Práctica 2

Gestor de transporte terrestre
LussajuBus

Asignatura

Programación Orientada a Objetos

Profesor

Jaime Alberto Guzmán Luna

Grupo: 01
Equipo: 7

Integrantes:

Samuel Hernández Duque - shernandezdu@unal.edu.co
Santiago Cardona Franco - sacardonaf@unal.edu.co

Universidad Nacional de Colombia

Sede Medellín

2024

ÍNDICE:

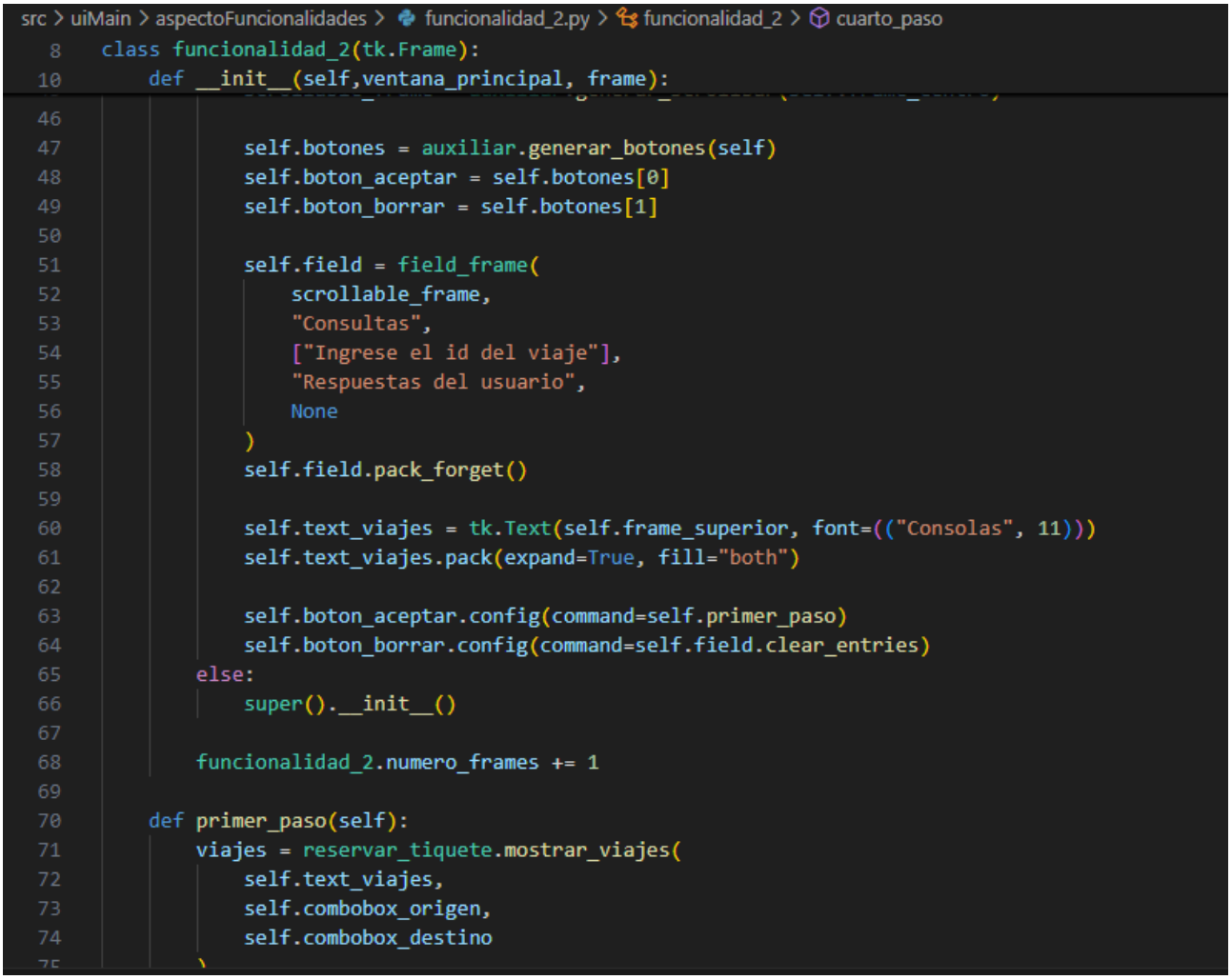
Contenido

src > uiMain > funcionalidades > funcionalidad1.py > ver_viajes > mostrar_viajes

```
11  class ver_viajes():
78      def tercera_pregunta(frame_funcionalidad):
93              return False
94
95      @staticmethod
96      def cuarta_pregunta(frame_funcionalidad):
97          respuesta = frame_funcionalidad.field_frame.getValue(
98              "Ingrese el número del asiento"
99          )
100
101         indice=frame_funcionalidad.field_frame.getValue(
102                 "Ingrese el id del viaje")
103         viaje = Empresa.buscar_viaje_por_id(indice)
104         ok=ae.excepcion_viaje(indice)
105         if ok=="ok":
106             ae.excepcion_asiento(respuesta,viaje.get_bus())
107
108
109
110         try:
111             asiento = viaje.buscar_asiento(respuesta)
112             if asiento != None and not asiento.is_reservado():
113                 frame_funcionalidad.field_frame.agregar_campo(
114                     "¿Por cuánto tiempo desea reservarlo?",
115                     True
116                 )
117         except:
118             messagebox.showerror("Error inesperado",
119                                  "Se ha producido un error inesperado pero puede continuar navegando en el programa
120
121     @staticmethod
```

src > uiMain > aspectoFuncionalidades > funcionalidad_1.py > funcionalidad_1 > __init__

```
9   class funcionalidad_1(tk.Frame):
11      def __init__(self,ventana_principal, frame):
29
30              self.botones = auxiliar.generar_botones(self)
31              self.boton_aceptar = self.botones[0]
32              self.boton_borrar = self.botones[1]
33
34              self.field_frame = field_frame(
35                  scrollable_frame,
36                  "Consultas",
37                  ["¿Desea ver más detalles sobre un viaje?"],
38                  "Respuestas del usuario",
39                  None
40              )
41
42              self.text = tk.Text(self.frame_superior, font=(("Consolas", 11)))
43              self.text.pack(expand=True, fill="both")
44
45              self.boton_aceptar.config(command=self.primer_paso)
46              self.boton_borrar.config(command=self.field_frame.clear_entries)
47
48              ver_viajes.mostrar_viajes(self)
49          else:
50              super().__init__()
51
52          funcionalidad_1.numero_frames += 1
53
54      def primer_paso(self):
55          caso = ver_viajes.primera_pregunta(self)
56
57          if caso == -1:
58              self.boton_aceptar.config(command=self.segundo_paso)
```


src > uiMain > aspectoFuncionalidades > funcionalidad_2.py > funcionalidad_2 > cuarto_paso
8 class funcionalidad_2(tk.Frame):
10 def __init__(self,ventana_principal, frame):
46
47 self.botones = auxiliar.generar_botones(self)
48 self.boton_aceptar = self.botones[0]
49 self.boton_borrar = self.botones[1]
50
51 self.field = field_frame(
52 scrollable_frame,
53 "Consultas",
54 ["Ingrese el id del viaje"],
55 "Respuestas del usuario",
56 None
57)
58 self.field.pack_forget()
59
60 self.text_viajes = tk.Text(self.frame_superior, font=(("Consolas", 11)))
61 self.text_viajes.pack(expand=True, fill="both")
62
63 self.boton_aceptar.config(command=self.primer_paso)
64 self.boton_borrar.config(command=self.field.clear_entries)
65 else:
66 super().__init__()
67
68 funcionalidad_2.numero_frames += 1
69
70 def primer_paso(self):
71 viajes = reservar_tiquete.mostrar_viajes(
72 self.text_viajes,
73 self.combobox_origen,
74 self.combobox_destino

Descripción general de la solución

Mediante el lenguaje Java y usando el paradigma de programación orientado a objetos, se desarrolla una aplicación de escritorio que permite manejar algunos de los procesos administrativos que se presentan en las terminales de transporte terrestre desde la perspectiva del funcionario al que uno acude cuando va a comprar un tiquete en persona. Para ello, se crean múltiples objetos con atributos y métodos con el fin de que al interactuar entre ellos se simulen dichos procesos. Los objetos están divididos en tres secciones: gestion, personas y transporte. La sección “gestion” se encarga de lo relacionado con las terminales, las empresas de transporte, los viajes, los tiquetes y el hospedaje. La sección “personas” se encarga de los relacionados con los conductores y los pasajeros. Por último, la sección “transporte” se encarga de lo relacionado con los buses, con los asientos y los tipos de asientos. En el dominio de las terminales de transporte es posible implementar un sinfín de cosas, pero en nuestro caso decidimos implementar las siguientes funcionalidades: ver viajes disponibles, reserva de tiquetes, gestión de tiquetes, hospedaje y opciones de administrador.

El proyecto como tal busca dar solución a la necesidad de unificar el transporte terrestre a nivel nacional mediante un sistema que coordine la relación de todos los entes que son participes en este sistema de transporte, tales como las empresas,las terminales, los pasajeros, conductores, entre otros; también pretendemos que se facilite el proceso de creación y almacenamiento de información relacionada con los agentes viajes públicos, para que no ocurran errores de información al pasarla entre diferentes sistemas y para agilizar el proceso de generación de viajes entre pasajero y entidades prestadoras de vehículos.

Por como está diseñado el proyecto inicialmente sólo puede haber un gestor que coordine toda la información a través de la plataforma que vamos a proveer y la idea es que pueda articular toda la información, desde la creación de terminales, buses y empresas; hasta la de personas y conductores al sistema

El sistema presenta serialización de la información, por ende para guardar los cambios realizados durante todo el programa sólo bastará con cerrarlo para que dicha información quede guardada y que al momento de abrirlo en otro momento la información se mantenga en orden para ser modificada de la manera más conveniente posible facilitando el proceso para el gestor de no tener que guardar la información en una base de datos externa o de tener que realizar todo el proceso completo cada vez que se inicie el programa; también el programa debe correr con relativa facilidad y velocidad únicamente generando excepciones por errores humanos y facilitando en ciertas partes el error por medio del manejo de excepciones básicas; en cuanto a la seguridad únicamente el gestor será quien tenga acceso a la información personal de cada usuario y a la información general del sistema y de las empresas, por lo que todo lo que ocurra con ella será responsabilidad del gestor

Descripción del diseño estático del sistema en la especificación UML

- Diagrama UML.pdf

Descripción de cada clase:

- ❖ *Paquete gestión:*

- ➔ Clase Empresa:

Se encarga de almacenar la información y las empresas que habitan en el sistema, cada empresa se identifica únicamente por su nombre y su relación con los conductores de muchos a uno, con las terminales es muchos a muchos, y con los viajes también es muchos a uno.

- ➔ Clase Habitación:

Se encarga de generar y almacenar las habitaciones que posee cada hospedaje que existe en el sistema, cada una se identifica por su número y su hospedaje y su relación con la clase Hospedaje es uno a muchos.

- ➔ Clase Hospedaje:

Se encarga de organizar, crear y almacenar los hospedajes que habitan en el sistema, se identifican por su nombre y su ubicación y su relación con las habitaciones es muchos a uno, y con los viajes es muchos a uno también.

- ➔ Clase Terminal:

Se encarga de administrar, crear y almacenar todas las terminales que pertenecen al programa, se identifican por su nombre y ubicación; y su relación con las empresas es muchos a muchos, y con los hospedajes es muchos a muchos.

- ➔ Clase Tiquete:

Se encarga de generar, almacenar la información, y los tiquetes que se creen durante el proceso del programa, se identifican los unos de los otros mediante el número de reserva y su relación con la clase Viaje es uno a muchos y con Pasajeros también es uno a muchos.

- ➔ Clase Viaje:

Se encarga de generar, almacenar la información y todos lo viajes que se creen durante la ejecución del programa, se identifican mediante el id y su relación con la clase Tiquete es muchos a uno, con Empresa es uno a muchos y con Bus es muchos a muchos.

- ❖ *Paquete personas:*

- ➔ Interfaz SuperPersona:

Se encarga de almacenar métodos abstractos que obligan a que la clase persona, conductor y pasajero sobreescriban.

- ➔ Clase Persona:

Se encarga de almacenar y proveer a la clase pasajero y conductor la información que ambas tienen en común pues estas heredan de persona, para las tres su identificador único es el número de identidad y no se relaciona directamente con ninguna otra clase.

- ➔ Clase Pasajero:

Se encarga de crear y de almacenar a todos los pasajeros y toda su información en el sistema, se relacionan directamente con la clase tiquete y su relación es muchos a uno.

- ➔ Clase Conductor:

Se encarga de crear y de almacenar a todos los conductores y toda su información en el sistema, se relacionan directamente con la clase Viajes y su relación es muchos a uno.

- ❖ *Paquete transporte:*

- ➔ Clase Asiento:

Se encarga de la creación y almacenamiento de asientos que pertenecen a un bus, su identificador único es su número y también tienen diferentes tipos que se los provee el enumerado Tipo Asiento, y su relación con la clase bus es uno a muchos y con el enumerado es uno a muchos también.

- ➔ Clase Bus:

Se encarga de la creación, almacenamiento y distribución de buses para llevar a cabo los viajes dentro del sistema, su identificador único son las placas, las cuáles se generan de manera aleatoria y no se repiten, se relacionan directamente con la clase asiento y tienen una relación de muchos a uno y con la clase viaje tiene una relación de muchos a uno.

- ➔ Enumerado Tipo Asiento:

Se encarga de almacenar 3 valores (Preferencial,Premium y Estándar) que son constantes y sirven para darle un tipo diferente a las sillas, tienen una relación con ellas de muchos a uno.

- ➔ Clase Abstracta Vehiculo:

Se encarga de tener los aspectos básicos de un vehículo de los cuales luego la clase bus puede heredar como las placas por ejemplo que son su identificador único, además tiene métodos que obligan a la clase bus a implementar, no se pueden instanciar directamente y no se relaciona con ninguna clase.

Descripción de la Implementación de características de programación orientada a objetos en el proyecto

- **Clases Abstracta (1) y Métodos Abstractos (1):**

Ubicado en el paquete gestorAplicación.transporte en la clase Vehiculo (Vehiculo es una clase abstracta y tiene un método abstracto llamado crearAsientos que recibe un parámetro entero)

```java
package gestorAplicación.transporte;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Random;

import gestorAplicación.personas.Conductor;

public abstract class Vehiculo implements Serializable {

    private static ArrayList<String> placas = new ArrayList<String>();
    private static final long serialVersionUID = 1613039627515609694L;
    private String placa;
    private Conductor conductor;

    public Vehiculo() {

    }

    public Vehiculo(String placa) {
        this.placa = placa;

    }

    public abstract void crearAsientos(int asientos);

    public String getPlaca() {
        return placa;
    }
```

Ubicado en el paquete gestorAplicación.transporte en la clase Bus (Bus hereda de la clase abstractaVehiculo)

```java

public class Bus extends Vehiculo implements Serializable {

```

Además está sobreescribiendo el método que está obligado a sobreescribir por heredarlo como abstracto de la clase Vehiculo con la misma firma, visibilidad y tipo de retorno

```
34      @Override
35      public void crearAsientos(int asientos) {
36          String letras = "ABCD";
37
38          for (int numero = 1; numero < asientos + 1; numero++) {
39              for (int letra = 0; letra < 4; letra++) {
40                  String numeroAsiento = String.valueOf(numero) + letras.charAt(letra);
41
42                  if (numero <= tiposAsientoFila[0]) {
43                      TipoAsiento tipo = TipoAsiento.PREFERENCIAL;
44                      this.asientos.add(new Asiento(numeroAsiento, tipo));
45                  } else if (numero <= tiposAsientoFila[1]) {
46                      TipoAsiento tipo = TipoAsiento.PREMIUM;
47                      this.asientos.add(new Asiento(numeroAsiento, tipo));
48                  } else {
49                      TipoAsiento tipo = TipoAsiento.ESTANDAR;
50                      this.asientos.add(new Asiento(numeroAsiento, tipo));
51                  }
52              }
53          }
54      }
55
```

El implementar un método abstracto en el programa ha sido de gran ayuda para evitar que se creen instancias de la clase Persona que por sí sóla no tiene sentido pero que agrupa muchas características de la clase Vehículo la cual hereda de esta, además obliga a sobreescribir el método abstracto crearAsiento por la clase bus para obligar a que este exista dentro de la clase que nos interesa instanciar y que a su vez facilite la generación del código en la clase principal

- **Interfaces (1) diferentes a los utilizados para serializar los objetos en el punto de la persistencia. Deberá ser propio del dominio a implementar.**

Interfáz SuperPersona ubicada en el paquete gestorAplicación.personas (Ejemplo de interfáz)

```
1  package gestorAplicación.personas;
2
3  import java.util.ArrayList;
4
5  import gestorAplicación.gestion.Tiquete;
6
7  public interface SuperPersona {
8      public void cancelarTiquete(Tiquete tiquete);
9
10     public Tiquete buscarTiquete(String numeroReserva);
11
12     public ArrayList<Tiquete> buscarTiquetes(String tipoTiquetes);
13
14     public void agregarTiquete(Tiquete tiquete);
15 }
16
```

Las interfaces son bastante útiles ya que proveen métodos de varios tipos, como abstractos, default, estáticos o privados, que además definen constantes únicamente creando una generalización y obligación al igual que en las clases abstractas a las clases hijas de definir los métodos abstractos y de

proveer métodos e información a las subclases que por sí solas no tienen mucho sentido pero que acompañadas sirven de muchos, además no se pueden instanciar, en nuestro caso lo usamos para obligar a las clases hijas en este caso a la clase Persona a tener que sobreescribir los métodos descritos en la interfaz que por defecto son públicos abstractos

- **Herencia (1)**

Ubicado en el paquete gestorAplicación.transporte en la clase Bus (Está clase está heredando de la clase Vehículo, por lo que se implementa la herencia)

```
1 package gestorAplicación.transporte;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5
6 import baseDatos.Deserializador;
7 import gestorAplicación.gestion.Hospedaje;
8 import gestorAplicación.gestion.Terminal;
9 import gestorAplicación.gestion.Viaje;
10
11 public class Bus extends Vehiculo implements Serializable {
```

Ubicada en el paquete gestorAplicación.personas en la clase Pasajero (Esta clase hereda de la clase Persona)

```
1 package gestorAplicación.personas;
2
3 import java.io.Serializable;
4 import java.time.LocalDate;
5 import java.util.ArrayList;
6
7 import gestorAplicación.gestion.Empresa;
8 import gestorAplicación.gestion.Hospedaje;
9 import gestorAplicación.gestion.Tiquete;
10 import gestorAplicación.gestion.Viaje;
11
12 public class Pasajero extends Persona implements Serializable {
13     private static final long serialVersionUID = -8124260530486820488L;
14     private static ArrayList<Pasajero> pasajeros = new ArrayList<Pasajero>();
15     private ArrayList<Tiquete> tiquetes = new ArrayList<Tiquete>();
16
```

Ubicada en el mismo paquete la clase Conductor (Hereda también de la clase Persona)

```java
package gestorAplicación.personas;

import gestorAplicación.gestion.Hospedaje;
import gestorAplicación.gestion.Terminal;
import gestorAplicación.gestion.Viaje;
import java.util.ArrayList;
import java.io.Serializable;

public class Conductor extends Persona implements Serializable {

    private static final long serialVersionUID = 1L;
    private static ArrayList<Conductor> conductores = new ArrayList<Conductor>();
    private ArrayList<Viaje> viajes = new ArrayList<Viaje>();
```

El uso de la herencia es fundamental en el programa para la reutilización y optimización del código, ya que por medio de la clase padres, todas las clases hijas pueden heredar los atributos comunes que posean, facilitando así el proceso de obtención de atributos y métodos comunes entre ellas y generando a su vez una relación entre ellas mediante el polimorfismo, en este caso implementamos dos herencias, tanto en la clase persona que es padre de pasajero y conductor como en la clase vehiculo la cual es únicamente padre de la clase bus

Ubicado en el paquete gestorAplicación.gestion en la clase Viaje (Está heredando de Object implícitamente)

```
167     @Override
168     public String toString() {
169         int origen = 11 - (getTerminalOrigen().getUbicacion().length());
170         String strOrigen = String.valueOf(origen);
171
172         int destino = 11 - (getTerminalDestino().getUbicacion().length());
173         String strDestino = String.valueOf(destino);
174
175         int id = 3 - getId().length();
176         String strId = String.valueOf(id);
177
178         String spaceOrigen;
179         String spaceDestino;
180         String spaceId;
181
182         if (origen == 0) {
183             spaceOrigen = "";
184         } else {
185             spaceOrigen = String.format("%" + strOrigen + "s", "");
186         }
187
188         if (destino == 0) {
189             spaceDestino = "";
190         } else {
191             spaceDestino = String.format("%" + strDestino + "s", "");
192         }
193
194         if (id == 0) {
195             spaceId = "";
196         } else {
197             spaceId = String.format("%" + strId + "s", "");
198         }
```

Además permite sobreescribir los métodos que la clase hereda de la clase Object como por ejemplo el toString en este caso

- **Ligadura dinámica (2) asociadas al modelo lógico de la aplicación.**

Ubicada en la Clase Interfaz en el paquete uiMain(Ejemplo ligadura dinámica)

```
1326        System.out.println();
1327
1328        Persona pasajero = Pasajero.buscarPasajero(idPasajero);
1329
1330        if (pasajero == null || ((Pasajero) pasajero).getTiquetes().isEmpty()) {
1331            System.out.println("No hay tiquetes asociados " + "con el número de identificación");
1332
1333            System.out.println();
1334        } else {
1335            ArrayList<Tiquete> tiquetesValidos = pasajero.buscarTiquetes("validos");
1336
1337            ArrayList<Tiquete> tiquetesVencidos = pasajero.buscarTiquetes("vencidos");
1338
1339            if (!tiquetesValidos.isEmpty()) {
1340                System.out.println("Tiqueres válidos");
```

Método buscar tiquetes en la clase Persona

```java
public ArrayList<Tiquete> buscarTiquetes(String tipoTiquetes) {
    return null;
}
```

Método buscar tiquetes sobreescrito en la clase pasajero

```java
public ArrayList<Tiquete> buscarTiquetes(String tipoTiquetes) {

    if (tipoTiquetes.equals("validos")) {
        ArrayList<Tiquete> tiquetesValidos = new ArrayList<Tiquete>();

        for (Tiquete tiquete : this.getTiquetes()) {
            tiquete.setViaje(Empresa.buscarViaje(tiquete.getViaje().getId()));
            if (tiquete.getViaje().getFecha().isAfter(LocalDate.now())) {
                tiquetesValidos.add(tiquete);
            }
        }

        return tiquetesValidos;
    } else if (tipoTiquetes.equals("vencidos")) {
        ArrayList<Tiquete> tiquetesVencidos = new ArrayList<Tiquete>();

        for (Tiquete tiquete : this.getTiquetes()) {
            tiquete.setViaje(Empresa.buscarViaje(tiquete.getViaje().getId()));
            if (tiquete.getViaje().getFecha().isBefore(LocalDate.now())) {
                tiquetesVencidos.add(tiquete);
            }
        }

        return tiquetesVencidos;
    } else {
        return null;
    }
}
```

Ubicada en la Clase Interfaz en el paquete uiMain(Ejemplo ligadura dinámica)

```java
            if (pasajero.buscarTiquete(numeroReserva) == null) {
                System.out.println("No se encontró ningún tiquete con el número de reserva "
                                   + numeroReserva);
                System.out.println();
```

Método buscar tiquete en la clase Persona

```java
public Tiquete buscarTiquete(String numeroReserva) {
    return null;
}
```

Método buscar tiquete sobreescrito en la clase pasajero

```
81      public Tiquete buscarTiquete(Viaje viaje) {
82          for (Tiquete tiquete : tiquetes) {
83              tiquete.setViaje(Empresa.buscarViaje(viaje.getId()));
84              if (tiquete.getViaje().equals(viaje)) {
85                  return tiquete;
86              }
87          }
88
89          return null;
90      }
91
92      public Tiquete buscarTiquete(String numeroReserva) {
93          for (Tiquete tiquete : tiquetes) {
94              if (tiquete.getNumeroReserva().equals(numeroReserva)) {
95                  return tiquete;
96              }
97          }
98
99          return null;
100     }
```

La ligadura dinámica es escencial y muy potente en cuanto a herencia se refiere ya que nos permite ejecutar el método más específico en caso de sobreescritura de un método desde una clase hija hacia una padre a pesar de haber generalizado el objeto para otros fines, lo cuál evita errores practicos que podrían dañar el código, por ejemplo esto lo aplicamos al hacer generalización del objeto de tipo pasajero para verlo como algo de tipo persona y cuando corremos el método buscar tiquete o buscar tiquetes se ejecuta el método definido en la clase pasajero, cumpliendo de esta manera con la ligadura dinámica

- **Atributos de clase (1) y métodos de clase (1)**

Ubicado en el paquete gestorAplicación.gestion en la clase Viaje (El ArrayList viajes de objetos de tipo viaje es un atributo de clase)

```
15
16 public class Viaje implements Serializable {
17     /**
18      *
19      */
20     private static final long serialVersionUID = 2760602559521284522L;
21     private static ArrayList<Viaje> viajes = new ArrayList<Viaje>();
22
23     private Terminal terminalOrigen;
24     private Terminal terminalDestino;
25     private Empresa empresa;
26     private LocalDate fecha;
27     private LocalTime hora;
28     private static int ids;
29     private String id;
30     private Bus bus;
31     private ArrayList<Tiquete> tiquetes = new ArrayList<Tiquete>();
32
```

La implementación de atributos de clase, es decir estáticos fueron de gran importancia para la generación del programa ya que por medio de arrays pudimos coleccionar todos los objetos que se van creando a lo largo del programa, ya que al no poderlos referenciar directamente durante el transcurso del programa la única manera que tenemos de acceder a todos los elementos es mediantes dicho arregle que los contenga a todos, y esto no solamente lo implementamos en la clase viaje sino en casi todas por la practicidad que esto genera.

Aquí unos cuantos ejemplos más:

Clase Bus en el paquete gestorAplicación.transporte

```
1 package gestorAplicación.transporte;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5
6 import baseDatos.Deserializador;
7 import gestorAplicación.gestion.Hospedaje;
8 import gestorAplicación.gestion.Terminal;
9 import gestorAplicación.gestion.Viaje;
10
11 public class Bus extends Vehiculo implements Serializable {
12
13     private static final long serialVersionUID = 2378919680109241789L;
14     private static ArrayList<Bus> buses = new ArrayList<Bus>();
15     private ArrayList<Asiento> asientos = new ArrayList<Asiento>();
16     private ArrayList<Viaje> viajes = new ArrayList<Viaje>();
17     private int[] tiposAsientoFila;
18     private int asiento;
```

Clase Hospedaje ubicada en el paquete gestorAplicación.gestion

```java
package gestorAplicación.gestion;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.ArrayList;

import gestorAplicación.personas.Conductor;
import gestorAplicación.transporte.Asiento;

public class Hospedaje implements Serializable {
    private static final long serialVersionUID = 3316398631943862366L;
    private static ArrayList<Hospedaje> hospedajes = new ArrayList<Hospedaje>();
    private ArrayList<Habitacion> habitaciones = new ArrayList<Habitacion>();
    private ArrayList<String> calificaciones = new ArrayList<String>();
    private double calificacion;
    private String nombre;
    private String ubicacion;
```

Clase Pasajero en el paquete gestorAplicación.personas

```java
package gestorAplicación.personas;

import java.io.Serializable;
import java.time.LocalDate;
import java.util.ArrayList;

import gestorAplicación.gestion.Empresa;
import gestorAplicación.gestion.Hospedaje;
import gestorAplicación.gestion.Tiquete;
import gestorAplicación.gestion.Viaje;

public class Pasajero extends Persona implements Serializable {
    private static final long serialVersionUID = -8124260530486820488L;
    private static ArrayList<Pasajero> pasajeros = new ArrayList<Pasajero>();
    private ArrayList<Tiquete> tiquetes = new ArrayList<Tiquete>();
```

Además pudimos implementar varios métodos estáticos, es decir, de clase que nos ayudaron al momento de la creación de las funcionalidades, por ejemplo para buscar objetos dentro del ArrayList que contiene a todos los objetos creados mediante el uso de atributos que los hagan únicos como identificadores en caso de viajes, pasajeros o conductores, de placas en caso de carros o de nombre en caso de empresas; además creamos otros métodos de clase para propósitos varios como por ejemplo la generación de matrículas de vehículos de manera aleatoria y la verificación de que estas sean únicas

En la clase Pasajero en el paquete gestorAplicación.personas

```java
public static Pasajero buscarPasajero(String nombre, String id) {
    for (Pasajero pasajero : pasajeros) {
        if (pasajero.nombre.equals(nombre) && pasajero.id != null) {
            if (pasajero.id.equals(id)) {
                return pasajero;
            }
        }
    }
    return null;
}
```

En la clase Terminal en el paquete gestorAplicación.gestion

```java
28     public static Terminal buscarTerminal(String ubicacion) {
29         ubicacion=ubicacion.toUpperCase();
30         for (Terminal terminal : terminales) {
31             if (terminal.getUbicacion().equals(ubicacion)) {
32                 return terminal;
33             }
34         }
35         return null;
36     }
```

En la clase Asiento en el paquete gestorAplicación.transporte

```java
69     public static Asiento buscarAsiento(String numero, String tipo) {
70         for (Asiento asiento : asientos) {
71             if (asiento.getNumero() != null && asiento.getTipoAsiento() != null) {
72                 if (asiento.getNumero().equals(numero) && asiento.getTipoAsiento().
73                         equals(TipoAsiento.valueOf(tipo))) {
74                     return asiento;
75                 }
76             }
77         }
78         return null;
79     }
```

En la clase Vehiculo en el paquete gestorAplicación.transporte

```
51      public static String generarPlaca() {
52          Random aleatorio = new Random();
53          ArrayList<String> letras = new ArrayList<String>();
54          String string = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
55          String[] parts = string.split("");
56          for (String letra : parts) {
57              letras.add(letra);
58          }
59          while (true) {
60              String r1 = letras.get(aleatorio.nextInt(26));
61              String r2 = letras.get(aleatorio.nextInt(26));
62              String r3 = letras.get(aleatorio.nextInt(26));
63              int r4 = aleatorio.nextInt(10);
64              int r5 = aleatorio.nextInt(10);
65              int r6 = aleatorio.nextInt(10);
66              String placa = r1 + r2 + r3 + "-" + r4 + r5 + r6;
67
68              if (verificarPlaca(placa)) {
69                  return placa;
70              }
71          }
72
73      }
74
75      public static boolean verificarPlaca(String placa) {
76          boolean ok = true;
77          for (String placa1 : placas) {
78              if (placa1.equals(placa)) {
79                  ok = false;
80              }
81          }
82          return ok;
83      }
```

- **Uso de constante (1 caso)**

Ubicada en la Interfaz dentro del paquete uiMain

```
567
568                             final String numeroAsiento2 = numeroAsiento;
569
```

Las constantes sirven para definir un valor en una variable que no se pueda modificar, lo cual es realmente útil especialmente en nuestro contexto del programa al momento de buscar algún objeto que no queramos modificar directamente desde un apartado del programa sino que queramos tenerlo de referencia para realizar otro tipo de operaciones sin que conlleven ningún riesgo, por ejemplo en este caso estamos reservando un asiento por un periodo de tiempo por lo cual no queremos que este pueda ser modificado de manera intencional o no durante el tiempo que esté reservado.

Además de ello implementamos una clase de tipo enumerado la cual define atributos constantes de la clase que en este caso será el tipo de asientos que pueden existir en un bus (preferencial, estandar y premium)

Ubicado en la clase tipo enumerado llamada TipoAsiento en el paquete gestorAplicación.transporte

```java
public enum TipoAsiento implements Serializable {
    ESTANDAR,
    PREMIUM,
    PREFERENCIAL;
}
```

- **Encapsulamiento (private, protected y public).**

Ubicado en la clase Vehiculo en el paquete gestorAplicación.transporte

```java
package gestorAplicación.transporte;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Random;

import gestorAplicación.personas.Conductor;

public abstract class Vehiculo implements Serializable {

    private static ArrayList<String> placas = new ArrayList<String>();
    private static final long serialVersionUID = 1613039627515609694L;
    private String placa;
    private Conductor conductor;

    protected Vehiculo() {

    }

    protected Vehiculo(String placa) {
        this.placa = placa;

    }

    public abstract void crearAsientos(int asientos);

    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }
```

Como norma general el encapsulamiento nos permite restringir el acceso a ciertos métodos o atributos desde clase externas a la propia, a subclases o a clases que no estén dentro del mismo paquete, nosotros decidimos adaptarnos a la convención de que los métodos se ponen privados y los métodos públicos, ya que esto genera un buen encapsulamiento en los elementos de una clase sin perder practicidad, además le generamos los métodos getter y setter a casi todos los atributos privados a excepción de algunos que no tienen sentido, ya que una vez creada la instancia no se puede o no se

debe cambiar dichos atributos porque afectaría el programa, además de ello pusimos los constructores de la clase abstracta Vehiculo como protected, ya que estas nunca pueden ser instanciadas y únicamente podrán ser accedidos por medio de clases hijas, todo esto resulta muy útil para que la información se mantenga lo más protegida posible durante el transcurso de creación y ejecución del programa

Ubicado en la clase Tiquete en el paquete gestorAplicación.gestion

```
13 public class Tiquete implements Serializable {
14     private static final long serialVersionUID = 5057370312141507904L;
15     private static ArrayList<Tiquete> tiquetes = new ArrayList<Tiquete>();
16     private static int numerosReserva = 1000000;
17     private Pasajero pasajero;
18     private Viaje viaje;
19     private Asiento asiento;
20     private String numeroReserva;
21     private Hospedaje hospedaje;
22 
23     public Tiquete () {
24         this.numeroReserva = String.valueOf(numerosReserva);
25     }
26 
27     public Tiquete(Pasajero pasajero, Viaje viaje, Asiento asiento, Hospedaje hospedaje) {
28         this.pasajero = pasajero;
29         this.viaje = viaje;
30         this.asiento = asiento;
31         this.numeroReserva = String.valueOf(numerosReserva);
32         this.hospedaje = hospedaje;
33         tiquetes.add(this);
34         numerosReserva++;
35     }
36 
37     public Tiquete(Pasajero pasajero, Viaje viaje, Asiento asiento) {
38         this.pasajero = pasajero;
39         this.viaje = viaje;
40         this.asiento = asiento;
41         this.numeroReserva = String.valueOf(numerosReserva);
42         this.hospedaje = null;
43         tiquetes.add(this);
44         numerosReserva++;
45     }
46 
```

- **Los siguientes conceptos asociados a la POO:**

o Sobrecarga de métodos(1 casos mínimo) y constructores(2 casos mínimo)

Ubicada en la clase Tiquete en el paquete gestorAplicación.gestion (Sobrecarga de constructores)

```java
public Tiquete () {
    this.numeroReserva = String.valueOf(numerosReserva);
}

public Tiquete(Pasajero pasajero, Viaje viaje, Asiento asiento, Hospedaje hospedaje) {
    this.pasajero = pasajero;
    this.viaje = viaje;
    this.asiento = asiento;
    this.numeroReserva = String.valueOf(numerosReserva);
    this.hospedaje = hospedaje;
    tiquetes.add(this);
    numerosReserva++;
}

public Tiquete(Pasajero pasajero, Viaje viaje, Asiento asiento) {
    this.pasajero = pasajero;
    this.viaje = viaje;
    this.asiento = asiento;
    this.numeroReserva = String.valueOf(numerosReserva);
    this.hospedaje = null;
    tiquetes.add(this);
    numerosReserva++;
}
```

Ubicada en la clase Asiento en el paquete gestorAplicación.transporte (Sobrecarga de constructores)

```java
15 public class Asiento implements Serializable {
16     private static final long serialVersionUID = 6674047871371131306L;
17     private static ArrayList<Asiento> asientos = new ArrayList<Asiento>();
18     private String numero;
19     private boolean reservado;
20     private LocalDateTime fechaReserva;
21     private TipoAsiento tipoAsiento;
22
23     public Asiento() {
24         this("Indefinido");
25         asientos.add(this);
26     }
27
28     public Asiento(String numero) {
29         this.numero = numero;
30         asientos.add(this);
31     }
32
33     public Asiento(String numero, TipoAsiento tipo) {
34         this.numero = numero;
35         this.tipoAsiento = tipo;
36         asientos.add(this);
37     }
```

Ubicada en la clase Tiquete en el paquete gestorAplicación.gestion (Sobrecarga de constructores)

```java
package gestorAplicación.gestion;

import java.io.Serializable;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import gestorAplicación.personas.Persona;
import gestorAplicación.transporte.Asiento;

public class Habitacion implements Serializable {
    private static final long serialVersionUID = 6655776532233087484L;
    private static ArrayList<Habitacion> habitaciones = new ArrayList<Habitacion>();
    private Hospedaje hospedaje;
    private String numeroHabitacion;
    private boolean reservada;
    private LocalDateTime fechaReserva;
    private String ubicacion;

    public Habitacion(String numeroHabitacion) {
        this.numeroHabitacion = numeroHabitacion;
    }

    public Habitacion(Hospedaje hospedaje, String numeroHabitacion, String ubicacion)
        this.hospedaje = hospedaje;
        this.numeroHabitacion = numeroHabitacion;
        this.ubicacion = ubicacion;
    }
```

La sobrecarga tanto de métodos como de constructores es fundamental a la hora de crear programas complejos, ya que habilitan varias vías en el proceso de creación de instancias para poder inicializar sus atributos de una manera diferente según sea necesario en cada caso, por ejemplo en nuestro caso fue de mucha ayuda cuando en habitación por ejemplo conocíamos el dato del hospedaje y la ubicación además del número podíamos crear una instancia e inicializarla con varios de sus atributos, mientras en otros casos en los que sólo conocíamos el número de la habitación usábamos tranquilamente el constructor que únicamente recibía ese argumento y no quedamos en problemas gracias a la sobrecarga de constructores, lo mismo pasa con los métodos, por ejemplo en el caso de buscar pasajeros, si teníamos únicamente el id o todos los datos del pasajero podríamos buscarlo sin ningún problema facilitando muchas cosas

Ubicado en la clase Empresa en el paquete gestoraAplicación.gestion (Esta y las siguientes 3 imágenes; ejemplo de Sobrecarga de métodos)

```java
30    public static ArrayList<Viaje> buscarViajes(LocalDate fecha) {
31        ArrayList<Viaje> viajes = new ArrayList<Viaje>();
32
33        for (Empresa empresa : empresas) {
34            for (Viaje viaje : empresa.getViajes()) {
35                if (fecha.equals(viaje.getFecha())) {
36                    viajes.add(viaje);
37                }
38            }
39        }
40
41        return viajes;
42    }
43
44    public static ArrayList<Viaje> buscarViajes(String origen, String destino) {
45        ArrayList<Viaje> viajes = new ArrayList<Viaje>();
46
47        if (destino.isBlank()) {
48            for (Empresa empresa : empresas) {
49                for (Viaje viaje : empresa.getViajes()) {
50                    if (origen.equals(viaje.getTerminalOrigen().getUbicacion())) {
51                        viajes.add(viaje);
52                    }
53                }
54            }
55        } else if (origen.isBlank()) {
56            for (Empresa empresa : empresas) {
57                for (Viaje viaje : empresa.getViajes()) {
58                    if (destino.equals(viaje.getTerminalDestino().getUbicacion())) {
59                        viajes.add(viaje);
60                    }
61                }
```

```
62              }
63          } else {
64              for (Empresa empresa : empresas) {
65                  for (Viaje viaje : empresa.getViajes()) {
66                      if (viaje.tieneSillas()) {
67                          if (origen.equals(viaje.getTerminalOrigen().getUbicacion())
68                                  && destino.equals(viaje.getTerminalDestino().getUbicacion())
69                                  && LocalDateTime.now().
70                                  isBefore(LocalDateTime.of(viaje.getFecha(), viaje.getHora()))) {
71                              viajes.add(viaje);
72                          }
73                      }
74                  }
75              }
76          }
77
78          return viajes;
79      }
80
81      public static ArrayList<Viaje> buscarViajes(LocalTime hora) {
82          ArrayList<Viaje> viajes = new ArrayList<Viaje>();
83
84          for (Empresa empresa : empresas) {
85              for (Viaje viaje : empresa.getViajes()) {
86                  if (hora.equals(viaje.getHora())) {
87                      viajes.add(viaje);
88                  }
89              }
90          }
91
92          return viajes;
93      }
94
```

```
    }

    public static ArrayList<Viaje> buscarViajes(String string) {
        ArrayList<Viaje> viajes = new ArrayList<Viaje>();

        for (Empresa empresa : empresas) {
            for (Viaje viaje : empresa.getViajes()) {
                if (string.equals(viaje.getId())) {
                    viajes.add(viaje);
                }
            }
        }

        return viajes;
    }

    public static Viaje buscarViaje(String id) {
        for (Empresa empresa : empresas) {
            for (Viaje viaje : empresa.getViajes()) {
                if (id.equals(viaje.getId())) {
                    return viaje;
                }
            }
        }

        return null;
    }

    public ArrayList<Viaje> getViajes() {
        return viajes;
    }
```

Ubicado en la clase Pasajero en el paquete gestorAplicación.personas (Sobrecarga de métodos)

```
79
80      public Tiquete buscarTiquete(Viaje viaje) {
81          for (Tiquete tiquete : tiquetes) {
82              tiquete.setViaje(Empresa.buscarViaje(viaje.getId()));
83              if (tiquete.getViaje().equals(viaje)) {
84                  return tiquete;
85              }
86          }
87
88          return null;
89      }
90
91      public Tiquete buscarTiquete(String numeroReserva) {
92          for (Tiquete tiquete : tiquetes) {
93              if (tiquete.getNumeroReserva().equals(numeroReserva)) {
94                  return tiquete;
95              }
96          }
97
98          return null;
99      }
```

o Manejo de referencias this para desambiguar y this() entre otras. 2 casos mínimo para cada caso

Ubicado en la clase Pasajero en el paquete gestorAplicación.personas (caso de this() y paso del this como referencia al obejto)

```
12 public class Pasajero extends Persona implements Serializable {
13      private static final long serialVersionUID = -8124260530486820488L;
14      private static ArrayList<Pasajero> pasajeros = new ArrayList<Pasajero>();
15      private ArrayList<Tiquete> tiquetes = new ArrayList<Tiquete>();
16
17      public Pasajero() {
18          this("Sin nombre","0");
19          pasajeros.add(this);
20      }
21
22      public Pasajero(String nombre, String id) {
23          this(nombre, id,"0000000000","No tiene");
24          pasajeros.add(this);
25      }
26
27      public Pasajero(String nombre, String idPasajero, String telefono, String correo) {
28          super(nombre, idPasajero, telefono, correo);
29          pasajeros.add(this);
30      }
31
```

Ubicado en la clase Hospedaje en el paquete gestorAplicación.gestion (caso de this() , this. Para desambiguar y paso del this como referencia del objeto mismo)

```
10 public class Hospedaje implements Serializable {
11     private static final long serialVersionUID = 3316398631943862366L;
12     private static ArrayList<Hospedaje> hospedajes = new ArrayList<Hospedaje>();
13     private ArrayList<Habitacion> habitaciones = new ArrayList<Habitacion>();
14     private ArrayList<String> calificaciones = new ArrayList<String>();
15     private double calificacion;
16     private String nombre;
17     private String ubicacion;
18 
19     public Hospedaje() {
20         this("Sin nombre","Sin ubicación");
21         hospedajes.add(this);
22     }
23 
24     public Hospedaje(String nombre, int pisos, int habitacionesPiso) {
25         this.nombre = nombre;
26         crearHabitaciones(pisos, habitacionesPiso);
27         hospedajes.add(this);
28     }
29 
30     public Hospedaje(String nombre, String ubicacion) {
31         this(nombre,3,5);
32         ubicacion=ubicacion.toUpperCase();
33         this.ubicacion = ubicacion;
34         hospedajes.add(this);
35     }
```

Clase Habitación ubicada en el paquete gestorAplicación.gestion(caso de this. Para desambiguar)

```
23     public Habitacion(String numeroHabitacion) {
24         this.numeroHabitacion = numeroHabitacion;
25     }
26 
27     public Habitacion(Hospedaje hospedaje, String numeroHabitacion, String ubicacion) {
28         this.hospedaje = hospedaje;
29         this.numeroHabitacion = numeroHabitacion;
30         this.ubicacion = ubicacion;
31     }
```

El uso de este tipo de ayudas que ofrece java para desambiguar casos es muy útil para evitar errores, como por ejemplo el usar this. Para desambiguar variables, el llamar a otros métodos para pasarles valores a asignar por defecto (elegidos por nosotros, no por el programa) y el paso a otros objetos del mismo objeto también a través del this para crear una conexión más específica entre un objeto y otro y así poderlos relacionar entre sí.

o Implementación de un caso de enumeración

Enumerado llamado TipoAsiento ubicado en el paquete gestorAplicación.transporte

```
1  package gestorAplicación.transporte;
2
3  import java.io.Serializable;
4
5  public enum TipoAsiento implements Serializable {
6      ESTANDAR,
7      PREMIUM,
8      PREFERENCIAL;
9  }
10
```

El uso de clases enumerados nos facilita el proceso de asignar constantes de cierto tipo a atributos de objetos como en este caso, en el cual decidimos crear el enumerado con los valores estándar, premium y preferencial los cuales son los tipos de los que pueden ser las sillas en un bus cada una de ellas con un precio y ubicación diferentes, lo cuál también en nuestro caso nos permite separar las secciones de cada bus haciendo a cada uno único y diferente; cabe aclarar que cualquier otro valor por fuera del enumerado generaría error.

Descripción de cada una de las 5 funcionalidades implementadas

Ver viajes disponibles: inicialmente se podrán ver todos los viajes disponibles por empresa en todas las terminales, además de poder filtrar los resultados por diferentes categorías como por ejemplo la fecha, hora, destino, origen, id y placa del bus. Se le pedirá al usuario que ingrese el id de un viaje en específico para ver los asientos disponibles organizados por el tipo de asiento. Por último, el usuario podrá congelar el asiento por un período de tiempo de su elección siempre y cuando sea antes de la fecha del viaje. Después de ese período de tiempo, el asiento estará disponible para reservar.

Diagrama Funcionalidad 1.pdf

Reserva de tiquetes: se le pedirá al usuario que ingrese el origen y el destino del viaje, y se mostrarán todos los viajes que cumplan con los requisitos de búsqueda. Luego, el usuario podrá escoger un viaje mediante el número de id y podrá ver todos los asientos disponibles. Se le pedirá al usuario que escoja un número de asiento e ingrese sus datos personales. Finalmente, se imprimirán los detalles de la reserva.

Diagrama Funcionalidad 2.pdf

Gestión de tiquetes: para empezar, se le pedirá al usuario que ingrese un número de identificación para buscar todos los tiquetes asociados con dicho número de identificación. En caso de que no se encuentre ningún tiquete se imprimirá un mensaje que diga que no hay ningún tiquete asociado. Luego, los tiquetes se mostrarán en dos categorías: tiquetes válidos y vencidos. El usuario podrá escoger cualquiera de los tiquetes mediante el número de reserva. Si el tiquete pertenece a los tiquete vencidos, se mostrarán más detalles acerca del viaje. En cambio, si el tiquete pertenece a los tiquetes válidos, el usuario tendrá que escoger entre dos opciones: cancelarlo o modificarlo. En caso de que el usuario decida cancelarlo, se liberará el asiento que ocupaba en el viaje. Si decide modificarlo, podrá cambiar de asiento o de viaje. Si cambia de asiento, se marcará el asiento que ocupaba como disponible y se le pedirá que escoja uno nuevo. Si decide cambiar de viaje, el asiento del usuario quedará marcado como disponible y se hará un proceso similar al de la reserva de tiquetes. Al final, se podrán ver los detalles de la reserva modificada.

Diagrama Funcionalidad 3.pdf

Hospedaje: primero, se le pedirá al usuario que ingrese un número de identificación para buscar todos los tiquetes asociados con dicho número de identificación. En caso de que no se encuentre ningún tiquete se imprimirá un mensaje que diga que no hay ningún tiquete asociado. En caso de que sí encuentre tiquetes, se mostrarán en pantalla los viajes por cada tiquete. Luego, se le pedirá al usuario que escoja un viaje en específico mediante el id. Para el viaje escogido se mostrarán los hospedajes disponibles que varían dependiendo del destino; luego, el usuario escoge el hospedaje que desee mediante el nombre y se modifica el tiquete para que incluya el servicio de hospedaje escogido, a lo que finalmente el usuario podrá elegir la habitación en la que se desea quedar y reservar por el tiempo que quiera, lo cual la pondrá como ocupada hasta que pase el tiempo de la reserva.

Diagrama Funcionalidad 4.pdf

Opciones de administrador: aquí el usuario podrá realizar las siguientes acciones; primero deberá escoger entre que desea modificar: Empresas, Terminales, Hospedajes (Hospedajes y Habitaciones), Viajes (Viajes y Tiquetes), Personal (Pasajeros y Conductores), o buses. Posteriormente podrá Agregar, Modificar, Ver, o Eliminar objetos de dichas clases (No podrá hacer todo para todas las clases); también habrá algunas que tendrán funciones extras como agregar conductor a una empresa o agregar una calificación a un hospedaje. Finalmente el usuario podrá decidir si desea abandonar el menú de administrador e ir al menú principal o si seguir modificando objetos.

Diagrama Funcionalidad 5.pdf

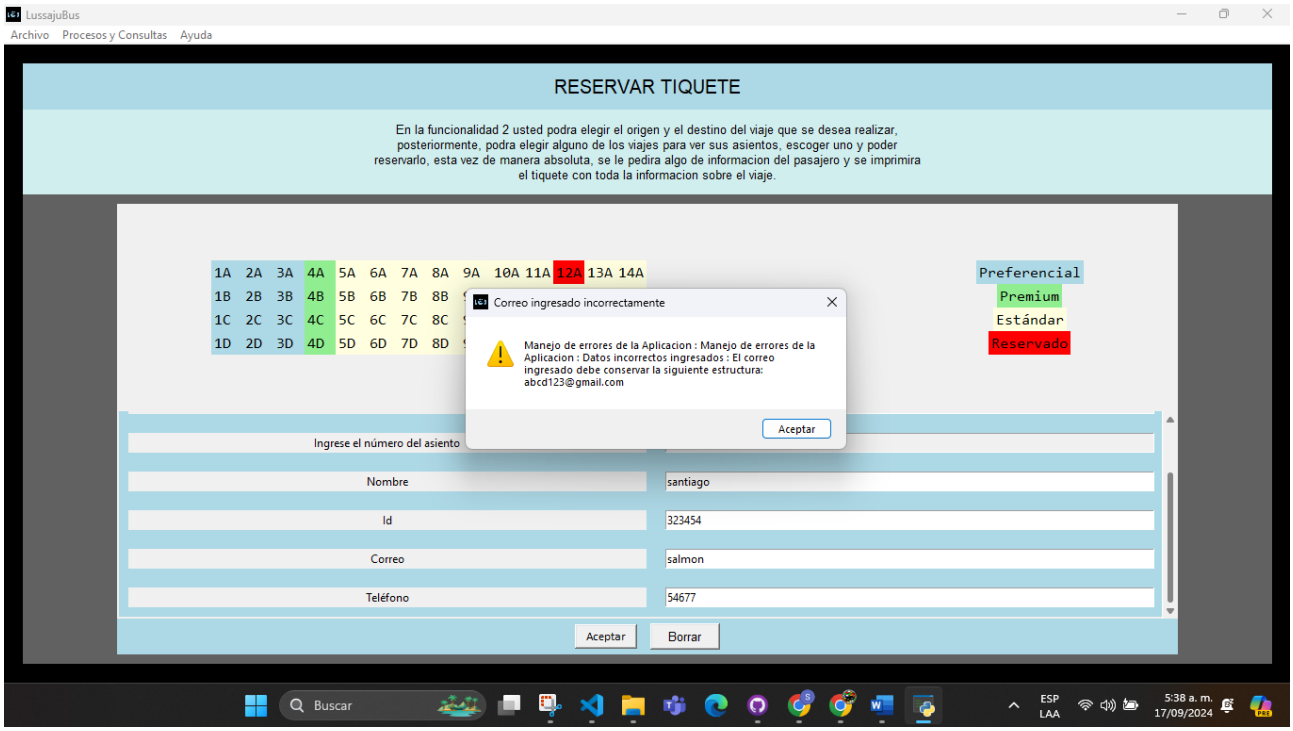
Manejo de excepciones

Para el manejo de excepciones, nosotros hemos decidido implementar dos subclases de la clase ErrorAplicacion que hereda a su vez de la clase Excepción, y estas dos subclases son: ExcepcionNoEncontrado la cual aporta el mensaje 'No se ha encontrado' al error que se mostrará al crear una instancia de esta clase y la cual irá ligada al mensaje dado por la clase ErrorAplicacion, el cual es: 'Manejo de errores de la Aplicacion', Y la otra clase es ExcepcionDatosIncorrectos, la cual aportará el mensaje: 'Datos incorrectos ingresados', ambas clases son usadas a lo largo del programa para tratar error relacionados con datos que el usuario ingresa y no existen los objetos con sus ids correspondientes o en el caso de datos incorrectos, por si el usuario ingresa un dato de manera incorrecta o no aceptada por el programa. Tuvimos un error con el primer mensaje, ya que este se imprimía dos veces siempre y no supimos como arreglarlo, porque cualquier cambio que le hiciéramos simplemente el código explotaba al acumular varias veces el mensaje por cada excepción que saliera causando un desbordamiento de pila con respecto a los mensajes.

Además estas clases implementaron otras subclases que complementan su mensaje para descrifrar y tratar correctamente el error correspondiente: Para el caso de Datos Incorrectos sus subclases son las siguientes:

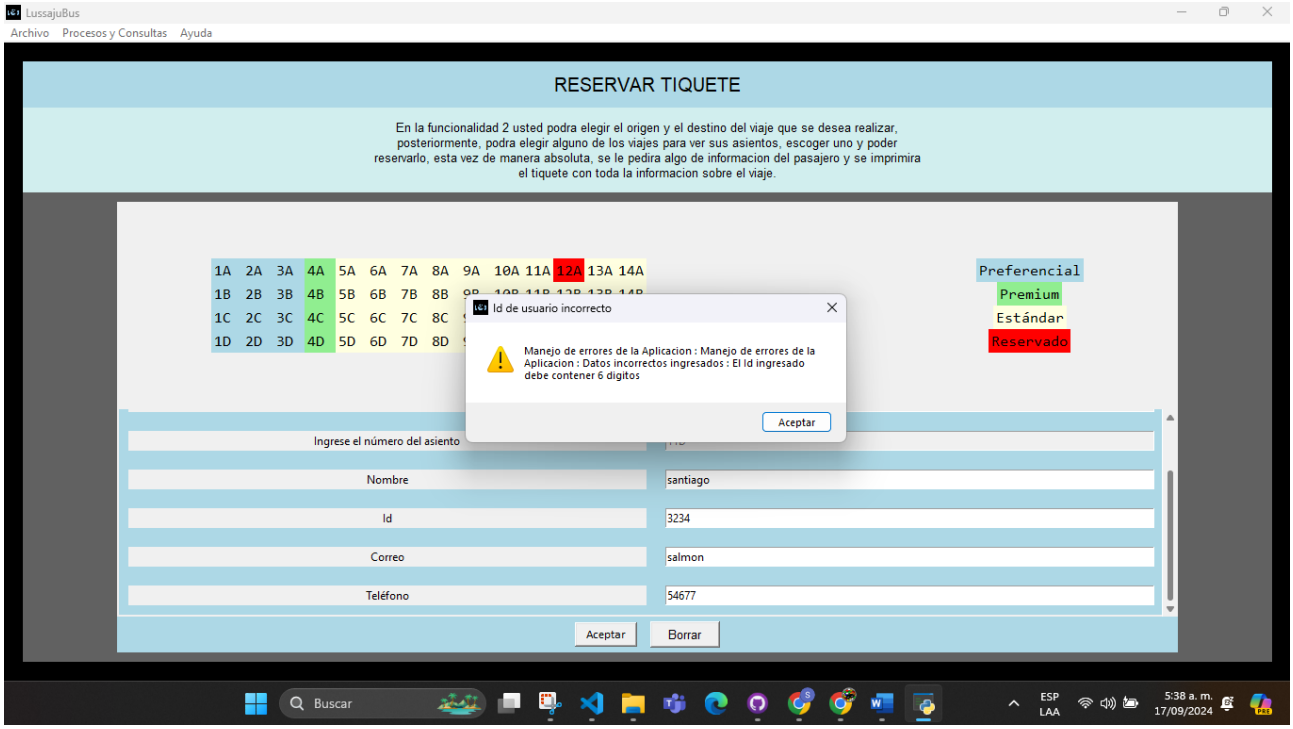
- ExcepcionCorreo: devuelve el mensaje: 'El correo ingresado debe conservar la siguiente estructura: abcd123@gmail.com' y se presenta cuando se le pide al usuario ingresar un correo y el correo ingresado por el usuario no conserva la estructura descrita anteriormente.

```
def cuarto_paso(self, viaje, asiento):
    if (ae.excepcion_id(self.field.entries["Id"].get())!="ok" or
        ae.excepcion_correo(self.field.entries["Correo"].get())!="ok" or
        ae.excepcion_telefono(self.field.entries["Teléfono"].get())!="ok"):
        self.cuarto_paso(self, viaje, asiento)

    else:
        reservar_tiquete.imprimir_tiquete(
        self.frame_superior,
        self.field,
        viaje,
        asiento)
        self.boton_aceptar.config(state="disabled")
        self.boton_borrar.config(state="disabled")
```



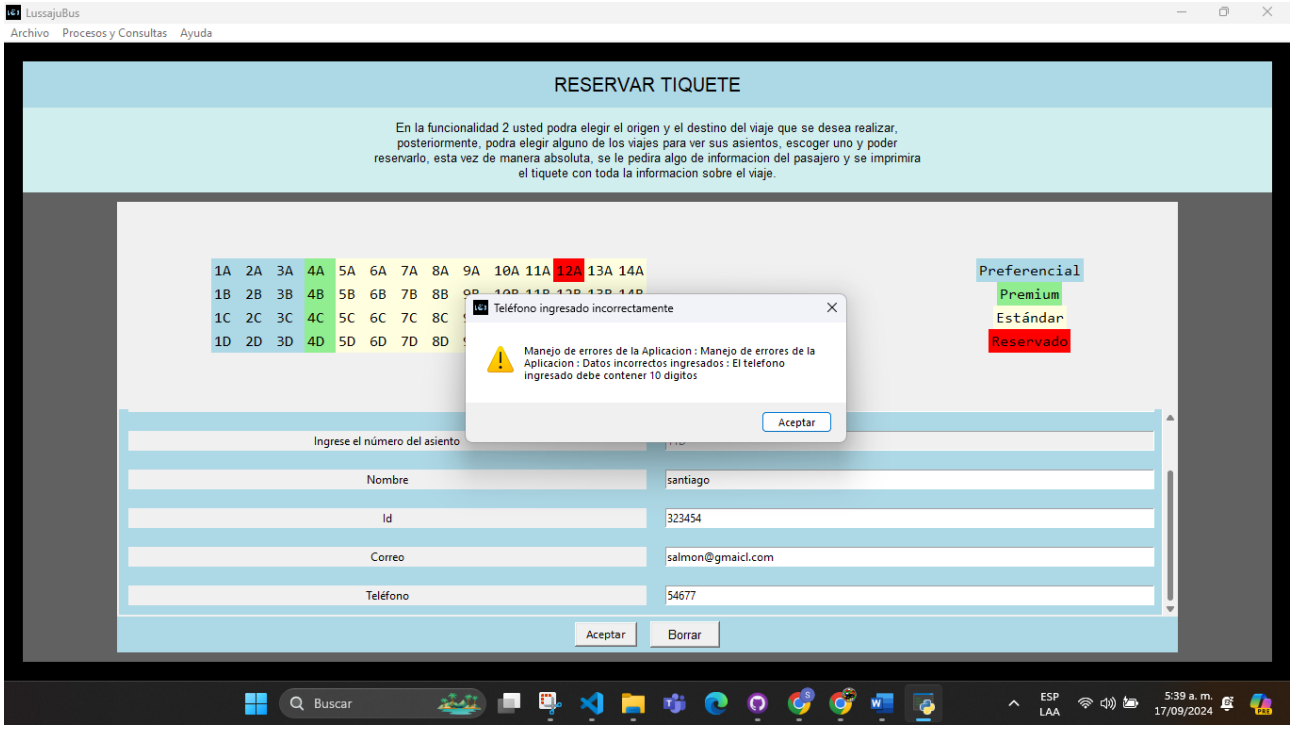
- ExcepcionID: devuelve el mensaje: 'El Id ingresado debe contener 6 digitos' y se presenta cuando el usuario debe ingresar su Id y este deberá de ser un Id válido es decir que cumpla con la condición anterior, por lo que esta excepción ayudará a verificar dicha información.

```
def cuarto_paso(self, viaje, asiento):
    if (ae.excepcion_id(self.field.entries["Id"].get())!="ok" or
        ae.excepcion_correo(self.field.entries["Correo"].get())!="ok" or
        ae.excepcion_telefono(self.field.entries["Teléfono"].get())!="ok"):
        self.cuarto_paso(self, viaje, asiento)

    else:
        reservar_tiquete.imprimir_tiquete(
        self.frame_superior,
        self.field,
        viaje,
        asiento)
        self.boton_aceptar.config(state="disabled")
        self.boton_borrar.config(state="disabled")
```



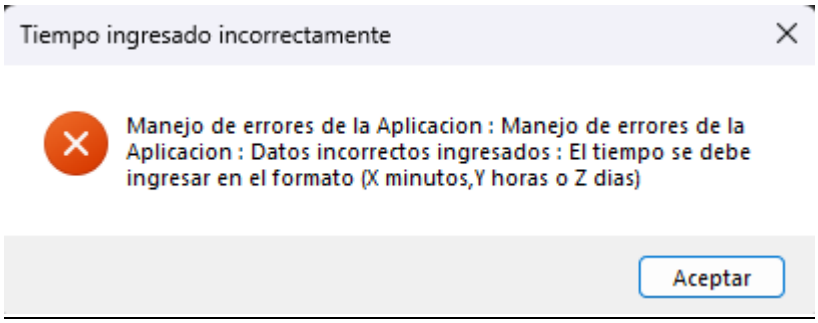
- ExcepcionTelefono: agrega el mensaje: 'El telefono ingresado debe contener 10 digitos'
  Y este servirá para verificar que el teléfono ingresado por el usuario tenga la estructura solicitada, de lo contrario por medio de verificaciones se pedirá que se vuelva a ingresar.

```
def cuarto_paso(self, viaje, asiento):
    if (ae.excepcion_id(self.field.entries["Id"].get())!="ok" or
        ae.excepcion_correo(self.field.entries["Correo"].get())!="ok" or
        ae.excepcion_telefono(self.field.entries["Teléfono"].get())!="ok"):
        self.cuarto_paso(self, viaje, asiento)

    else:
        reservar_tiquete.imprimir_tiquete(
        self.frame_superior,
        self.field,
        viaje,
        asiento)
        self.boton_aceptar.config(state="disabled")
        self.boton_borrar.config(state="disabled")
```



- ExcepcionTiempo:agrega el mensaje: ‘El tiempo se debe ingresar en el formato(X minutos, Y horas o Z dias)’ Y esto ayuda para comprobar que el tiempo que se separa un viaje o una habitación si pueda ser tomado de manera correcta y en el formato correcto, de otra manera generará una excepción de este tipo que puede ser tratada adecuadamente.

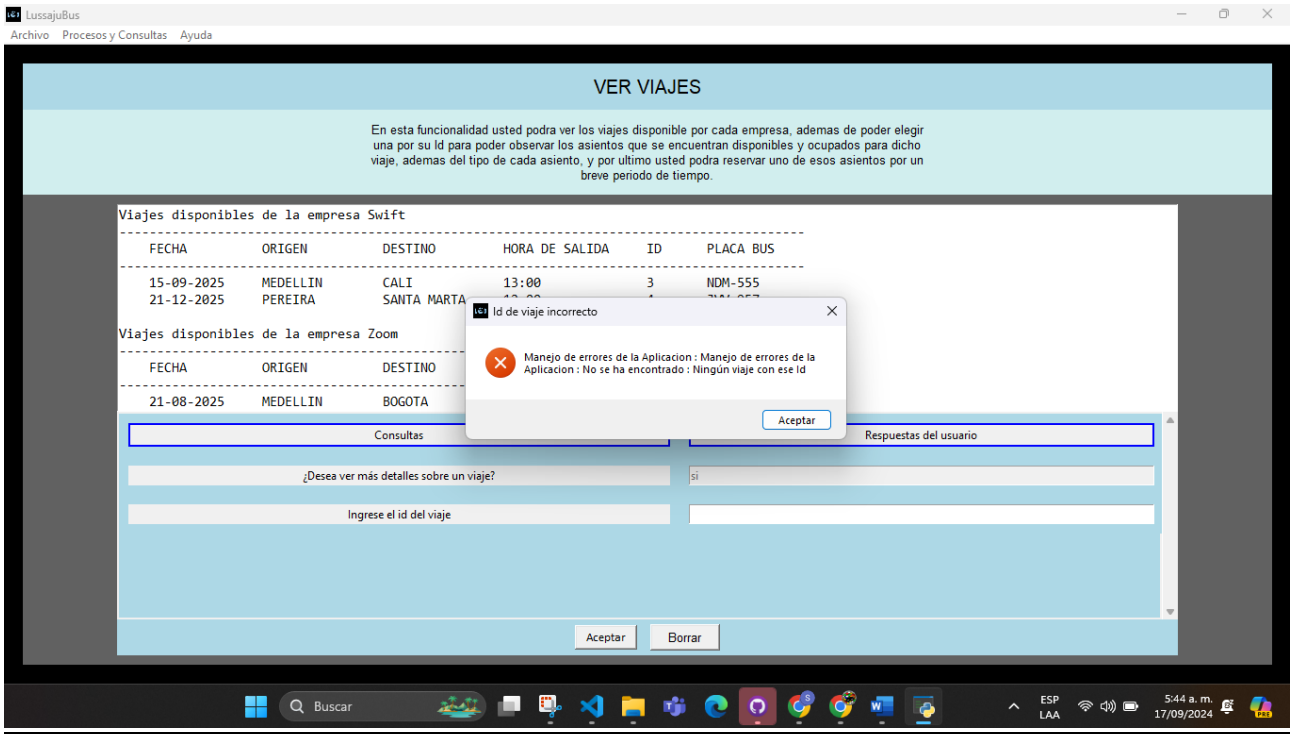
```
def quinto_paso(self):
    ver_viajes.quinta_pregunta(self)
    if (ae.excepcion_tiempo!="ok"):
        self.quinto_paso()
    else:
        self.field_frame.entries[
        "¿Por cuánto tiempo desea reservarlo?"
    ].config(state="disabled")

        self.boton_aceptar.config(state="disabled")
        self.boton_borrar.config(state="disabled")
```



- Excepción Valores Vacios: agrega el mensaje: 'El valor de los siguientes campos está vacío: ' y hace

que al darle click al botón aceptar se verifique que todos los campos del field_frame estén completamente diligenciados, de lo contrario se lanzará un error de este tipo que podrá ser tratado posteriormente.

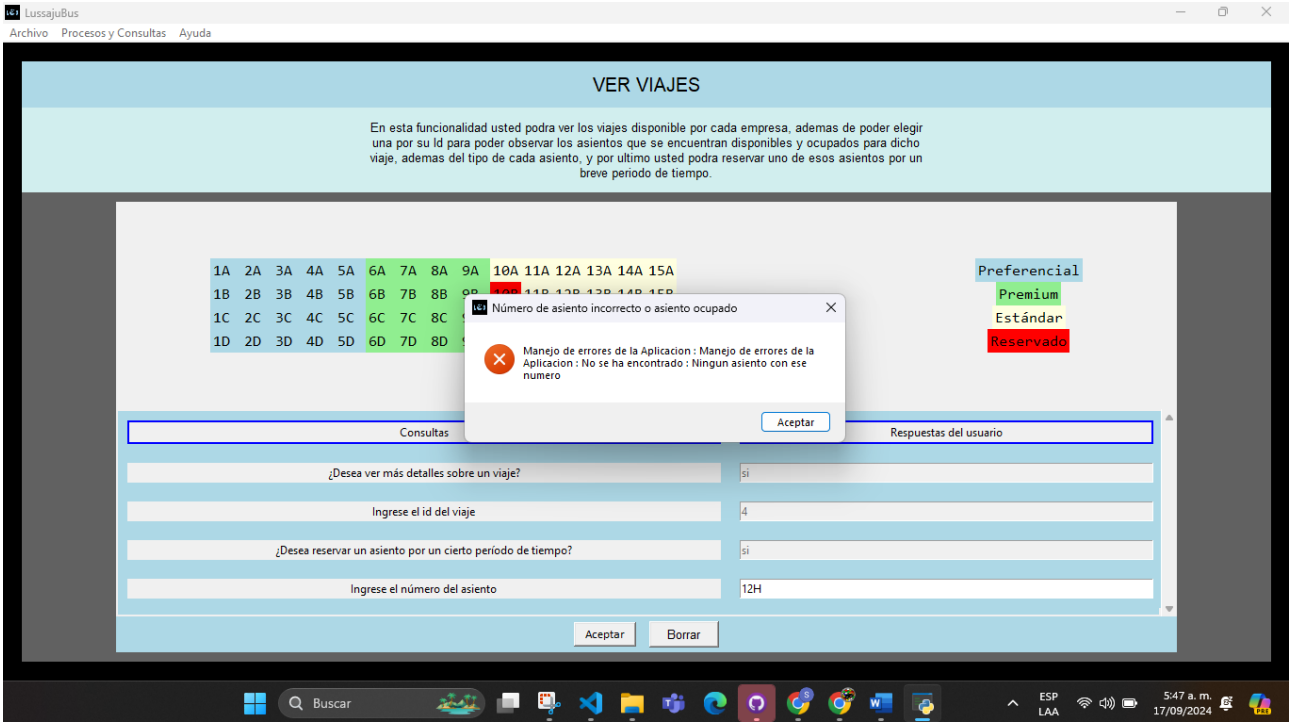
Finalmente no pudimos implementar el método como tal por falta de tiempo, pero su funcionamiento evita que existan espacios en blanco al presionar el botón aceptar, funciona correctamente, sólo que no lo pudimos implementar. Actualmente este error puede ser reemplazado por otros según el contexot:



Para el caso de ExcepcionNoEncontrado sus subclases son las siguientes:

- ExcepcionAsiento: genera el siguiente mensaje: 'Ningun asiento con ese numero'
  Y se encarga de verificar que el asiento si exista o que si esté disponible para un respectivo bus, esto se hace a través de unos métodos que se comentarán más adelante

```
@staticmethod
def cuarta_pregunta(frame_funcionalidad):
    respuesta = frame_funcionalidad.field_frame.getValue(
        "Ingrese el número del asiento"
    )

    indice=frame_funcionalidad.field_frame.getValue(
            "Ingrese el id del viaje")
    viaje = Empresa.buscar_viaje_por_id(indice)
    ok=ae.excepcion_viaje(indice)
    if ok=="ok":
        ae.excepcion_asiento(respuesta,viaje.get_bus())
```

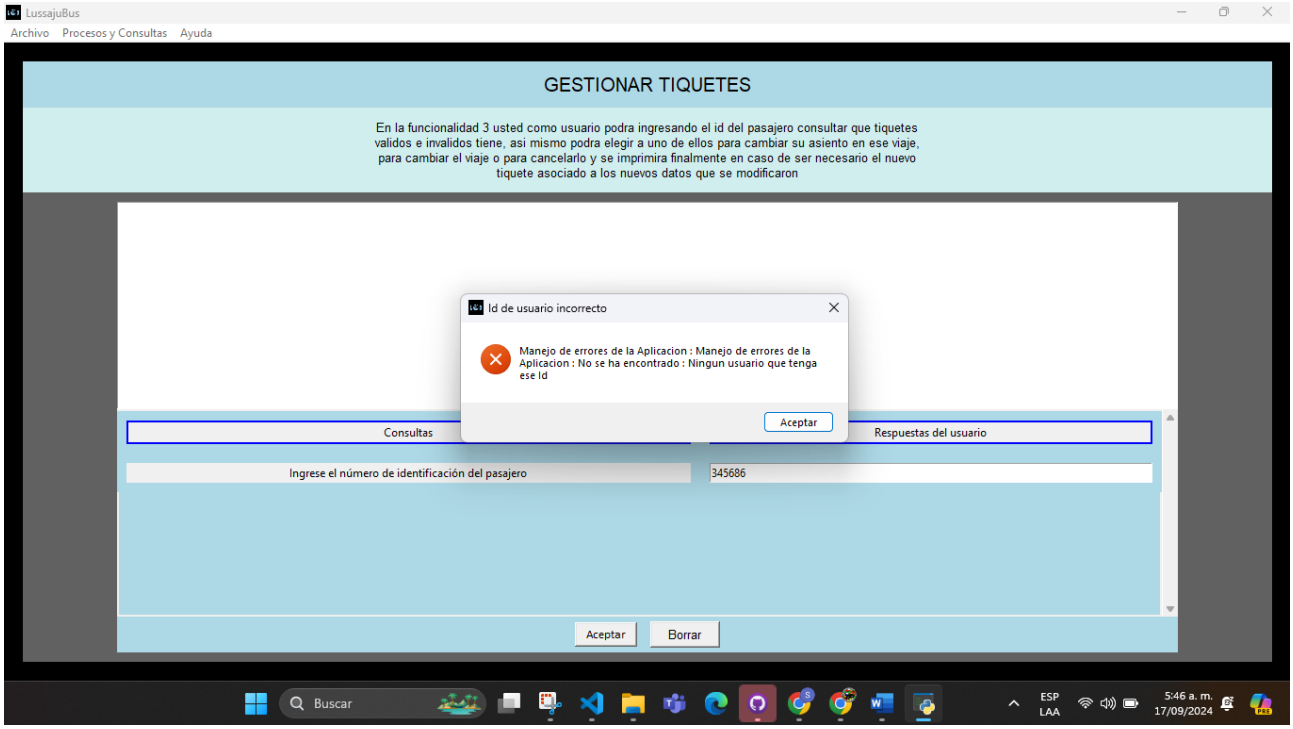


- ExcepcionHospedaje: genera el siguiente mensaje: 'Ningun hospedaje con ese nombre' y se encarga de verificar que el hospedaje exista para cualquier viaje, es decir que el nombre del hospedaje exista, de lo contrario se manejaría el error, esta es la descripción como tal de los métodos que implementan esta jerarquía de errores.

La sección de Hospedaje no la pudimos programar por falta de tiempo, por lo que dicha funcionalidad en la parte de Python y la 5 no están dentro del sistema

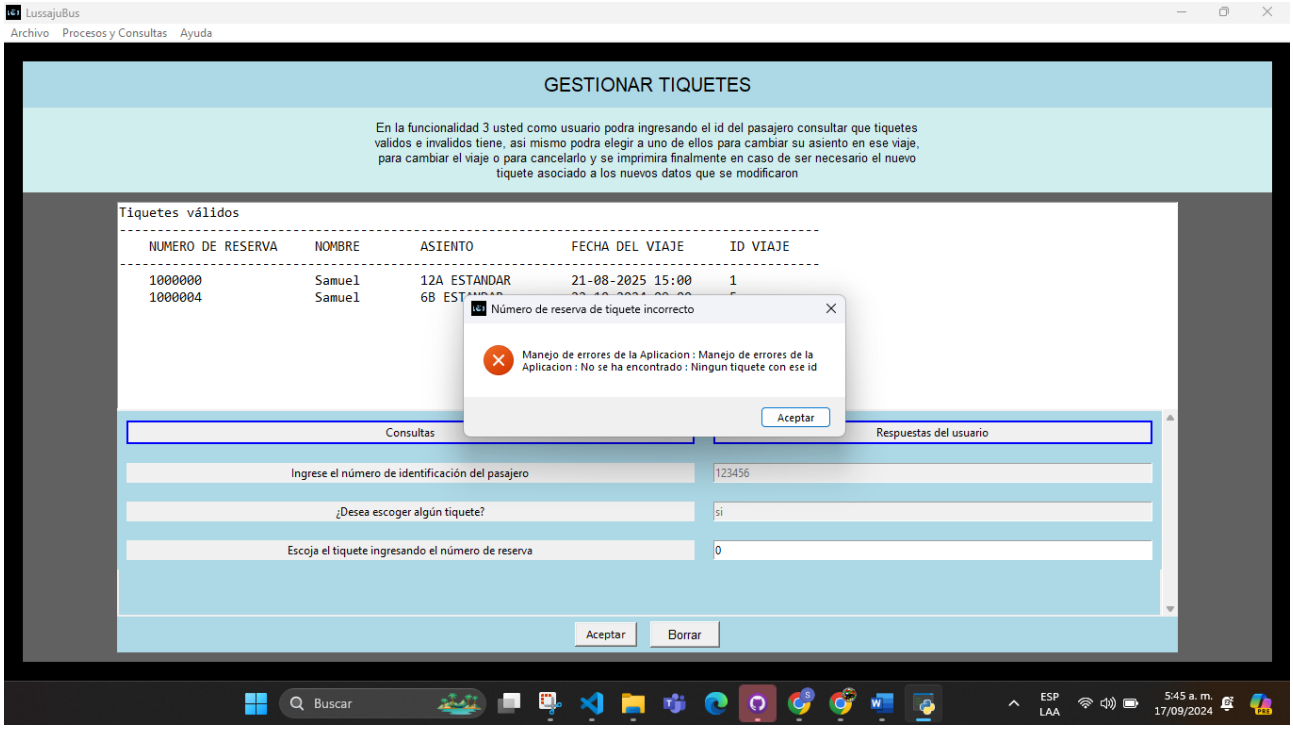
- ExcepcionIdUsuario: añade el siguiente mensaje: 'Ningun usuario que tenga ese Id' Y se encarga de verificar que exista algún usuario que tenga dicho Id dentro de la base de datos, de lo contrario se generará este error y se deberá de tratar de la manera adecuada.

```
@classmethod
def mostrar_tiquetes(cls, frame_funcionalidad):
    numero_identificacion = frame_funcionalidad.field_frame.getValue(
        "Ingrese el número de identificación del pasajero"
    )

    if (
        exepciones.excepcion_id(numero_identificacion) == "ok"
        and exepciones.excepcion_id_usuario(numero_identificacion) == "ok"
    ):
        pasajero = Pasajero.buscar_pasajero(numero_identificacion)
```



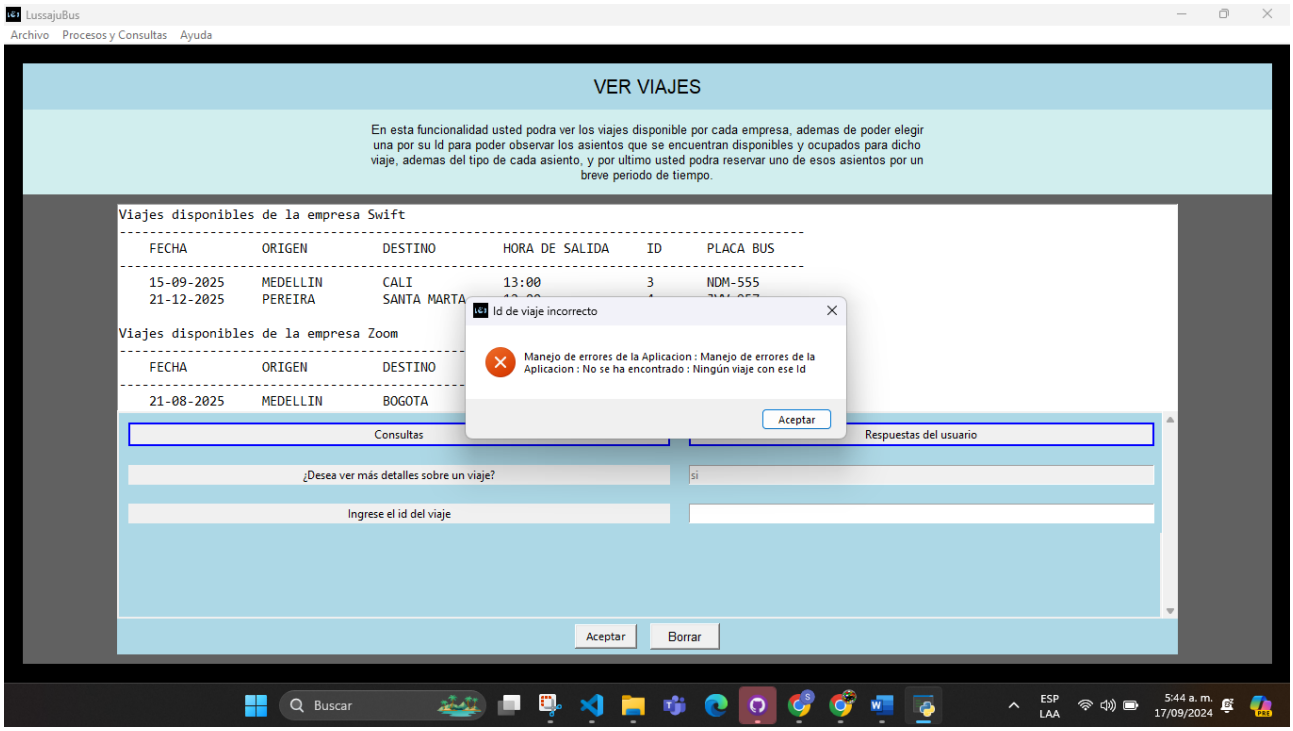
- ExcepcionTiquete:Es similar a las clase anteriores, su mensaje es el siguiente: 'Ningun tiquete con ese id' Y se usa para verificar que existe un tiquete según su número de reserva.

```
@classmethod
def seleccionar_tiquete(cls, frame_funcionalidad):
    numero_tiquete = frame_funcionalidad.field_frame.getValue(
        "Escoja el tiquete ingresando el número de reserva"
    )

    if exepciones.excepcion_tiquete(numero_tiquete) == "ok":
        tiquete = cls.PASAJERO.buscar_tiquete_por_reserva(numero_tiquete)

        if tiquete in cls.TIQUETES_VENCIDOS:
            frame_funcionalidad.text.config(state="normal")
            frame_funcionalidad.text.delete("1.0", "end")
            frame_funcionalidad.text.insert("end", "-" * 93 + "\n")
            frame_funcionalidad.text.insert(
```



- ExcepciónViaje: También es muy similar a las clases anteriores, esta en particular agrega el siguiente mensaje: 'Ningún viaje con ese Id' Y sirve para verificar que existe algún viaje que tengo un Id pasado como parámetro dentro de las funciones que describen el funcionamiento de estos errores.

```
@staticmethod
def mostrar_viajes(text, combobox_origen, combobox_destino):
    text.config(state="normal")
    text.delete("1.0", "end")
    origen = combobox_origen.get()
    destino = combobox_destino.get()

    viajes = Empresa.buscar_viajes_por_origen_destino(origen, destino)

    for viaje in viajes:
        if (ae.excepcion_viaje(viaje.get_id()))=="ok":
            print("ok")
```



También se nos pedía crear dos clases que fueran hijas de algún de las dos principales creadas por nosotros, en este caso EcepcionNo Encontrado y ExcepcionDatosIncorrectos, de las cuales una de ellas pudiera verificar si algún dato ingresado por el usuario es correcto o si pertenece a la base de datos del programa, y como básicamente la mayoría de nuestras clases de errores tienen esa función cualquiera de ellas podría hacer la veces de lo que pide el enunciado de la prática en ese sentido; y por otra parte se nos pedía otra clase para manejar errores, la cual sacara error si al darle al botón de aceptar en un fieldframe esta me compruebe si hay espacios sin llenar, es decir, en blanco dentro del field_frame y si es así que sacara una excepción para pedirle al usuario que diligencie todos los campos del field_frame para poder proseguir con el desarrollo de la funcionalidad; esta otra clase también fue descrita e implementada anteriormente y se llamaExcepcionValoresVacios.
Finalmente, para poder implementar la jerarquía de errores de la mejor manera se ha decidido crear un archivo que contenga todos los métodos que implementen dichas clases de errores para que puedan ser usadas dentro de las funcionalidades simplemente como métodos y que permita realizar las verificaciones necesarias para el desarrollo del programa, el archivo que contiene a todos esos métodos se llama: auxiliar_excepciones.py y el archivo que contiene a todas las excepciones se llama:excepciones.py

Excepciones:

```python
class ErrorAplicacion(Exception):
    def __init__(self, mensaje='Manejo de errores de la Aplicacion'):
        self.mensaje = mensaje
        super().__init__(self.mensaje)

class ExcepcionNoEncontrado(ErrorAplicacion):
    def __init__(self, mensaje='No se ha encontrado'):
        super().__init__(f"{ErrorAplicacion().mensaje} : {mensaje}")

class ExcepcionDatosIncorrectos(ErrorAplicacion):
    def __init__(self,mensaje2='Datos incorrectos ingresados'):
        super().__init__(f"{ErrorAplicacion().mensaje} : {mensaje2}")

class ExcepcionAsiento(ExcepcionNoEncontrado):
    def __init__(self,mensaje2='Ningun asiento con ese numero'):
        super().__init__(f"{ExcepcionNoEncontrado().mensaje} : {mensaje2}")

class ExcepcionCorreo(ExcepcionDatosIncorrectos):
    def __init__(self,mensaje2='El correo ingresado debe conservar la siguiente estructura: abcd123@gmail.com'):
        super().__init__(f"{ExcepcionDatosIncorrectos().mensaje} : {mensaje2}")

class ExcepcionHospedaje(ExcepcionNoEncontrado):
    def __init__(self,mensaje2='Ningun hospedaje con ese nombre'):
        super().__init__(f"{ExcepcionNoEncontrado().mensaje} : {mensaje2}")

class ExcepcionId(ExcepcionDatosIncorrectos):
    def __init__(self,mensaje2='El Id ingresado debe contener 6 digitos'):
        super().__init__(f"{ExcepcionDatosIncorrectos().mensaje} : {mensaje2}")
```

```python
class ExcepcionHospedaje(ExcepcionNoEncontrado):
    def __init__(f"{ExcepcionNoEncontrado().mensaje} : {mensaje2}")

class ExcepcionId(ExcepcionDatosIncorrectos):
    def __init__(self,mensaje2='El Id ingresado debe contener 6 digitos'):
        super().__init__(f"{ExcepcionDatosIncorrectos().mensaje} : {mensaje2}")

class ExcepcionIdUsuario(ExcepcionNoEncontrado):
    def __init__(self,mensaje2='Ningun usuario que tenga ese Id'):
        super().__init__(f"{ExcepcionNoEncontrado().mensaje} : {mensaje2}")

class ExcepcionTelefono(ExcepcionDatosIncorrectos):
    def __init__(self,mensaje2='El telefono ingresado debe contener 10 digitos'):
        super().__init__(f"{ExcepcionDatosIncorrectos().mensaje} : {mensaje2}")

class ExcepcionTiempo(ExcepcionDatosIncorrectos):
    def __init__(self,mensaje2='El tiempo se debe ingresar en el formato (X minutos,Y horas o Z dias)'):
        super().__init__(f"{ExcepcionDatosIncorrectos().mensaje} : {mensaje2}")

class ExcepcionTiquete(ExcepcionNoEncontrado):
    def __init__(self, mensaje2='Ningun tiquete con ese id'):
        super().__init__(f"{ExcepcionNoEncontrado().mensaje} : {mensaje2}")

class ExcepcionValoresVacios(ExcepcionDatosIncorrectos):
    def __init__(self,mensaje2='El valor de los siguientes campos está vacío: '):
        super().__init__(f"{ExcepcionDatosIncorrectos().mensaje} : {mensaje2}")

class ExcepcionViaje(ExcepcionNoEncontrado):
    def __init__(self, mensaje2='Ningún viaje con ese Id'):
        super().__init__(f"{ExcepcionNoEncontrado().mensaje} : {mensaje2}")
```

Métodos de implementación de las excepciones:

```python
from excepciones import ExcepcionId
from excepciones import ExcepcionCorreo
from excepciones import ExcepcionTelefono
from excepciones import ExcepcionTiempo
from excepciones import ExcepcionIdUsuario
from excepciones import ExcepcionViaje
from excepciones import ExcepcionTiquete
from excepciones import ExcepcionHospedaje
from excepciones import ExcepcionAsiento
from excepciones import ExcepcionValoresVacios

from field_frame import field_frame

from gestorAplicación.gestion.empresa import Empresa
from gestorAplicación.gestion.viaje import Viaje
from gestorAplicación.gestion.tiquete import Tiquete
from gestorAplicación.personas.pasajero import Pasajero
from gestorAplicación.gestion.terminal import Terminal
from gestorAplicación.gestion.hospedaje import Hospedaje
from gestorAplicación.transporte.bus import Bus

from tkinter import messagebox
import tkinter as tk
import re
```

```python
def excepcion_id(id):
    try:
        if len(str(id))!=6 or int(id)<0:
            raise ExcepcionId()
    except ExcepcionId as e:
        messagebox.showwarning("Id de usuario incorrecto",e)
    except:
        messagebox.showwarning("Id de usuario incorrecto","El id de usuario debe ser un número")
    else:
        return "ok"

def excepcion_correo(correo):
    try:
        patron=r'^[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z]{2,}$'
        if not re.match(patron,correo):
            raise ExcepcionCorreo()
    except ExcepcionCorreo as e:
        messagebox.showwarning("Correo ingresado incorrectamente",e)
    except:
        messagebox.showwarning("Correo ingresado incorrectamente","El correo ingresado es incorrecto, ingréselo nue
    else:
        return "ok"

def excepcion_telefono(telefono):
    try:
        if len(str(telefono))!=10 or int(telefono)<0:
            raise ExcepcionTelefono()
    except ExcepcionTelefono as e:
        messagebox.showwarning("Teléfono ingresado incorrectamente",e)
```

```python
def excepcion_telefono(telefono):
    try:
        if len(str(telefono))!=10 or int(telefono)<0:
            raise ExcepcionTelefono()
    except ExcepcionTelefono as e:
        messagebox.showwarning("Teléfono ingresado incorrectamente",e)
    except:
        messagebox.showwarning("Teléfono ingresado incorrectamente","El teléfono ingresado es incorrecto, ingréselo
    else:
        return "ok"

def excepcion_tiempo(tiempo):
    try:
        patron=r'^[0-9]+ (días?|horas?|minutos?)$'
        if not re.match(patron,tiempo):
            raise ExcepcionTiempo()
    except ExcepcionTiempo as e:
        messagebox.showerror("Tiempo ingresado incorrectamente",e)
    except:
        messagebox.showerror("Tiempo ingresado incorrectamente","El tiempo ingresado es incorrecto, ingréselo nueva
    else:
        return "ok"

def excepcion_id_usuario(id):
    ok=excepcion_id(id)
    try:
        if ok=="ok":
            ok=False
            for pasajero in Pasajero.pasajeros:
                if str(pasajero.get_id())==str(id):
                    ok=True
```

```python
def excepcion_id_usuario(id):
            ok=False
            for pasajero in Pasajero.pasajeros:
                if str(pasajero.get_id())==str(id):
                    ok=True
            if ok==False:
                raise ExcepcionIdUsuario()

    except ExcepcionId as e:
        messagebox.showerror("Id de usuario incorrecto",e)
    except ExcepcionIdUsuario as e:
        messagebox.showerror("Id de usuario incorrecto",e)
    except:
        messagebox.showerror("Id de usuario incorrecto","El id de usuario debe ser un número")
    else:
        return "ok"

def excepcion_viaje(id):
    try:
        ok=False
        for empresa in Empresa.get_empresas():
            for viaje in empresa.get_viajes():
                if str(viaje.get_id())==str(id):
                    ok=True
        if ok==False:
            raise ExcepcionViaje()
    except ExcepcionViaje as e:
        messagebox.showerror("Id de viaje incorrecto",str(e))
    except:
        messagebox.showerror("Id de viaje incorrecto","El id de empresa debe ser un número")
    else:
```

```python
def excepcion_tiquete(numero_reserva):
        int(numero_reserva)
        ok=False
        for pasajero in Pasajero.get_pasajeros():
            for tiquete in pasajero.get_tiquetes():
                if str(tiquete.get_numero_reserva())==str(numero_reserva):
                    ok=True
        if ok==False:
            raise ExcepcionTiquete()
    except ExcepcionTiquete as e:
        messagebox.showerror("Número de reserva de tiquete incorrecto",e)
    except:
        messagebox.showerror("Número de reserva de tiquete incorrecto","El número de reserva del tiquete debe ser u
    else:
        return "ok"

def excepcion_hospedaje(nombre):
    try:
        ok=False
        for terminal in Terminal.get_terminales():
            for hospedaje in terminal.get_hospedajes():
                if hospedaje.get_nombre()==str(nombre):
                    ok=True
        if ok==False:
            raise ExcepcionHospedaje()
    except ExcepcionHospedaje as e:
        messagebox.showerror("Nombre de hospedaje incorrecto",e)
    except:
        messagebox.showerror("Nombre de hospedaje incorrecto","El nombre del hospedaje debe ser una cadena de carac
    else:
        return "ok"
```

```python
141 def excepcion_asiento(numero,bus:Bus):
142     try:
143         ok=False
144         for asiento in bus.get_asientos():
145             if str(asiento.get_numero())==str(numero):
146                 ok=True
147         if ok==False:
148             raise ExcepcionAsiento()
149     except ExcepcionAsiento as e:
150         messagebox.showerror("Número de asiento incorrecto o asiento ocupado",e)
151     except:
152         messagebox.showerror("Número de asiento incorrecto o asiento ocupado",
153                              "El numero del asiento no se ha puesto correctamente, ingréselo nuevamente")
154     else:
155         return "ok"
156
157 def valores_vacios(field:field_frame):
158     try:
159         ok=True
160         lista=[]
161         value:tk.Entry
162         for key,value in field.entries.items():
163             if (value.get()==""):
164                 ok=False
165                 lista.append(field.labels[key])
166         if ok==False:
167             raise ExcepcionValoresVacios()
168     except ExcepcionValoresVacios as e:
169         messagebox.showwarning("No se han llenado todos los campos",e+lista)
170     except:
171         messagebox.showwarning("No se han llenado todos los campos",
```

```python
def valores_vacios(field:field_frame):
    try:
        ok=True
        lista=[]
        value:tk.Entry
        for key,value in field.entries.items():
            if (value.get()==""):
                ok=False
                lista.append(field.labels[key])
        if ok==False:
            raise ExcepcionValoresVacios()
    except ExcepcionValoresVacios as e:
        messagebox.showwarning("No se han llenado todos los campos",e+lista)
    except:
        messagebox.showwarning("No se han llenado todos los campos",
                               "Debe diligenciar todos los campos para poder continuar")
    else:
        return "ok"

```

FieldFrame:

A continuación se muestra el código que implementamos para la creación de la clase FieldFrame que hereda de Frame de tkinter:

```
import tkinter as tk

class field_frame(tk.Frame):
    def __init__(
        self,
        parent,
        tituloCriterios,
        criterios,
        tituloValores,
        valores = None,
        habilitado = None
    ):
        super().__init__(parent, bg='lightblue')
        self.pack(fill="both")

        self.tituloCriterios = tituloCriterios
        self.criterios = criterios
        self.tituloValores = tituloValores
        self.valores = valores
        self.habilitado = habilitado

        self.labels = {}
        self.entries = {}

        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)

        if self.tituloCriterios!=None:
            frame_tituloCriterios=tk.Frame(
                self,
                highlightbackground="blue",
                highlightthickness=2
            )
            frame_tituloCriterios.grid(row=0,column=0,sticky='news',padx=10,pady=10)
            label=tk.Label(frame_tituloCriterios,text=self.tituloCriterios)
            label.pack()
        if self.tituloValores!=None:
            frame_tituloValores=tk.Frame(
                self,
                highlightbackground="blue",
                highlightthickness=2
            )
            frame_tituloValores.grid(row=0,column=1,sticky='news',padx=10,pady=10)
            label=tk.Label(frame_tituloValores,text=self.tituloValores)
            label.pack()

        if self.criterios != None:
            for criterio in self.criterios:
                label = tk.Label(self, text=str(criterio))
                label.grid(
                    row=self.criterios.index(criterio)+1,
                    column=0,
                    sticky='news',
                    padx=10,
                    pady=10
```

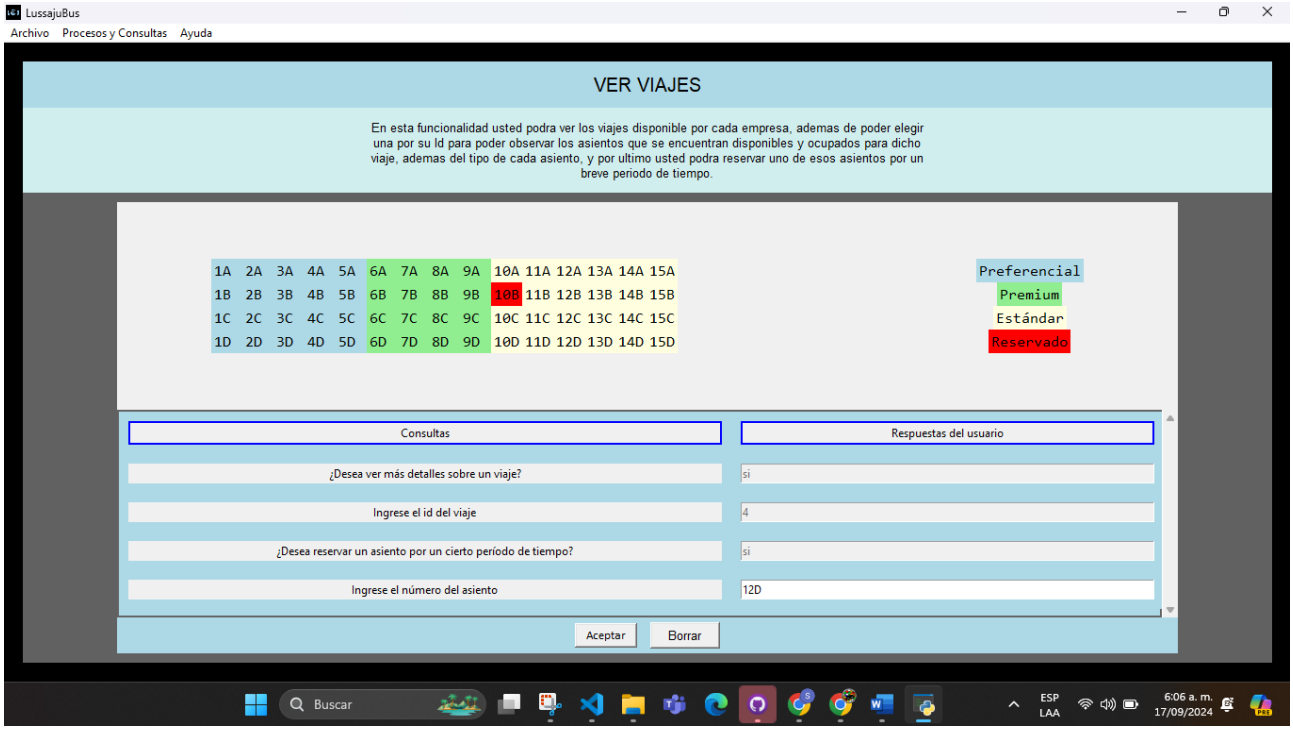
```
            )
            self.labels[criterio] = label

            entry = tk.Entry(self)
            entry.grid(
                row=self.criterios.index(criterio)+1,
                column=1,
                sticky='news',
                padx=10,
                pady=10
            )
            self.entries[criterio] = entry

    def getValue(self, criterio):
        return self.entries[criterio].get()

    def agregar_campo(self, criterio, desabilitar_anterior):
        label = tk.Label(self, text=criterio)
        label.grid(
            row=len(self.labels)+1,
            column=0,
            sticky='news',
            padx=10,
            pady=10
        )
        self.labels[criterio] = label

        if desabilitar_anterior:
            self.entries[list(self.entries.keys())[-1]].config(state="disabled")
        entry = tk.Entry(self)
        entry.grid(
            row=len(self.entries)+1,
            column=1,
            sticky='news',
            padx=10,
            pady=10
        )
        self.entries[criterio] = entry

    def clear_entries(self):
        for entry in self.entries.values():
            entry.delete(0, "end")
```

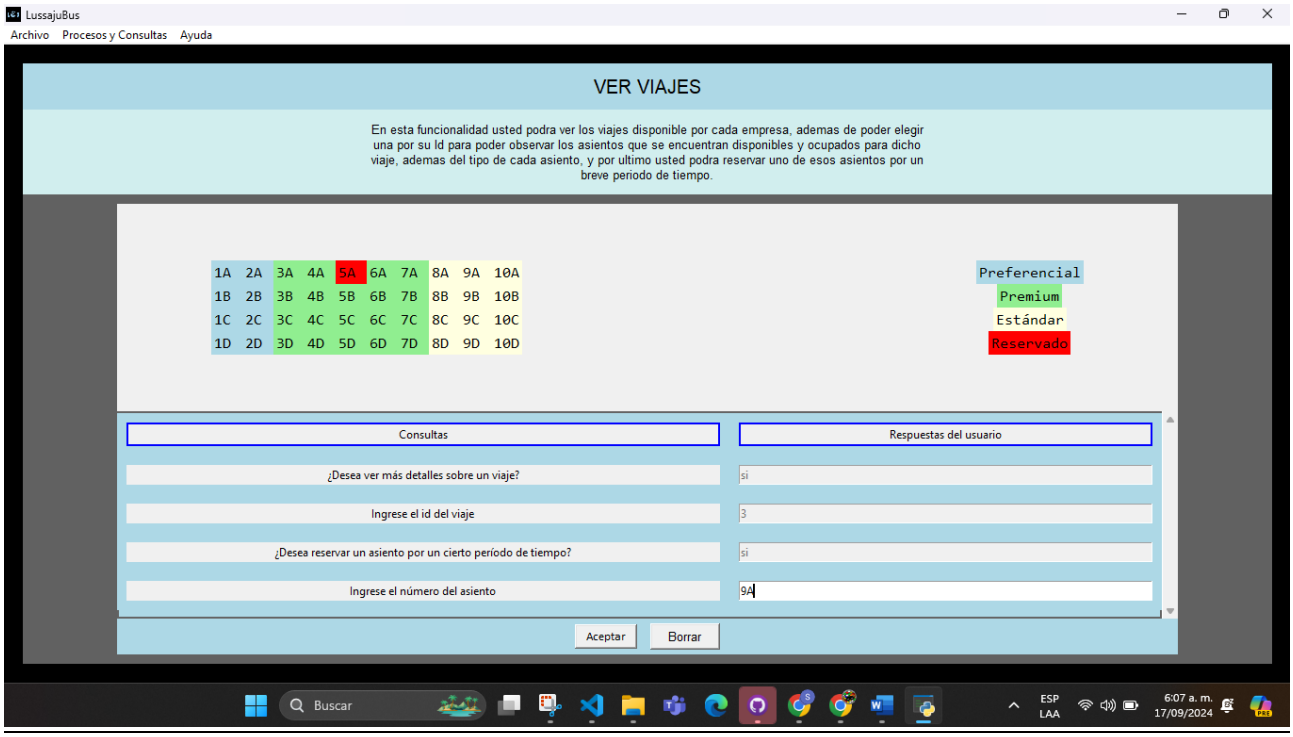
Para la lógica de la clase nosotros hicimos lo siguiente:

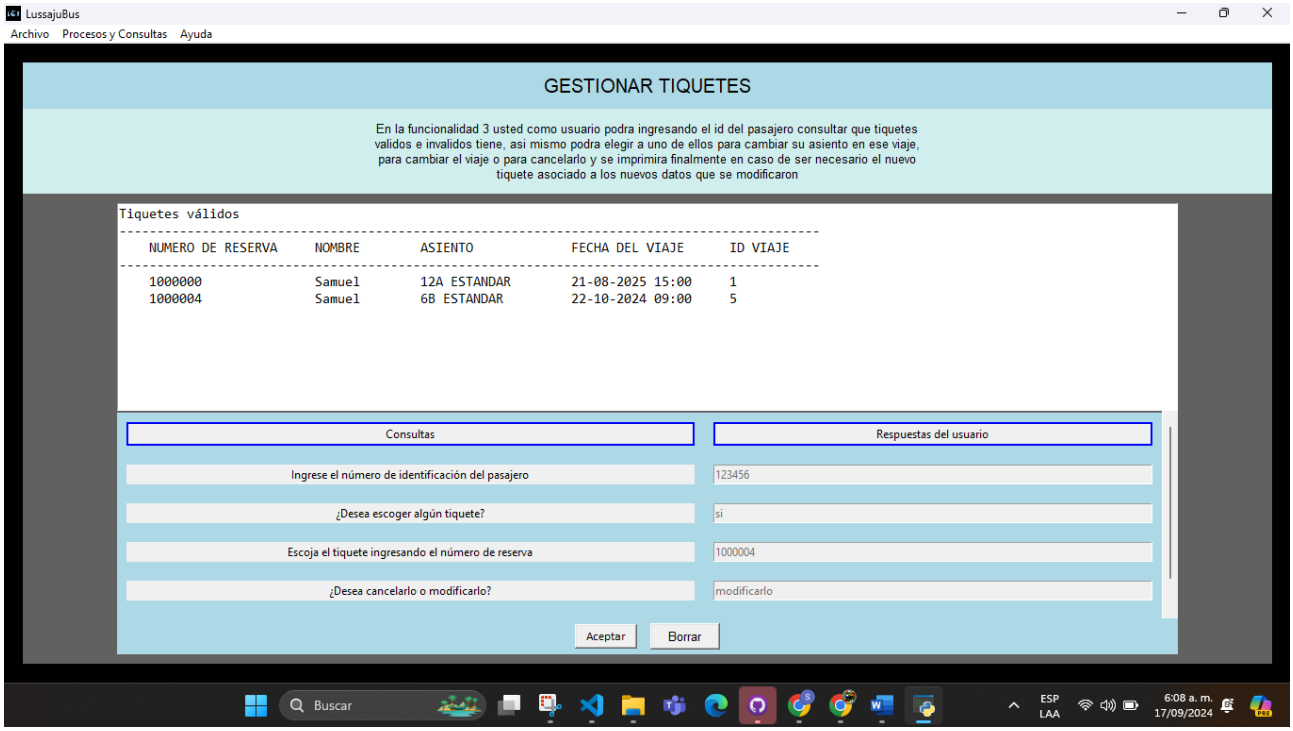
Primero importamos la Liberia tkinter de Python para realizar los procesos, luego creamos una clase llamada Tkinter que hereda de la clase Frame de tk, además dentro del inicializador se admiten unas variables y se inicializan como variables de instancia, estás son todas las requeridas más una extra que es el frame original, es decir, el frame sobre el cual se va a crear la clase fieldframe, ambos títulos determianan el título de la tabla que genera el fieldframe, el atributo criterios nos muestra las consultas o preguntas que se le harán al usuario y en el atributo valores se crearán los entries para que el usuario pueda ingresar información si su configuración lo permite o sino que simplemente la pueda observar. Además se configuro los parámetros para que el espaciado del grid sea centrado en el frame a la hora de crear el field frame y no se tengan problemas relacionados con el espaciado de la tabla, también se crearon dos variable s de instancia, las cuales son values y entries que ambas son diccionarios, en las cuales los valores criterio serán las clasves y los valores respectivos serán el objeto de tipo label para los enunciados o criterios (labels) o serán los entries para el caso de los Entrys que permitirán el ingreso de datos por parte del usuario al sistema. Además falta mencionar que se crea una instancia de la clase padre y se le asigna un color al fieldframe de azul claro.
Luego en la parte lógica del field frame, ocurre que se rectifica si se ingresan valores para los títulos de

ambas columnas y en caso de ser así, se crea un label que tiene borde azul para diferencia el título del resto y se le asigna el título como tal para ambos casos, si el título no está no se pone, Y ya posteriormente por cada criterios se procede a poner dentro de la cuadrícula destinada para el fieldframe cada uno de los campos que aprecerán inicialmente en el programa, Luego la clase posee 5 métodos, uno de ellos borrará el valor de todos los entries que estén activos actualmente y esto se ejecuta mediante el botón borrar, otro método importante es el getValue que se requería como tal para el proyecto y su función es retornar el valor exacto de un entry según su labe o criterio correspondiente, otro método es agregar campos que permite agregar un nuevo campo vacía para llenar y se pondrá dentro del fieldframe antiguo aumentando su capacidad, cabe aclarar que cuando el field frame se hace muy grande y sobrepasa el tamaño de su frame, se ha creado una scroll bar que nos permitirá desplazarnos alrededor del fieldframe rápidamente. Y por último los últimos dos métodos que conforman a nuestro fieldframe son ocultar o mostrar campos que su función está resumida en su nombre,, respectivamente, se encargan de ocultar o mostrar una lista de campos con la condición de que ya estén creadas dentro del fieldframe, por lo que lo único que ha´ra será ocultarlas y mostrarlas en el momento adecuado; es una alternativa al método agregar valores.

La clase Fieldframe, ha sido de gran ayuda durante el proceso de creación de las funcionalidades, ya que facilita por medio de la tabulación la forma en la que el usuario ingresa los datos al sistema, además es una forma más organizada de llevar el programa, sin retrasos, de manera muy secuencial y fácil de depurar, por lo que el haber organizado el programa mediante el ingreso de datos pro parte del usuario hacia el sistema de esta manera ha mejorado la eficacia, el control y ha mejorado muchos otros aspectos del código relacionados con la legibilidad del mismo




LussajuBus
Archivo Procesos y Consultas Ayuda
VER VIAJES
En esta funcionalidad usted podra ver los viajes disponible por cada empresa, ademas de poder elegir una por su Id para poder observar los asientos que se encuentran disponibles y ocupados para dicho viaje, ademas del tipo de cada asiento, y por ultimo usted podra reservar uno de esos asientos por un breve periodo de tiempo.
1A 2A 3A 4A 5A 6A 7A 8A 9A 10A
1B 2B 3B 4B 5B 6B 7B 8B 9B 10B
1C 2C 3C 4C 5C 6C 7C 8C 9C 10C
1D 2D 3D 4D 5D 6D 7D 8D 9D 10D
Preferencial
Premium
Estándar
Reservado
Consultas
Respuestas del usuario
¿Desea ver más detalles sobre un viaje?
si
Ingrese el id del viaje
3
¿Desea reservar un asiento por un cierto período de tiempo?
si
Ingrese el número del asiento
9A
Aceptar
Borrar
Buscar
ESP LAA
6:07 a. m. 17/09/2024


LussajuBus
Archivo Procesos y Consultas Ayuda
GESTIONAR TIQUETES
En la funcionalidad 3 usted como usuario podra ingresando el id del pasajero consultar que tiquetes validos e invalidos tiene, asi mismo podra elegir a uno de ellos para cambiar su asiento en ese viaje, para cambiar el viaje o para cancelarlo y se imprimira finalmente en caso de ser necesario el nuevo tiquete asociado a los nuevos datos que se modificaron
Tiquetes válidos
NUMERO DE RESERVA NOMBRE ASIENTO FECHA DEL VIAJE ID VIAJE
1000000 Samuel 12A ESTANDAR 21-08-2025 15:00 1
1000004 Samuel 6B ESTANDAR 22-10-2024 09:00 5
Consultas
Respuestas del usuario
Ingrese el número de identificación del pasajero
123456
¿Desea escoger algún tiquete?
si
Escoja el tiquete ingresando el número de reserva
1000004
¿Desea cancelarlo o modificarlo?
modificarlo
Aceptar
Borrar
Buscar
ESP LAA
6:08 a. m. 17/09/2024

```
class ver_viajes():
    def segunda_pregunta(frame_funcionalidad):



        except:
            messagebox.showerror("Error inesperado",
                                 "Se ha producido un error inesperado pero puede continuar navegando en el prograr
            return 0
        frame_funcionalidad.text.pack_forget()

        auxiliar.asientos(frame_funcionalidad.frame_superior, viaje)

        frame_funcionalidad.field_frame.agregar_campo(
            "¿Desea reservar un asiento por un cierto período de tiempo?",
            True
        )
        return -1

    @staticmethod
    def tercera_pregunta(frame_funcionalidad):
        respuesta = frame_funcionalidad.field_frame.getValue(
            "¿Desea reservar un asiento por un cierto período de tiempo?"
        ).lower()

        if respuesta == "si":
            frame_funcionalidad.field_frame.agregar_campo(
                "Ingrese el número del asiento",
                True
            )
            return True
        elif respuesta == "no":
            return False
```

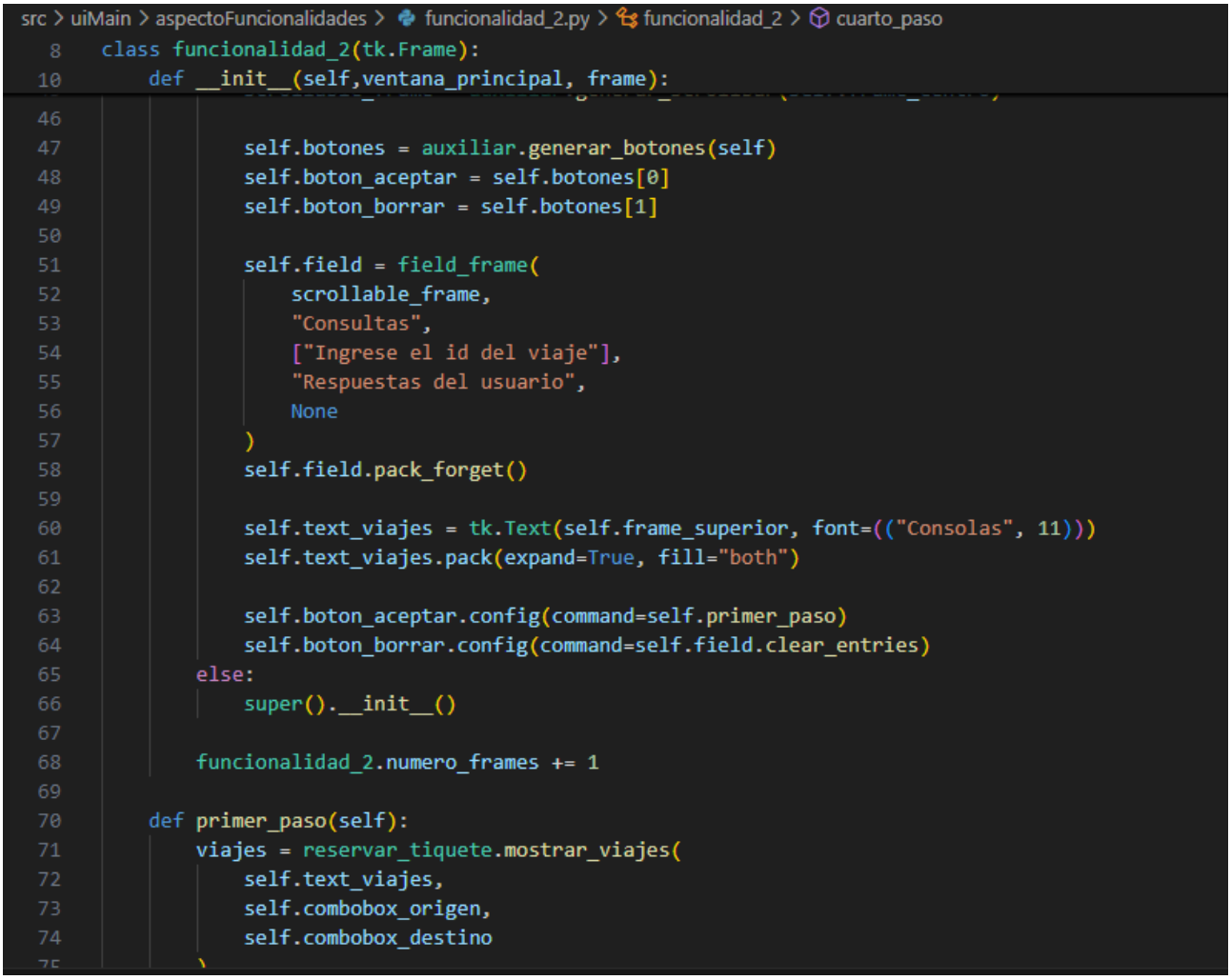
src > uiMain > funcionalidades > funcionalidad1.py > ver_viajes > mostrar_viajes

```
11  class ver_viajes():
78      def tercera_pregunta(frame_funcionalidad):
93          return False
94
95      @staticmethod
96      def cuarta_pregunta(frame_funcionalidad):
97          respuesta = frame_funcionalidad.field_frame.getValue(
98              "Ingrese el número del asiento"
99          )
100
101         indice=frame_funcionalidad.field_frame.getValue(
102                 "Ingrese el id del viaje")
103         viaje = Empresa.buscar_viaje_por_id(indice)
104         ok=ae.excepcion_viaje(indice)
105         if ok=="ok":
106             ae.excepcion_asiento(respuesta,viaje.get_bus())
107
108
109
110         try:
111             asiento = viaje.buscar_asiento(respuesta)
112             if asiento != None and not asiento.is_reservado():
113                 frame_funcionalidad.field_frame.agregar_campo(
114                     "¿Por cuánto tiempo desea reservarlo?",
115                     True
116                 )
117         except:
118             messagebox.showerror("Error inesperado",
119                                  "Se ha producido un error inesperado pero puede continuar navegando en el programa
120
121     @staticmethod
```

src > uiMain > aspectoFuncionalidades > funcionalidad_1.py > funcionalidad_1 > __init__

```
class funcionalidad_1(tk.Frame):
    def __init__(self,ventana_principal, frame):

            self.botones = auxiliar.generar_botones(self)
            self.boton_aceptar = self.botones[0]
            self.boton_borrar = self.botones[1]

            self.field_frame = field_frame(
                scrollable_frame,
                "Consultas",
                ["¿Desea ver más detalles sobre un viaje?"],
                "Respuestas del usuario",
                None
            )

            self.text = tk.Text(self.frame_superior, font=(("Consolas", 11)))
            self.text.pack(expand=True, fill="both")

            self.boton_aceptar.config(command=self.primer_paso)
            self.boton_borrar.config(command=self.field_frame.clear_entries)

            ver_viajes.mostrar_viajes(self)
        else:
            super().__init__()

        funcionalidad_1.numero_frames += 1

    def primer_paso(self):
        caso = ver_viajes.primera_pregunta(self)

        if caso == -1:
            self.boton_aceptar.config(command=self.segundo_paso)
```

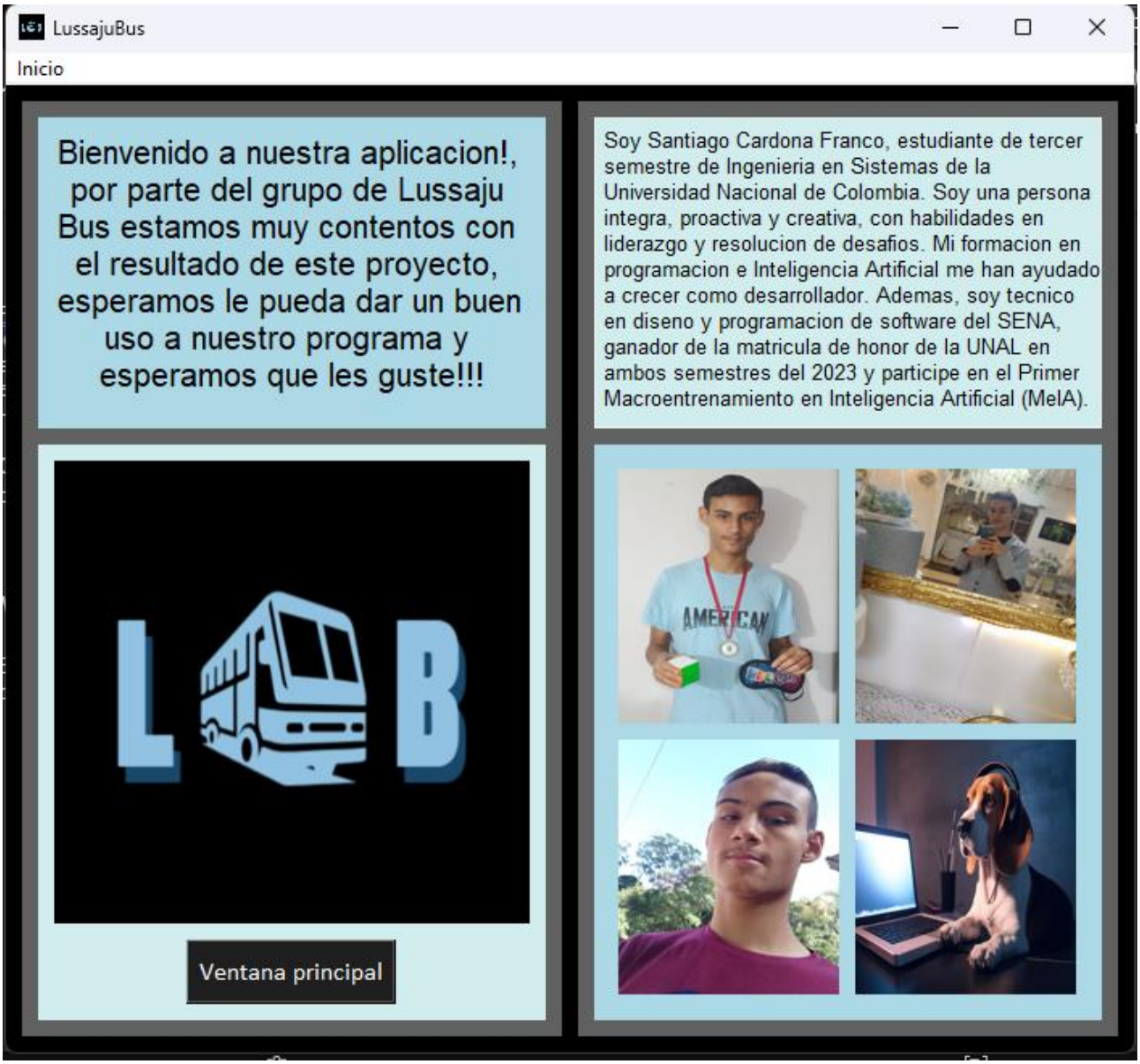
Ln 13, Col 36 Spaces: 4


src > uiMain > aspectoFuncionalidades > funcionalidad_2.py > funcionalidad_2 > cuarto_paso
8 class funcionalidad_2(tk.Frame):
10 def __init__(self,ventana_principal, frame):
46
47 self.botones = auxiliar.generar_botones(self)
48 self.boton_aceptar = self.botones[0]
49 self.boton_borrar = self.botones[1]
50
51 self.field = field_frame(
52 scrollable_frame,
53 "Consultas",
54 ["Ingrese el id del viaje"],
55 "Respuestas del usuario",
56 None
57)
58 self.field.pack_forget()
59
60 self.text_viajes = tk.Text(self.frame_superior, font=(("Consolas", 11)))
61 self.text_viajes.pack(expand=True, fill="both")
62
63 self.boton_aceptar.config(command=self.primer_paso)
64 self.boton_borrar.config(command=self.field.clear_entries)
65 else:
66 super().__init__()
67
68 funcionalidad_2.numero_frames += 1
69
70 def primer_paso(self):
71 viajes = reservar_tiquete.mostrar_viajes(
72 self.text_viajes,
73 self.combobox_origen,
74 self.combobox_destino

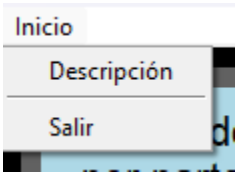
Manual de usuario

El programa de LussajuBus se trata de manejar terminales, empresas, buses, conductores, hospedajes y pasajeros para agendar viajes y organizar el sistema de transporte público en Colombia.

Inicialmente al abrir el programa le aparecerá la siguiente pantalla:

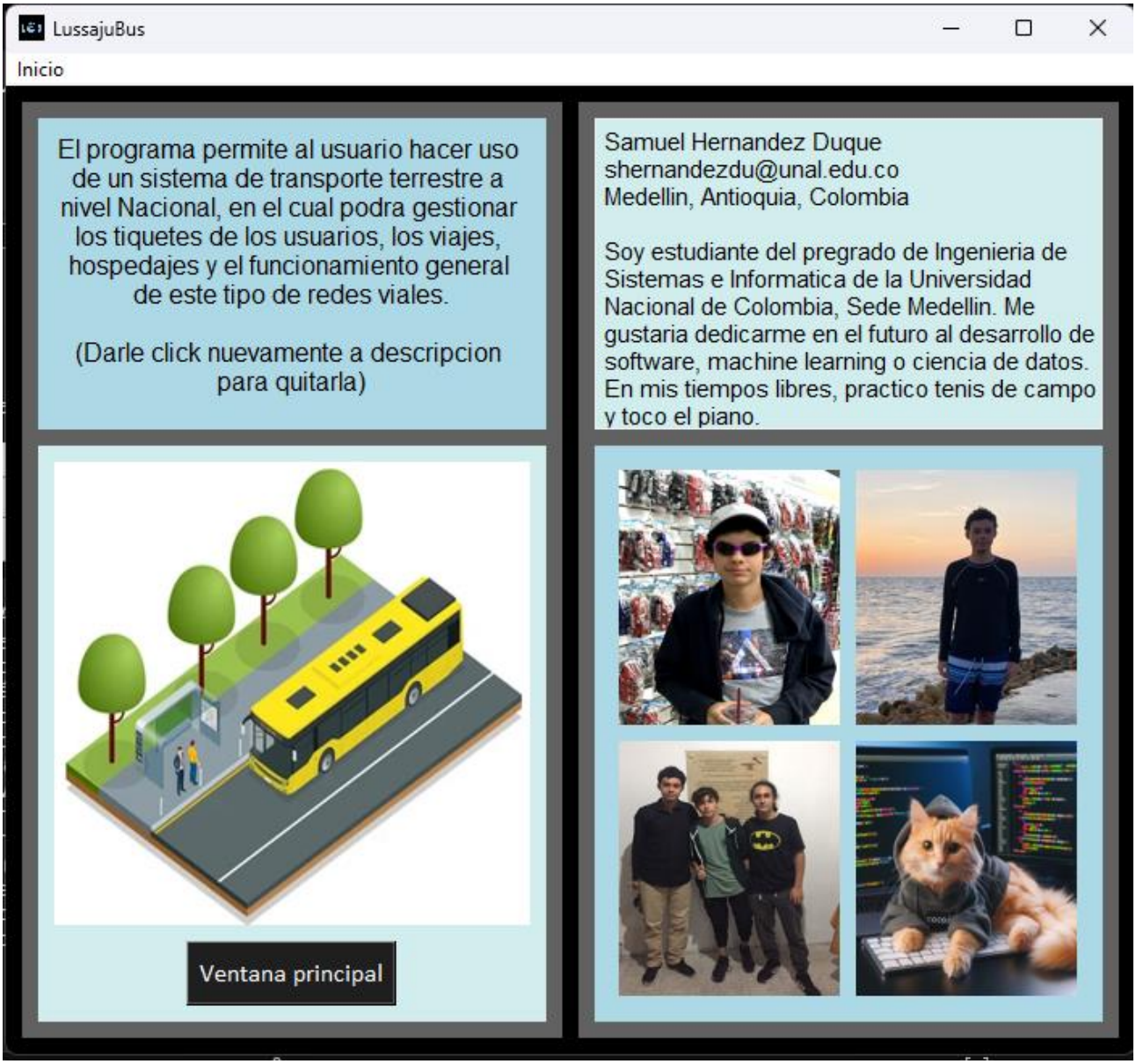


La cual está dividia en varias parte, inicialmente, se le dará la bienvenida al programa mediante el mensaje de la parte superior izquierda, en la parte superior aprece un botón que es un menú, al abrirlo se mostrarán las siguientes opciones:

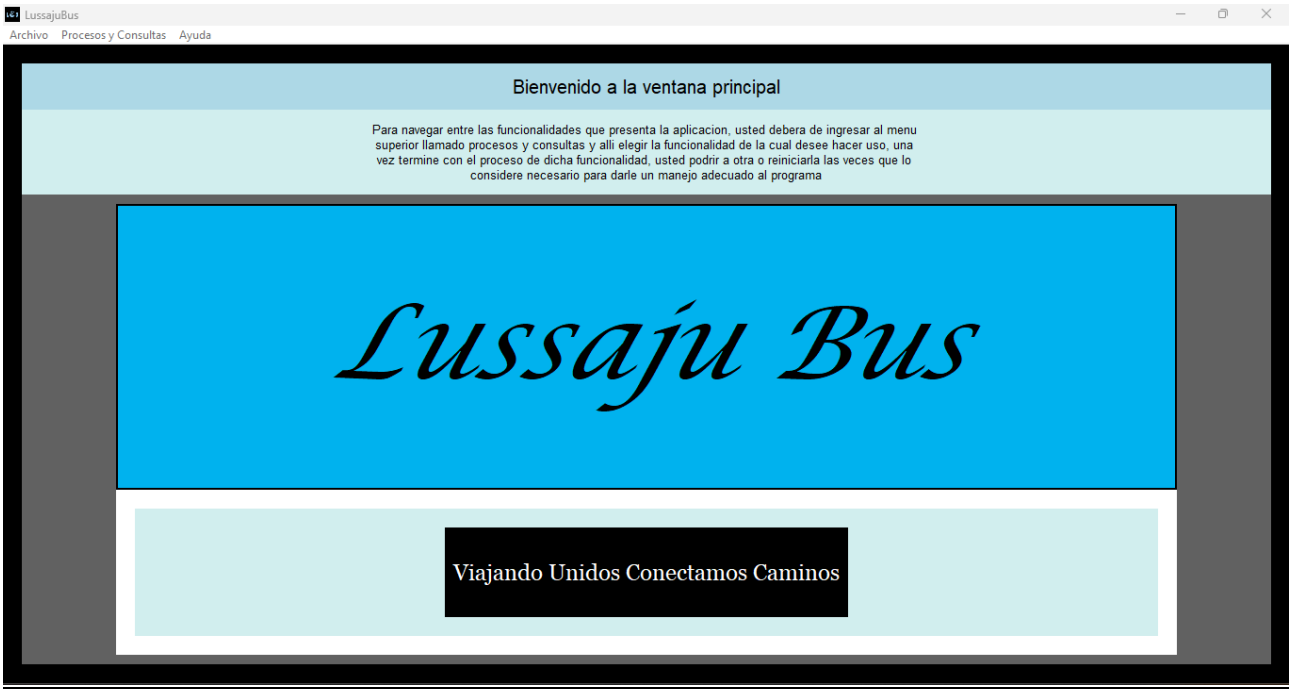


Al darle click a descripción pasará que ahora en vez de tener el texto de bienvenida mencionado anteriormente, se mostrará una breve descripción de que puede realizar usted como usuario dentro

del sistema de LussajuBus, por otra parte, si se le da click al botón de salir la aplicación se saldrá con éxito, volviendo a la ventana anterior, en la parte superior derecha usted podrá encontrar una breve descripción de la hoja de vida de cada programador que ayudó a la creación de este programa y en la parte inferior derecha se observarán 4 imágenes relacionadas con cada uno de los programadores, las imágenes son presentadas a modo de mosaico y si usted como usuario le da click a la hoja de vida del desarrollador, podrá observar que la hoja de vida cambia y las imágenes también para mostrar a otro desarrollador del proyecto, esto ocurre, ya que la ventana de arriba a la derecha es en realidad un botón; en la parte inferior izquierda encontramos otra ventana que nos mostrará inicialmente el logo de LussajuBus y que al pasar el mouse por encima de ella cambiará la imagen y en total mostrará 5 imágenes relacionadas con el proyecto, y finalmente por esta parte aparecerá un botón de color negro que dice Ventana principal y que al darle click al botón le dirigirá a otra ventana que es la principal:



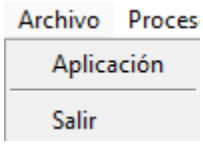
Dentro de la ventana principal se podrá observar inicialmente lo siguiente:



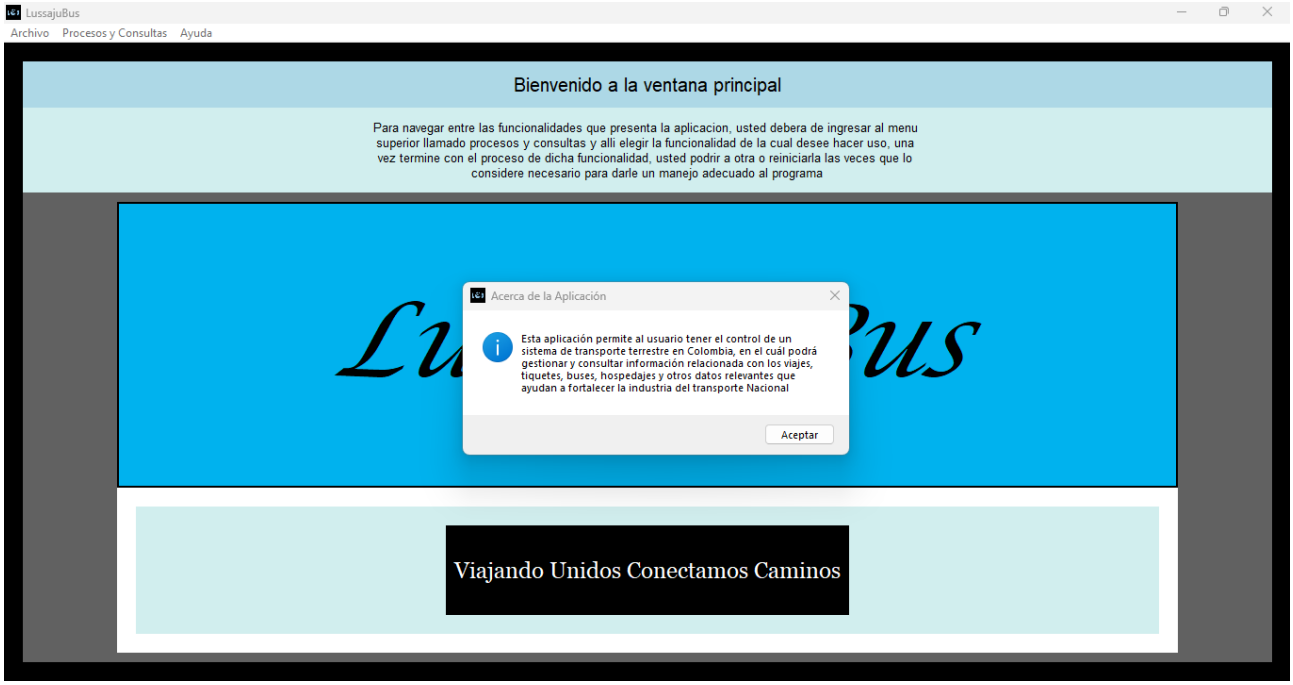
Lo cual nos muestra en el centro el nombre de la compañía, junto con el slogan, además de que en la parte superior se nos da la bienvenida a la ventana principal que contiene todas las funcionalidades relacionadas con el sistema de transporte que se ofrece, además de proporcionar una breve descripción de como operar el sistema.

En la parte superior izquierda podremos encontrar un menú que contiene 3 menús desplegables:

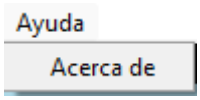
- Archivo:



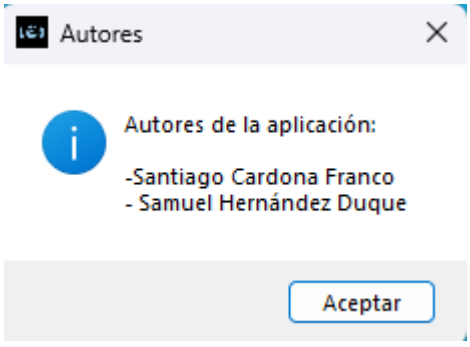
Este menú tendrá dos opciones, las cuales son Aplicación que al darle click nos mostrará una breve descripción sobre la aplicación y el botón salir que al darle click nos redirigirá a la ventana de inicio que fue la anterior a esta.



- Ayuda:



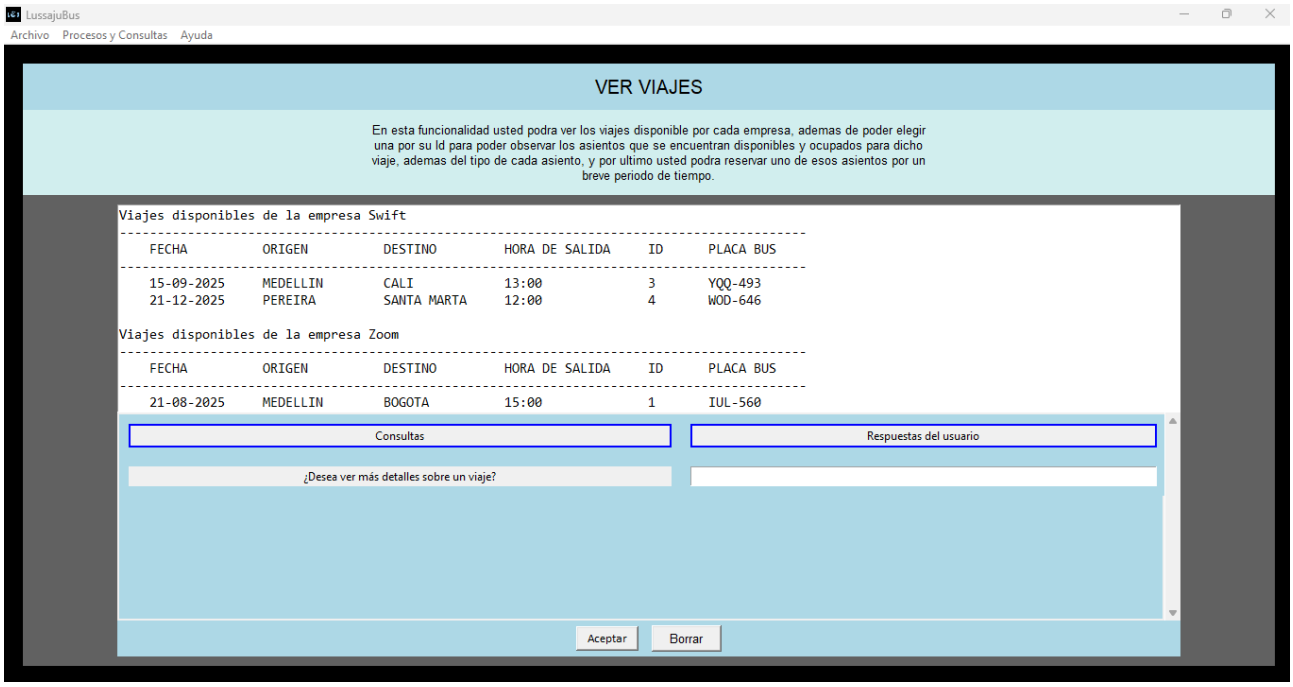
Nos mostrará una única opción, la cual es ‘Acerca de ‘ y que al darle click nos sacará otra ventana emergente como la anterior en la cual se muestra el nombre de ambos autores de la aplicación en nuestro caso:



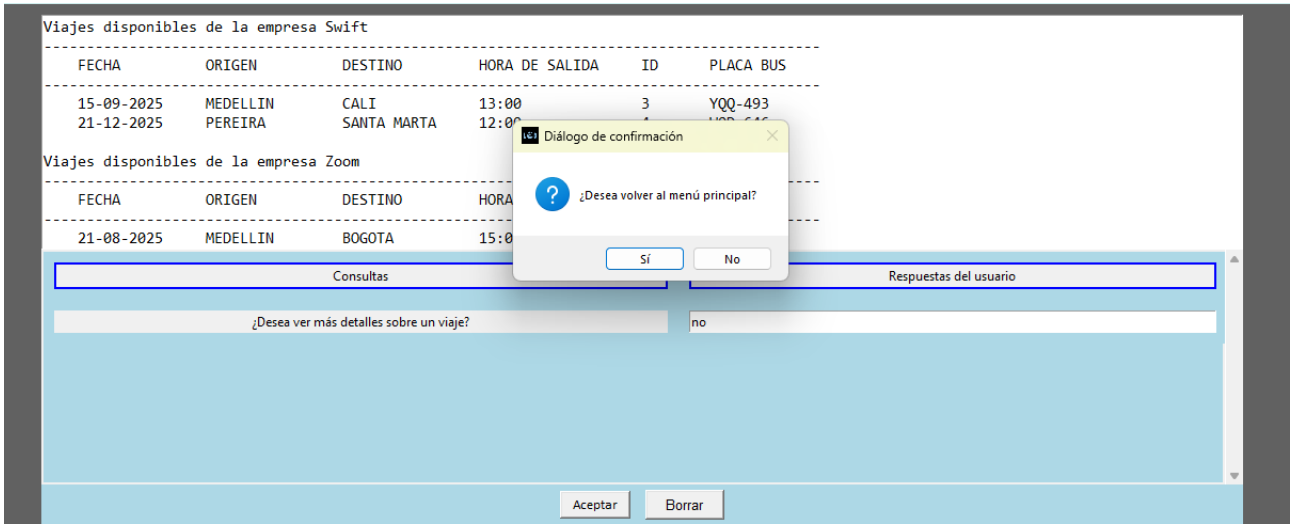
- Procesos y Consultas:

Por último en el menú de procesos y consultas se nos desplegarán 3 opciones, cada una de ellas representa una de las posibles opciones que tiene el gestor de la aplicación para realizar consultas y modificaciones en el sistema, estas son las 3 primeras funcionalidades como tal que aparecían en el programa anterior, y estas son: Ver viajes, Reservar tiquetes y Gestionar tiquetes, al darle click a ccada una de ella nos aprecerán su funcionalidad y una descripción general sobre lo que se puede hacer con dicha funcionalidad, a continuación una guía rápida:

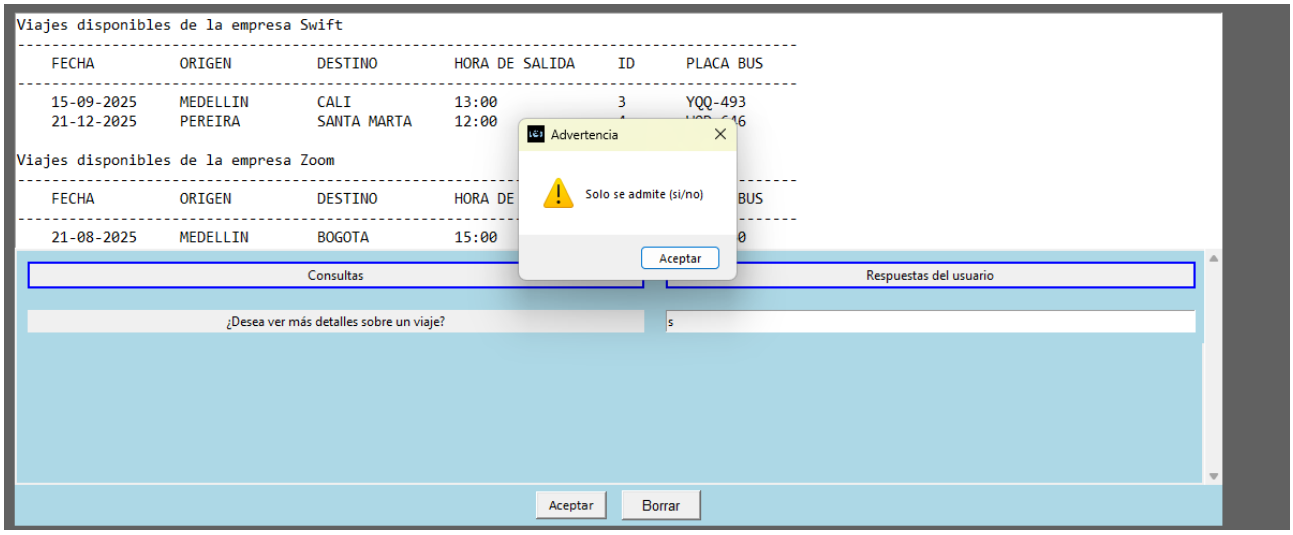
- Ver viajes:



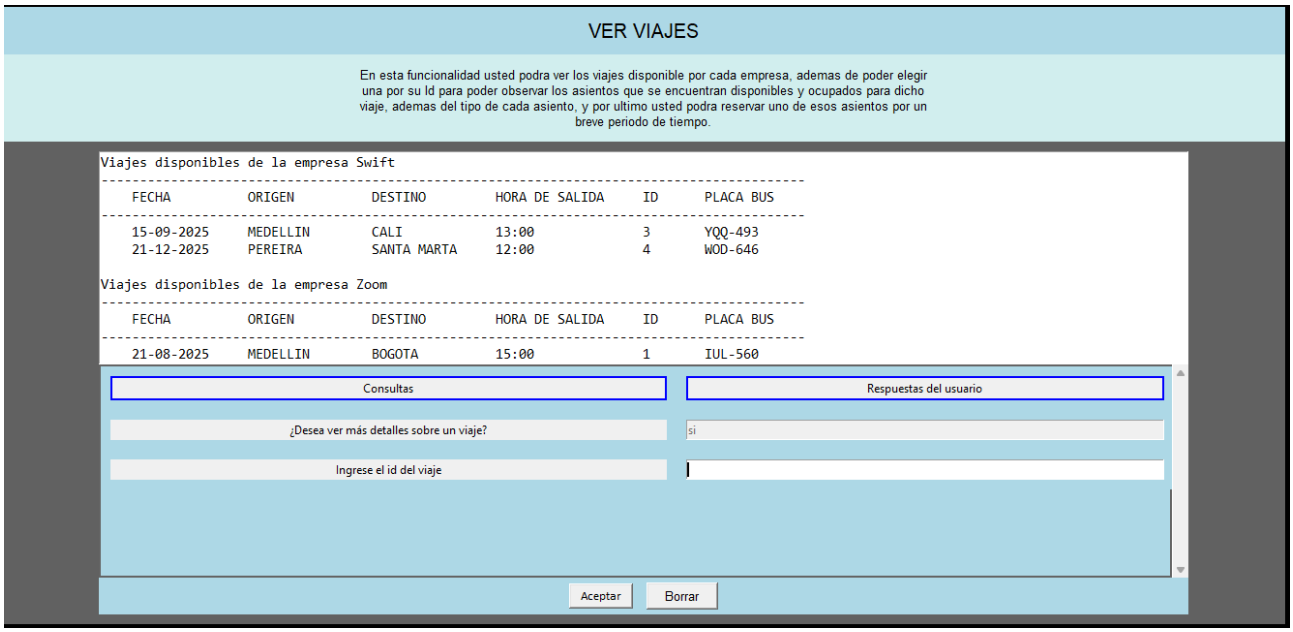
Inicialmente se nos mostrarán los viajes disponibles por cada empresa y podremos verlos todos, para posteriormente si deseamos ver información sobre un viaje escribiremos sí en el cuadro a la derecha, en caso de escribir no:



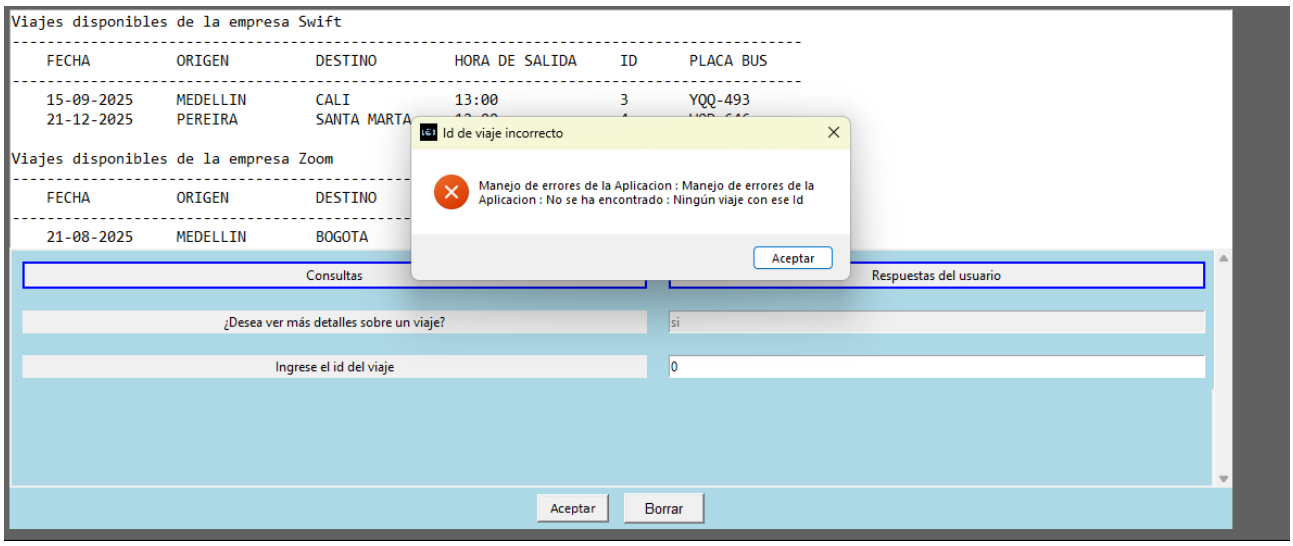
O en caso de nos escribir nada o escribir otra cosa:



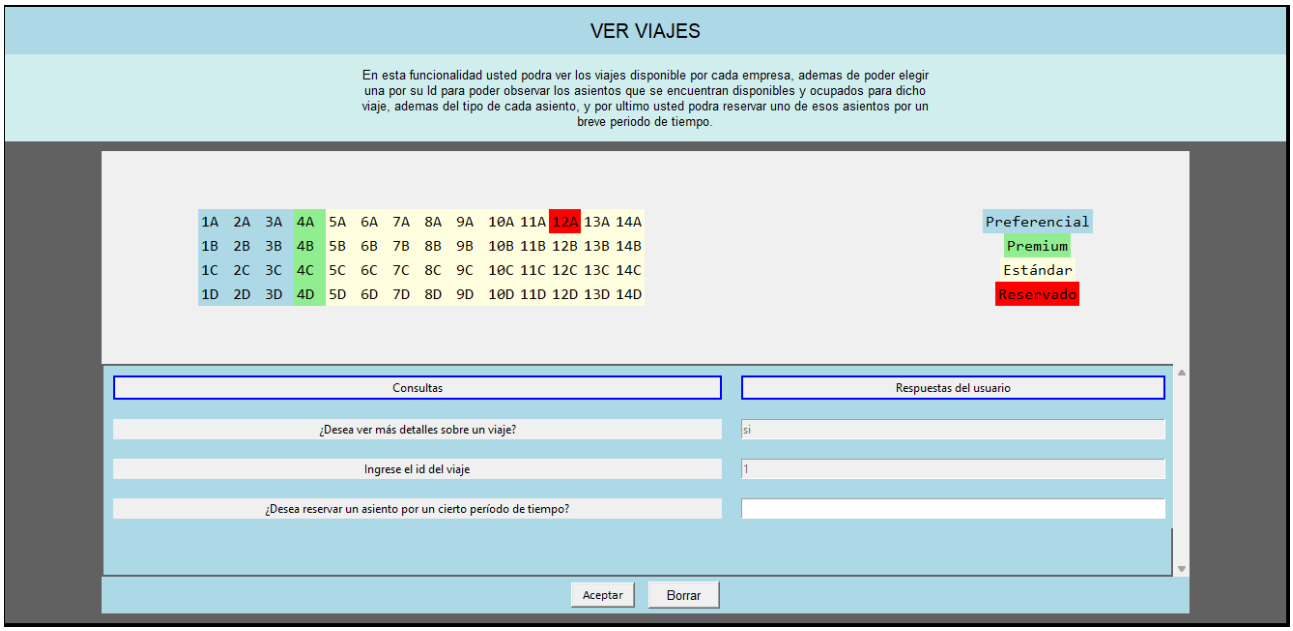
Luego de escribir que sí nos pedirá ingresar el id del viaje:



SI ingresamos uno que no existe nos sacará error:

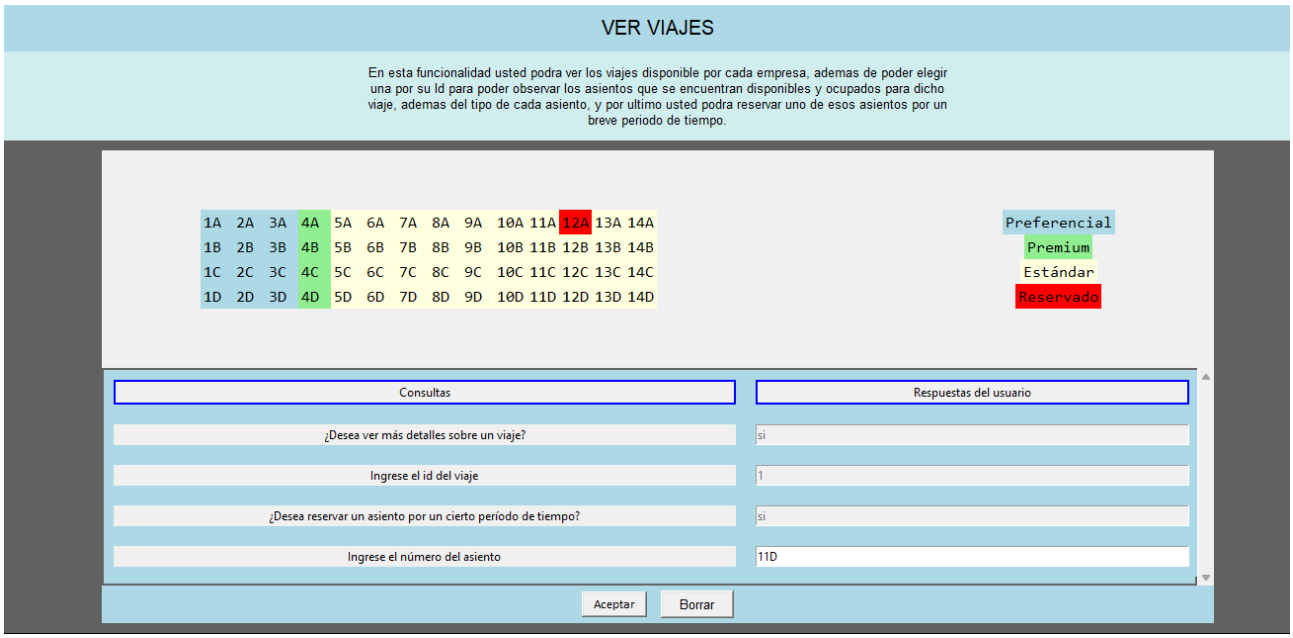


Luego se nos mostrará una gráfica que simula la ubicación de los asientos en el bus, junto con su tipo y si está reservado o no:



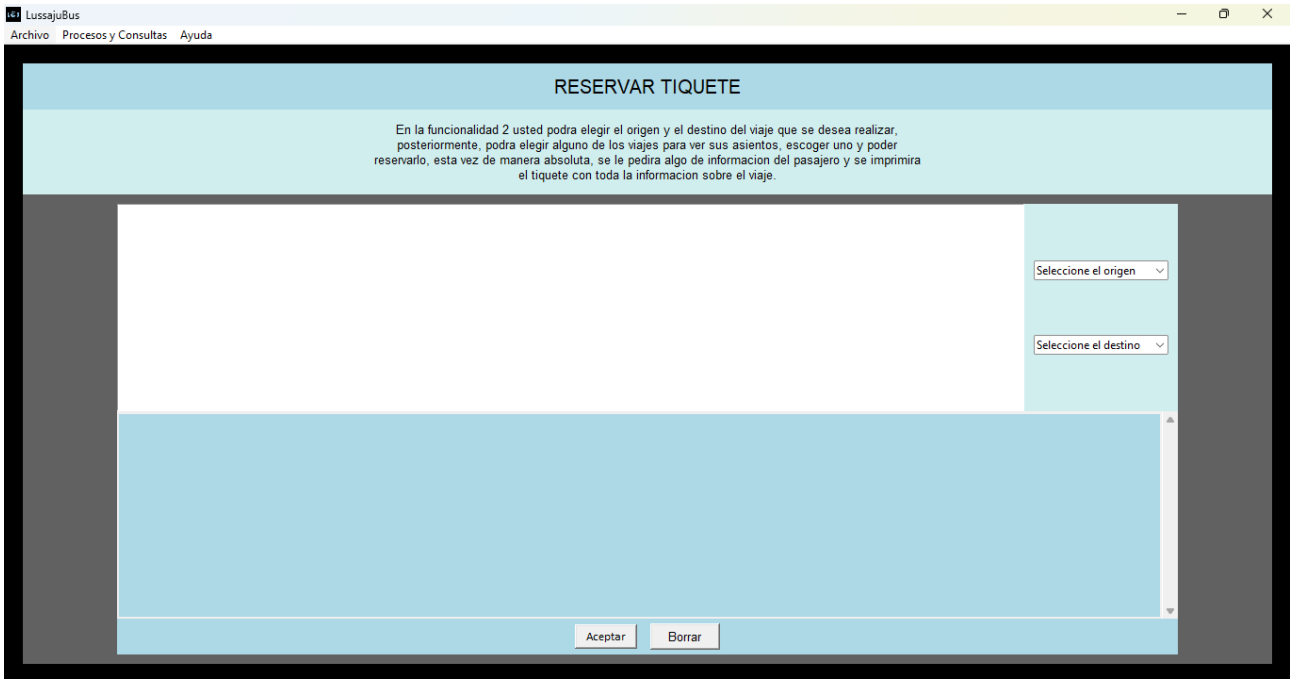
Posteriormente se nos preguntará si deseamos reservar un asiento por un periodo de tiempo, si decimos que no nos preguntará si queremos redirigirnos al menú principal y si respondemos otra cosa nos sacará una advertencia como la anterior; al responder que si :

Se nos pedirá el asiento: si el asiento no existe o está reservado nos sacará error, de lo contrario, se nos preguntará por cuánto tiempo deseamos reservar el asiento

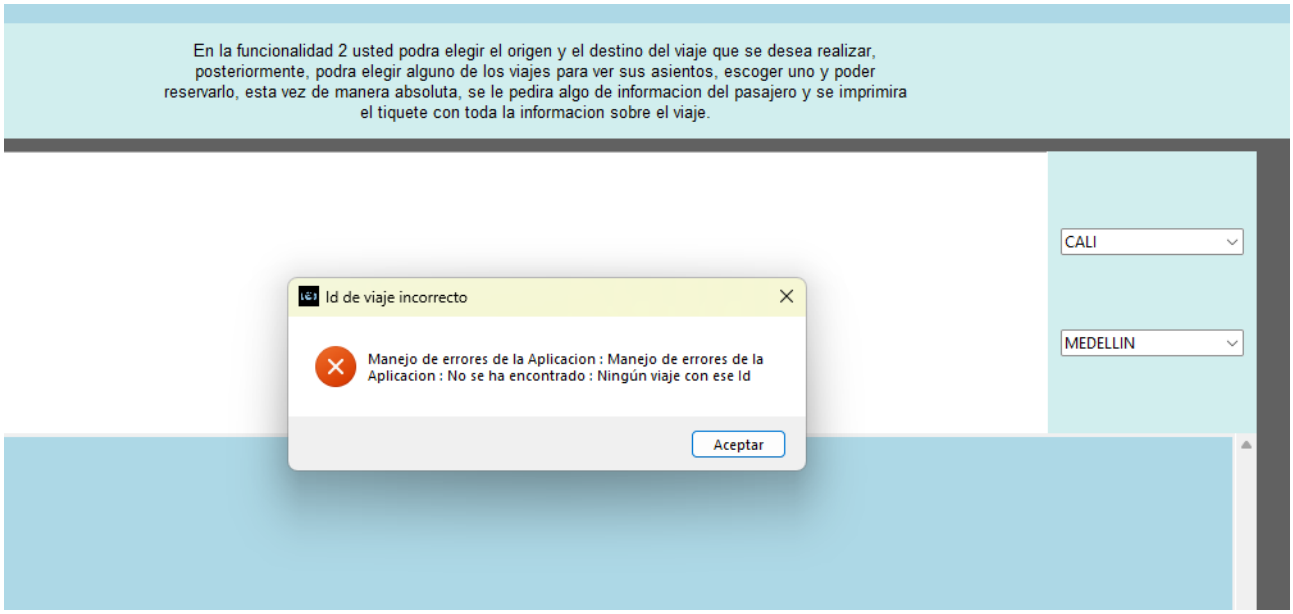


Y finalmente se imprimirá asiento reservado exitosamente (en nuestro caso, no siempre funciona esta última parte de la primera funcionalidad)

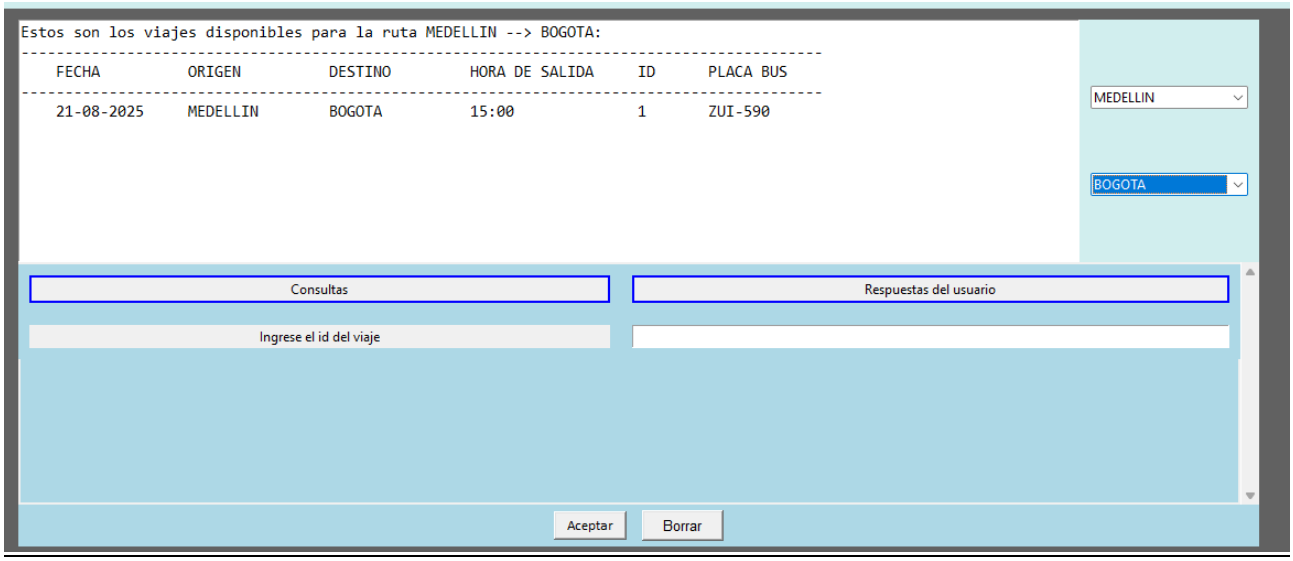
-Reservar Tiquetes:
Inicialmente se mostrará la siguiente ventana:



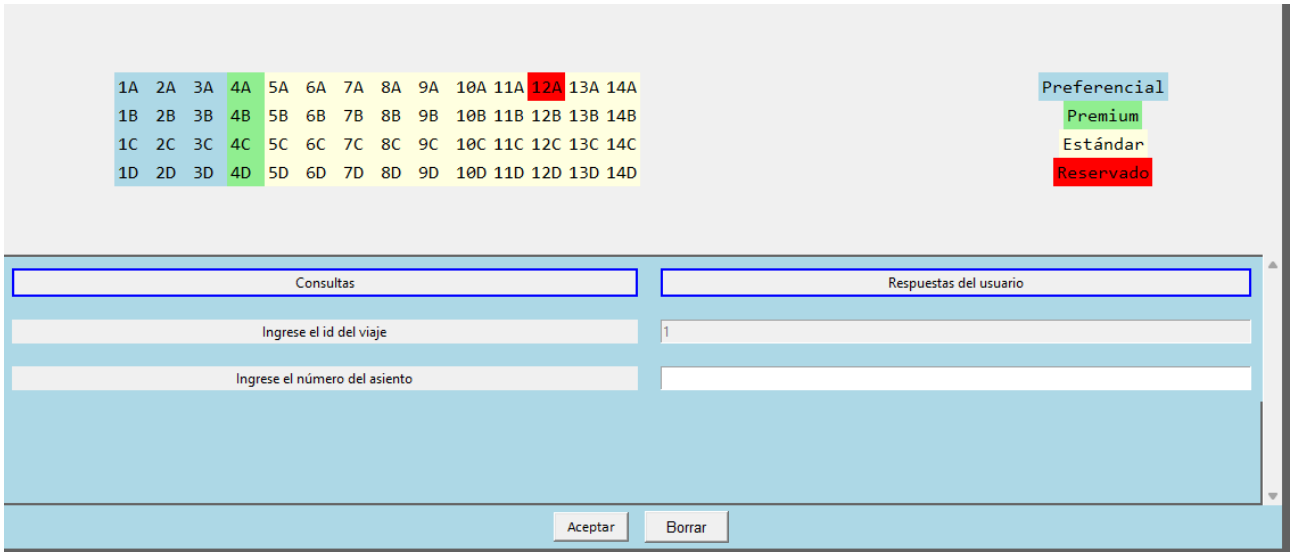
En los botones de la derecha se le deberá de dar click y elegir la ciudad de origen y destino para poder reservar un viaje y posteriormente darle click al botón aceptar, en caso de no exista ningún viaje, saldrá un mensaje de error y se deberá de elegir otro origen y/u otro destino:



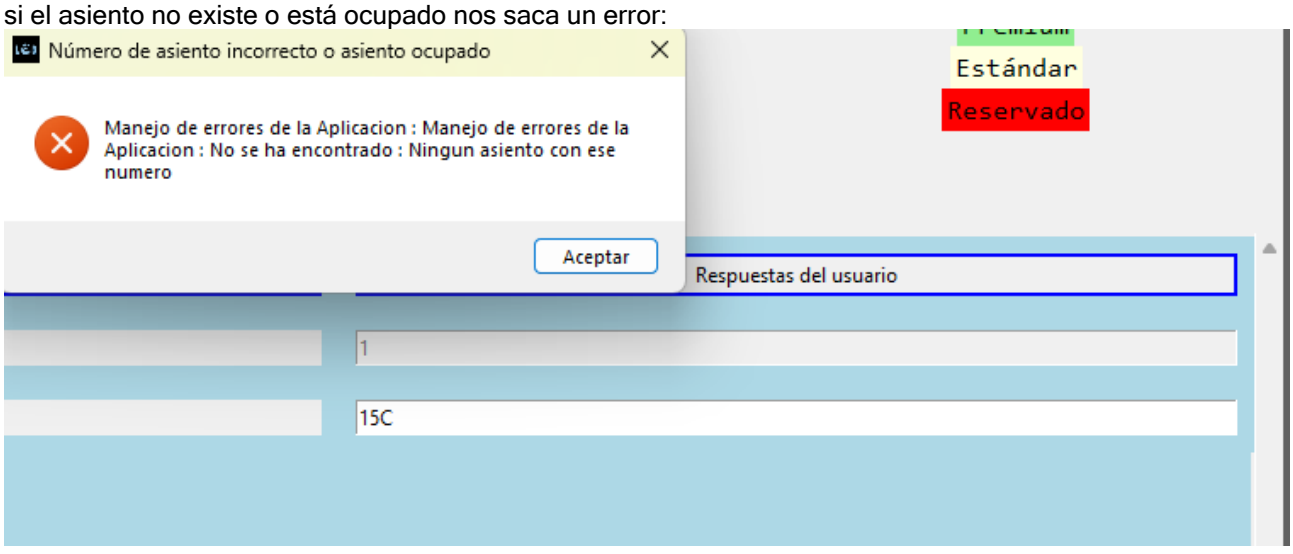
Si encuentralos viajes se mostrarán en la parte superior del centro:



Se pedirá que se ingrese el id del viaje, si se ingresa mal, sacará el mismo error anterior.

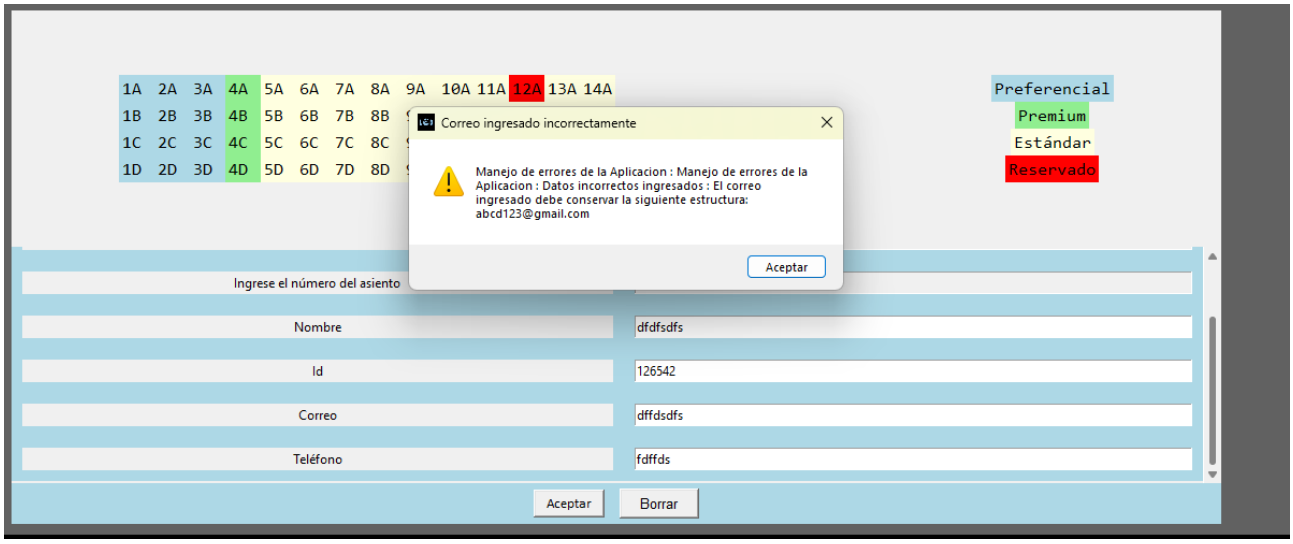


Luego se nos mostrará el bus con sus asientos y se nos pide que lijamos un asiento para reservarlo, si el asiento no existe o está ocupado nos saca un error:



En caso de ser correcto, aparecerá lo siguiente:

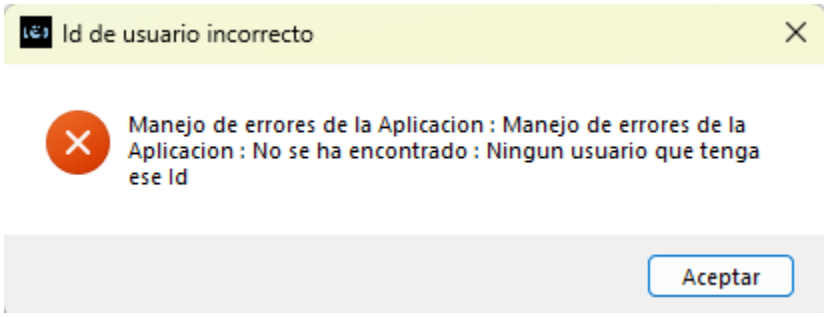
Un formulario que le pedirá al usuario que ingrese su nombre, teléfono, id y correo para poder imprimir el tiquete, para ello se debe tener en cuenta que el id debe tener 6 dígitos, el teléfono 10 y que el correo debe tener la estructura san123@gmail.com , de lo contrario se marcará su error correspondiente y no le permitirá al usuario imprimir su tiquete hasta que los datos no estén correctos:



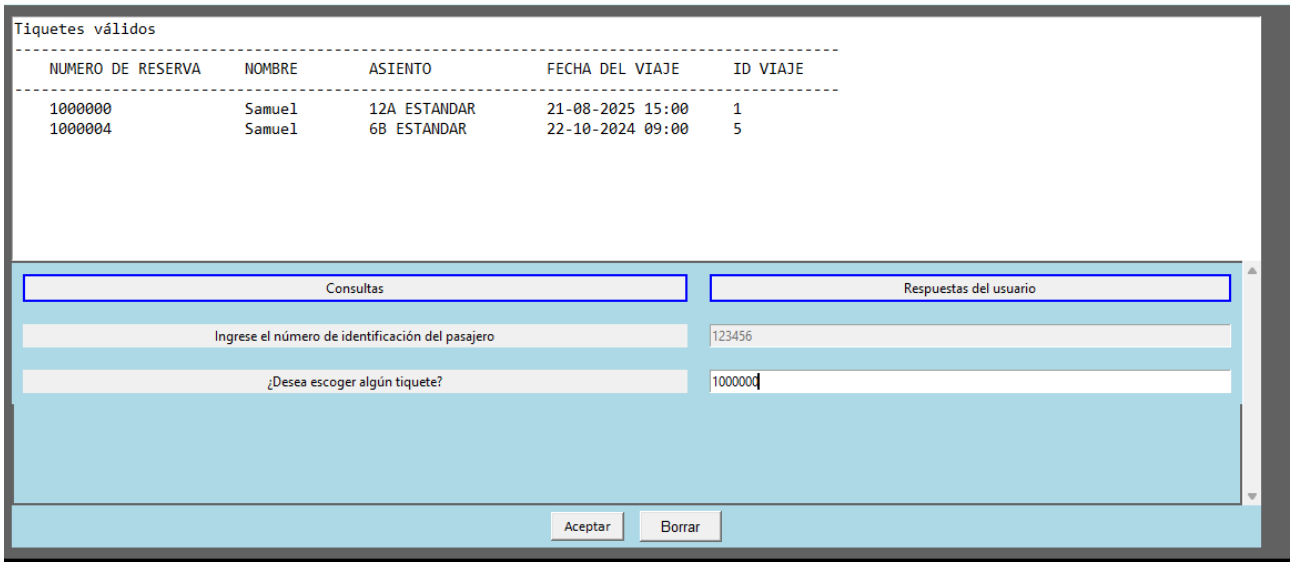
Finalmente, se imprimirá el tiquete resrvado, junto con varios datos relevantes para el proceso y si desea redirigirse a otra funcionalidad, siempre lo podrá hacer por medio del menú superior.

-Gestionar tiquetes,:
Esta funcionalidad es muy similar a las anteriores solamente que nos permite que ingresando el id de un usuario podamos cancelar, modificar el asiento o modificar el viaje elegido por el usuario del pasajero:

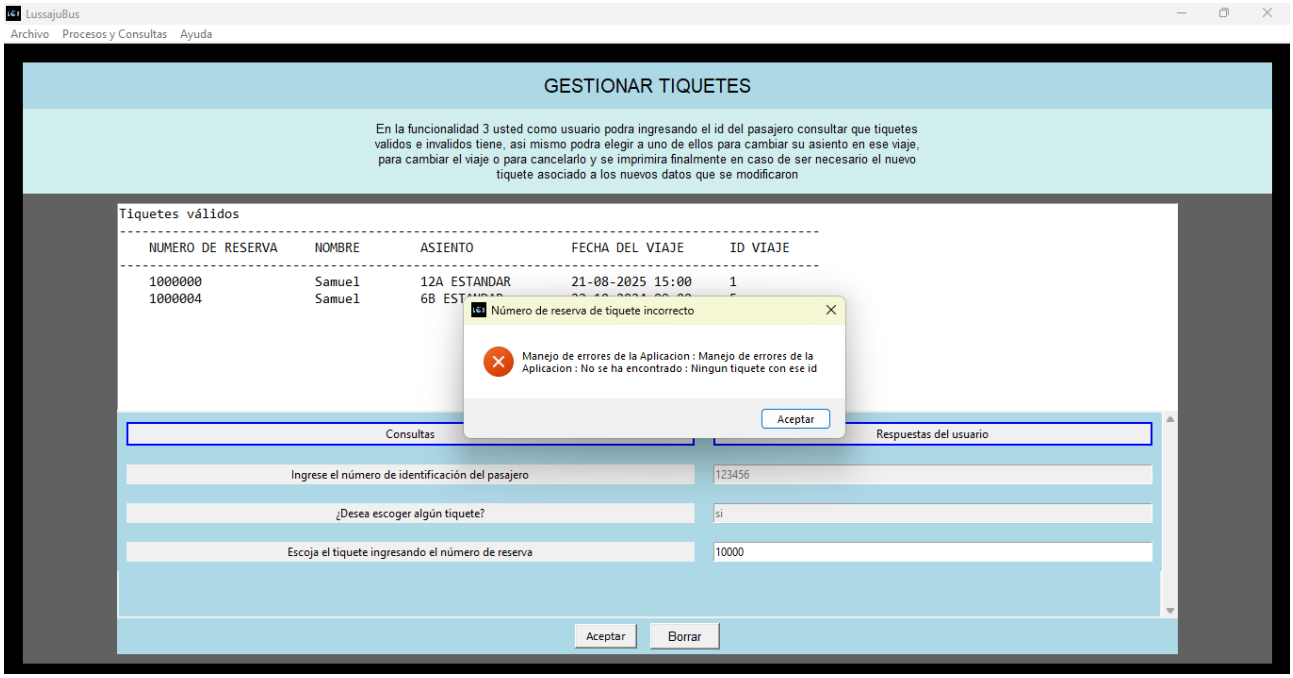
Si se ingresa un id incorrecto, saca error:



Si el id coincide con algún usuario, se pedirá que se ingrese el número de reserva del tiquete y se mostrarán los disponibles por pantalla:



Si se ingresa mal el número de reserva sacará error:



Anteriormente se pedirá revisar si si se quiere elegir algún tiquete o si no, en caso de que no se preguntará si se desea redirigir al menú principal; si el id es correcto se pregunta si se desea modificar el viaje o cancelarlo, si se ingresa cancelarlo, se imrimirá ‘Tiquete cancelado con éxito’ y se acabará esa parte del

programa:

Si se desea modificarlo aparecerá los posibles viajes por lo cuales lo puede modificar para el mismo destino o si desea cambiar púnicamente el asiento aparecerán los asientos disponibles para el mismo viaje.

Con esto culminamos el viaje por el manual de usuario de nuestra aplicación, esperamos que le haya sido de gran ayuda.