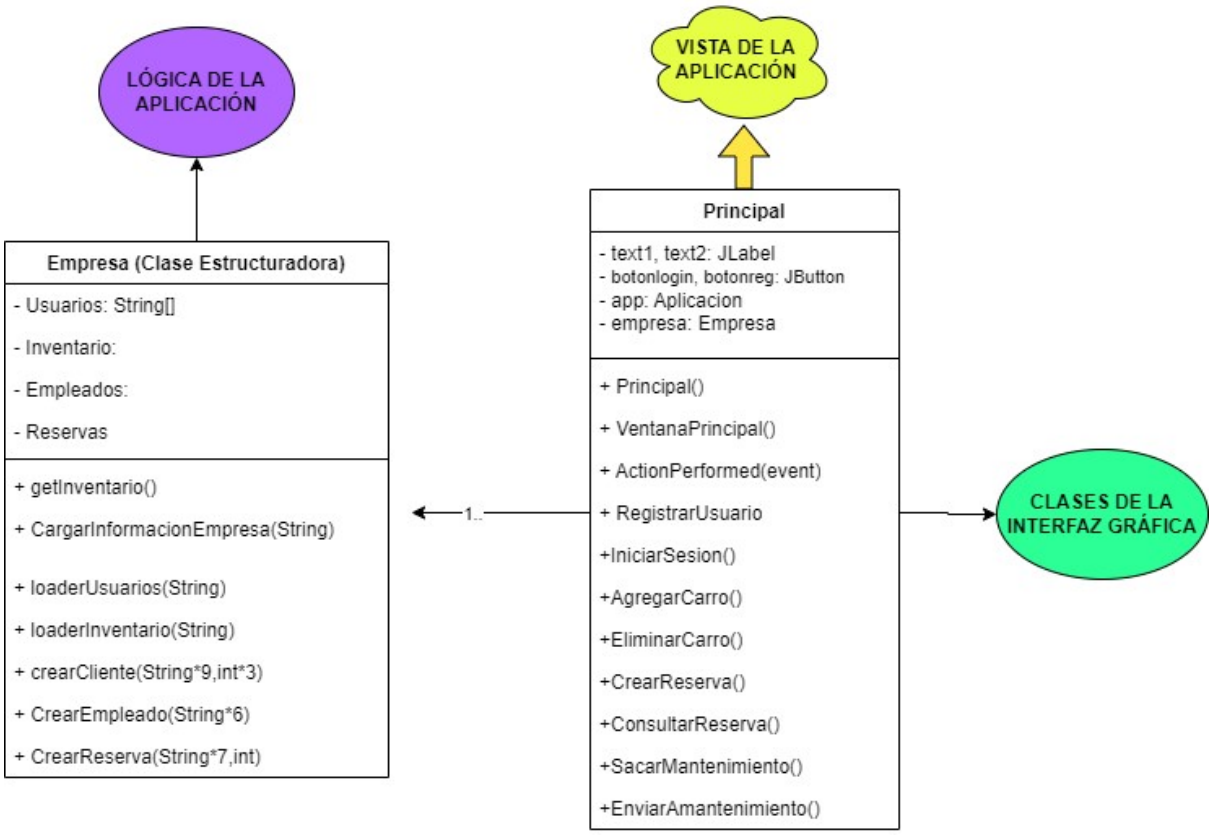
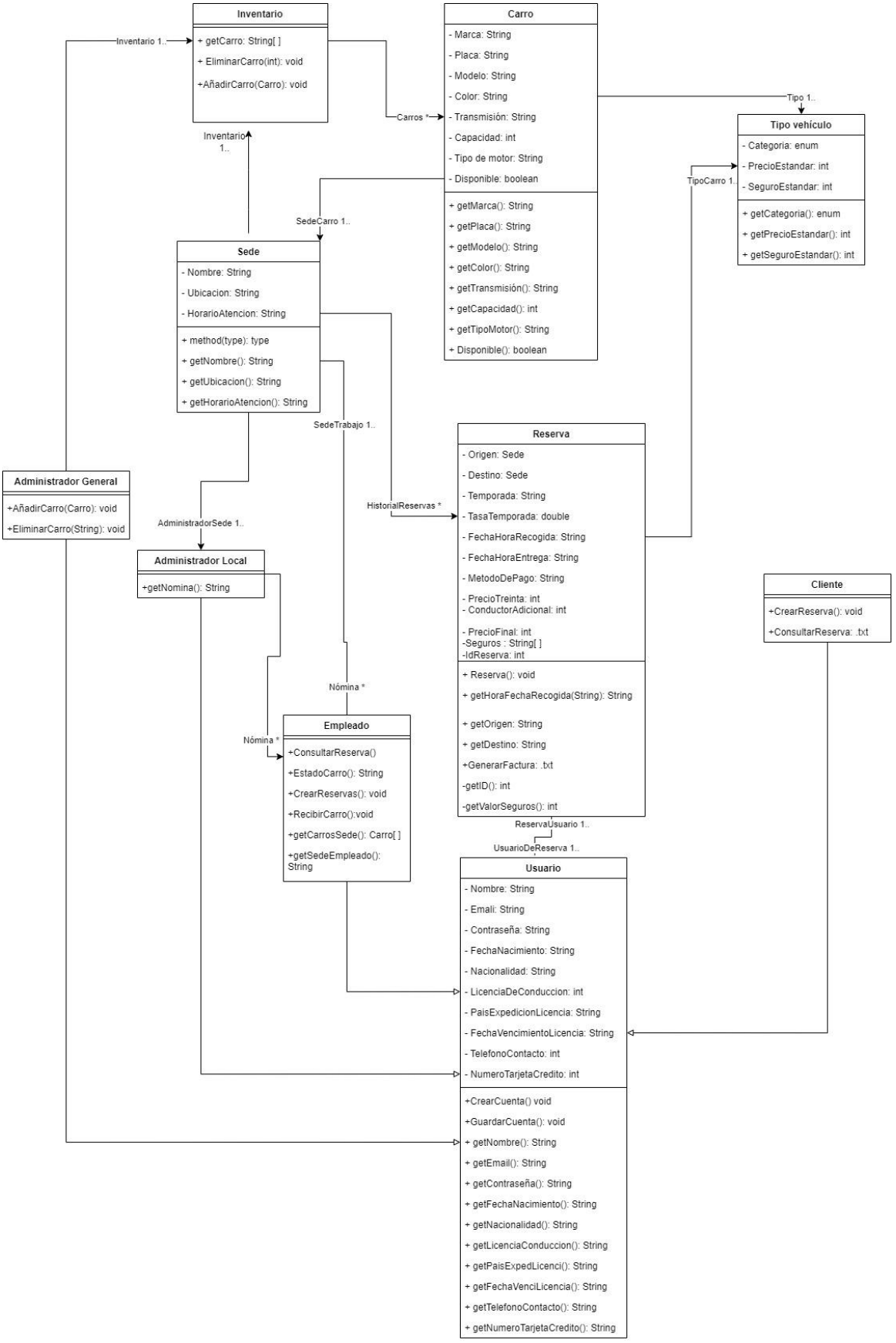
Grupo 6
Documento de Diseño
Proyecto 3

1. **Planeación del acoplamiento.**



En este diagrama planeamos la estructuración de la interfaz por medio de una clase “Principal”. Esta clase tiene los componentes interactivos necesarios para conectarse con la interfaz pero también es el eje entre la clase empresa y la interfaz. La clase empresa fue la clase estructuradora de la lógica dónde se aglomeran todos los métodos fundamentales para resolver los requerimientos funcionales de la aplicación. Adicionalmente, desde la clase Principal planeamos correr el programa con el fin de que sea el portal para mostrar consola con elementos gráficos de la aplicación.

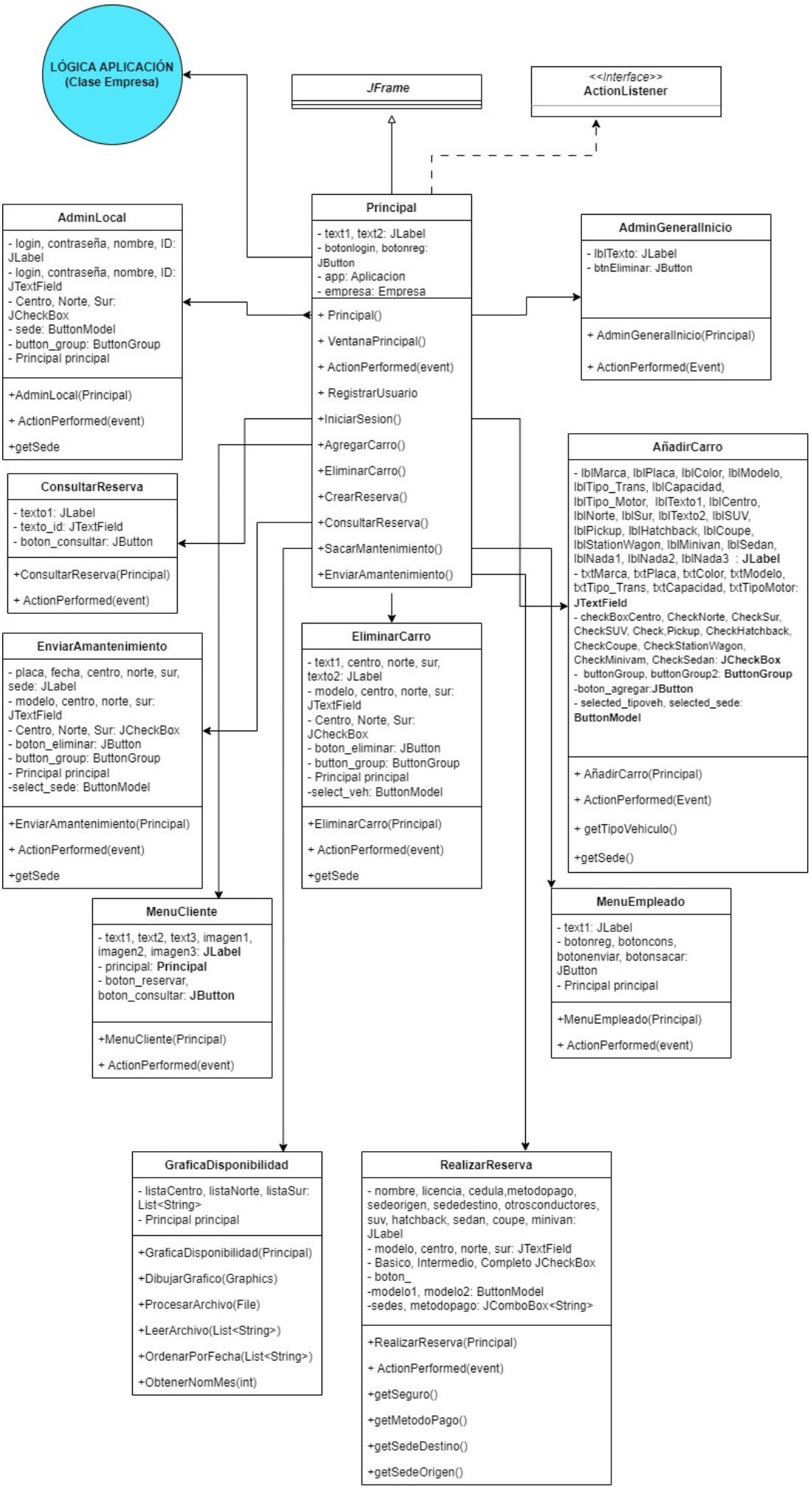
2. Diagrama de clases de la lógica del programa.



Este diagrama relaciona las clases relevantes entre los objetos que van a transmitir y gestionar la información de los requerimientos. Generalmente la clase que se relaciona con el problema le envía una solicitud a la consola, para que el usuario ingrese unos datos en la interfaz gráfica. Estos datos se envían a una clase que en la mayoría de las veces creará una instancia del objeto y lo guardará. Al tener esta información, la clase estructuradora se encarga de resolver cualquier lógica adicional necesaria del requerimiento y genera la funcionalidad para guardar la información en un archivo .txt.

3. **Diagrama de clases de la interfaz gráfica.**

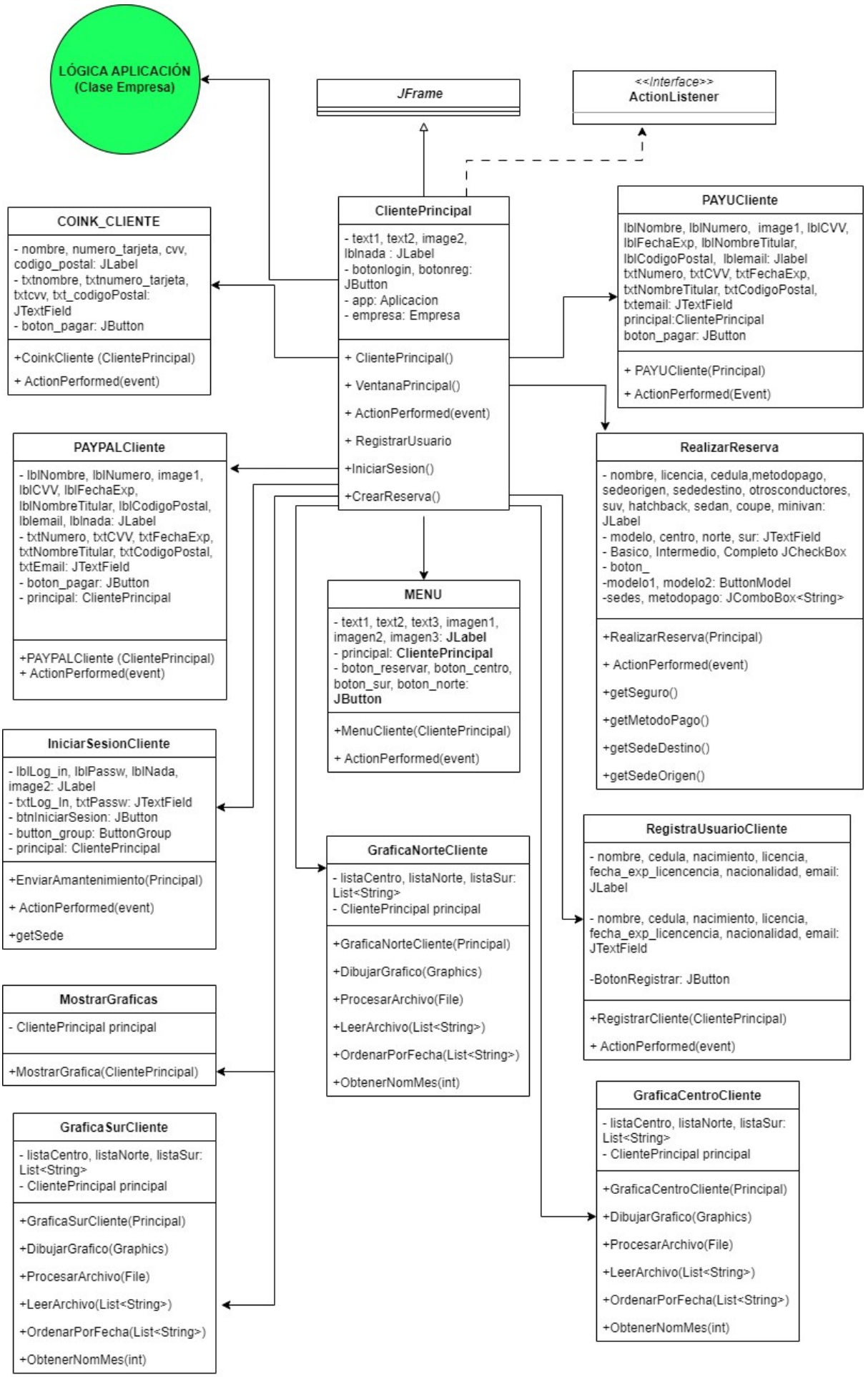
En este diagrama hicimos la planeación sobre cómo se estructuraría la interfaz gráfica del programa. Para establecer una conexión entre la lógica del programa y la interfaz creamos la clase Principal para invocar los métodos de la clase estructuradora, Empresa. Una vez con la lógica traída dentro de una sola clase podemos crear todas las demás clases que contendrán el código para cada una de las ventanas de la plataforma. Todas estas se conectan con la clase principal por medio de una instancia en cada una de ellas y tendrán los métodos necesarios para que el usuario interactúe. Particularmente, se establecen las implementaciones de la interfaz *ActionListener* para que cada uno de los elementos que requieran de respuesta por parte del programa trabajen. También, las clases heredan la lógica de la clase y librerías de *JFrame* donde se utilizan los componentes como paneles, iconos, botones, cuadros de texto y modelos para configurar la consola visualmente y hacer que toda la aplicación funcione.


LÓGICA APLICACIÓN
(Clase Empresa)
JFrame
<<Interface>>
ActionListener
Principal
- text1, text2: JLabel
- botonlogin, botonreg: JButton
- app: Aplicacion
- empresa: Empresa
+ Principal()
+ VentanaPrincipal()
+ ActionPerformed(event)
+ RegistrarUsuario
+IniciarSesion()
+AgregarCarro()
+EliminarCarro()
+CrearReserva()
+ConsultarReserva()
+SacarMantenimiento()
+EnviarAmantenimiento()
AdminLocal
- login, contraseña, nombre, ID: JLabel
- login, contraseña, nombre, ID: JTextField
- Centro, Norte, Sur: JCheckBox
- sede: ButtonModel
- button_group: ButtonGroup
- Principal principal
+AdminLocal(Principal)
+ ActionPerformed(event)
+getSede
AdminGeneralInicio
- lblTexto: JLabel
- btnEliminar: JButton
+ AdminGeneralInicio(Principal)
+ ActionPerformed(Event)
ConsultarReserva
- texto1: JLabel
- texto_id: JTextField
- boton_consultar: JButton
+ConsultarReserva(Principal)
+ ActionPerformed(event)
AñadirCarro
- lblMarca, lblPlaca, lblColor, lblModelo, lblTipo_Trans, lblCapacidad, lblTipo_Motor, lblTexto1, lblCentro, lblNorte, lblSur, lblTexto2, lblSUV, lblPickup, lblHatchback, lblCoupe, lblStationWagon, lblMinivan, lblSedan, lblNada1, lblNada2, lblNada3 : JLabel
- txtMarca, txtPlaca, txtColor, txtModelo, txtTipo_Trans, txtCapacidad, txtTipoMotor: JTextField
- checkBoxCentro, CheckNorte, CheckSur, CheckSUV, Check,Pickup, CheckHatchback, CheckCoupe, CheckStationWagon, CheckMinivam, CheckSedan: JCheckBox
- buttonGroup, buttonGroup2: ButtonGroup
-boton_agregar:JButton
- selected_tipoveh, selected_sede: ButtonModel
+ AñadirCarro(Principal)
+ ActionPerformed(Event)
+ getTipoVehiculo()
+getSede()
EnviarAmantenimiento
- placa, fecha, centro, norte, sur, sede: JLabel
- modelo, centro, norte, sur: JTextField
- Centro, Norte, Sur: JCheckBox
- boton_eliminar: JButton
- button_group: ButtonGroup
- Principal principal
-select_sede: ButtonModel
+EnviarAmantenimiento(Principal)
+ ActionPerformed(event)
+getSede
EliminarCarro
- text1, centro, norte, sur, texto2: JLabel
- modelo, centro, norte, sur: JTextField
- Centro, Norte, Sur: JCheckBox
- boton_eliminar: JButton
- button_group: ButtonGroup
- Principal principal
-select_veh: ButtonModel
+EliminarCarro(Principal)
+ ActionPerformed(event)
+getSede
MenuCliente
- text1, text2, text3, imagen1, imagen2, imagen3: JLabel
- principal: Principal
- boton_reservar, boton_consultar: JButton
+MenuCliente(Principal)
+ ActionPerformed(event)
MenuEmpleado
- text1: JLabel
- botonreg, botoncons, botonenviar, botonsacar: JButton
- Principal principal
+MenuEmpleado(Principal)
+ ActionPerformed(event)
GraficaDisponibilidad
- listaCentro, listaNorte, listaSur: List<String>
- Principal principal
+GraficaDisponibilidad(Principal)
+DibujarGrafico(Graphics)
+ProcesarArchivo(File)
+LeerArchivo(List<String>)
+OrdenarPorFecha(List<String>)
+ObtenerNomMes(int)
RealizarReserva
- nombre, licencia, cedula,metodopago, sedeorigen, sededestino, otrosconductores, suv, hatchback, sedan, coupe, minivan: JLabel
- modelo, centro, norte, sur: JTextField
- Basico, Intermedio, Completo JCheckBox
- boton_
-modelo1, modelo2: ButtonModel
-sedes, metodopago: JComboBox<String>
+RealizarReserva(Principal)
+ ActionPerformed(event)
+getSeguro()
+getMetodoPago()
+getSedeDestino()
+getSedeOrigen()

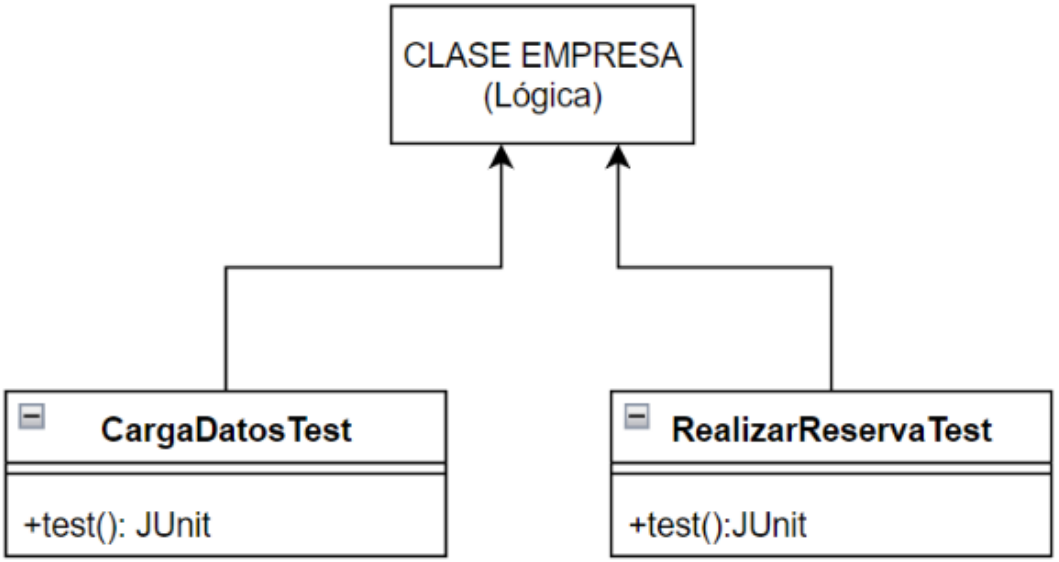
4. **Responsabilidades**

Responsabilidad	Rol
Registrar la información básica de un usuario para poder almacenarla en la base de usuarios.	Creación de usuarios
Reservar toda la información básica de los usuarios así como utilizarla para el posterior log-in de cada persona	Almacenador de usuarios
Crear una reserva de vehículo en la aplicación	Generador reserva
Ajustar el precio a pagar en la reserva conforme los seguros adquiridos y las tarifas por temporadas de la empresa	Caracterizador precio reserva
Permitir al empleado saber en que estado esta un vehículo y conocer su disponibilidad para el efecto de una reserva	Rastreador estado de vehículos
Mantener la persistencia de la aplicación, donde toda la información registrada quede procesada en facturas.	Archivador facturas
Permitir al administrador conocer la nómina de cada sede y tratar a los empleados como un usuario de la aplicación	Gestión de empleados
Permitir que el administrador general gestione el inventario de la empresa	Gestión de inventario
Contener toda la información acerca de los vehículos de la empresa.	Almacenador de inventario
Fuente de información de todos los vehículos existentes en la empresa junto con sus características respectivas	Base de datos

5. Diagrama de la Aplicación (SOLO CLIENTE)


LÓGICA APLICACIÓN
(Clase Empresa)
JFrame
<<Interface>>
ActionListener
COINK_CLIENTE
- nombre, numero_tarjeta, cvv, codigo_postal: JLabel
- txtnombre, txtnumero_tarjeta, txtcvv, txt_codigoPostal: JTextField
- boton_pagar: JButton
+CoinkCliente (ClientePrincipal)
+ ActionPerformed(event)
ClientePrincipal
- text1, text2, image2, lblnada : JLabel
- botonlogin, botonreg: JButton
- app: Aplicacion
- empresa: Empresa
+ ClientePrincipal()
+ VentanaPrincipal()
+ ActionPerformed(event)
+ RegistrarUsuario
+IniciarSesion()
+CrearReserva()
PAYUCliente
lblNombre, lblNumero, image1, lblCVV, lblFechaExp, lblNombreTitular, lblCodigoPostal, lblemail: Jlabel
txtNumero, txtCVV, txtFechaExp, txtNombreTitular, txtCodigoPostal, txtemail: JTextField
principal:ClientePrincipal
boton_pagar: JButton
+ PAYUCliente(Principal)
+ ActionPerformed(Event)
PAYPALCliente
- lblNombre, lblNumero, image1, lblCVV, lblFechaExp, lblNombreTitular, lblCodigoPostal, lblemail, lblnada: JLabel
- txtNumero, txtCVV, txtFechaExp, txtNombreTitular, txtCodigoPostal, txtEmail: JTextField
- boton_pagar: JButton
- principal: ClientePrincipal
+PAYPALCliente (ClientePrincipal)
+ ActionPerformed(event)
RealizarReserva
- nombre, licencia, cedula,metodopago, sedeorigen, sededestino, otrosconductores, suv, hatchback, sedan, coupe, minivan: JLabel
- modelo, centro, norte, sur: JTextField
- Basico, Intermedio, Completo JCheckBox
- boton_
-modelo1, modelo2: ButtonModel
-sedes, metodopago: JComboBox<String>
+RealizarReserva(Principal)
+ ActionPerformed(event)
+getSeguro()
+getMetodoPago()
+getSedeDestino()
+getSedeOrigen()
MENU
- text1, text2, text3, imagen1, imagen2, imagen3: JLabel
- principal: ClientePrincipal
- boton_reservar, boton_centro, boton_sur, boton_norte: JButton
+MenuCliente(ClientePrincipal)
+ ActionPerformed(event)
IniciarSesionCliente
- lblLog_in, lblPassw, lblNada, image2: JLabel
- txtLog_In, txtPassw: JTextField
- btnIniciarSesion: JButton
- button_group: ButtonGroup
- principal: ClientePrincipal
+EnviarAmantenimiento(Principal)
+ ActionPerformed(event)
+getSede
GraficaNorteCliente
- listaCentro, listaNorte, listaSur: List<String>
- ClientePrincipal principal
+GraficaNorteCliente(Principal)
+DibujarGrafico(Graphics)
+ProcesarArchivo(File)
+LeerArchivo(List<String>)
+OrdenarPorFecha(List<String>)
+ObtenerNomMes(int)
RegistraUsuarioCliente
- nombre, cedula, nacimiento, licencia, fecha_exp_licencencia, nacionalidad, email: JLabel
- nombre, cedula, nacimiento, licencia, fecha_exp_licencencia, nacionalidad, email: JTextField
-BotonRegistrar: JButton
+RegistrarCliente(ClientePrincipal)
+ ActionPerformed(event)
MostrarGraficas
- ClientePrincipal principal
+MostrarGrafica(ClientePrincipal)
GraficaCentroCliente
- listaCentro, listaNorte, listaSur: List<String>
- ClientePrincipal principal
+GraficaCentroCliente(Principal)
+DibujarGrafico(Graphics)
+ProcesarArchivo(File)
+LeerArchivo(List<String>)
+OrdenarPorFecha(List<String>)
+ObtenerNomMes(int)
GraficaSurCliente
- listaCentro, listaNorte, listaSur: List<String>
- ClientePrincipal principal
+GraficaSurCliente(Principal)
+DibujarGrafico(Graphics)
+ProcesarArchivo(File)
+LeerArchivo(List<String>)
+OrdenarPorFecha(List<String>)
+ObtenerNomMes(int)

6. **Diagrama UML Pruebas JUnit**



7. **Análisis desarrollo proyecto**

- **¿Qué cosas salieron bien y qué cosas salieron mal?**

Inicialmente, durante las 3 iteraciones del proyecto tuvimos varios aciertos para que como equipo logramos sacar exitosamente el programa basado en el alquiler de carros de manera tecnológica e innovadora. Antes de comenzar el proyecto fue fundamental entender y conocer el nuevo lenguaje al que nos estábamos enfrentando, pues la programación en java orientada a objetos era nueva para todos los del grupo, para ello la realización de los talleres fue crucial para sentar bases y así empezar de manera satisfactoria el proyecto. El problema era claro desde el principio y entre el equipo tuvimos reuniones fructuosas que permitían desarrollar las diferentes ideas de los integrantes y así obtener una primera iteración muy acertada. Consideramos que estas reuniones ayudaron mucho en el progreso de la aplicación, por lo tanto continuamos haciéndolas para concretar y finalizar las últimas iteraciones. Adicionalmente, opinamos que todos los integrantes teniamos manejo del tema, lo que simplificaba la tarea a la hora de plantear soluciones frente a los problemas establecidos.

Por otra parte, existen aspectos en donde el grupo pudo haber mejorado; el documento de análisis de la primera iteración pudo ser mejor, pues faltó incluir las

características de cada categoría de carro, así como las tarifas de los seguros, conductores adicionales y métodos de pago. Sin embargo, estos pequeños fallos fueron arreglados e incluidos en la siguiente iteración. Otro aspecto que pudo haber sido mejorable fue la elaboración del boceto de la interfaz gráfica del proyecto, ya que faltó incluir algunas ventanas que eran necesarias para el correcto desarrollo de la aplicación.

No obstante, estamos satisfechos con el trabajo realizado durante las 3 iteraciones, por lo tanto, las cosas que salieron bien opacaron a las que salieron mal, siendo estas últimas muy mínimas.

- ¿Qué decisiones resultaron acertadas y qué decisiones fueron problemáticas?

Para la realización del proyecto tuvimos que tomar varías decisiones. En primer lugar, tuvimos que definir las estructuras de datos que íbamos a usar para la realización de los requerimientos. En nuestro caso utilizamos hash maps para manejar la información de los usuarios, empleados, reservas e inventario. A grandes rasgos, esta decisión no nos presentó inconvenientes, ya que logramos definir bien los métodos para trabajar con dichas estructuras y presentar los datos correctamente a la hora de entregar el proyecto.

Por otro lado, para manejar las versiones del proyecto utilizamos Git, esta herramienta se nos hizo de gran ayuda para no perder el trabajo previamente realizado y por ende las entregas 1, 2 y 3 eran coherentes entre sí. No obstante, antes de empezar a programar tuvimos que hacer un paso de previo de abstracción, durante esta etapa tuvimos que pensar en el qué mas no en el cómo, es decir, imaginar una solución del proyecto sin pensar en detalles de la implementación, durante este paso no tuvimos en cuenta aspectos importante como el uso genérico de los usuarios de la aplicación (cliente, empleado y administradores), las tarifas de alquiler (temporada alta y baja), los métodos de pago, conductores adicionales y los seguros.

Estos detalles causaron que a la hora de implementar la solución tuviéramos que replantear nuestra solución. Por ejemplo, tuvimos que replantear nuestra clase controladora, que es donde tenemos las funciones que modifican la información del inventario y la consulta de reservas.

-¿Qué tipo de problemas tuvieron durante el desarrollo de los proyectos y a qué se debieron?

Iniciar este proyecto significó un gran reto para nosotros. Aproximarnos por primera vez a la programación orientada a objetos desde la entrega uno nos puso a pensar con una lógica nueva y más compleja. Además, el IDE de Eclipse también fue un ambiente nuevo de desarrollo donde tuvimos que adaptarnos rápido a la nueva tecnología. Conforme avanzamos en los proyectos entendimos mejor la abstracción requerida para resolver problemas y poner en práctica soluciones más eficientes. Un reto grande consistió en plasmar las decisiones y estrategias que planeamos en los diagramas UML y documentos de diseño en la implementación de la aplicación; nos dimos cuenta que el proceso era iterativo y no siempre salía como se esperaba. Al iniciar la materia era incierto sobre cómo debíamos organizar los proyectos y como entender los errores que nos daba el programa constantemente Unos aspectos claros que concluimos que son necesarios para la programación orientada a objetos son:

- Tener un entendimiento claro del problema y con base en ello planear un diseño que sea claro y sencillo
- Ser cuidadosos en pensar las abstracciones. ¿Qué objetos o elementos son claves para el contexto? ¿Qué deberían hacer dichos elementos?
- Mantener una comunicación efectiva entre el grupo durante el proceso de ideación y desarrollo
- Utilizar la herencia para apalancarse de métodos que estén agrupados en un componente y mantener la cohesión
- Hacer uso de una clase estructuradora que sea de utilidad para acoplar el programa

Estos aspectos y muchos más fueron los que aprendimos sobre la marcha y por medio de horas de trabajo entendimos que nos harían la vida más fácil para desarrollar software en esta materia. Es importante seguir trabajando en la aplicación de pruebas para cada vez hacer programas más sólidos y profesionales.

-