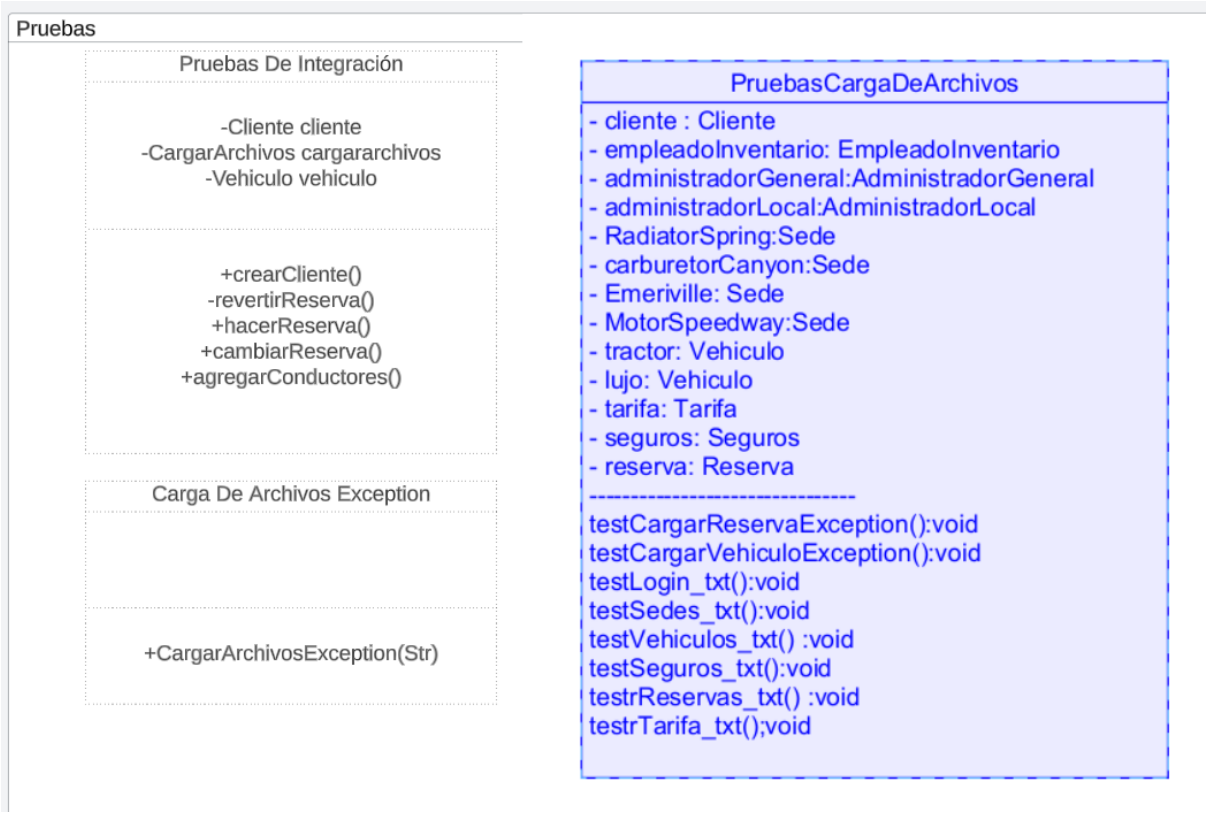
Daniel Humberto Velez Rojas - 202221703
Eric Sebastian Alarcón - 202220287
Andres Botero Ruiz - 202223503
Manolo Hernández Rojas - 202224469

Adiciones:
En este proyecto 3 debemos realizar adiciones más no cambios. Ahora debemos emitir las facturas en formato pdf en el momento que se entrega el vehículo y se paga el servicio. También debemos agregar características adicionales al móvil, tales como: tipo de vehículo y porcentaje adicional de la prima de vehículos, por otra parte hay que integrar una pasarela de pagos simulada que acepte tarjetas de crédito. Por último una aplicación exclusiva para cliente y tener pruebas automatizadas.

Diseño de pruebas:
Realizamos las pruebas automatizadas en base a los principios de diseño que vimos en clase. Primero vimos que las pruebas de regresión se aplicarán por las secciones más importantes del código (refiriéndonos a requerimiento y relaciones) utilizamos Junit como el medio para poner a correr las pruebas.

UML prueba de carga de archivos:



Explicación de las pruebas:

Pruebas de integración:
Se prueban tres funcionalidades de hacer una reserva: Iniciar una reserva, cambiar información de una reserva ya existente, y agregar conductores adicionales a una reserva. Hay un BeforeEach que crea siempre al mismo cliente (Guido) que hace todas las reservas, y también hay un @AfterEach que revierte cada reserva

Pruebas de Carga de Datos:
Cargan todos los documentos de texto para convertirlos en instancias de objetos, luego se consulta información sobre dichos objetos para verificar que se cargaron bien. También se prueban dos excepciones, ambas de cuando se pide que se carga un objeto que no existe en el documento de texto con la excepcion CargaArchivosException.java

Manejo de Errores:
Para manejar los errores decidimos seguir la siguiente implementación: Se intenta ejecutar algo en la interfaz, la interfaz le pide al modelo y si el modelo retorna un error, la interfaz mostrará un mensaje de error al usuario sin que la aplicación se apague.

A continuación se encuentra una lista con todos los posibles casos de errores que maneja la aplicación:

Para todos los tipos de usuario y en login:

- Si no se llenan todas las casillas con datos

Dentro de Cliente:

Reserva:
- Si no hay carros disponibles
- Si tiene bloqueada la tarjeta
- Si la fecha no está en el formato indicado
- Si la hora no está dentro de los horarios hábiles para la sede

Alquiler:
- Si la reserva no existe
- Si la hora de devolución no está en los horarios disponibles
- Si se entrega en la sede diferente a la indicada

Entregar Vehículo:
- Si la placa no existe
- Si el ID de reserva no existe
- Si no exista la placa y no exista el ID de reserva
- Si se entrega en la sede diferente a la indicada

Dentro de Administrador Local:

Modificar Trabajador
- Si no existe trabajador

Dentro de Admin General:

Revisar Mantenimiento
- Si no hay vehículos que revisar pero se confirma el enviar a mantenimiento
  Dar de baja y generar log
- Si la placa no existe

Dentro de Empleado Mostrador:

- Si no existe el vehículo al mandar a revisar

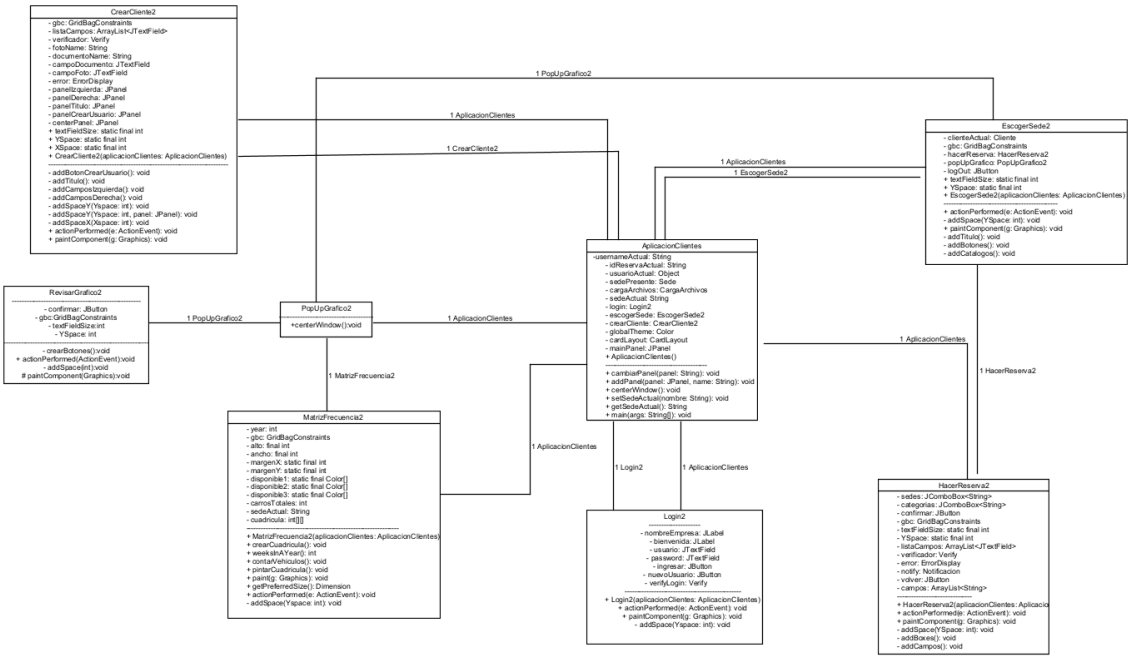
Dentro de Empleado Inventario:

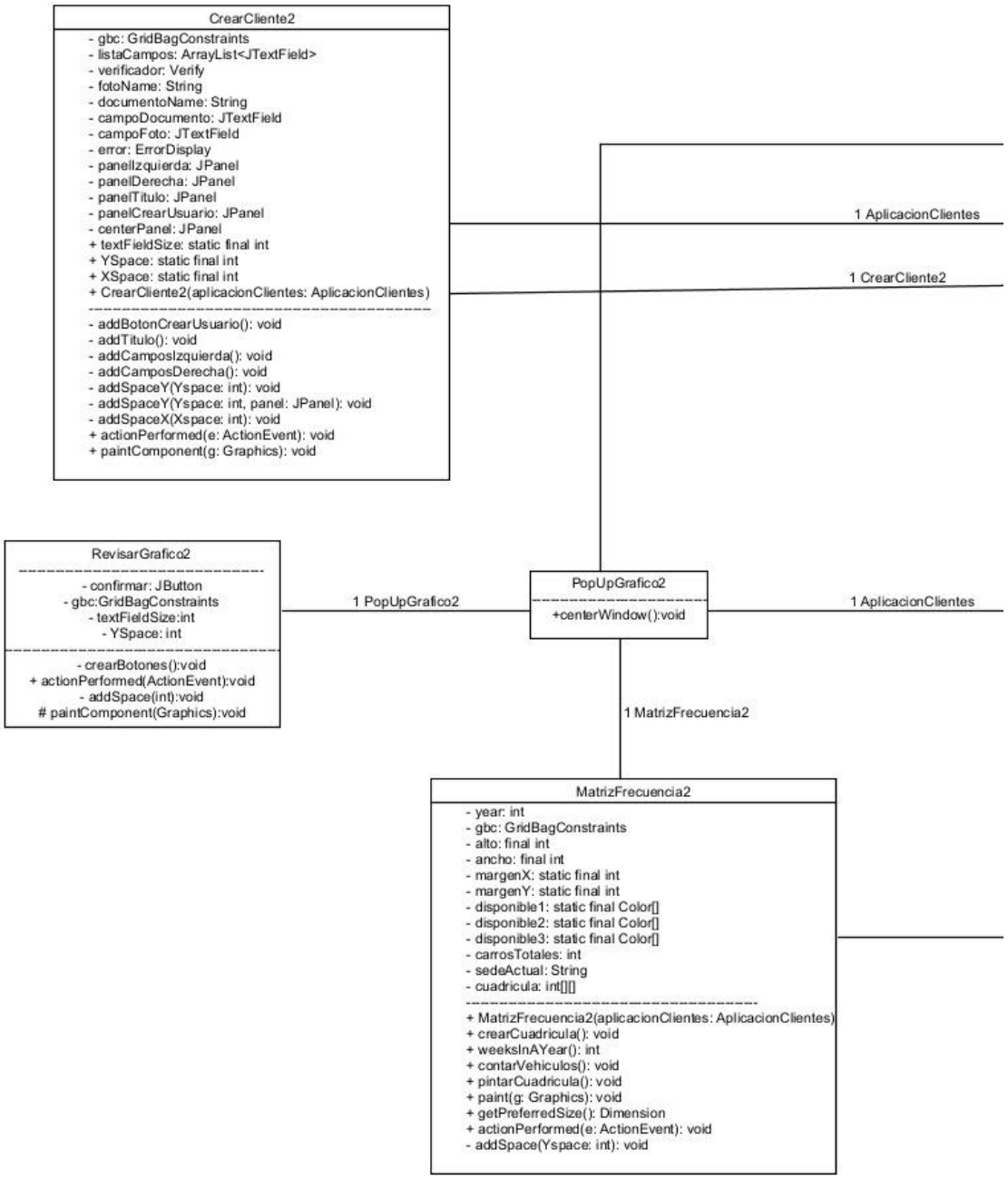
- Si no hay vehículos que revisar pero se confirma el enviar a mantenimiento

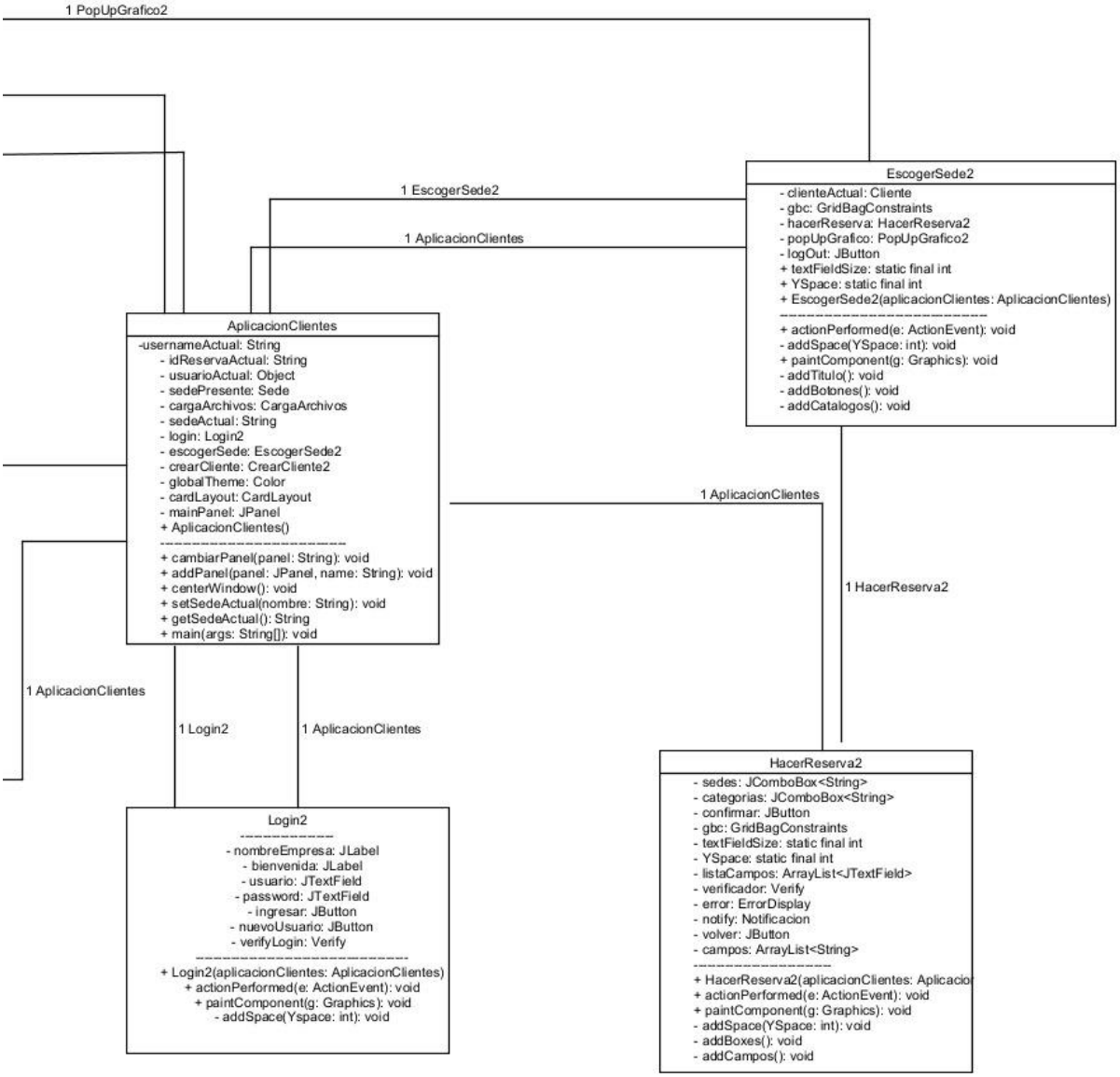
Dentro de Login

- Si al momento de iniciar sesión no existen las credenciales escritas

UML Aplicación cliente:




CrearCliente2
- gbc: GridBagConstraints
- listaCampos: ArrayList<JTextField>
- verificador: Verify
- fotoName: String
- documentoName: String
- campoDocumento: JTextField
- campoFoto: JTextField
- error: ErrorDisplay
- panelIzquierda: JPanel
- panelDerecha: JPanel
- panelTitulo: JPanel
- panelCrearUsuario: JPanel
- centerPanel: JPanel
+ textFieldSize: static final int
+ YSpace: static final int
+ XSpace: static final int
+ CrearCliente2(aplicacionClientes: AplicacionClientes)
- addBotonCrearUsuario(): void
- addTitulo(): void
- addCamposIzquierda(): void
- addCamposDerecha(): void
- addSpaceY(Yspace: int): void
- addSpaceY(Yspace: int, panel: JPanel): void
- addSpaceX(Xspace: int): void
+ actionPerformed(e: ActionEvent): void
+ paintComponent(g: Graphics): void
1 AplicacionClientes
1 CrearCliente2
RevisarGrafico2
- confirmar: JButton
- gbc:GridBagConstraints
- textFieldSize:int
- YSpace: int
- crearBotones():void
+ actionPerformed(ActionEvent):void
- addSpace(int):void
paintComponent(Graphics):void
1 PopUpGrafico2
PopUpGrafico2
+centerWindow():void
1 AplicacionClientes
1 MatrizFrecuencia2
MatrizFrecuencia2
- year: int
- gbc: GridBagConstraints
- alto: final int
- ancho: final int
- margenX: static final int
- margenY: static final int
- disponible1: static final Color[]
- disponible2: static final Color[]
- disponible3: static final Color[]
- carrosTotales: int
- sedeActual: String
- cuadricula: int[][]
+ MatrizFrecuencia2(aplicacionClientes: AplicacionClientes)
+ crearCuadricula(): void
+ weeksInAYear(): int
+ contarVehiculos(): void
+ pintarCuadricula(): void
+ paint(g: Graphics): void
+ getPreferredSize(): Dimension
+ actionPerformed(e: ActionEvent): void
- addSpace(Yspace: int): void



En la parte superior se encuentra el diagrama de la nueva clase a implementar que se nos pedía llamada AplicacionClientes.

Aplicación PDF:
Para este requerimiento se usó la librería iText. Esta genera una PDF a partir del método .add(new Paragraph (“ ”)) que añade el texto al documento y .add(new Image(ImageDataFactory.create(imagePath))); para añadir la firma en foto como se muestra en la imagen de abajo.

```java
public void generarPdf(String idReserva,String placa, String sedeActual,String nombre) {
        try {
                File folder = new File("./facturas/");
                folder.mkdirs();
    PdfWriter writer = new PdfWriter("./facturas/"+idReserva+"-EntregaVehiculo.pdf");
    PdfDocument pdf = new PdfDocument(writer);
    Document document = new Document(pdf);

    document.add(new Paragraph("=======================FACTURA========================="));
    document.add(new Paragraph("Nombre de la persona que lo devuelve: "+ nombre));
    document.add(new Paragraph("id de la Reserva: "+ idReserva));
    document.add(new Paragraph("carro devuelto: " + placa));
    document.add(new Paragraph("Sede en la que se devolvió "+ sedeActual));
    document.add(new Paragraph("Este documento es valido para cualquier reclamo"));
    document.add(new Paragraph("Firma del Administrador General:"));

    String imagePath = "./imagenes/firma.png";
    Image img = new Image(ImageDataFactory.create(imagePath));
    document.add(img);

    document.close();

} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (MalformedURLException e) {
                e.printStackTrace();
        }
}
```

Pagos Con Tarjeta:

Para la nueva funcionalidad de pagos mediante pasarelas creamos una funcionalidad en donde dependiendo la pasarela elegida, se crea una nueva clase de esa pasarela para manejar la información y mostrarla en la interfaz desde la clase
+Pagos(String, double, String, boolean)

```java
try {

        Class clase = Class.forName("RentadoraModelo."+pasarela);
        this.pasarela = (MetodoTransaccion) clase.getDeclaredConstructor(null).newInstance(null);
```