- patente: String
- capacidad: int
- pasajerosActuales: int
- costoMantencion: double
- mapaBus: Map<String, Pasajero>
+ Bus(patente: String, pasajerosActuales: int, costoMantencion: double, mapaBus: Map<String, Pasajero>)
+ Bus(patente: String, pasajerosActuales: int)
+ Bus()
+ getPatente(): String
+ setPatente(patente: String): void
+ getCapacidad(): int
+ setCapacidad(capacidad: int): void
+ getPasajerosActuales(): int
+ setPasajerosActuales(pasajerosActuales: int): void
+ getCostoMantencion(): double
+ setCostoMantencion(costoMantencion: double): void
+ setMapaBus(key: String, value: Pasajero): void
+ getMapaBus(key: String): Pasajero
+ asignarPasajero(pasajero: Pasajero): boolean
+ getMapaBus(): Map<String, Pasajero>
1
contiene
*
*
Pasajero
- rut: String
- nombre: String
+ Pasajero(rut: String, nombre: String)
+ getRut(): String
+ setRut(rut: String): void
+ getNombre(): String
+ setNombre(nombre: String): void

4. Lógica de Negocio

El sistema implementa lógica de negocio para garantizar que no se exceda la capacidad del bus. Si un bus está lleno, no se permite la asignación de más pasajeros, y se genera un mensaje de advertencia para informar al usuario.

4.1. Ejemplo de Flujo de Trabajo

1. Un bus es creado con una capacidad fija de 44 pasajeros.
2. Se asigna un pasajero al bus utilizando su RUT como identificador en el mapa.
3. Si el bus alcanza su capacidad, se notifica al sistema que no es posible agregar más pasajeros.
4. El sistema también permite consultar los pasajeros actuales de un bus mediante su RUT.

5. Relación con los Requisitos del SIA

5.1 SIA 1.2 EC: Inclusión de Atributos y Métodos en las Clases

- **Código Correspondiente:**
    - **Bus:**
        - **Atributos: patente, capacidad, pasajerosActuales, costoMantencion, mapaBus.**
        - **Métodos: asignarPasajero(), getMapaBus(), setMapaBus(), entre otros.**
    - **Pasajero:**
        - **Atributos: rut, nombre, tickets (lista de tickets adquiridos por el pasajero).**
        - **Métodos: Getters y setters para rut, nombre y tickets.**
    - **Ruta:**
        - **Atributos: id, destino, distancia, horarios, buses.**
        - **Métodos: agregarHorario(), agregarBusARuta(), entre otros.**

- **Empresa:**
  - **Métodos: Para agregar, eliminar y modificar Bus y Ruta, gestionar Pasajero, entre otros.**
- **SIAP:**
  - **Método: inicializarDatosPorDefecto() para la inicialización de datos predeterminados.**

5.2 SIA 1.3 EC: Asociación entre Clases (Relaciones de Agregación y Composición)

Código Correspondiente:

- **Métodos en Empresa para agregar, eliminar y modificar buses y rutas:**
  - **agregarBusEmpresa()**
  - **eliminarBusEmpresa()**
  - **modificarBus()**
  - **agregarRutaEmpresa()**
  - **eliminarRutaEmpresa()**
  - **modificarRutaEmpresa()**
- **Métodos en Bus para gestionar pasajeros:**
  - **asignarPasajero()**
  - **getMapaBus()**
- **Métodos en Pasajero para gestionar tickets:**
  - **addTicket()**
  - **getTickets()**

5.3 SIA 1.4 EC: Métodos para Modificación y Consulta

- **Código Correspondiente:**
    - **Métodos en Empresa para agregar, eliminar y modificar buses y rutas:**
        - **agregarBusEmpresa()**
        - **eliminarBusEmpresa()**
        - **modificarBus()**
        - **agregarRutaEmpresa()**
        - **eliminarRutaEmpresa()**
        - **modificarRutaEmpresa()**
    - **Métodos en Bus para gestionar pasajeros:**
        - **asignarPasajero()**
        - **getMapaBus()**

5.4 SIA 1.5 EC: Gestión de Horarios en Ruta

- **Código Correspondiente:**
    - **Ruta:**
        - **Tiene un Map<LocalDate, LocalDateTime> para almacenar horarios:**
            - **Map<LocalDate, LocalDateTime> horarios**
        - **Métodos para agregar horarios:**
            - **agregarHorario(LocalDate fecha, LocalDateTime hora)**

5.5 SIA 1.7 EC: Inicialización de Datos Predeterminados

- **Código Correspondiente:**
    - **Método en SIAP para cargar datos predeterminados:**
        - **inicializarDatosPorDefecto()**

5.6 SIA 1.8 EC: Validación y Manejo de Errores

- **Código Correspondiente:**
    - **Manejo de errores y validaciones en Empresa:**
        - **Verificación de si una ruta o un bus existe antes de realizar operaciones (e.g., en buscarRutaEnEmpresa, obtenerRuta).**

6. Implementación Técnica

El código está escrito en Java y hace uso de colecciones como HashMap para gestionar los datos de los pasajeros. La implementación sigue principios básicos de programación orientada a objetos, donde cada bus y pasajero es representado como una entidad independiente.

6.1. Entorno de Desarrollo

- **Lenguaje**: Java
- **Entorno de Desarrollo**: NetBeans e IntelliJ IDEA
- **Frameworks**: Ninguno (aplicación básica basada en Java SE)

6.2. Estructura del Código

El código está dividido en dos clases principales:

- **Bus.java**: Gestiona los datos del bus y sus pasajeros.
- **Pasajero.java**: Almacena la información de cada pasajero.

7. Conclusiones

Este sistema proporciona una forma eficiente de gestionar la asignación de pasajeros a los buses y llevar un registro claro de los costos de mantención de cada vehículo. Al utilizar un mapa para almacenar los pasajeros, se mejora la eficiencia al realizar búsquedas por RUT, y la capacidad fija garantiza que no se asignen más pasajeros de los permitidos.

7.1. Próximo Avance

- Ampliar el sistema para permitir la eliminación de pasajeros del bus.
- Incluir un módulo de reportes para visualizar el uso y los costos de mantención de los buses a lo largo del tiempo.
- Implementar validaciones adicionales para asegurar la integridad de los datos (como verificar que el RUT tenga un formato válido antes de asignarlo).
- Inclusión de ventanas y Base de dato basada en SQLite