Diseño DPOO Entrega 3 - Proyecto 3

*La entrega se está realizando sobre el documento previamente enviado**

Santiago Cabra Chavez 202110929
Manuel Gomez Salazar 202020415
Santiago Linares 201915789

Ventana de inicio:
En la clase VentanaDeInicio del paquete interfaz es dónde se ubica el main de la aplicación. En el main se inicia el panel de login y se cargan los datos para el modelo. El modelo está conectado a cada una de las clases ya que se usó lo hecho en el proyecto 1 para completar este proyecto.

PanelLogin:
En el panel login se hace la verificación de los datos de cada usuario y se identifica si ese usuario es administrador, recepcionista o empleado. Dependiendo del tipo de usuario se genera el menú correspondiente a ese usuario.

VentanaEmpleado:
En esta ventana están los botones correspondientes a las funciones de empleado los cuales son: agregar servicio a habitación y agregar producto a habitación. Para cada uno de esos botones se genera una nueva ventana en la cual se registra la habitación a la cual se le quiere agregar el producto o el servicio y le agrega ese servicio o producto. En caso de que no se encuentre la habitación o no se llene el campo de la habitación se le pide al usuario que llene el texto o cambie la habitación que se ingresó.

VentanaAdministrador:
En esta ventana están los botones correspondientes a las funciones del administrador los cuales son: nueva habitación, nuevo producto, nuevo servicio, nuevo usuario, eliminar producto, eliminar servicio, y eliminar usuario. Cada uno de estos botones abre una ventana nueva en la cual se le pide al administrador los datos pertinentes para poder agregar o eliminar los servicios o productos que se deseen.

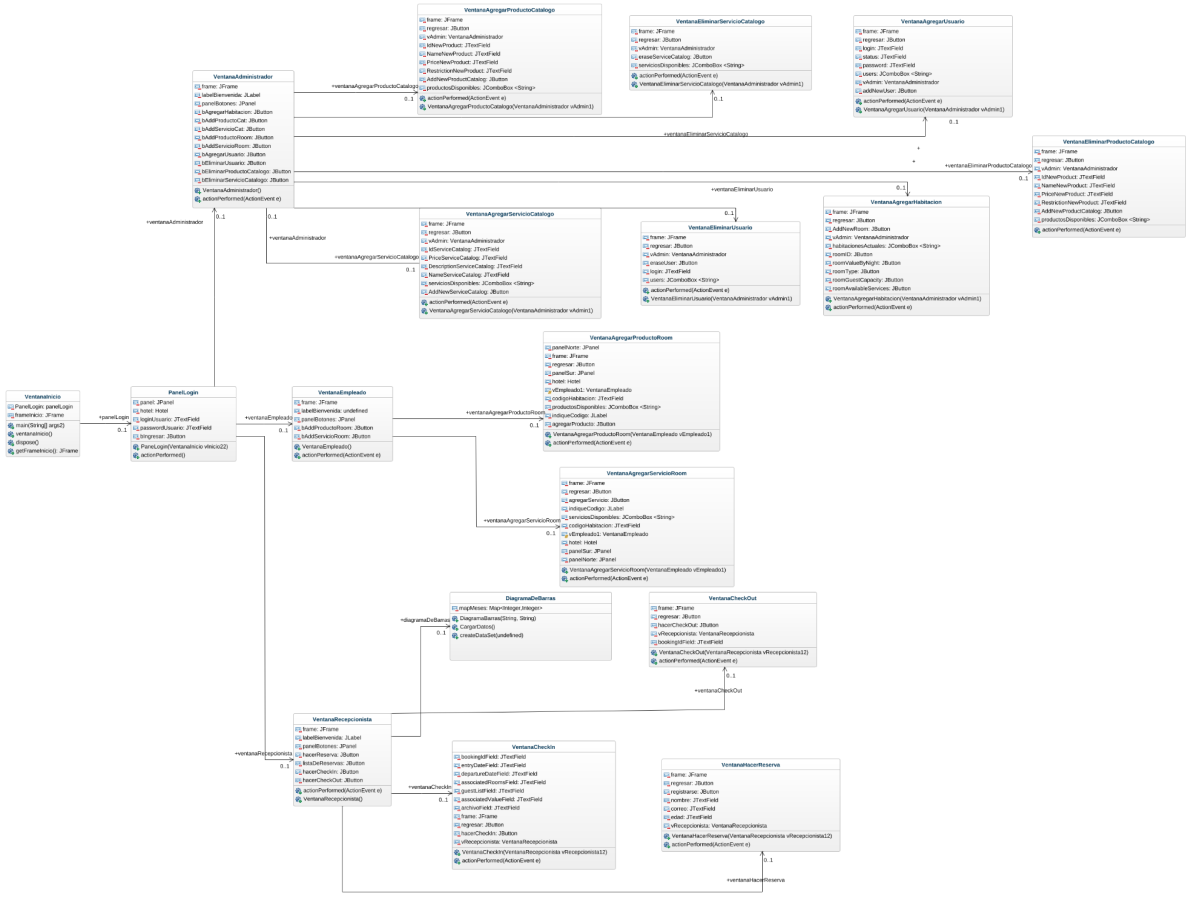
DiagramaBarras:
Esta es la clase que genera el diagrama que muestra la ocupación por los meses del hotel, esta opción se accede desde la VentanaRecepcionista. Para lograr esto se usó JFreeChart. Los

datos para graficar se obtuvieron leyendo el archivo de reservas y guardandolos en un mapa. Por cada reserva que se iniciaba en un mes se le agregaba un 1 a la llave correspondiente a ese mes en el Hash Map. Posteriormente se le asigna el valor de cada mes a la columna correspondiente y se grafica en una ventana aparte.

EstadisticasProductos: al Igual que la ventana de ocupación de habitaciones en el año. se genera un diagrama donde se analiza la cantidad de productos disponibles dentro del hotel.

EstadisticasServicios: Se crea un diagrama con JFreeChart para ver la cantidad de servicios ofrecidos en el Hotel.

Diagrama de Clases de alto Nivel para la interfaz gráfica



*Si existe dificultad para la lectura del diagrama se hara un envio del archivo original aparte**

Carga de datos:

La lectura de los datos se hizo desde la clase Hotel.

```
private static Map<Integer, Habitacion> mapaHabitaciones = new LinkedHashMap<>();//
private static Map<Integer, Producto> mapaProductos = new LinkedHashMap<>();//
                                                                            //
private static Map<Integer, Servicio> mapaServicios = new LinkedHashMap<>();//
                                                                            //
private static Map<Integer, Reserva> mapaReservas = new LinkedHashMap<>();//

private static Map<String, Usuario> mapaUsuarios = new LinkedHashMap<>();//
```

Se inicia creando Hash maps en los cuales se va a actualizar la información de la lectura de los textos. Los hashmaps fueron creados para que las llaves fueran el id del objeto (idHabitacion, idProducto, idServicio, idReserva, loginUsuario) y el valor sea el objeto correspondiente. Se optaron por hash maps para que cuando el recepcionista, empleado o administrador tenga que acceder a algún objeto, este pueda hacerlo de una forma rápida y digitando

Cada una de las funciones de carga de datos (cargarServicios, cargarProductos, cargarUsuarios, cargarHabitaciones, cargarReservas) toma el archivo .txt correspondiente, separa los datos de cada linea por (;) y crea el objeto con los datos del txt.

Hotel:

```
public class Hotel {
    private static Map<Integer, Habitacion> mapaHabitaciones = new LinkedHashMap<>();//
    private static Map<Integer, Producto> mapaProductos = new LinkedHashMap<>();//
                                                                                //
    private static Map<Integer, Servicio> mapaServicios = new LinkedHashMap<>();//
                                                                                //
    private static Map<Integer, Reserva> mapaReservas = new LinkedHashMap<>();//

    private static Map<String, Usuario> mapaUsuarios = new LinkedHashMap<>();//
```

Para empezar, creamos el Hotel con 5 mapas diferentes, un mapa de habitaciones, uno de productos, uno de servicios, uno con las reservas del hotel y el ultimo con los usuarios que tienen acceso al hotel.

```
public static void cargarInfoHotel() throws FileNotFoundException, IOException {
    cargarHabitaciones();
    cargarProductos();
    cargarServicios();
    cargarReservas();
    cargarUsuarios();
```

Luego cargamos los datos al hotel.

Para las funciones de cargar datos de los mapas que conforman el hotel usamos hashMaps los cuales guardan la información del correspondiente archivo .txt donde se este guardando la informacion, esta carga de datos se tiene que hacer cada vez que se inicie el hotel.

```java
public static void cargarProductos() throws FileNotFoundException, IOException {// Función que va a crear un hash
                                                                                 // map de los productos que se van a
                                                                                 // ofrecer en el hotel. Se carga a
                                                                                 // partir de un archivo .txt que
                                                                                 // está en la carpeta de data
    try (BufferedReader brProductos = new BufferedReader(new FileReader("./data/productos.txt"))) {
        String linea = "";
        System.out.println("Se empezaron a cargar los productos");
        while ((linea = brProductos.readLine()) != null) {
            String[] partes = linea.split(";"); // Separa la linea por los ;
            int productId = Integer.parseInt(partes[0]);
            String productName = partes[1];
            int productValue = Integer.parseInt(partes[2]);
            String productRestrictions = partes[3];
            Producto producto = mapaProductos.get(productId);
            if (producto == null) {
                producto = new Producto(productId, productValue, productName, productRestrictions);
                mapaProductos.put(productId, producto);
            }
        }
    } catch (NumberFormatException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    System.out.println("Se cargaron los productos");
}
```

Usuarios:

```java
public abstract class Usuario
{
    private String login;
    private String password;
    private String userType;

    public Usuario(String login, String password, String userType)
    {
        this.login = login;
        this.password = password;
        this.userType = userType;
    }

    public String getLogin()
    {
        return login;
    }

    public String getPassword()
    {
        return password;
    }

    public String getUserType()
    {
        return userType;
    }
```

La Interfaz de usuario es la base de las clases de: Administrador, empleado y recepcionista. donde se crean con un nombre o LogIn, una contraseña y el tipo de usuario que es (esto es importante debido a que cada usuario tiene acceso a funciones distintas).

Administrador:

El administrador tiene varias funciones unicas,

+ addProductCatalog(): Void. El administrador va a poder agregar un producto con su precio respectivo al catálogo para que los clientes del hotel puedan usarlo.

```java
public void addProductCatalog(Map<Integer, Producto> products, Integer id, String name, Integer value,
        String textFile, String locationRestrictions) {
    // Crear objeto Product con los datos recibidos
    Producto newProduct = new Producto(id, value, name, locationRestrictions);

    // Agregar producto al Map
    products.put(id, newProduct);

    // Escribir datos en archivo de texto
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(textFile, true))) {
        writer.newLine();
        writer.write(id + ";" + value + ";" + name + ";" + locationRestrictions);
        System.out.println("Producto agregado existosamente");

    } catch (IOException e) {
        System.err.println("Error al escribir en archivo: " + e.getMessage());
    }

}
```

+ addServiceCatalog(): Void. El administrador va a poder agregar un servicio con su precio respectivo al catálogo para que los clientes del hotel puedan usarlo.

```java
public void addServiceCatalog(Map<Integer, Servicio> services, Integer id, String name, Integer value,
        String textFile, String description) {
    // Crear objeto Product con los datos recibidos
    Servicio newService = new Servicio(id, name, value, description);

    // Agregar producto al Map
    services.put(id, newService);

    // Escribir datos en archivo de texto
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(textFile, true))) {
        writer.newLine();
        writer.write(id + ";" + value + ";" + name + ";" + description);
        System.out.println("Servicio agregado existosamente");

    } catch (IOException e) {
        System.err.println("Error al escribir en archivo: " + e.getMessage());
    }

}
```

+ deleteProductCatalog(): Void. El administrador va a poder eliminar un producto del catálogo del hotel.

```java
public void deleteProductCatalog(Map<Integer, Producto> products, Integer id, String textFile) {
    // Obtener el producto correspondiente al ID

    Producto deletedProduct = products.get(id);

    // Si el producto existe en el Map

    if (deletedProduct != null) {
        // Eliminar producto del Map

        products.remove(id);

        // Eliminar línea correspondiente al producto en el archivo de texto

        try (BufferedReader reader = new BufferedReader(new FileReader(textFile));
             BufferedWriter writer = new BufferedWriter(new FileWriter(textFile + ".temp"))) {
            String currentLine;
            while ((currentLine = reader.readLine()) != null) {
                String[] fields = currentLine.split(";");
                if (!fields[0].equals(String.valueOf(id))) {
                    writer.write(currentLine);
                    writer.newLine();
                }
            }
        } catch (IOException e) {
            System.err.println("Error al eliminar producto: " + e.getMessage());
        }

        // Renombrar archivo temporal y borrar el original

        try {
            java.nio.file.Files.move(java.nio.file.Paths.get(textFile + ".temp"), java.nio.file.Paths.get(textFile),
                    java.nio.file.StandardCopyOption.REPLACE_EXISTING);
        } catch (IOException e) {
            System.err.println("Error al renombrar archivo temporal: " + e.getMessage());
        }
    } else {
        System.out.println("El producto con ID " + id + " no existe en el Map.");
    }
}
```

+ deleteServiceCatalog(): Void. El administrador va a poder eliminar un servicio del catálogo del hotel.

```java
public void deleteServiceCatalog(Map<Integer, Servicio> services, Integer id, String textFile) {
    // Obtener el Servicio correspondiente al ID

    Servicio deletedService = services.get(id);

    // Si el Servicio existe en el Map

    if (deletedService != null) {
        // Eliminar Servicio del Map

        services.remove(id);

        // Eliminar línea correspondiente al Servicio en el archivo de texto

        try (BufferedReader reader = new BufferedReader(new FileReader(textFile));
             BufferedWriter writer = new BufferedWriter(new FileWriter(textFile + ".temp"))) {
            String currentLine;
            while ((currentLine = reader.readLine()) != null) {
                String[] fields = currentLine.split(";");
                if (!fields[0].equals(String.valueOf(id))) {
                    writer.write(currentLine);
                    writer.newLine();
                }
            }
        } catch (IOException e) {
            System.err.println("Error al eliminar producto: " + e.getMessage());
        }

        // Renombrar archivo temporal y borrar el original

        try {
            java.nio.file.Files.move(java.nio.file.Paths.get(textFile + ".temp"), java.nio.file.Paths.get(textFile),
                    java.nio.file.StandardCopyOption.REPLACE_EXISTING);
        } catch (IOException e) {
            System.err.println("Error al renombrar archivo temporal: " + e.getMessage());
        }
    } else {
        System.out.println("El producto con ID " + id + " no existe en el Map.");
    }
}
```

+ addProductRoom(): Void. El administrador va a poder agregar un producto con su precio respectivo a la habitacion para que los clientes del hotel puedan usarlo.

```java
public void addProductRoom(String archivo, int idBuscado, String nuevoProducto,
        Map<Integer, Habitacion> habitaciones) {
    try {
        // Lee todas las lineas del archivo en una lista
        List<String> lineas = Files.readAllLines(Paths.get(archivo), Charset.defaultCharset());
        boolean encontrado = false;

        // Itera sobre las lineas y actualiza la línea correspondiente
        for (int i = 0; i < lineas.size(); i++) {
            String linea = lineas.get(i);
            String[] elementos = linea.split(";");

            // Si se encuentra la linea correspondiente al ID buscado
            if (Integer.parseInt(elementos[0]) == idBuscado) {
                encontrado = true;

                // Agrega el nuevo producto a la tercera columna
                if (elementos.length > 2 && !elementos[3].isEmpty()) {
                    elementos[3] += "," + nuevoProducto;
                } else {
                    elementos[3] = nuevoProducto;
                }

                // Actualiza la línea en la lista
                linea = String.join(";", elementos);
                lineas.set(i, linea);

                // Actualiza la habitación correspondiente en el HashMap
                Habitacion habitacion = habitaciones.get(idBuscado);
                habitacion.getConsumptionRecord().add(nuevoProducto);
                actualizarHabitacion(habitacion);

                break; // Termina el bucle porque ya se actualizó la línea correspondiente
            }
        }
```

```java
        // Si no se encontró el ID buscado, lanza una excepción
        if (!encontrado) {
            throw new RuntimeException("No se encontró el id buscado.");
        }

        // Escribe todas las lineas actualizadas al archivo
        Files.write(Paths.get(archivo), lineas, Charset.defaultCharset());

        System.out.println("Producto agregado exitosamente.");
    } catch (IOException e) {
        System.out.println("Error al agregar el producto: " + e.getMessage());
    }
}
```

+ addServiceRoom(): Void. El administrador va a poder agregar un servicio con su precio respectivo a la habitacionpara que los clientes del hotel puedan usarlo.

```java
public void addServiceRoom(String archivo, int idBuscado, String nuevoServicio,
        Map<Integer, Habitacion> habitaciones) {
    try {
        // Lee todas las lineas del archivo en una lista
        List<String> lineas = Files.readAllLines(Paths.get(archivo), Charset.defaultCharset());
        boolean encontrado = false;

        // Itera sobre las lineas y actualiza la línea correspondiente
        for (int i = 0; i < lineas.size(); i++) {
            String linea = lineas.get(i);
            String[] elementos = linea.split(";");

            // Si se encuentra la linea correspondiente al ID buscado
            if (Integer.parseInt(elementos[0]) == idBuscado) {
                encontrado = true;

                // Agrega el nuevo producto a la tercera columna
                if (elementos.length > 2 && !elementos[3].isEmpty()) {
                    elementos[3] += "," + nuevoServicio;
                } else {
                    elementos[3] = nuevoServicio;
                }

                // Actualiza la línea en la lista
                linea = String.join(";", elementos);
                lineas.set(i, linea);

                // Actualiza la habitación correspondiente en el HashMap
                Habitacion habitacion = habitaciones.get(idBuscado);
                habitacion.getConsumptionRecord().add(nuevoServicio);
                actualizarHabitacion(habitacion);

                break; // Termina el bucle porque ya se actualizó la línea correspondiente
            }
        }
```

```java
        // Si no se encontró el ID buscado, lanza una excepción
        if (!encontrado) {
            throw new RuntimeException("No se encontró el id buscado.");
        }

        // Escribe todas las lineas actualizadas al archivo
        Files.write(Paths.get(archivo), lineas, Charset.defaultCharset());

        System.out.println("Servicio agregado exitosamente.");
    } catch (IOException e) {
        System.out.println("Error al agregar el servicio: " + e.getMessage());
    }
}
```

+ removeServiceRoom(): Void. El administrador va a poder eliminar un servicio de la habitacion correspondiente del hotel.

```java
public void removeServiceRoom(String archivo, int idBuscado, String servicioEliminar,
Map<Integer, Habitacion> habitaciones) {
    try {
        // Abre el archivo para lectura y escritura
        BufferedReader reader = new BufferedReader(new FileReader(archivo));
        FileWriter writer = new FileWriter(archivo + ".tmp");

        String linea;
        boolean encontrado = false;

        // Lee cada línea del archivo
        while ((linea = reader.readLine()) != null) {
            // Divide la línea por el carácter ";" para obtener los elementos
            String[] elementos = linea.split(";");

            // Si el segundo elemento es el id buscado
            if (Integer.parseInt(elementos[0]) == idBuscado) {
                encontrado = true;

                // Verifica si la habitación tiene el producto a eliminar
                String[] productos = elementos[3].split(",");
                List<String> productosList = new ArrayList<>(Arrays.asList(productos));

                if (!productosList.remove(servicioEliminar)) {
                    throw new RuntimeException("La habitación no tiene el producto a eliminar.");
                }

                // Reemplaza la línea original por la línea actualizada
                elementos[3] = String.join(",", productosList);
                linea = String.join(";", elementos);

                // Actualiza la habitación correspondiente en el HashMap
                Habitacion habitacion = habitaciones.get(idBuscado);
                habitacion.getConsumptionRecord().remove(servicioEliminar);
                actualizarHabitacion(habitacion);
            }
```

```java
            // Escribe la línea en el archivo temporal
            writer.write(linea + "\n");
        }

        // Si no se encontró el id buscado, lanza una excepción
        if (!encontrado) {
            throw new RuntimeException("No se encontró el id buscado.");
        }

        // Cierra el lector y escritor de archivos
        reader.close();
        writer.close();

        // Reemplaza el archivo original con el archivo temporal
        File originalFile = new File(archivo);
        File tempFile = new File(archivo + ".tmp");

        if (!tempFile.renameTo(originalFile)) {
            throw new RuntimeException("No se pudo actualizar el archivo.");
        }

        System.out.println("Producto eliminado exitosamente.");
    } catch (IOException e) {
        System.out.println("Error al eliminar el producto: " + e.getMessage());
    }
}
```

+ removeProductRoom(): Void. El administrador va a poder eliminar un producto de la habitacion correspondiente del hotel.

```java
public void removeProductRoom(String archivo, int idBuscado, String productoEliminar,
        Map<Integer, Habitacion> habitaciones) {
    try {
        // Abre el archivo para lectura y escritura
        BufferedReader reader = new BufferedReader(new FileReader(archivo));
        FileWriter writer = new FileWriter(archivo + ".tmp");

        String linea;
        boolean encontrado = false;

        // Lee cada línea del archivo
        while ((linea = reader.readLine()) != null) {
            // Divide la línea por el carácter ";" para obtener los elementos
            String[] elementos = linea.split(";");

            // Si el segundo elemento es el id buscado
            if (Integer.parseInt(elementos[0]) == idBuscado) {
                encontrado = true;

                // Verifica si la habitación tiene el producto a eliminar
                String[] productos = elementos[3].split(",");
                List<String> productosList = new ArrayList<>(Arrays.asList(productos));

                if (!productosList.remove(productoEliminar)) {
                    throw new RuntimeException("La habitación no tiene el producto a eliminar.");
                }

                // Reemplaza la línea original por la línea actualizada
                elementos[3] = String.join(",", productosList);
                linea = String.join(";", elementos);

                // Actualiza la habitación correspondiente en el HashMap
                Habitacion habitacion = habitaciones.get(idBuscado);
                habitacion.getConsumptionRecord().remove(productoEliminar);
                actualizarHabitacion(habitacion);
            }
```

```java
            // Escribe la línea en el archivo temporal
            writer.write(linea + "\n");
        }

        // Si no se encontró el id buscado, lanza una excepción
        if (!encontrado) {
            throw new RuntimeException("No se encontró el id buscado.");
        }

        // Cierra el lector y escritor de archivos
        reader.close();
        writer.close();

        // Reemplaza el archivo original con el archivo temporal
        File originalFile = new File(archivo);
        File tempFile = new File(archivo + ".tmp");

        if (!tempFile.renameTo(originalFile)) {
            throw new RuntimeException("No se pudo actualizar el archivo.");
        }

        System.out.println("Producto eliminado exitosamente.");
    } catch (IOException e) {
        System.out.println("Error al eliminar el producto: " + e.getMessage());
    }
}
```

+ addRoom(): Void. El administrador va a poder agregar una habitación con su respectiva capacidad, tipo de habitación, descripción y el identificador se va a generar automáticamente para la habitación. Si se agrega una habitación esta debe de quedar guardada para que las próximas veces pueda ser usada.

```java
public void addRoom(Map<Integer, Habitacion> rooms, int roomId, int valueByNight, String roomType,
        int guestCapacity, String availableServices, String filePath) throws IOException {
    // Convierte el String de servicios en una lista de Strings

    ArrayList<String> servicesList = new ArrayList<String>();
    String[] servicesCodes = availableServices.split(",");
    for (String code : servicesCodes) {
        servicesList.add(code.trim());
    }

    Habitacion newRoom = new Habitacion(roomId, false, servicesList, new ArrayList<String>(),
            new ArrayList<String>(), valueByNight, guestCapacity, roomType, 0);
    rooms.put(roomId, newRoom);

    // Escribe la información de la habitación en el archivo de texto
    BufferedWriter writer = new BufferedWriter(new FileWriter(filePath, true));
    writer.write(newRoom.getId() + ";" + newRoom.getOccupancyStatus() + ";"
            + String.join(",", newRoom.getServices()) + ";" + String.join(",", newRoom.getConsumptionRecord()) + ";"
            + String.join(",", newRoom.getGuestList()) + ";" + newRoom.getValueByNight() + ";"
            + newRoom.getGuestCapacity() + ";" + newRoom.getRoomType() + ";" + newRoom.getTotalValue());
    writer.newLine();
    writer.close();
}
```

+ deleteRoom(): Void. El administrador va a poder eliminar una habitación del sistema del hotel.

```java
public void deleteRoom(Integer roomId, Map<Integer, Habitacion> habitaciones, String rutaArchivo)
        throws IOException {
    habitaciones.remove(roomId);

    File archivo = new File(rutaArchivo);
    File archivoTemporal = new File("temp.txt");

    BufferedReader lector = new BufferedReader(new FileReader(archivo));
    BufferedWriter escritor = new BufferedWriter(new FileWriter(archivoTemporal));

    String linea;
    while ((linea = lector.readLine()) != null) {
        String[] atributos = linea.split(";");
        if (Integer.parseInt(atributos[0]) != roomId) {
            escritor.write(linea + "\n");
        }
    }

    lector.close();
    escritor.close();

    archivo.delete();
    archivoTemporal.renameTo(new File(rutaArchivo));
}
```

+ addUser(): Usuario. El administrador tiene la facultad de crear un nuevo usuario en el sistema, para que en caso de que haya cambio de personal, estos puedan hacer uso del mismo a partir de su cargo.

```java
public void addUser(String login, String password, String userType, Map<String, Usuario> usuarios,
        String rutaArchivo) throws IOException {

    Usuario usuario;

    if (userType.equals("administrador")) {
        usuario = new Administrador(login, password, userType);
    } else if (userType.equals("recepcionista")) {
        usuario = new Recepcionista(login, password, userType);
    } else {
        usuario = new Empleado(login, password, userType);
    }

    usuarios.put(login, usuario);

    FileWriter escritor = new FileWriter(rutaArchivo, true);
    escritor.write(login + ";" + password + ";" + userType + "\n");
    escritor.close();

}
```

+ deleteUser(): El administrador tiene la capacidad de eliminar un usuario del sistema del hotel.

```java
public void deleteUser(String login, Map<String, Usuario> usuarios, String rutaArchivo) throws IOException {

    usuarios.remove(login);

    FileWriter escritor = new FileWriter(rutaArchivo);

    for (Usuario usuario : usuarios.values()) {
        escritor.write(usuario.getLogin() + ";" + usuario.getPassword() + ";" + usuario.getUserType() + "\n");
    }
    escritor.close();
}
```

+ getUsers(): List. El administrador puede obtener una lista completa de los usuarios del sistema con sus respectivos correos y claves.

```java
public void getUsers(Map<String, Usuario> usuarios) {
    for (Usuario usuario : usuarios.values()) {
        System.out.println("Login: " + usuario.getLogin());
        System.out.println("Password: " + usuario.getPassword());
        System.out.println("UserType: " + usuario.getUserType());
        System.out.println("--------------------");
    }
}
```

+ getBooking(): List. El administrador va a poder obtener una lista completa de las reservas activas del hotel en ese momento.

```java
public void getBookings(Map<Integer, Reserva> reservas) {
    for (Map.Entry<Integer, Reserva> entry : reservas.entrySet()) {
        Integer idReserva = entry.getKey();
        Reserva reserva = entry.getValue();
        System.out.println("Reserva #" + idReserva);
        System.out.println("Date: " + reserva.getDate());
        System.out.println("Associated rooms: " + reserva.getRoomsList());
        System.out.println("Guest list: " + reserva.getGuestList());
        System.out.println("Associated value: " + reserva.getAssociatedValue());
        System.out.println("-----------------------------------");
    }
}
```

Empleado:

+ empleado(): se crea el empleado con las caracteriscticas

```
public Empleado(String login, String password, String userType) {
    super(login, password, userType);
```

el empleado tambien tiene como funciones el poder agregar o eliminar productos o servicios a las habitaciones de los huespedes del hotel.

Recepcionista:

+ modifyBooking(): Void. El recepcionista va a poder modificar una reserva dado el String correspondiente al ID de la reserva. Modificar también implica cancelar la reserva en caso de ser necesario.

+ checkOut(): void El recepcionista va a poder cerrar la estadía de un huésped en el sistema, guardando la información de la reserva en un documento de reservas terminadas exitosamente.

+crearHuesped(): crea un huesped nuevo para agregar a la reserva

```
public Huesped crearHuesped(String name, int age, String email, String Archivo, int guestID) {
    // Crear un nuevo huésped
    Huesped nuevoHuesped = new Huesped(name, age, email, guestID);

    try {
        FileWriter fw = new FileWriter(Archivo, true); // true para que se agregue al final del archivo
        PrintWriter pw = new PrintWriter(fw);
        pw.println(nuevoHuesped.getGuestID() + ";" + nuevoHuesped.getName() + ";" + nuevoHuesped.getAge() + ";" + nuevoHuesped.getEmail() + ";" + nue
        pw.close();
        System.out.println("Huésped guardado con éxito en el archivo "+ Archivo);
    } catch (IOException e) {
        System.out.println("Error al guardar el huésped en el archivo "+ Archivo );
        e.printStackTrace();
    }

    return nuevoHuesped;

}
```

+ checkIn(): void El recepcionista va a poder generar la reserva y estadía de un cliente en el sistema.

```java
public void checkIn(Integer bookingId, String entryDate, String departureDate, ArrayList<Integer> associatedRooms, ArrayList<String> guestList, Integ
    // Verificar si las habitaciones ya están reservadas para las fechas de entrada y salida
    ArrayList<Reserva> reservas = getBookingsList(Archivo);
    for (Reserva r : reservas) {
        if ((entryDate.equals(r.getEntryDate()) || departureDate.equals(r.getDepartureDate())) ||
            r.associatedRooms.stream().anyMatch(associatedRooms::contains)) {
            System.out.println("Las habitaciones ya están reservadas para las fechas de entrada y salida especificadas");
            return;
        }
    }



    // Crear una nueva reserva
    Reserva nuevaReserva = new Reserva(bookingId, entryDate, departureDate, associatedRooms, guestList, associatedValue);

    // Guardar la reserva en un archivo de texto
    try {
        FileWriter fw = new FileWriter(Archivo, true); // true para que se agregue al final del archivo
        PrintWriter pw = new PrintWriter(fw);
        pw.println(nuevaReserva.getBookingId() + ";" + nuevaReserva.getEntryDate() + ";" + nuevaReserva.getDepartureDate() + ";" + nuevaReserva.getRo
        pw.close();
        System.out.println("Reserva guardada con éxito en el archivo "+ Archivo);
    } catch (IOException e) {
        System.out.println("Error al guardar la reserva en el archivo "+ Archivo);
        e.printStackTrace();
    }
}
```

```java
public ArrayList<Reserva> getBookingsList(String archivo) {
    ArrayList<Reserva> bookingsList = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(archivo))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] bookingData = line.split(",");
            Integer bookingId = Integer.parseInt(bookingData[0]);
            String entryDate = bookingData[1];
            String departureDate = bookingData[2];
            ArrayList<Integer> associatedRooms = new ArrayList<>();
            for (String roomId : bookingData[3].split("-")) {
                associatedRooms.add(Integer.parseInt(roomId));
            }
            ArrayList<String> guestList = new ArrayList<>();
            for (String guestName : bookingData[4].split("-")) {
                guestList.add(guestName);
            }
            Integer associatedValue = Integer.parseInt(bookingData[5]);
            Reserva booking = new Reserva(bookingId, entryDate, departureDate, associatedRooms, guestList, associatedValue);
            bookingsList.add(booking);
        }
    } catch (IOException e) {
        System.out.println("Error al leer el archivo reservas.txt");
        e.printStackTrace();
    }

    return bookingsList;
}
```

+ getBookingsList() List :El recepcionista tiene la facultad de ver todas las reservas que se encuentren activas en el hotel con el fin de ofrecerle una habitación a un huésped.

huesped:

huesped es donde se aloja y se crea toda la informacion de los huespedes para asociarlos a una reserva:

```java
public class Huesped {
    private int guestID;
    private String name;
    private int age;
    private String email;
    private int roomID;
    private int bookingID;

    public Huesped(String name, int age, String email) {
        this.guestID = 0;
        this.name = name;
        this.age = age;
        this.email = email;
    }
```

```java
    public int getAge() {
        return this.age;
    }

    public String getName() {
        return this.name;
    }

    public String getEmail() {
        return this.email;
    }

    public int getGuestID() {
        return this.guestID;
    }

    public int getRoomID() {
        return this.roomID;
    }

    public int getBookingID() {
        return this.bookingID;
    }
```

existe una funcion llamada generateID() la cual crea un codigo único para identificar al huesped.

```java
  public static int generateID() {
       String chars = "0123456789";
       StringBuilder sb = new StringBuilder();
       Random random = new Random();
       for (int i = 0; i < 3; i++) {
         sb.append(chars.charAt(random.nextInt(chars.length())));
       }
       int ID = Integer.parseInt(sb.toString());
       return ID;
    }
}
```

Habitación:

La clase habitación almacena toda la info necesaria para construir una habitación y usar sus datos para la creación de una reserva

```java
Habitacion(Integer roomId, Boolean occupancyStatus, ArrayList<String> availableServices, ArrayList<String> consumptionRecord,
 ArrayList<String> guestList, Integer valueByNight, Integer guestCapacity, String roomType, Integer totalValue)

is.roomId = roomId;
is.occupancyStatus = occupancyStatus;
is.availableServices = availableServices;
is.consumptionRecord=consumptionRecord;
is.guestList = guestList;
is.valueByNight = valueByNight;
is.guestCapacity = guestCapacity;
is.roomType = roomType;
is.totalValue = totalValue;

int getId()

turn roomId;

Boolean getOccupancyStatus()

turn occupancyStatus;

ArrayList<String> getServices()

turn availableServices;

ArrayList<String> getConsumptionRecord()

turn consumptionRecord;

ArrayList<String> getGuestList()

turn guestList;

Integer getValueByNight()

turn valueByNight;

Integer getGuestCapacity()

turn guestCapacity;

String getRoomType()

turn roomType;
```

reserva:

La clase reserva almacena toda la info necesaria para construir y almacenar una reserva

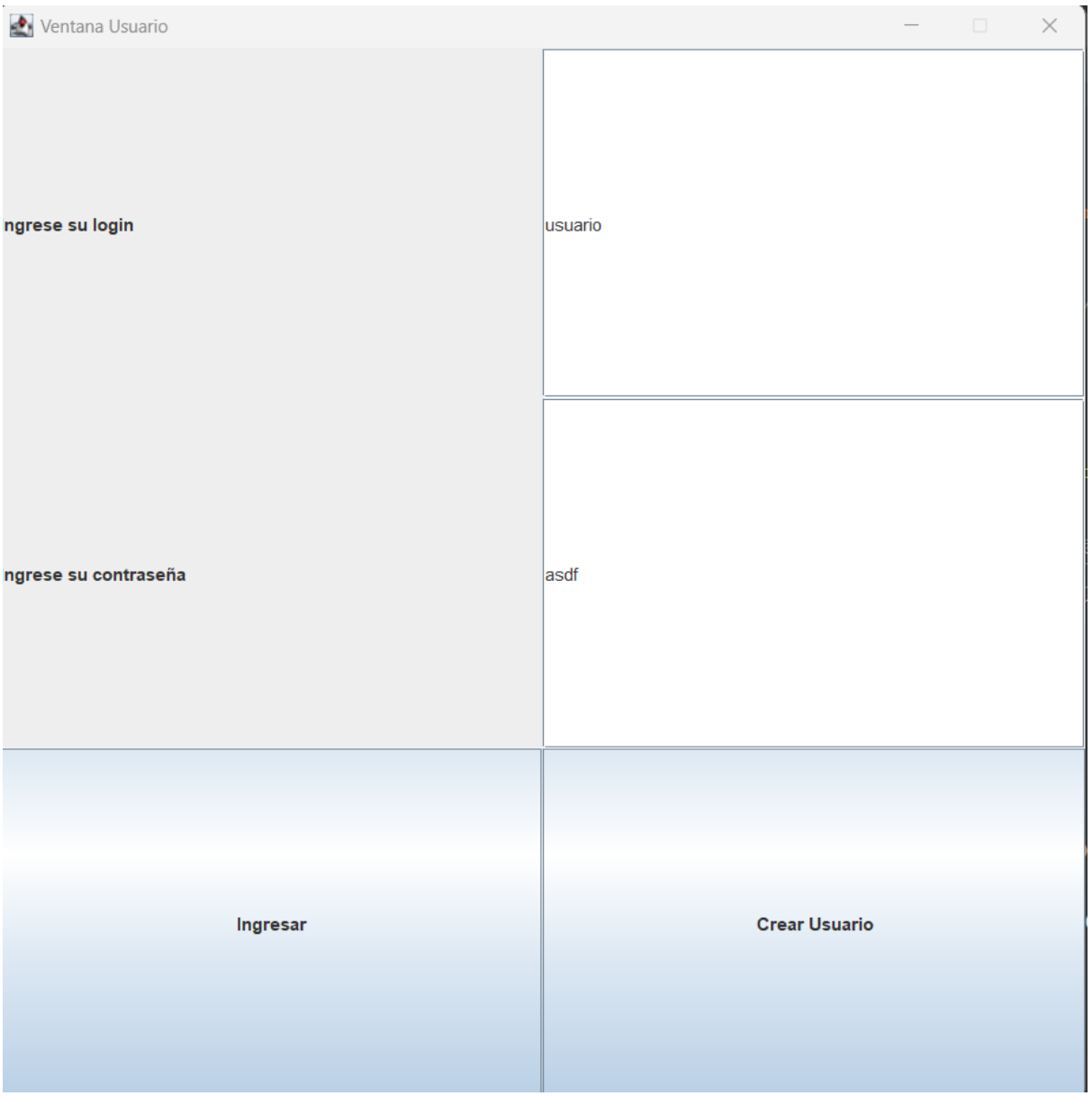
```java
public  Reserva(Integer bookingId, String entryDate, String departureDate, ArrayList<Integer> associatedRooms, ArrayList<String> guestList, Integer associatedValue)
{
    this.bookingId = bookingId;
    this.entryDate = entryDate;
    this.departureDate = departureDate;
    this.associatedRooms = associatedRooms;
    this.guestList = guestList;
    this.associatedValue = associatedValue;
}
public String getDate()
{
    String fechas_reservadas = entryDate+"-"+departureDate;

    return fechas_reservadas;
}
public Integer getAssociatedValue()
{
    return associatedValue;
}
public ArrayList<String> getGuestList()
{
    return guestList;
}
public ArrayList<Integer> getRoomsList()
{
    return associatedRooms;
}
public Integer getBookingId()
{
    return bookingId;
}
```

Interfaz Grafica de Usuario

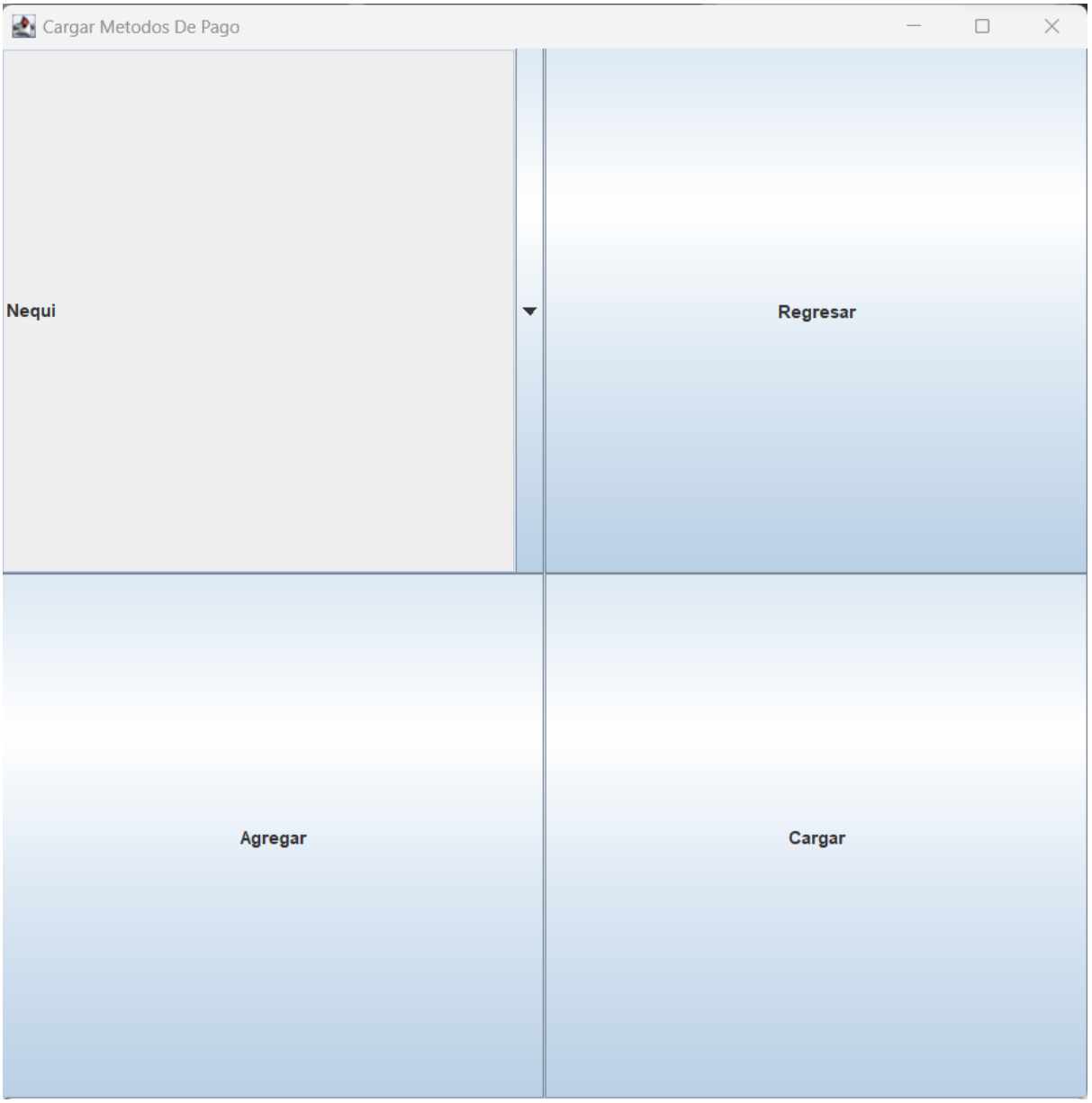
Para la entrega 3 del proyecto de DPOO, se desarrolló una interfaz de usuario, la cual permitió a un cliente ingresar a la plataforma del hotel y realizar procedimientos como:

- Cargar plataforma de pago
- Pagar una reserva
- Hacer una reserva
- Revisar la disponibilidad a partir de la observación de las reservas

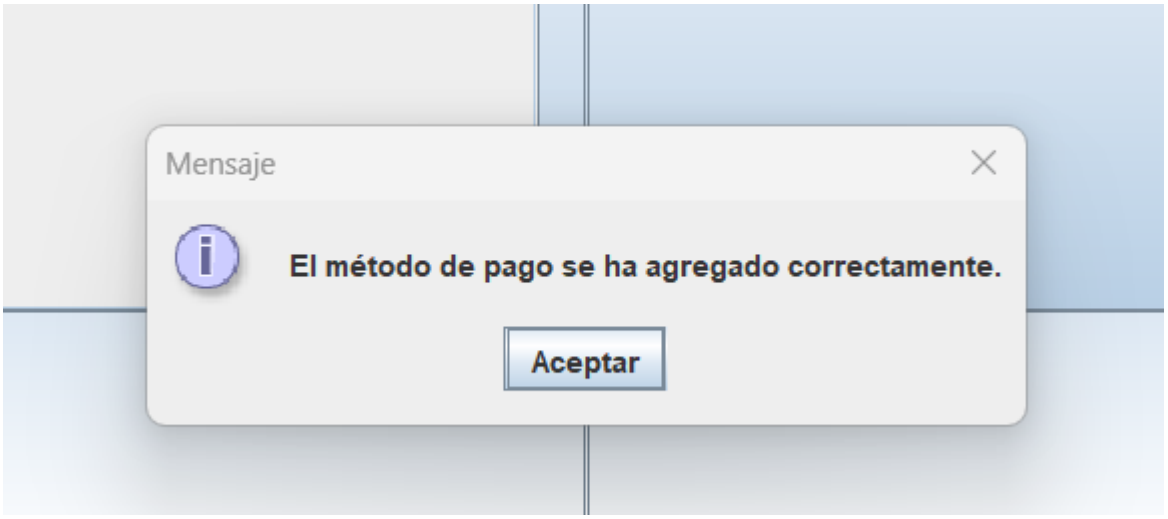
El funcionamiento de la interfaz de usuario es el mismo que el de los empleados del Hotel, se posee un usuario y una contraseña, los cuales estan almacenados en un archivo .txt



FUNCIONAMIENTO CARGA DE METODOS DE PAGO

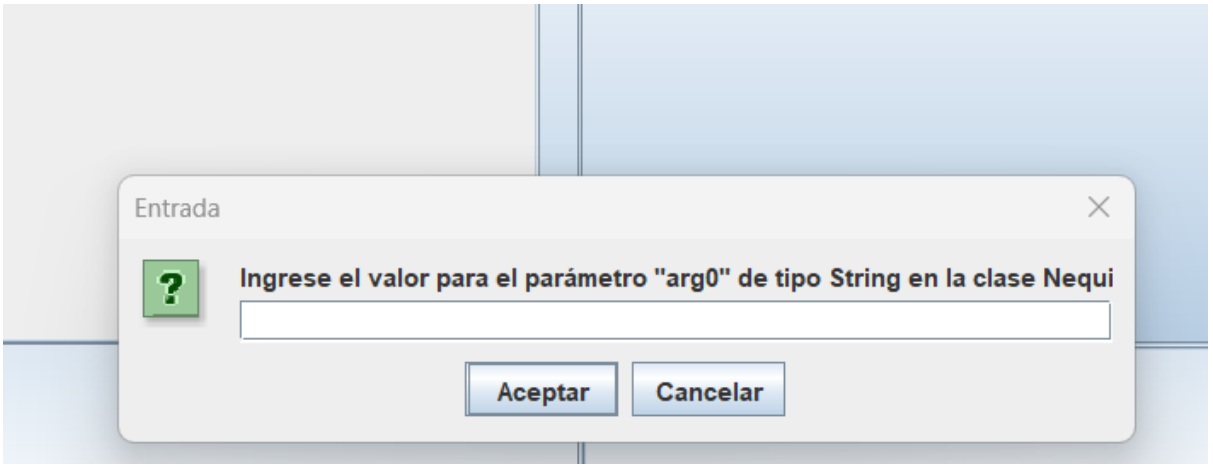


Para poder realizar una carga efectiva y dinámica de los métodos de pago seleccionados por el usuario se utilizó un sistema en el cual en un frame de Swing se incluyeron 3 JButtons y un JComboBox, en este ultimo estaban alojados los nombres de los metodos de pago disponibles para ser cargados, estos surgen de un archivo .txt llamado “metodosDePago.txt”, en este punto el usuario selecciona haciendo click en el nombre del metodo de pago que desea cargar y presiona en el boton agregar:

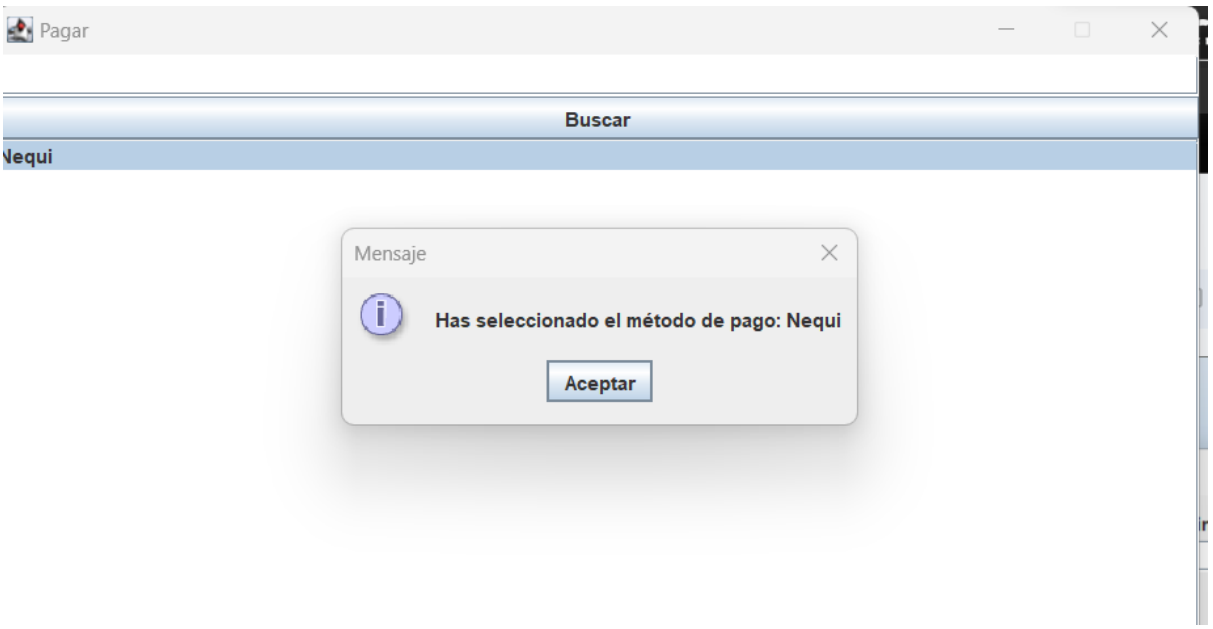


Al presionar en el boton agregar, el nombre del metodo de pago se agrega a una lista ArrayList<String> , una vez el usuario se siente satisfecho con los metodos de pago que ha agregado a la lista presiona en cargar, lo que genera que por cada elemento de la lista, se creen objetos de las clases que se estan cargando.

Un ejemplo de esto es que si se carga solamente la clase “Nequi”, se va a pedir al usuario que ingrese su numero de telefono y el monto total que tiene en esta cuenta, lo ultimo es para que al momento de pagar se cuente con suficiente balance para pagar la reserva.



Al finalizar de ingresar todos los parametros se abre una nueva pestaña que se llama “Pagar”, en esta se van a ver los metodos de pago que fueron cargados por el usuario, se debe de seleccionar uno:



Una vez seleccionado se ingresa el ID de una reserva en la parte de arriba, y si se cuenta con el suficiente balance en la cuenta, y el ID de la reserva existe, se procede con el pago.

Ademas de esto cuando se paga, se genera una factura que se agrega en la carpeta “data” del proyecto con la informacion.