Práctica 1

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

1. 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)

```
1 package gestorAplicación.transporte;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5 import java.util.Random;
6
7 import gestorAplicación.personas.Conductor;
8
9 public abstract class Vehiculo implements Serializable {
10
11     private static ArrayList<String> placas = new ArrayList<String>();
12     private static final long serialVersionUID = 1613039627515609694L;
13     private String placa;
14     private Conductor conductor;
15
16     public Vehiculo() {
17
18     }
19
20     public Vehiculo(String placa) {
21         this.placa = placa;
22
23     }
24
25     public abstract void crearAsientos(int asientos);
26
27     public String getPlaca() {
28         return placa;
29     }
```

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

```
10
11 public class Bus extends Vehiculo implements Serializable {
12
```

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
public static Terminal buscarTerminal(String ubicacion) {
    ubicacion=ubicacion.toUpperCase();
    for (Terminal terminal : terminales) {
        if (terminal.getUbicacion().equals(ubicacion)) {
            return terminal;
        }
    }
    return null;
}
```

En la clase Asiento en el paquete gestorAplicación.transporte

```java
public static Asiento buscarAsiento(String numero, String tipo) {
    for (Asiento asiento : asientos) {
        if (asiento.getNumero() != null && asiento.getTipoAsiento() != null) {
            if (asiento.getNumero().equals(numero) && asiento.getTipoAsiento().
                    equals(TipoAsiento.valueOf(tipo))) {
                return asiento;
            }
        }
    }
    return null;
}
```

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 = 505737031214150790 4L;
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     }
32 
```

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

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.

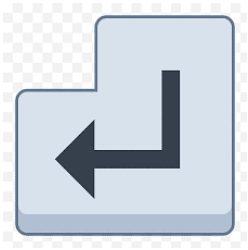
Inicialmente al abrir el programa se le dará la Bienvenida al Menú Principal y le pedirá que escoja una opción entre:

```
1. Ver viajes disponibles
2. Reservar tiquete
3. Gestionar tiquetes
4. Agregar servicio de hospedaje
5. Opciones de administrador
6. Salir
```

Posteriormente usted deberá ingresar el número de la opción que desea utilizar, cada opción hará algo diferente y se resume en el nombre.

A continuación comenzaremos a describir cada una de las funciones que ofrece el programa:

1. Ver viajes disponible**:** Al escribir 1 y darle al enter :



Se mostrarán todos los viajes que existen separados por cada empresa, de cada uno de ellos se mostrarán sus datos más relevantes como la fecha del viaje, el origen, el destino, la hora de salida, el id del viaje (esté será el identificador único de cada viaje) y la placa del bus.

Como la información puede ser mucha y los viajes pueden ser difíciles de encontrar usted podrá filtrar los viajes por alguna de las características dichas anteriormente por lo que si desea filtrar por alguna categoría deberá escribir “si” a la pregunta “¿Desea filtrar por alguna categoría?” para luego poner el nombre de la categoría por la que desea filtrar y luego la información; por ejemplo, al filtrar por Destino puedo poner Cali y se mostrarán todos los viajes que tengan a Cali como destino, en caso de no querer filtrar los viajes, se responderá a la pregunta “¿Desea filtrar por alguna categoría?” con “no”.

Luego de esto se mostrará la pregunta ¿Desea ver más detalles sobre un viaje?” en caso de responder “no” se le redirigirá al menú principal; en caso de responder “sí” se le pedirá que ingrese el id del viaje, por lo que usted como usuario deberá de ingresar el id del viaje del cual estemos interesados en guardar un asiento por un periodo de tiempo, si el viaje con el id escrito no existe se le redirigirá al menú principal.

Después de ingresar el id se le mostrará algo como esto:

```
ASIENTOS DISPONIBLES

PREFERENCIAL
                --
1A  1B  1C  1D      |
2A  2B  2C  2D      |
3A  3B  3C  3D      |
                --
PREMIUM
                --
4A  4B  4C  4D      |
                --
ESTANDAR
                --
5A  5B  5C  5D      |
6A  6B  6C  6D      |
7A  7B  7C  7D      |
8A  8B  8C  8D      |
9A  9B  9C  9D      |
10A 10B 10C 10D  |
11A 11B 11C 11D   |
     12B 12C 12D   |
13A 13B 13C 13D  |
14A 14B 14C 14D  |
                --
```

Esos serán los asientos disponibles para el viaje con el id que escogimos, los asientos estarán separados en PREFERENCIAL, PREMIUM y ESTÁNDAR. Cada asiento tendrá su respectivo número y letra y la ubicación dentro del bus donde la cabeza del bus será la parte superior y la cola la parte inferior; cada bus tendrá diferentes asientos y filas por cada sección pero todos conservan el mismo orden, es decir, PREFERENCIAL, PREMIUM, ESTÁNDAR. Si un asiento no aparece como en este caso el asiento 12A es porque dicho asiento ya está reservado por lo cual no se puede escoger, mientras que el resto de asientos sí.

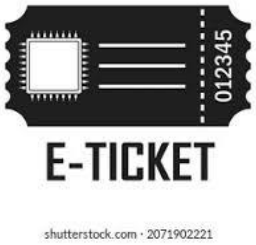
Posterior a esto le aparecerá la pregunta ¿Desea reservar un asiento por un cierto período de tiempo? a la que si se responde “no” le redirigirá al menú principal, pero al responder “si” se le hará otra pregunta: Ingrese el número del asiento: en la cual usted deberá ingresar el número y la letra (en mayúscula) del asiento que se quiere reservar por un tiempo, si el asiento no existe o está ocupado deberá elegir otro hasta que ingrese uno disponible, luego se le preguntará ¿Por cuánto tiempo desea reservarlo? (minutos/horas/dias) para responder a esta pregunta usted deberá escribir el número, luego dejar un espacio en blanco y luego escribir lo que corresponda (minuto/s, hora/s, dia/s), por ejemplo: 5 minutos, 1 hora, 7 dias.

Finalmente le mostrará el siguiente mensaje:

Asiento reservado exitosamente

Con el cuál se dará por terminada esta primera función del programa, al usted ver este mensaje tendrá separado ese asiento para usted durante el tiempo que ha sido reservado (si el asiento no se reserva definitivamente en la funcionalidad 2 no tendrá el asiento durante el viaje, esta sólo es una reserva temporal).

2. **Reservar tiquete:** Al escribir 2 y darle al enter :



Se nos pedirá que ingresemos el origen y el destino del viaje en el cual vamos a reservar nuestro tiquete, podemos saber qué viajes son los disponibles mediante la función descrita anteriormente para que en esta únicamente tengamos que ingresar la ciudad de origen y destino para filtrar los viajes (**CUIDADO** : las ciudades se deben ingresar en mayúscula, exactamente como aparecen en la primera función, ya que de no ser así puede que no se encuentren los viajes); luego de esto se nos filtrarán todos los viajes que cumlan con las condiciones de ciudad de origen y ciudad de destino, cada uno junto con su fecha, origen, destino, hora de salida, id del viaje y placa del bus.

Posterior a esto debemos de ingresar el id único del viaje del cual deseemos reservar el tiquete, en caso de que el id no exista o no esté dentro de los viajes mostrados en este último filtro el programa nos imprimirá un mensaje diciendo que no se encontró ningún viaje con ese id y nos redirigirá al menú principal; si el id si es válido nuevamente se nos imprimirá la estructura del bus con todos sus asientos:

```
ASIENTOS DISPONIBLES:

PREFERENCIAL
              --
1A  1B  1C  1D   |
2A  2B  2C  2D   |
3A  3B  3C  3D   |
              --
PREMIUM
              --
4A  4B  4C  4D   |
              --
ESTANDAR
              --
5A  5B  5C  5D   |
6A  6B  6C  6D   |
7A  7B  7C  7D   |
8A  8B  8C  8D   |
9A  9B  9C  9D   |
10A 10B 10C 10D  |
11A 11B 11C 11D  |
   12B 12C 12D   |
13A 13B 13C 13D  |
14A 14B 14C 14D  |
              --
Ingrese el número del asiento:
```

En esta parte nuevamente deberemos de elegir el asiento que deseamos reservar pero esta vez de manera permanente, todo con las mismas especificaciones dadas en la función 1.

Tras esto se nos pedirán los datos personales para completar la creación del tiquete, estos datos son: Nombre completo, Número de identificación, Teléfono y Correo.

Para tener en cuenta: El Nombre sólo servirá como referencia en ciertos casos, el Número de identificación es el realmente importante, este debe consistir en 6 dígitos (por ejemplo: 123456), sino el programa lo pedirá nuevamente, el teléfono y correo servirán como medios de contacto en caso de cualquier inconveniente, estos se generarán como datos al registrar el tiquete reservado y estos deben de cumplir con ciertas indicaciones: el número de teléfono deberá de tener 10 dígitos juntos sin espacios, guiones, ni letras (por ejemplo: 1234567890); y el correo deberá de conservar la estructura: abcdefg@wxyz.com; en caso de que el teléfono o el correo no cumplan con estas condiciones se le pedirá que lo ingrese nuevamente.

Finalmente se generará el tiquete, el cuál tendrá un número de identificación único y contendrá los siguientes datos acerca del pasajero y el viaje: Nombre del pasajero, Id del pasajero, Teléfono, Correo, Asiento, Empresa, Placa del bus, Id del viaje, Fecha y hora, Origen y Destino. El tiquete lucirá algo así:

```
---------------------------------
  Tiquete No.1000008
---------------------------------
Nombre del pasajero: Juan Roldan
Id del pasajero: 583629
Teléfono: 7253926252
Correo: JuanR@gmail.com
Asiento: 4D PREMIUM
Empresa: Jet
Placa del bus: GKG-899
Id del viaje: 7
Fecha y hora: 2026-07-25 20:10
Origen: PEREIRA
Destino: CALI
---------------------------------
```

Cuando aparezca el tiquete, ya se le habrá reservado el asiento de manera permanente, culmina con la función 2 y se le redirigirá al menú principal.

3. **Gestionar Tiquetes:** Al escribir 3 y darle al enter :

Para esta tercera funcionalidad inicialmente se pedirá el número de identificación del pasajero al cual deseamos gestionarle o modificarle el tiquete, si ingresamos un número que no sea el id de algún pasajero entonces le sacará el siguiente mensaje: No hay tiquetes asociados con el número de identificación y se le redirigirá al menú principal; en caso de que el número de identificación del pasajero si sea válido le imprimirá un cuadro como este:

```
Tiqueres válidos
-------------------------------------------------------------------------------------------------
    NUMERO DE RESERVA     NOMBRE        ASIENTO            FECHA DEL VIAJE      ID VIAJE
-------------------------------------------------------------------------------------------------
    1000002               Juan          5A PREMIUM         15-09-2025 13:00     3
    1000006               Juan          11D ESTANDAR       25-07-2026 20:10     7
```

El cual nos mostrará los tiquetes válidos que tiene dicho pasajero.

¿Qué es un tiquete válido? Un tiquete válido es un tiquete de un viaje que todavía no ha ocurrido, es decir un viaje en el cual la fecha del viaje es posterior a la fecha actual y por ende los tiquetes vencidos son aquellos en los que la fecha del viaje ya expiró.

Teniendo presente esto se mostrarán los tiquetes válidos juntos con alguna de su información: Número de reserva, Nombre, Asiento, Fecha del viaje e Id del viaje.

Luego de ello se nos hace la siguiente pregunta: ¿Desea escoger algún tiquete? (si/no) si escribimos "no", entonces se nos redirigirá al menú principal, si escribimos "si", nos pedirá que ingresemos el número de la reserva para escoger el tiquete que deseemos gestionar, si el número de reserva ingresado no coincide con ningún tiquete mostrado, entonces nos mostrará lo siguiente:
No se encontró ningún tiquete con el número de reserva y nos mandará nuevamente al menú principal; pero en caso de si encontrarlo nos imprimirá esto:

```
¿Qué desea hacer?
1. Cancelarlo
2. Modificarlo
```

Se deberá de ingresar el número de la operación que deseamos realizar con dicho tiquete, ya sea cancelarlo o modificarlo; si se escribe otro número vuelve a salir lo mismo, pero si escribimos cualquier otra cosa nos redirigirá al menú principal mostrando el tiquete sin ninguna modificación.

- Si se ingresa 1 y se decide cancelar el tiquete, el sistema se encargará de cancelar el tiquete, por lo que liberará el asiento que antes se había reservado, el usuario no tendrá más ese tiquete asignado y se le redirigirá al menú principal.

- Si se ingresa 2 y se decide modificar el tiquete podrá elegir entre dos opciones: si cambiar el asiento o cambiar el viaje.

En caso de ingresar 2:

```
¿Qué desea hacer?
1. Cambiar de asiento
2. Elegir otro viaje
```

Se debe de ingresar el número de la operación que deseamos realizar, si se escribe cualquier otra cosa nos redirigirá al menú principal mostrando el tiquete sin ninguna modificación.

- Si elegimos 1 esta vez nos mostrará los asientos disponibles para que podamos elegir uno y posteriormente imprimirá el tiquete con la modificación de asiento realizada:

```
ASIENTOS DISPONIBLES:

PREFERENCIAL
                --
1A  1B  1C  1D   |
2A  2B  2C  2D   |
3A  3B  3C  3D   |
                --
PREMIUM
                --
4A  4B  4C  4D   |
                --
ESTANDAR
                --
5A  5B  5C  5D   |
6A  6B  6C  6D   |
7A  7B  7C  7D   |
8A  8B  8C  8D   |
9A  9B  9C  9D   |
10A 10B 10C 10D  |
11A 11B 11C      |
12A 12B 12C 12D  |
13A 13B 13C 13D  |
                --
```

```
Tiquete modificado exitosamente
---------------------------------
   Tiquete No.1000006
---------------------------------
Nombre del pasajero: Juan
Id del pasajero: 123321
Teléfono: null
Correo: null
Asiento: 2C PREFERENCIAL
Empresa: Jet
Placa del bus: GKG-899
Id del viaje: 7
Fecha y hora: 2026-07-25 20:10
Origen: PEREIRA
Destino: CALI
---------------------------------
```

- Si elegimos 2 se nos pedirá ingresar el Origen y el Destino del nuevo viaje que vamos a elegir y de ahí en adelante continúa exactamente igual que con la función 2, hasta al final generar nuevamente otro tiquete con el nuevo viaje que generó el usuario.

Finalmente se imprimirá el tiquete con el mensaje: Tiquete modificado exitosamente
Lo cuál significa que ya el tiquete está modificado, que se ha terminado la tercera función del programa y que se le redirigirá al menú principal.

4. **Agregar servicio de hospedaje:** Al escribir 4 y darle al enter :

Inicialmente se nos pedirá que ingresemos el número de identificación del pasajero al cual le deseamos agregar el hospedaje, si no se encuentra al pasajero por medio del número de id ingresado imprimirá el siguiente mensaje: El pasajero no ha reservado tiquetes para ningún viaje y le redirigirá al menú principal, pero en caso de si encontrarlo se mostrarán por pantalla los Viajes que tiene actualmente reservados dicho pasajero junto con ciertos datos sobre ellos como: Fecha, Origen, Destino, Hora de salida, Id, Placa del bus y Asiento.

Posteriormente, se le hará la siguiente pregunta:
¿Desea agregar el servicio de hospedaje a algún viaje? (si/no)

Si se responde que no se le redirigirá al menú principal, y si responde que si, se le pedirá que ingrese el id del viaje sobre el cual desea agregar el servicio de hospedaje, si el viaje no se encuentra por su id sacará el siguiente mensaje:
No se encontró ningún viaje con el número de id
Y se le redirigirá al menú principal; en caso contrario le mostrará los hospedajes que están disponibles en la ciudad de destino del viaje a realizar, cada una junto con su Nombre, Calificación, y Habitaciones disponibles; de la siguiente manera:

Suponiendo que la ciudad destino es BOGOTÁ:

```
Hospedajes disponibles en BOGOTÁ:
------------------------------------------------------------
  NOMBRE    CALIFICACIÓN     HABITACIONES DISPONIBLES
------------------------------------------------------------
  Nest      0.0 estrellas    20

Ingrese el nombre del hospedaje que desea:
```

Luego deberá de ingresar el nombre del hospedaje en el cual se desea hospedar durante el viaje, si el hospedaje ingresado no coincide con alguno de los mostrados se imprimirá lo siguiente:
El hospedaje a no está disponible para este destino y se le redirigirá al menú principal;
De coincidir el nombre, es decir, de ingresar el nombre del hospedaje de manera correcta, se mostrarán las habitaciones que tiene el hospedaje, cuáles están disponibles, cuáles están ocupadas, y en cuánto tiempo se liberarán, esto se muestra de la siguiente manera:

```
Habitaciones disponibles:
------------------------------------------------------------
  NÚMERO DE HABITACIÓN     RESERVADA     DISPONIBLE EN
------------------------------------------------------------
  101                 No          null
  102                 No          null
  103                 No          null
  104                 No          null
  105                 No          null
  201                 No          null
  202                 No          null
  203                 No          null
  204                 Sí          1 día 18 horas 45 minutos
  205                 No          null
  301                 No          null
  302                 No          null
  303                 No          null
  304                 No          null
  305                 No          null
  401                 No          null
  402                 No          null
  403                 No          null
  404                 No          null
  405                 No          null
```

Importante: *el primer número de cada habitación es el piso y los otro dos serán el apartamento si la habitación no está reservada se puede agregar al viaje ingresando su número, si una habitación ya está reservada se podrá observar en la última columna en cuánto tiempo se liberará para poder ser elegida.*

- **NOTA:** también tenga en cuenta que el tiempo para que esté disponible nuevamente la habitación será el tiempo durante el que se reserva desde que inicia el viaje.

Después de ingresar el número de la habitación (Si no existe la habitación con el número indicado se volverá a preguntar) se nos preguntará: ¿por cuánto tiempo vamos a reservar la habitación?

**CUIDADO : ** para responder a esta pregunta usted deberá escribir el número, luego dejar un espacio en blanco y luego escribir lo que corresponda (hora/s, dia/s), por ejemplo: 1 hora, 7 dias.

Finalmente le sacará un mensaje diciendo que su habitación ha en el hospedaje ha sido correctamente asignada a su viaje, con lo cuál termina la cuarta función del programa y lo redirige al menú principal.

5. **Opciones de administrador:** Al escribir 5 y darle al enter :

Para culminar con la guía de cómo usar el programa falta explicar el funcionamiento de la quinta función, es decir todo lo que puede hacer un administrador en el programa, aquí ya la cosa se pone un poco más técnica, pero la funcionalidad es bastante simple, al ingresar le aparecerá lo siguiente:

```
Bienvenido Administrador
¿Qué desea modificar?
1. Empresas
2. Hospedajes
3. Terminales
4. Viajes
5. Personal
6. Buses
7. Volver
Ingrese el número de la operación:
```

Por cada una de las opciones que se ingrese se podrá modificar aspectos varios de cada uno (recuerde que se ingrese el número de la opción a ejecutar)

A continuación vamos a explorar por cada opción que se puede realizar:

- ***Empresas:***

```
Ingrese una opción
1. Agregar
2. Ver
3. Eliminar
4. Volver
```

Se pueden agregar empresas al sistema (1), ver todas las empresas que pertenecen al sistema (2) ó eliminar empresas del sistema (3), además se puede agregar o eliminar conductores de las empresas dentro de sus respectivos apartados; la opción 4 se usa para volver al apartado anterior.

- ***Hospedajes:***

```
Ingrese una opción
1. Agregar
2. Ver
3. Eliminar
4. Volver
```

Se pueden agregar hospedajes y habitaciones al sistema (1), ver todos los hospedajes que pertenecen al sistema, su calificación y cuántas habitaciones tienen (2) ó eliminar hospedajes o habitaciones del sistema (3), además se puede agregar calificaciones a un hospedaje en específico (de tener varias, estas se promedian para dar un único resultado), la opción 4 se usa para volver al apartado anterior.

- ***Terminales:***

```
Ingrese una opción
1. Agregar
2. Ver
3. Eliminar
4. Volver
```

Se pueden agregar terminales al sistema (1), ver todas las terminales que pertenecen al sistema (2) ó eliminar terminales del sistema (3), además se puede agregar o eliminar empresas de las terminales dentro de sus respectivos apartados; la opción 4 se usa para volver al apartado anterior.

- ***Viajes:***

```
Ingrese una opción
1. Agregar
2. Modificar
3. Ver
4. Eliminar
5. Volver
```

Se pueden agregar Viajes y Tiquetes al sistema de manera indirecta(1), modificar ciertos elementos de un viaje o de un tiquete, como la ciudad de Origen y Destino, o el conductores por dar algunos ejemplos,también se puede ver todos los Viajes y Tiquetes que pertenecen al sistema(3) ó eliminar Hospedajes y Pasajeros que hagan parte del sistema (4),, la opción 5 se usa para volver al apartado anterior. Para agregar un Viaje o un Tiquete únicamente hay que responder a todas las preguntas que vayan surgiendo respecto al bus, pasajero, viaje, etc.

- ***Personal:***

```
Ingrese una opción
1. Agregar
2. Modificar
3. Ver
4. Eliminar
5. Volver
```

Se pueden agregar Pasajeros y Conductores al sistema (1), modificar información referente a Conductores y Pasajeros específicos (2), ver todos los Pasajeros y Conductores que pertenecen al sistema (3) ó eliminar Pasajeros y Conductores del sistema (4), la opción 5 se usa para volver al apartado anterior.

- ***Buses:***

```
Ingrese una opción
1. Agregar
2. Ver
3. Eliminar
4. Volver
```

Se pueden agregar buses al sistema (1), ver todos los buses que pertenecen al sistema (2) ó eliminar buses del sistema (3); la opción 4 se usa para volver al apartado anterior.

- ***Volver:***

Esta opción simplemente se utiliza para volver al menú principal y salir del modo administrador.

CONSIDERACIONES IMPORTANTES ⚡ :

- Se recomienda única y exclusivamente hacer uso de las opciones de eliminar objetos cuando se esté completamente seguro de que el objeto no tiene información relevante dentro del programa, ya que si esto no se tiene en cuenta puede generar error al consultar cierto tipo de información (OJO: Usted fue advertido).
- Un ejemplo del literal anterior es: eliminar un bus que ya tiene viajes asignados, por lo que si se intenta ver el viaje el código sacará error.
- Además, cada vez que se modifique o se realice algún cambio en el modo administrador, este le hará la pregunta de si desea continuar en el modo administrador o salir al menú principal, ya se deja a consideración del usuario el seguir explorando el uso del programa y de las infinitas posibilidades que este ofrece respecto al sistema terrestre unificado de transporte en Colombia.

Finalmente la última opción dentro del menú principal sirve para cerrar el programa permitiendo que todo lo que se haya realizado se guarde para una próxima oportunidad; si el programa se cierra bruscamente, es decir, de otra manera que no sea por medio de esta opción, toda la información que se haya agregado y modificado se perderá por lo que siempre hay que recordar cerrar el programa por este medio. □

Con esto hemos culminado con esta guía básica acerca del funcionamiento de LussajuBus, esperamos que haya sido de su agrado y ¡a unificar el transporte juntos! □□□