Gabriel Padilla - 202224552
María Calle - 202220202
Juan Ernesto Pinto - 202021839

Documento de Diseño Proyecto 1 DPO

1) Elementos relevantes del análisis

a. Requerimientos funcionales

1	Generar un archivo de log con el historial de un vehículo	Persistencia
2	Conocer las características básicas de un vehículo	Inventario de vehículos
3	Acceder / consultar los vehículos por categorías	
4	Conocer el estado de un vehículo (alquilado / disponible / en mantenimiento...)	
5	Registrar la compra de vehículos nuevos y su ubicación en una sede particular (administrador general)	
6	Registrar a un cliente con: • Datos personales • Datos de licencia • Datos de medio de pago	Reservas y alquiler
7	Crear una reserva indicando: • El tipo de carro que se quiere (categoría) • La sede donde se recogerá y entregará • La fecha y la hora de recogida y entrega • Seguros incluidos • Conductores adicionales	
8	Generar cobros de los alquileres: • Cobro preliminar del 30% al hacer la reserva • Cobro restante total cuando el vehículo se recoge	
9	Bloquear tarjeta de crédito del cliente durante su alquiler y desbloquearla una vez devuelve el vehículo	
10	Generar tarifa del alquiler con base en: • Categoría del vehículo • Rango de fechas del alquiler • Si el vehículo será entregado y recogido en la misma sede • Seguros incluidos por el cliente • Conductores adicionales	
11	Registrar un conductor adicional para el vehículo	
12	Registrar el traslado de un carro entre sedes (reserva especial para cliente interno)	
13	Gestionar la información básica de cada sede (administrador general)	Sedes
14	Registrar y manejar información de los empleados de la sede (administrador local)	
15	Actualizar el estado de un carro (ser limpiado antes de volver a estar disponible)	

b. Requerimientos no funcionales

- Seguridad: el sistema debe estar protegido contra accesos no autorizados a ciertas funcionalidades (distinción de funcionalidades para clientes / empleados)
- Persistencia: en caso de que el sistema se caiga en alguna de las sedes, la información de reservas, pagos, inventario, etc. debe quedar guardada en archivos
- Actuación: la aplicación debe poder manejar el n´umero de usuarios requerido sin ninguna afectación en rendimiento
- Usabilidad: la aplicación debe ser fácil de usar y comprender para cada tipo de usuario
- Fiabilidad: el sistema debe ser confiable y servirles a los distintos usuarios para sus requisitos
- Mantenimiento y escalabilidad: el sistema debe ser fácil de actualizar, mantener y escalar hacia arriba o abajo según se anecesario

c. Restricciones

- La aplicación debe estar hecha en Java y la interfaz debe estar basada en consola
- Todos los usuarios del sistema deben tener un login y un password: todos usarán la misma aplicación pero dependiendo del tipo de usuario, las opciones serán diferentes (cliente / empleado)
- En cuánto a la persistencia, la información debe almacenarse en archivos, planos o binarios, dentro de una carpeta, y se debe suponer que solo la aplicación va a escribir y leer los archivos en esta carpeta
- La carpeta destinada a la persistencia no puede ser la misma donde se encuentre el código fuente de la aplicación

2) Responsibility driven design

a. Objetos y roles

i. Information holders:
   - Base de datos, que contiene archivos destinados a guardar información sobre sedes, inventario de vehículos, reservas, pagos y clientes
   - Inventario de vehículos, que guarda todos los vehículos que son propiedad de la empresa en categorías

ii. Service provider:
   - Archivo, que escribe todos los archivos destinados a la persistencia con base en los eventos de la aplicación

iii. Coordinator:
   - Aplicación alquiler, que delega y encamina las tareas a los demás componentes de la aplicación

iv. Controller:
   - Reservas, que reúne toda la información importante para actualizar estados y calcular la tarifa, que despliega acciones importantes en otros componentes

b. Responsabilidades

i. *Conocimiento*: los information holders, es decir la base de datos y el inventario de vehículos deben cumplir con una responsabilidad de conocimiento: la base de datos debe conocer toda la actividad de la

aplicación y el inventaria de vehículos debe contener todos los vehículos que son propiedad de la empresa

ii. *Acciones*: el objeto archivo debe ser capaz de escribir todos los archivos para garantizar la persistencia

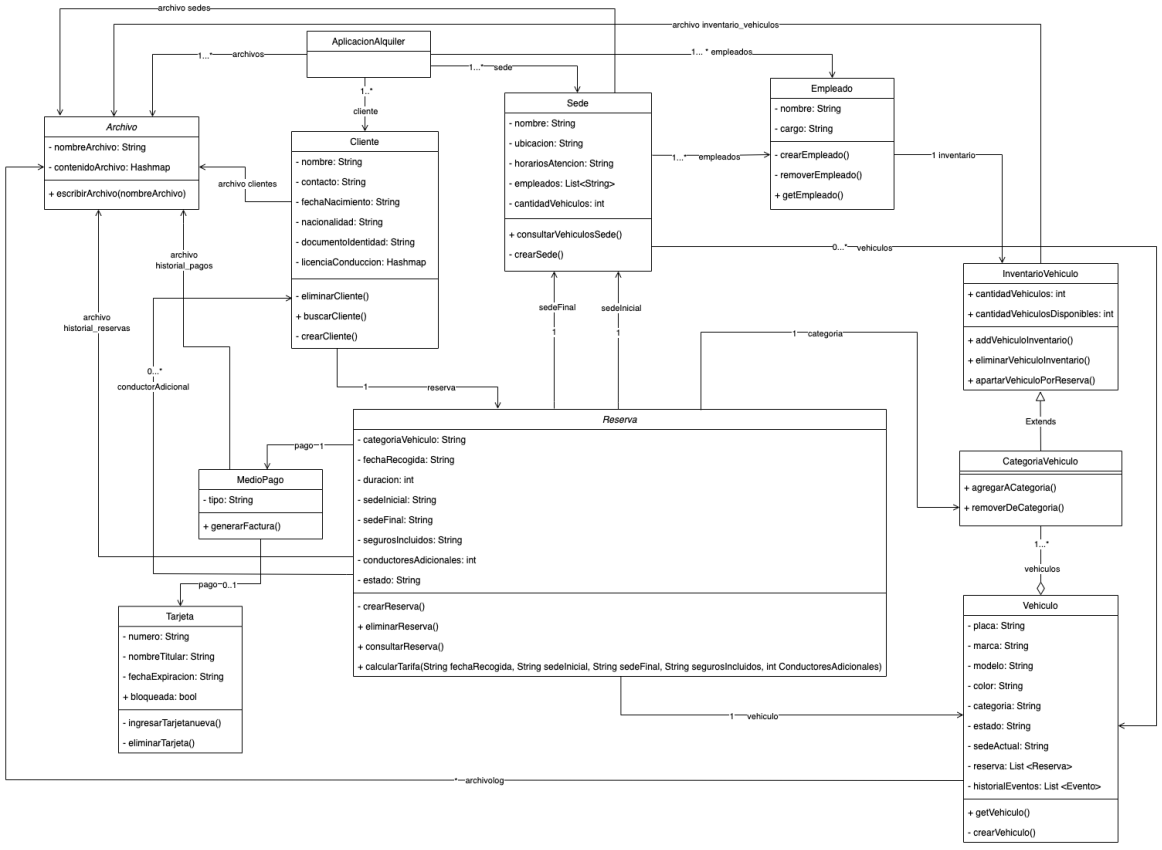
iii. *Acciones*: la aplicación alquiler debe cumplir con la responsabilidad de delegar y encaminar las tareas a los demás componentes de la aplicación

iv. *Decisiones*: el objeto reservas debe ser capaz de tomar decisiones que determinen el comportamiento de otros elementos en la aplicación para cumplir con sus fines

c. Colaboraciones

El coordinator, aplicación alquiler, delega y encamina las tareas solicitadas por los usuarios a los distintos componentes de la aplicación. Los componentes de la aplicación cumplen con sus respectivos roles, que giran en torno a los estímulos del corazón de la aplicación: las reservas (controller). El coordinator y controller actúan de forma encadenada, a su vez consultando el inventario de vehículos, que les provee el conocimiento que necesitan para poder ejecutar sus acciones armoniosamente, y, en el caso de las reservas, le permite tomar decisiones que guían el comportamiento de otros componentes. Todas las acciones de los componentes de la aplicación se registran en archivos, que son escritos por un service provider (objeto archivo), que alimenta simultáneamente el principal information holder de la aplicación: la base de datos.

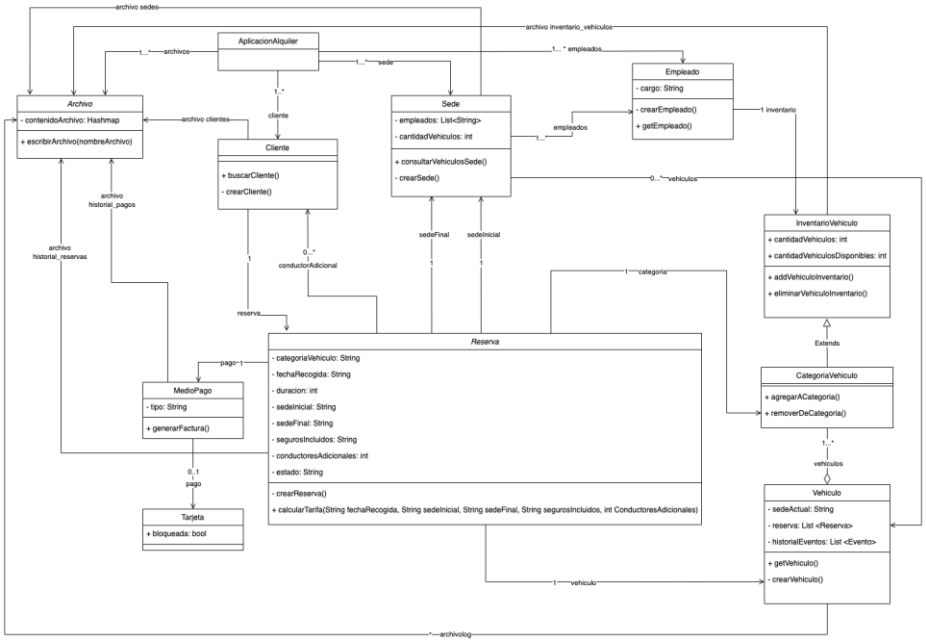
3) Diagrama de clases con atributos, métodos, cardinalidades y relaciones



Justificación de decisiones importantes:

- La clase AplicacionAlquiler se relaciona con las clases Cliente y Empleado, que permiten acceder a las distintas funcionalidades de la aplicación de acuerdo con el tipo de usuario. Dentro de la clase Empleado se especifica el tipo de empleado en el atributo cargo (admin general / admin local / empleado), que filtra con más detalle las funcionalidades del usuario.
- La subclase CategoriaVehiculo hereda de la superclase InventarioVehiculo, ya que ambas comparten los atributos cantidadVehiculos y cantidadVehiculosDisponibles, que son de interés para las clases Reserva y Sede. La subclase CategoriaVehiculo tiene una relación de agregación con la clase Vehiculo, pues una categoría contiene vehículos. De esta manera se construye la idea que hay un inventario, clasificado en categorías que contienen vehículos.
- Una reserva se asocia, tanto con una categoría como con un vehículo, pues para consultar la disponibilidad de vehículos se debe conocer la información de la categoría, y para cambiar el estado de un vehículo en las fechas de la reserva, se debe tener acceso a la clase Vehiculo
- Para poder generar la tarifa de una reserva, usamos la función calcularTarifa() en la clase Reserva, que se asocia con las clases Sede y CategoriaVehiculo para reunir la información necesaria para calcular la tarifa de un cliente. Esta función tiene como parámetros la fecha de recogida, la sede inicial y final, los seguros incluidos y los conductores adicionales; todos factores que afectan la tarifa. El valor resultante de la función calcularTarifa() después nos permitirá hacer los cobros del 30% y el cobro final
- La clase Reserva se asocia con la clase Cliente de dos formas: cada cliente tiene una reserva asociada, pero también, si a la hora de hacer una reserva se quiere agregar a un conductor, este se agregará como otro cliente, ya que el cliente y el conductor adicional comparten los mismos atributos
- El traslado de carros entre sedes se hará de la misma forma que se hace una reserva convencional, solo que el cliente tendrá unos atributos específicos para reconocerlo como un cliente interno

4) **Diagrama de clases de alto nivel con todas las clases y relaciones, pero no todos los métodos y atributos (relaciones más claras)**



5) **En relación con la persistencia**

Todas las acciones de la aplicación se guardarán en archivos planos para garantizar la persistencia de la información. Habrá una carpeta que guardará diversos archivos de persistencia. Estructuraremos esta carpeta de la siguiente manera:

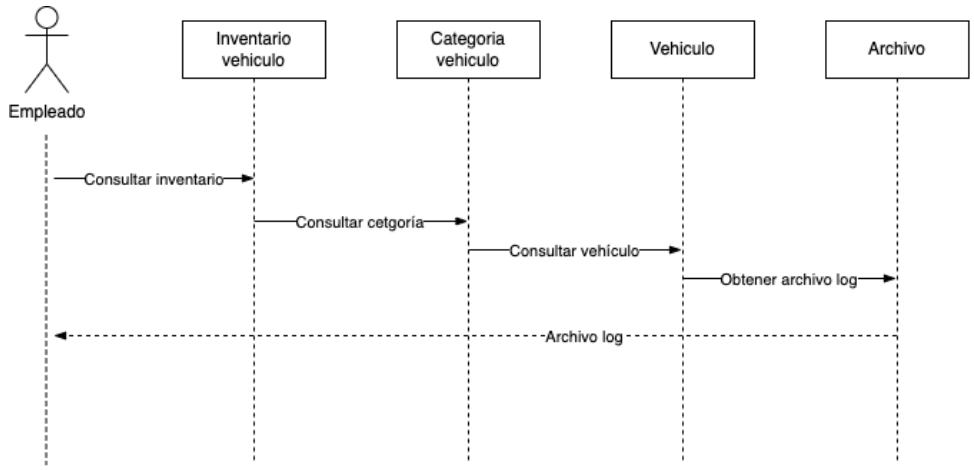
Persistencia (carpeta) >

- Archivo: sedes
- Archivo: inventario_vehiculos (estructurado en categorias que contienen vehiculos)
- Archivo: historial_reservas
- Archivo: historial_pagos
- Archivo: clientes

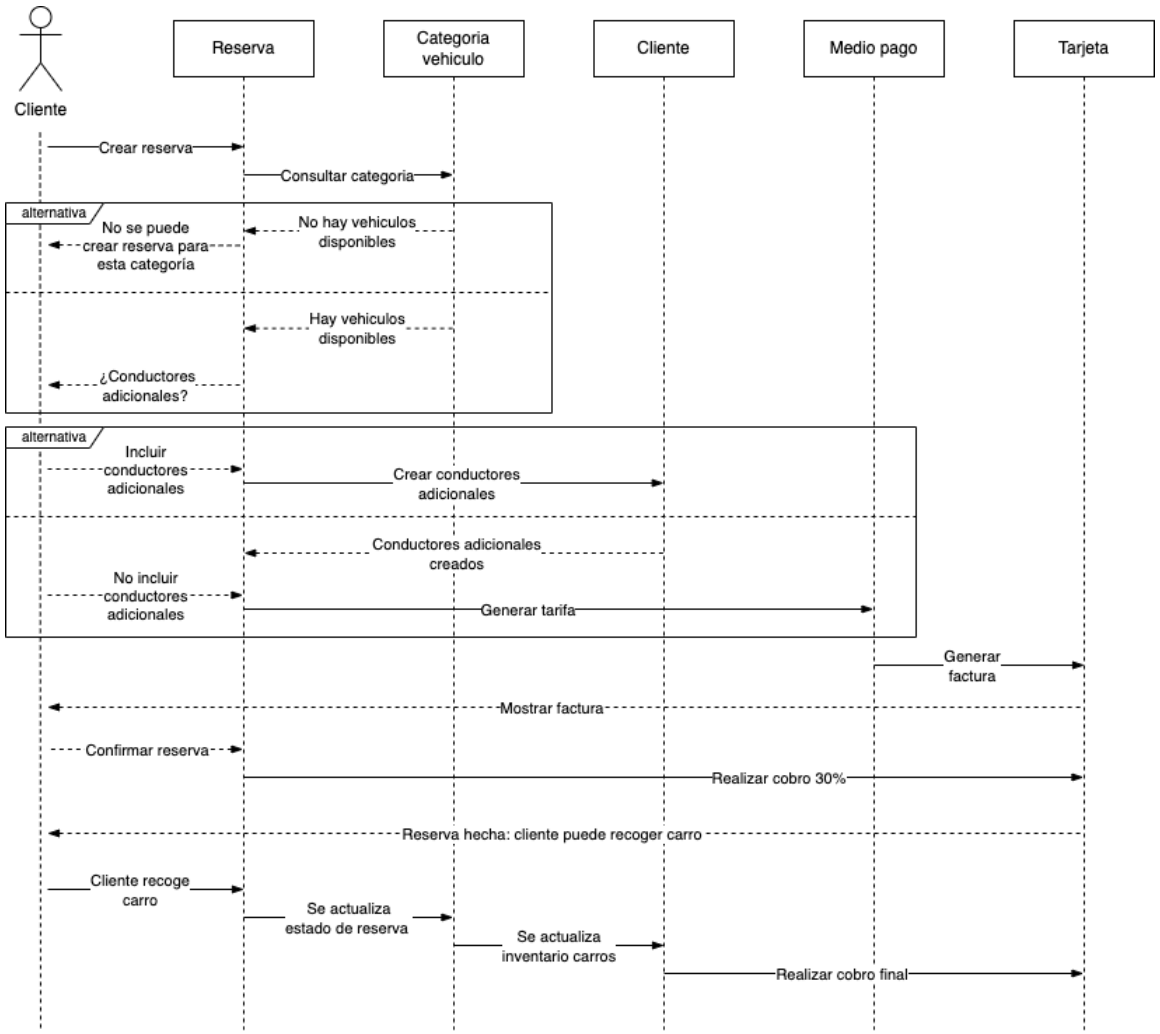
Las clases Sede, InventarioVehiculo, Reserva, MedioPago y Cliente están asociadas a la clase archivo para escribir los eventos de estas clases en los archivos correspondientes usando el método escribirArchivo(). De esta forma se garantizará la persistencia de toda la información de la aplicación alquiler.

6) **Diagramas de secuencia para funcionalidades críticas**

a. <u>Generar un archivo de log para historial de un vehículo</u>



b. <u>Crear una reserva:</u>
   i. Acceder/consultar los vehículos por categoría
   ii. Registrar conductor adicional
   iii. Generar tarifa
   iv. Bloquear tarjeta de crédito
   v. Generar cobros


Cliente
Reserva
Categoria vehiculo
Cliente
Medio pago
Tarjeta
Crear reserva
Consultar categoria
alternativa
No se puede crear reserva para esta categoría
No hay vehiculos disponibles
Hay vehiculos disponibles
¿Conductores adicionales?
alternativa
Incluir conductores adicionales
Crear conductores adicionales
Conductores adicionales creados
No incluir conductores adicionales
Generar tarifa
Generar factura
Mostrar factura
Confirmar reserva
Realizar cobro 30%
Reserva hecha: cliente puede recoger carro
Cliente recoge carro
Se actualiza estado de reserva
Se actualiza inventario carros
Realizar cobro final