Proyecto #2 – Documento diseño

Introducción y Objetivos del Proyecto

El proyecto "Ferretería Rent-A-Car" busca mejorar la experiencia de alquiler de vehículos mediante una aplicación de interfaz intuitiva. Esta solución permitirá a usuarios (clientes, empleados y administradores) gestionar eficazmente todos los aspectos del alquiler de automóviles, como reservas, mantenimiento y administración de flotas.

Objetivos del Proyecto

Objetivo General: Desarrollar y lanzar una solución de software que simplifique la gestión de alquileres de vehículos, mejore la eficiencia y ofrezca una excelente experiencia de usuario a través de una interfaz gráfica basada en Swing o JavaFX.

Objetivos Específicos:

Mejorar la interfaz gráfica de usuario en comparación con la interfaz de consola existente.

Facilitar la gestión de vehículos, reservas y alquileres a través de la interfaz gráfica.

Incorporar una visualización en tiempo real de la disponibilidad de vehículos.

Optimizar la administración de usuarios y roles, incluyendo clientes, empleados y administradores.

Adaptar el proyecto anterior (Proyecto #1) a los nuevos requisitos.

Análisis de Requisitos

Requisitos Funcionales:

Gestión de Usuarios: Registro, inicio y cierre de sesión para usuarios. Administración de cuentas de clientes y empleados para administradores.

Gestión de Vehículos: Agregar, modificar y eliminar vehículos para usuarios autorizados.

Gestión de Reservas y Alquileres: Realizar y gestionar reservas para clientes, procesar alquileres y devoluciones para empleados.

Gestión de Tarifas y Seguros: Definir y modificar tarifas y opciones de seguros para administradores.

Gestión de Sedes: Añadir y gestionar sedes para superadministradores.

Reportes: Funcionalidad para generar reportes de vehículos, reservas y alquileres.

Requisitos No Funcionales:

Usabilidad: La interfaz debe ser intuitiva para todos los usuarios.

Rendimiento: Respuesta en menos de 2 segundos bajo carga normal.

Seguridad: Protección de datos sensibles y transacciones seguras.

Mantenibilidad: Código bien documentado y estructurado.

Compatibilidad: Compatible con principales versiones de Java y sistemas operativos.

Requisitos Específicos para la Interfaz Gráfica y la Lógica de Dominio

Interfaz Gráfica:

Proporcionar una experiencia de usuario intuitiva y coherente en todas las pantallas.

Facilitar la navegación y gestión de alquileres, reservas, vehículos y clientes.

Implementar controles de entrada de datos con validaciones.

Visualización clara de la disponibilidad de vehículos.

Lógica de Dominio:

Manejar la creación, actualización y eliminación de datos.

Aplicar reglas de negocio para tarifas y seguros.

Integrar la lógica de mantenimiento y administración de vehículos.

Procesar la autenticación y autorización de usuarios.

Ofrecer búsqueda y filtrado eficiente de información.

Descripción del Diseño del Proyecto #1

El Proyecto #1 tenía una interfaz basada en consola que permitía la administración básica de alquileres y reservas, con comandos de texto y visualización de datos en consola, limitando las funcionalidades y sin representación gráfica.

Cambios Propuestos para Adaptarlo al Proyecto #2

Para el Proyecto #2, se realizaron las siguientes modificaciones:

Transición de Consola a GUI para una interacción más intuitiva.

Mejora de la interactividad mediante elementos gráficos.

Optimización del flujo de trabajo para mayor eficiencia.

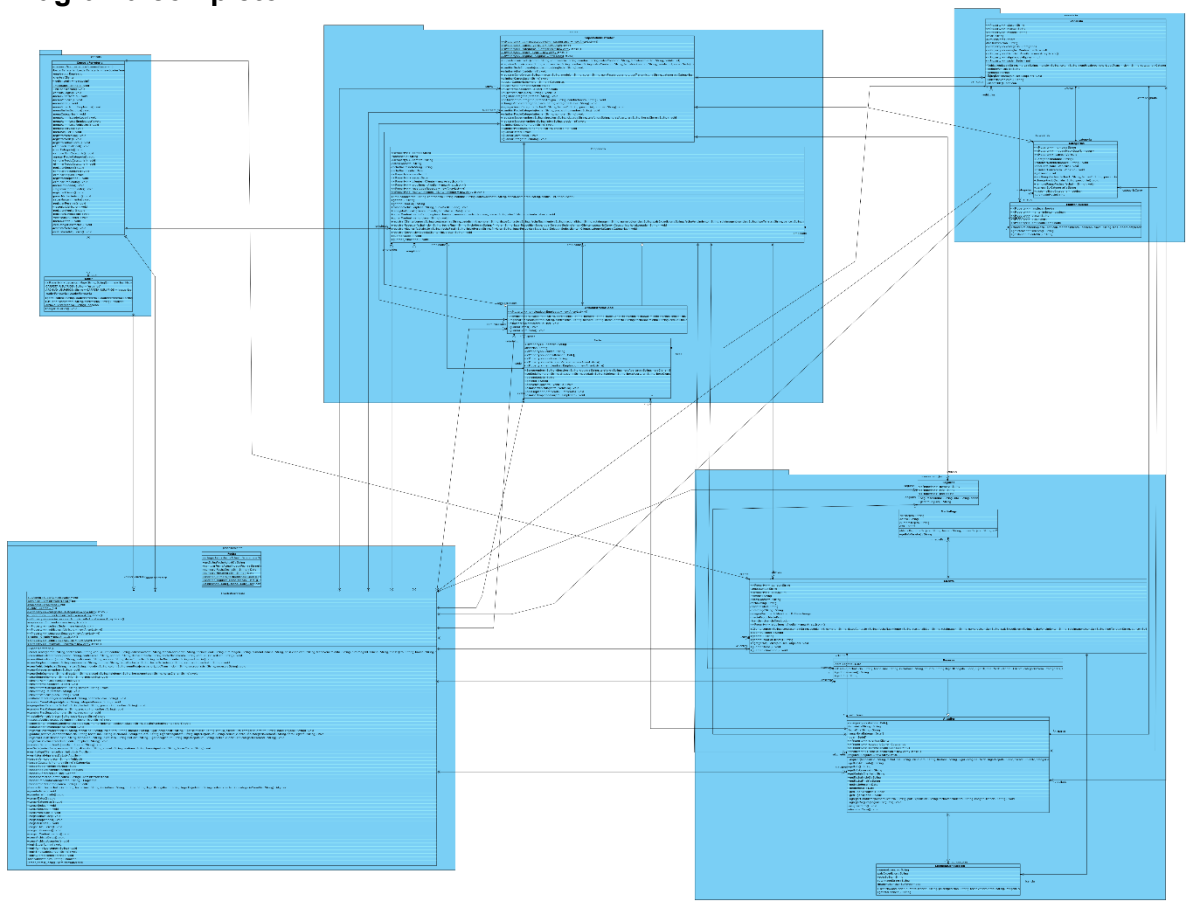
Visualización gráfica de datos complejos como la disponibilidad de vehículos.

Unificación de estilos para coherencia y profesionalismo.

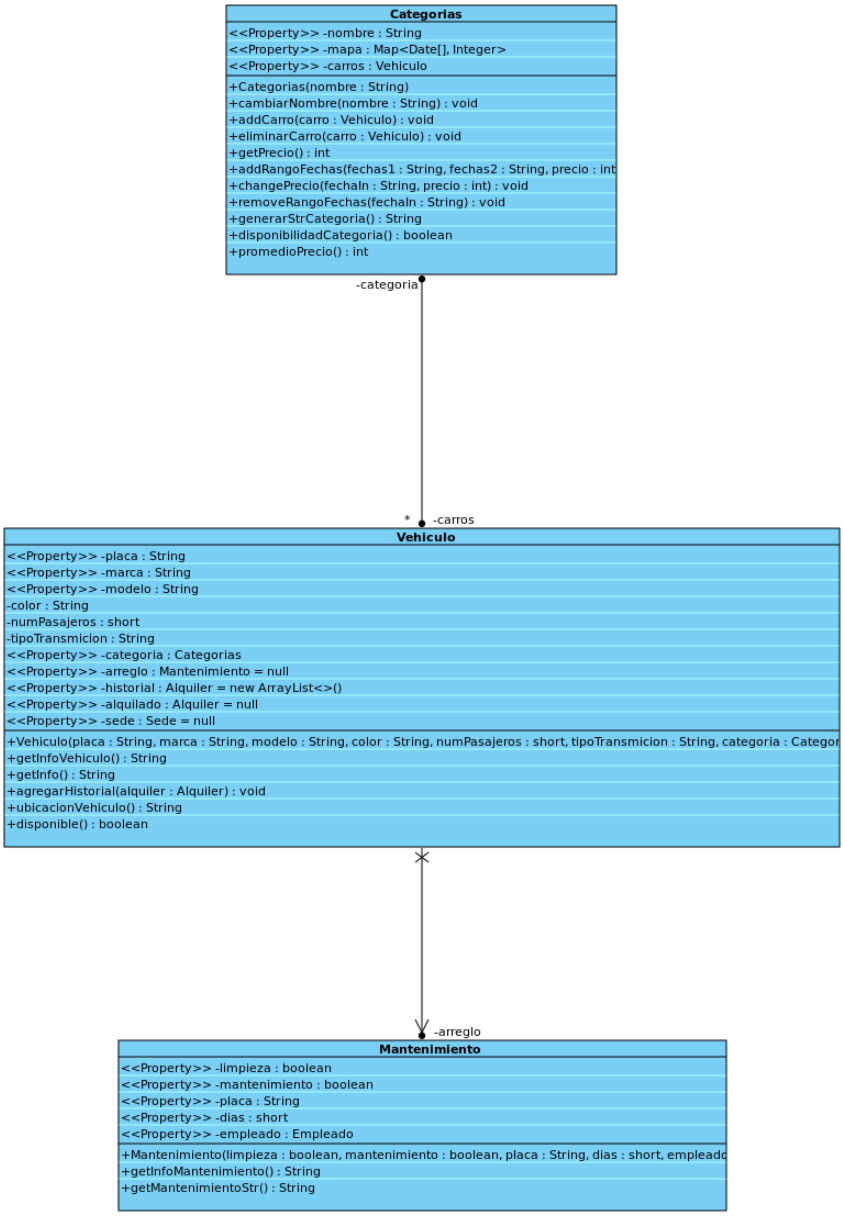
Siguiente se presentan los diagramas UML del proyecto antes y después de la implementación de la GUI

1. Diagrama de clases de proyecto pre-interfaz

1.1. Diagrama completo

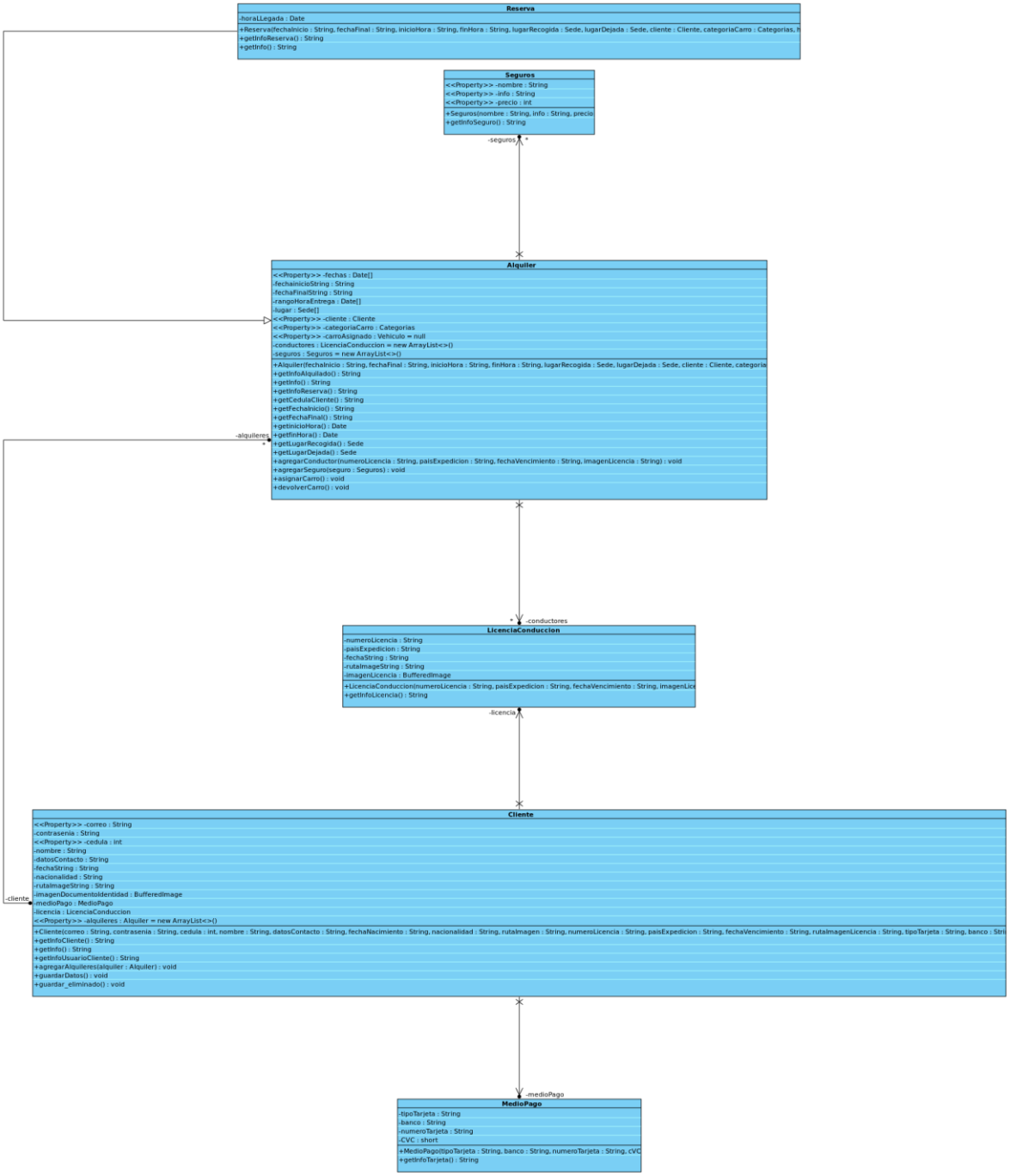


1.2. Diagrama inventario

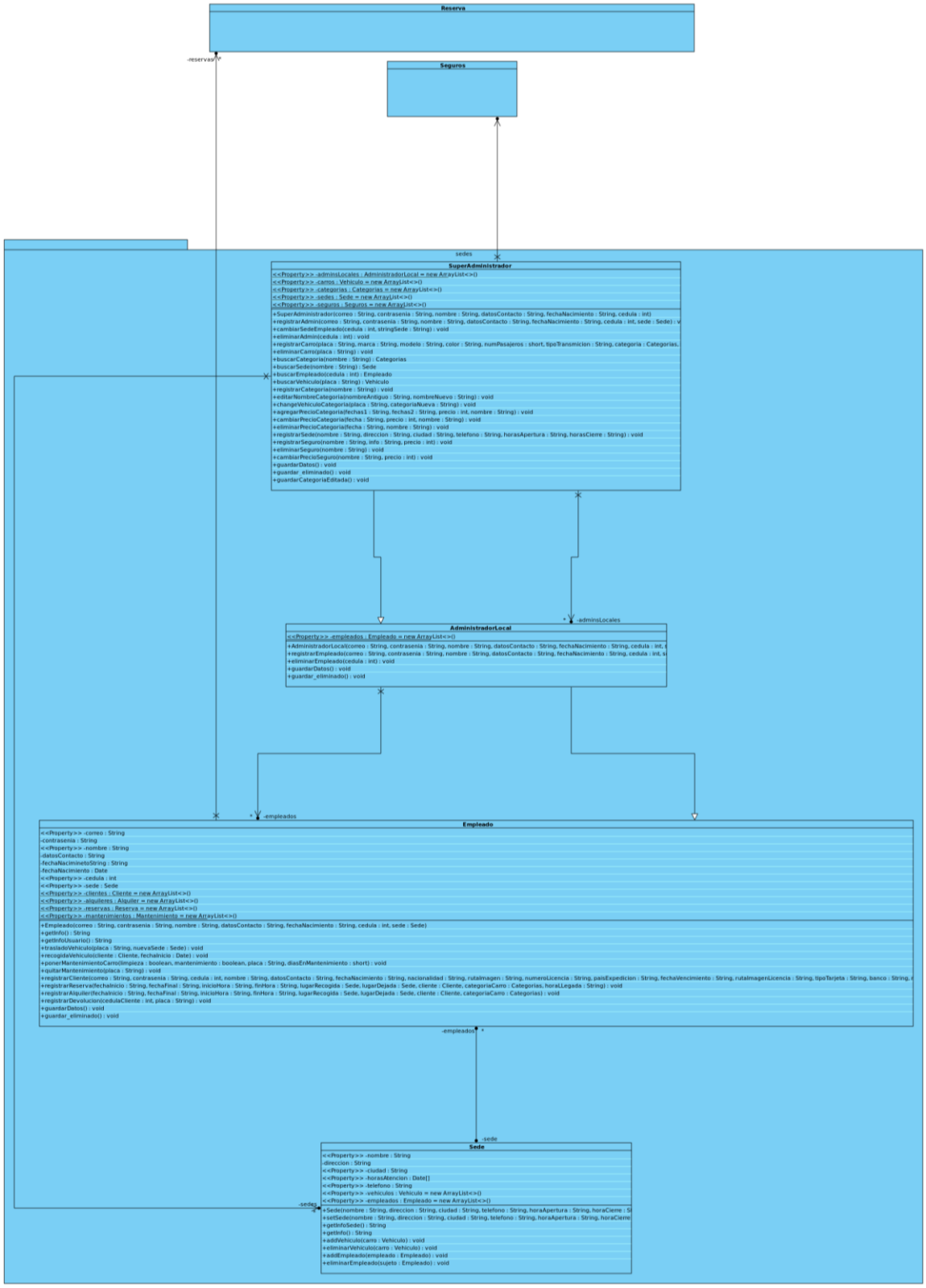


1.3. Diagrama procesamiento

1.4. Diagrama reserva-alquiler

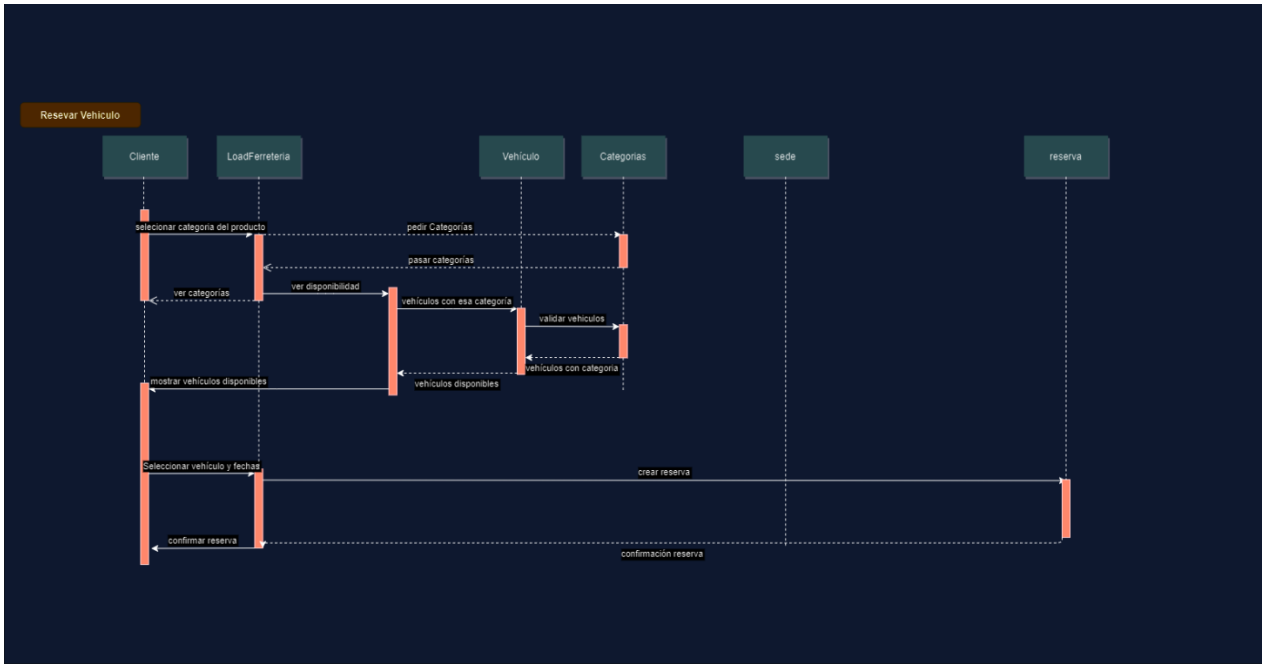


1.5. Diagrama sedes


Reserva
-reservas
Seguros
sedes
SuperAdministrador
<<Property>> -adminsLocales : AdministradorLocal = new ArrayList<>()
<<Property>> -carros : Vehiculo = new ArrayList<>()
<<Property>> -categorias : Categorias = new ArrayList<>()
<<Property>> -sedes : Sede = new ArrayList<>()
<<Property>> -seguros : Seguros = new ArrayList<>()
+SuperAdministrador(correo : String, contrasenia : String, nombre : String, datosContacto : String, fechaNacimiento : String, cedula : int)
+registrarAdmin(correo : String, contrasenia : String, nombre : String, datosContacto : String, fechaNacimiento : String, cedula : int, sede : Sede) :
+cambiarSedeEmpleado(cedula : int, stringSede : String) : void
+eliminarAdmin(cedula : int) : void
+registrarCarro(placa : String, marca : String, modelo : String, color : String, numPasajeros : short, tipoTransmicion : String, categoria : Categorias,
+eliminarCarro(placa : String) : void
+buscarCategoria(nombre : String) : Categorias
+buscarSede(nombre : String) : Sede
+buscarEmpleado(cedula : int) : Empleado
+buscarVehiculo(placa : String) : Vehiculo
+registrarCategoria(nombre : String) : void
+editarNombreCategoria(nombreAntiguo : String, nombreNuevo : String) : void
+changeVehiculoCategoria(placa : String, categoriaNueva : String) : void
+agregarPrecioCategoria(fechas1 : String, fechas2 : String, precio : int, nombre : String) : void
+cambiarPrecioCategoria(fecha : String, precio : int, nombre : String) : void
+eliminarPrecioCategoria(fecha : String, nombre : String) : void
+registrarSede(nombre : String, direccion : String, ciudad : String, telefono : String, horasApertura : String, horasCierre : String) : void
+registrarSeguro(nombre : String, info : String, precio : int) : void
+eliminarSeguro(nombre : String) : void
+cambiarPrecioSeguro(nombre : String, precio : int) : void
+guardarDatos() : void
+guardar_eliminado() : void
+guardarCategoriaEditada() : void
-adminsLocales
AdministradorLocal
<<Property>> -empleados : Empleado = new ArrayList<>()
+AdministradorLocal(correo : String, contrasenia : String, nombre : String, datosContacto : String, fechaNacimiento : String, cedula : int,
+registrarEmpleado(correo : String, contrasenia : String, nombre : String, datosContacto : String, fechaNacimiento : String, cedula : int,
+eliminarEmpleado(cedula : int) : void
+guardarDatos() : void
+guardar_eliminado() : void
-empleados
Empleado
<<Property>> -correo : String
-contrasenia : String
<<Property>> -nombre : String
-datosContacto : String
-fechaNaciminetoString : String
-fechaNacimiento : Date
<<Property>> -cedula : int
<<Property>> -sede : Sede
<<Property>> -clientes : Cliente = new ArrayList<>()
<<Property>> -alquileres : Alquiler = new ArrayList<>()
<<Property>> -reservas : Reserva = new ArrayList<>()
<<Property>> -mantenimientos : Mantenimiento = new ArrayList<>()
+Empleado(correo : String, contrasenia : String, nombre : String, datosContacto : String, fechaNacimiento : String, cedula : int, sede : Sede)
+getInfo() : String
+getInfoUsuario() : String
+trasladoVehiculo(placa : String, nuevaSede : Sede) : void
+recogidaVehiculo(cliente : Cliente, fechaInicio : Date) : void
+ponerMantenimientoCarro(limpieza : boolean, mantenimiento : boolean, placa : String, diasEnMantenimiento : short) : void
+quitarMantenimiento(placa : String) : void
+registrarCliente(correo : String, contrasenia : String, cedula : int, nombre : String, datosContacto : String, fechaNacimiento : String, nacionalidad : String, rutaImagen : String, numeroLicencia : String, paisExpedicion : String, fechaVencimiento : String, rutaImagenLicencia : String, tipoTarjeta : String, banco : String,
+registrarReserva(fechaInicio : String, fechaFinal : String, inicioHora : String, finHora : String, lugarRecogida : Sede, lugarDejada : Sede, cliente : Cliente, categoriaCarro : Categorias, horaLLegada : String) : void
+registrarAlquiler(fechaInicio : String, fechaFinal : String, inicioHora : String, finHora : String, lugarRecogida : Sede, lugarDejada : Sede, cliente : Cliente, categoriaCarro : Categorias) : void
+registrarDevolucion(cedulaCliente : int, placa : String) : void
+guardarDatos() : void
+guardar_eliminado() : void
-empleados
-sede
Sede
<<Property>> -nombre : String
-direccion : String
<<Property>> -ciudad : String
<<Property>> -horasAtencion : Date[]
<<Property>> -telefono : String
<<Property>> -vehiculos : Vehiculo = new ArrayList<>()
<<Property>> -empleados : Empleado = new ArrayList<>()
+Sede(nombre : String, direccion : String, ciudad : String, telefono : String, horaApertura : String, horaCierre : S
+setSede(nombre : String, direccion : String, ciudad : String, telefono : String, horaApertura : String, horaCierre
+getInfoSede() : String
+getInfo() : String
+addVehiculo(carro : Vehiculo) : void
+eliminarVehiculo(carro : Vehiculo) : void
+addEmpleado(empleado : Empleado) : void
+eliminarEmpleado(sujeto : Empleado) : void
-sedes

2. Diagramas de secuencia

2.1. Reservar-Alquilar vehículo



3.1.1 Cliente: Todo comienza con el cliente. Esta entidad representa a la persona o sistema que interactúa con la aplicación.

3.1.2. Seleccionar categoría del producto: El cliente inicia el proceso seleccionando una categoría de producto. Esto podría referirse a diferentes tipos de vehículos, como sedanes, SUVs, camionetas, etc.

3.1.3. Ver categorías: El cliente solicita ver las categorías disponibles.

3.1.4. Pedir Categorías: LoadFerreteria consulta las categorías disponibles. Para hacer esto, se comunica con la clase "Categorías".

3.1.5. Pasar categorías: Una vez que las categorías se han recuperado, se pasan de vuelta al cliente a través de LoadFerreteria.

3.1.6. Ver disponibilidad: Después de elegir una categoría, el cliente solicita ver la disponibilidad de vehículos dentro de esa categoría.

3.1.7. Vehículos con esa categoría: LoadFerreteria consulta a la clase "Vehículo" para obtener los vehículos disponibles en la categoría seleccionada.

3.1.8. Validar vehículos: Se realiza una validación de los vehículos, si están disponibles para la reserva o cualquier otro criterio necesario.

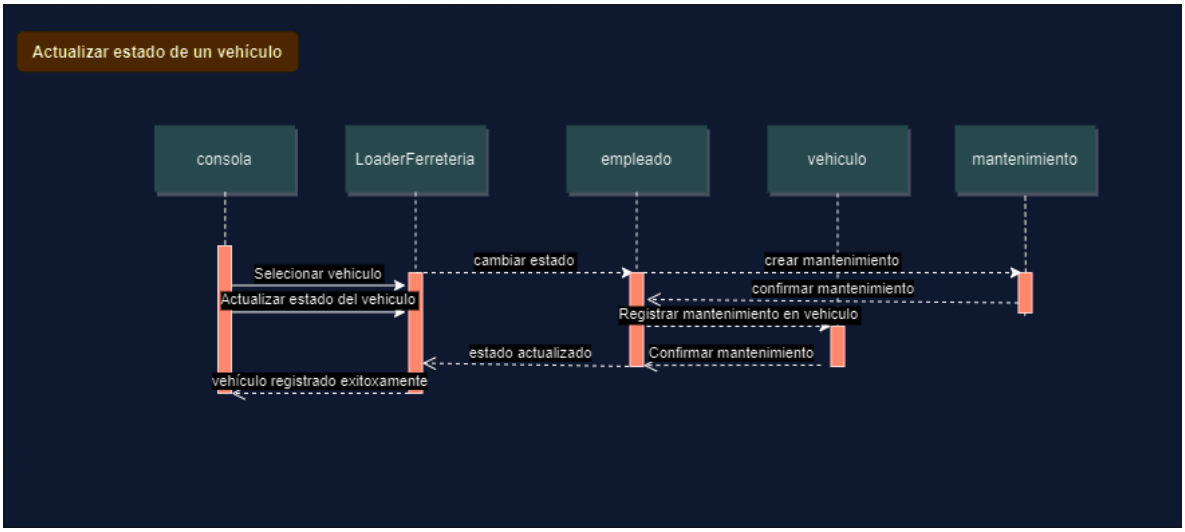
3.1.9. Mostrar vehículos disponibles: Una vez validados, los vehículos disponibles se muestran al cliente.

3.1.10. Seleccionar vehículo y fechas: El cliente elige un vehículo específico y las fechas para las cuales desea reservar.

3.1.11. Crear reserva: Se crea una reserva en el sistema con los detalles proporcionados por el cliente, esta acción se comunica con la clase "reserva".

3.1.12. Confirmar reserva: Una vez creada la reserva, se envía una confirmación al cliente.

3.2 Actualizar estado del vehículo



3.2.1 Consola: Un usuario (probablemente un empleado o administrador) interactúa con una interfaz de la aplicación. El proceso inicia cuando el usuario selecciona un vehículo específico desde la consola para actualizar su estado.

3.2.2 Actualizar estado del vehículo: Una vez que el vehículo es seleccionado, el usuario procede a actualizar el estado de dicho vehículo. Esto podría referirse a cambiar su disponibilidad, condiciones, ubicación, etc.

3.2.3 Cambiar estado: El empleado cambia o actualiza el estado del vehículo basándose en la información proporcionada desde la consola.

3.2.4 Registrar mantenimiento en vehículo: Tras cambiar el estado, si el vehículo requiere mantenimiento, el empleado procede a registrar esta información.

3.2.5 Confirmar mantenimiento: Luego de registrar el mantenimiento, se envía una confirmación al sistema.

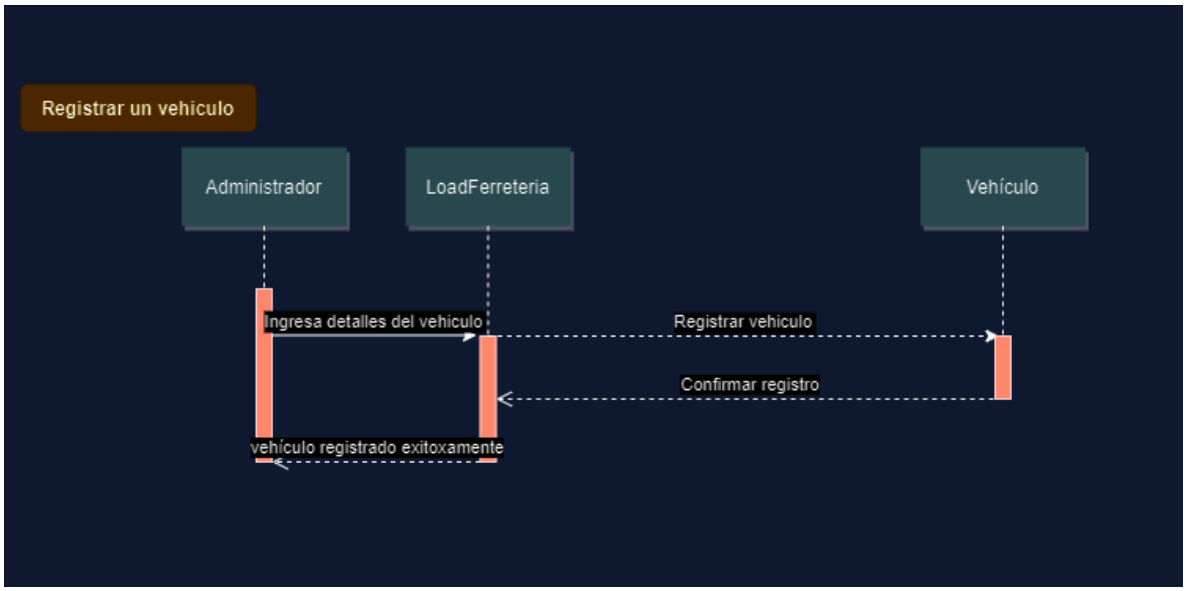
3.2.6 Crear mantenimiento: Se crea un registro de mantenimiento en el sistema para el vehículo en cuestión.

3.2.7 Confirmar mantenimiento: Se confirma que el mantenimiento ha sido registrado correctamente.

3.2.8 LoaderFerreteria: Una vez que el estado del vehículo ha sido actualizado y el mantenimiento registrado (si es necesario), LoaderFerreteria envía un mensaje de confirmación de vuelta a la consola.

3.2.9 Vehículo registrado exitosamente: Esta es la confirmación que se muestra al usuario, indicando que el proceso se completó con éxito.

3.3 Registrar un vehículo



3.3.1 Administrador: Todo comienza con el administrador, que es la figura encargada de añadir nuevos vehículos al sistema.

3.3.2 Ingresa detalles del vehículo: El administrador inicia el proceso ingresando todos los detalles pertinentes del vehículo que desea registrar. Estos detalles pueden incluir información como marca, modelo, año, número de placa, etc.

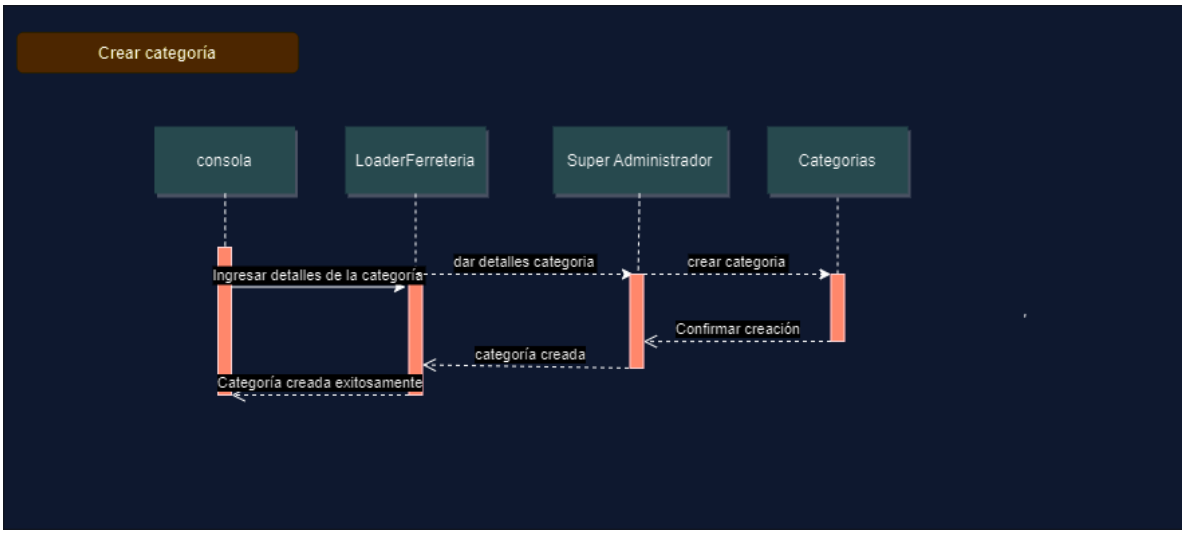
3.3.3. **Registrar vehículo:** Una vez que el administrador ha ingresado todos los detalles, LoadFerreteria toma esa información y procede con el proceso de registro del vehículo en el sistema.

3.3.4 Confirmar registro: Una vez que LoadFerreteria envía la solicitud para registrar el vehículo, la clase "Vehículo" procesa esa información y, si todo es correcto, confirma el registro. Esto significa que el vehículo ha sido añadido con éxito al sistema.

3.3.5 LoadFerreteria: Tras recibir la confirmación de que el vehículo ha sido registrado, LoadFerreteria informa al administrador acerca del éxito del proceso.

3.3.6 Vehículo registrado exitosamente: Esta confirmación se muestra al administrador para informarle que el vehículo ha sido añadido correctamente.

3.4 Crear categoría



3.4.1 Ingresar detalles de la categoría: Desde la consola, se ingresan los detalles de la nueva categoría que se desea crear. Estos detalles pueden incluir el nombre de la categoría, una descripción, entre otros.

3.4.2 Dar detalles categoría: Tras recibir los detalles de la nueva categoría desde la consola, LoaderFerreteria pasa esta información al Super Administrador.

3.4.3 Crear categoría: Una vez que recibe los detalles de la categoría del LoaderFerreteria, el Super Administrador inicia el proceso de creación de la nueva categoría.

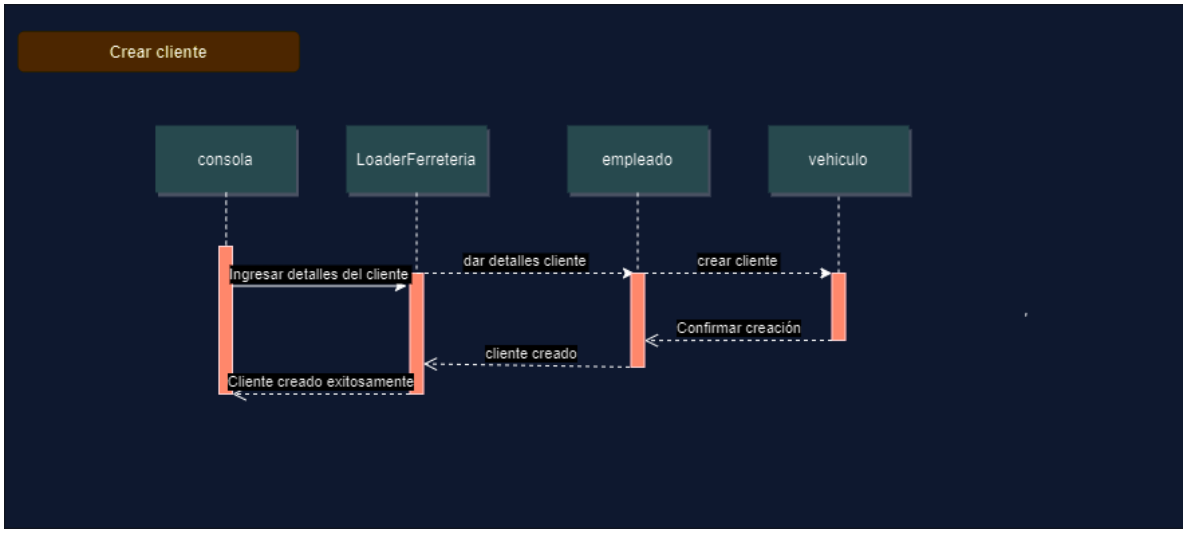
3.4.4 Confirmar creación: Tras recibir la solicitud de creación de la nueva categoría por parte del Super Administrador, la clase "Categorías" procesa la información , y si todo está en orden, confirma que la categoría ha sido creada exitosamente.

3.4.5. **Super Administrador:** Una vez que recibe la confirmación de la creación exitosa de la categoría por parte de "Categorías", notifica a LoaderFerreteria.

3.4.6 LoaderFerreteria: Al ser informado del éxito en la creación de la categoría, LoaderFerreteria informa a la consola.

3.4.7 Categoría creada exitosamente: Esta es la confirmación que recibe la consola, indicando que la categoría se ha añadido con éxito al sistema.

3.5 Crear cliente



3.5.1 **Ingresar detalles del cliente:** El usuario (Empleado, administrador, etc) ingresa los datos del nuevo cliente en la consola. Estos detalles pueden incluir nombre, dirección, número de teléfono, entre otros.

3.5.2 **Dar detalles cliente:** Después de que la consola recibe la información del cliente, LoaderFerreteria toma esos detalles y los pasa al empleado.

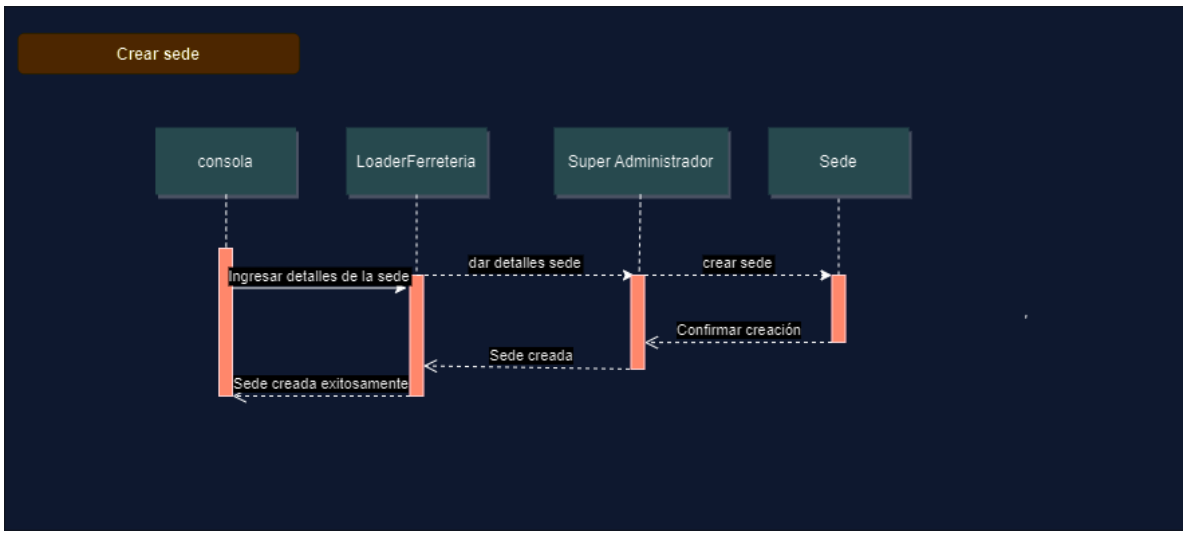
3.5.3 **Crear cliente:** Con la información proporcionada por LoaderFerreteria, el empleado crea el nuevo cliente en la base de datos o sistema correspondiente.

3.5.4 **Confirmar creación:** Tras procesar la información y registrar al nuevo cliente, el empleado confirma la creación exitosa del cliente.

3.5.5 **LoaderFerreteria:** Una vez que el empleado confirma que el cliente ha sido creado con éxito, LoaderFerreteria recibe esta confirmación.

3.5.6 **Cliente creado exitosamente:** LoaderFerreteria notifica a la consola que el cliente ha sido registrado con éxito en el sistema.

3.6 Crear sede



3.6.1 Ingresar detalles de la sede: Desde la consola, el usuario proporciona los datos requeridos para la nueva sede, que podrían incluir ubicación, capacidad, número de teléfono, entre otros.
3.6.2 Dar detalles sede: Una vez que la consola ha recopilado la información de la sede, LoaderFerreteria toma esos datos y los transmite al Super Administrador.

3.6.3 Crear sede: Con los datos proporcionados por LoaderFerreteria, el Super Administrador procede a registrar la nueva sede en la base de datos o sistema correspondiente.

3.6.4 Confirmar creación: Después de registrar la sede, el Super Administrador confirma que la sede ha sido creada con éxito.

3.6.5 LoaderFerreteria: Al recibir la confirmación de que la sede ha sido registrada con éxito por el Super Administrador, LoaderFerreteria procesa esta confirmación.

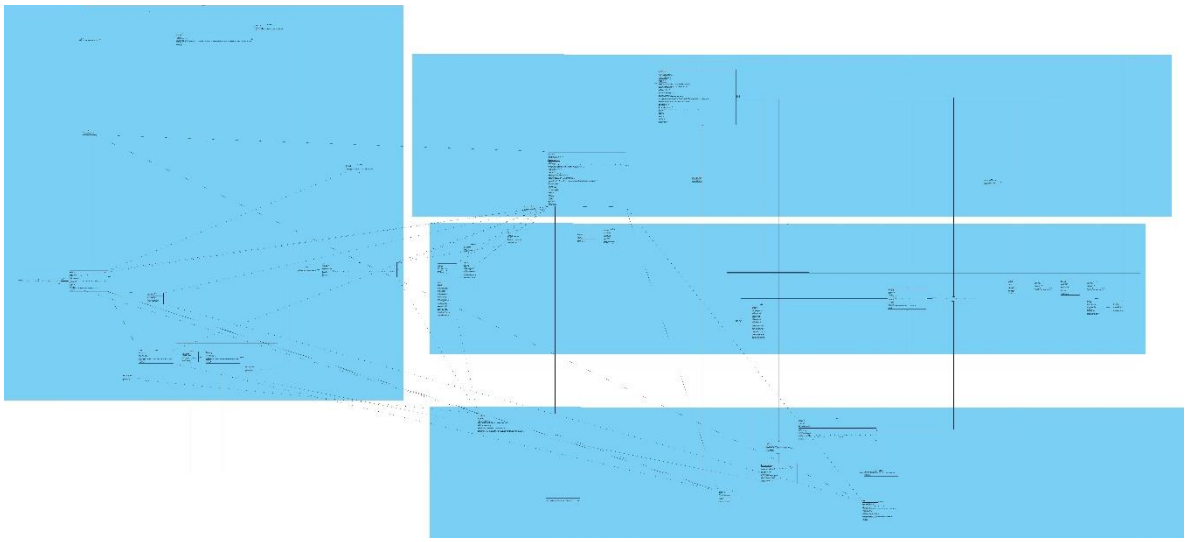
3.6.6 Sede creada exitosamente: Finalmente, LoaderFerreteria notifica a la consola que la sede ha sido creada exitosamente.

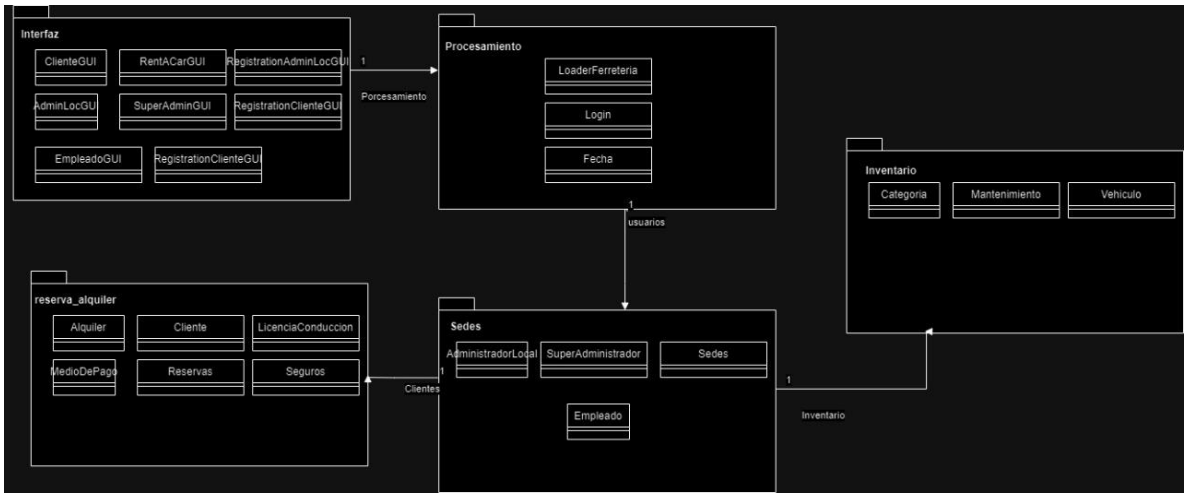
3. Diagramas de clases Post-interfaz

Ahora para esta parte solo se mostrar las partes que fueron fundamentalmente modificadas, especialmente la interfaz como nuevo modulo del programa, y el diagrama de clases global

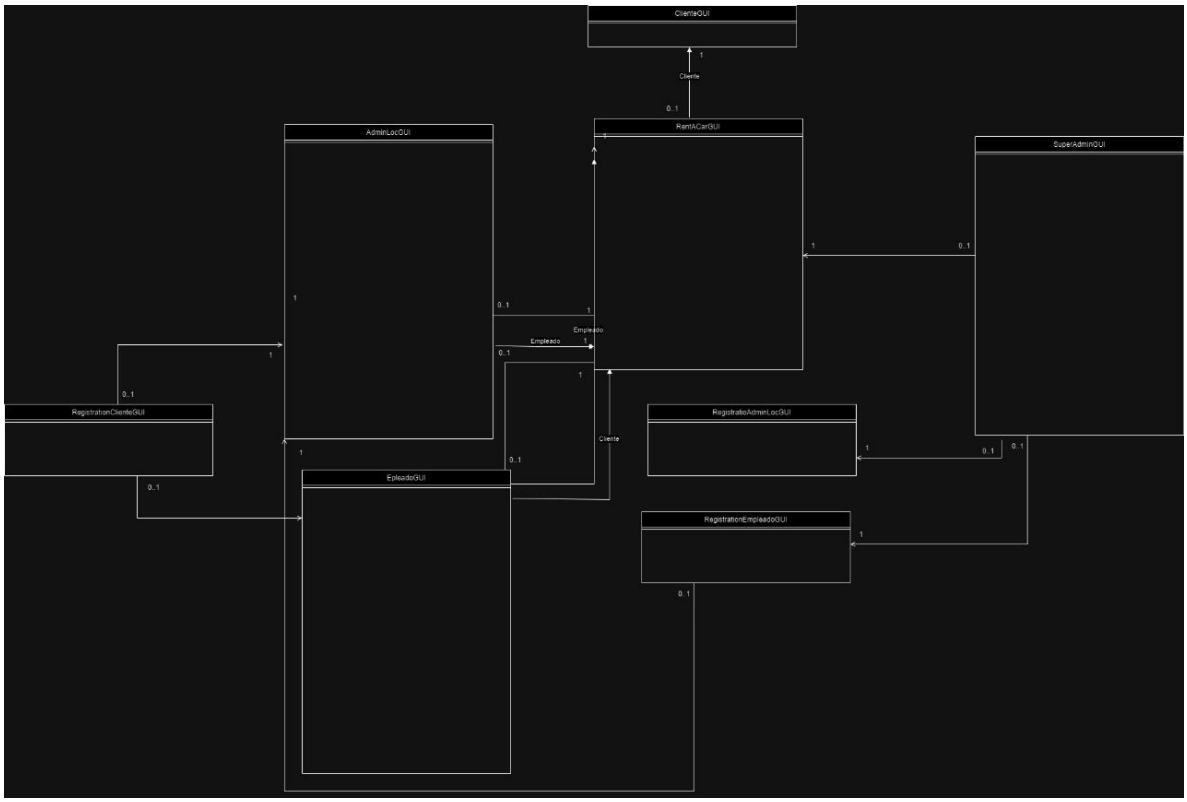
3.1 Diagrama de clases global y modelo de mundo

se adjunta los diagramas en el documento, pero en el directorio se encuentran los archivos originales para poder visualizarlos en una resolución mayor

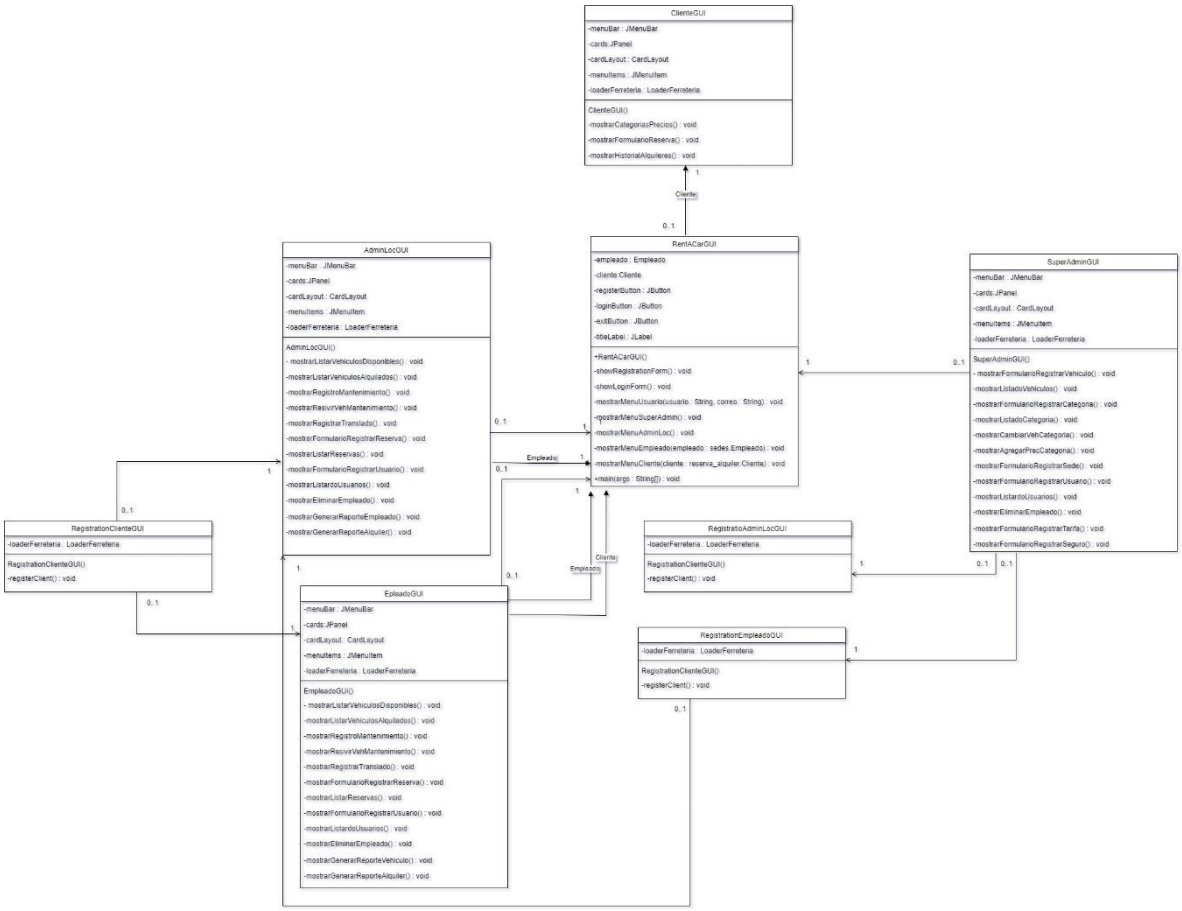




3.2 Diagrama de clases de la interfaz



3.3 Diagrama de métodos de la interfaz



Como ultima aclaración no se actualizaron los diagramas de secuencia de las operaciones que se pueden realizar en el programa ya que esencialmente para la lógica del programa siguen la misma lógica que la consola además de que el tomar los datos y enviarlos se hace en el mismo método sin requerir el llamado de clases relacionadas etc. Esto producto de la compatibilidad de Loadererreteria con las relaciones hacia la interfaz

4. Glosario

4.1 LoadFerreteria: Es una clase intermediaria que maneja las operaciones relacionadas con la renta de vehículos.

4.2. Categorías: Es una clase que administra las diferentes categorías de vehículos disponibles.

4.3. Vehículo: Es una clase que administra la información de los vehículos.

4.4. Sede: Es una clase relacionada con el lugar o ubicación donde se encuentra el vehículo.

4.5. Reserva: Esta entidad representa la reserva en sí y cualquier lógica o almacenamiento relacionado con ella.

4.6 Empleado: Esta clase representa al empleado que está interactuando con el sistema. Es quien efectivamente realiza algunas acciones.

4.7 Mantenimiento; Esta clase se encarga de gestionar y almacenar la información relacionada con el mantenimiento de los vehículos.