UNIVERSIDAD NACIONAL DE COLOMBIA

SISTEMA DE GESTIÓN DE VENTA DE VUELOS- PRACTICA1-GRUPO2-EQUIPO7

INTEGRANTES

Juan Carlos Largo Buenahora, Harrison Zuleta Montoya, María Alejandra Muñoz González, Santiago Andrés Mejía García y Santiago Martínez Ríos

DESCRIPCIÓN

Descripción general

Este programa busca permitir la compra y reasignación de boletos de vuelo, servicios de la aerolínea, con la ventaja de un sistema de inicio de sesión que permite reconocer al usuario y su historial de compras, millas adquiridas y los descuentos que le permiten adquirir estas millas. Se busca permitir la experiencia tradicional de las aerolíneas en sus páginas web o en general la experiencia de compra de un vuelo hasta la confirmación de vuelo.

Descripción técnica

En el desarrollo del programa se busca dar una aplicación real a los conocimientos adquiridos acerca de la POO (Programación Orientada a Objetos) y sus pilares, los cuales son:

Abstracción: por medio del cual sacamos características del mundo real para darle una funcionalidad y acercamiento realista al problema o meta de la aplicación, un ejemplo muy claro son las características y métodos de las clases principales que involucra nuestro programa las cuales son asiento, Usuario, Pasajero, Boleto, Vuelo y Maleta que son las bases sobre las que se construye todo nuestro programa al sacar pedazos reconocibles de estos objetos o seres en la vida real, permitiéndonos la creación de funcionalidades como la compra, reasignación, descuento y gestión de usuario. Ejemplo directo, nuestra clase Maleta, la cual abstrae de su característica de ser un objeto tridimensional, sus medidas nos ayudan con el cálculo de la cuenta general del vuelo, y definir ciertas restricciones de peso y tamaño para todas sus instancias en los diferentes espacios que definimos que se puede transportar.

```java
package gestorAplicacion.Aerolinea;

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

import gestorAplicacion.Cuenta.*;

import static uiMain.Estetica.*;

public class Maleta implements Serializable {

    private static final long serialVersionUID = 1L;
    private int id;

    private int peso;
    private int largo;
    private int ancho;
    private int alto; // Al fin y al cabo es un volumen

    private Pasajero pasajero; //
    private Boleto boleto;
    private String destino_origen;
    private String estado;
```

Encapsulamiento: por medio del cual permitimos que el programa sea seguro ante el uso irreconocible de los atributos de la clase y limitamos la interacción de las clases de forma que armonicen para el propósito que buscamos, un ejemplo claro de esto es la privatización de los atributos de las clases core de nuestro programa, obligando a hacer uso de herramientas que aunque permiten el acceso a estos atributos , nos ayuda a mantener un seguimiento claro de las interacciones de los atributos, manejo de los métodos e infracciones de las clases en general en pos de la realización de las funcionalidades y acciones que buscamos entre ellos, haciendo que el programa sea fácil de seguir, documentar y entender en su totalidad, impidiendo igualmente errores e interacciones indeseadas.

Un ejemplo directo de esta aplicación como dijimos son los atributos privados de las clases core, que son accedidos en las diferentes funcionalidades por medio de métodos, haciendo que haya un seguimiento claro de cuando se accede a qué cosa de cada funcionalidad

Ejemplo: acceso al atributo asiento y vuelo de la clase boleto, el acceso al asiento de la clase vuelo inmediatamente sacada de boleto.

Clase Boleto:

```
14  public class Boleto implements Serializable {
15
16      private static final long serialVersionUID = 1L;
17
18      private static int cont = 0;
19      private int id;
20
21      // Atributos
22      private String tipo;
23      private Usuario user;
24      private String status = "Pendiente";
25      private String origen;
26      private String destino;
27      private boolean checkInRealizado = false;
28      private ArrayList<ServiciosEspeciales> serviciosContratados = new A
29      private ArrayList<Animal> mascotas = new ArrayList<>();
30      private int cantidadMascotasCabina = 0;
31      private int cantidadMascotasBodega = 0;
32
33      private float valor;
34
35      private ArrayList<Maleta> equipaje = new ArrayList<>();
36      private Asiento asiento;
```

Clase vuelo:

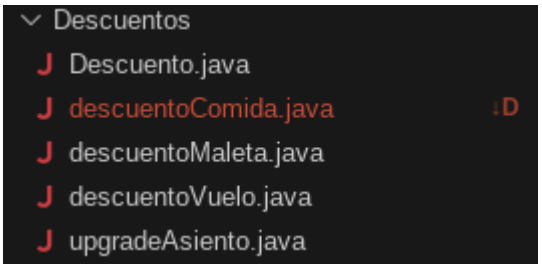
```java
public class Vuelo implements Serializable { // se crea la clase e implementa serializacion
    private static final long serialVersionUID = 1L;

    ArrayList<Asiento> asientos = new ArrayList<>();
    private final String AEROLINEA;
    private final String ID;
    private String horarioSalida;
    private String horarioLlegada;
    private final String DESTINO;
    private final String ORIGEN;
    private ArrayList<Maleta> equipajes = new ArrayList<>();
```

Acceso atributos privados vuelo, asiento de boleto, y asientos de vuelo:

```java
private static void millasAsiento(Usuario user, Descuento descuento) {

    Boleto boleto = selecBoleto(user);
    Asiento asiento = boleto.getAsiento();
    // se verifica que el asiento sea economico
    // si es vip ya no se puede mejorar

    if (asiento.getTipo() == "Economico") {

        // Hacer asiento vip
        ArrayList<Asiento> asientos = (boleto.getVuelo()).getAsientos();

        printNegrita(colorTexto("Asientos disponibles", "morado"));
        salto();
        for (Asiento asientoTemp : asientos) {
            if (asientoTemp.getTipo().equals("Vip")) {
                identacion(asientoTemp.getInfo(), 2);
            }
        }

        salto();
        promptIn("Por favor, seleccione el número del asiento deseado: ");
        int indexAsiento = inputI();

        // ... Cambiar y reasignar todo
        Asiento newAsiento = asientos.get(indexAsiento - 1);
        boleto.upgradeAsiento(asiento, newAsiento);
        descuento.aplicarDescuento(boleto);
```

Polimorfismo: que permite aumentar el rango de acciones o limitarlas para poder definir las acciones o funcionalidades desde diferentes enfoques, desde la limitación que tienen los animales en general en el transporte de los vuelos, a las restricciones que tienen perros y gatos por individual, a pesar de ser precisamente de tipo animal, haciendo que sea más diversa la forma en que podamos enfocar restricciones, acciones o características que esperamos de estos objetos que hemos abstraído de la realidad.

Un ejemplo de esto es la lista de tipo descuento que tiene Boleto, en donde son almacenados los descuentos de comida, maleta, vuelo y asiento.

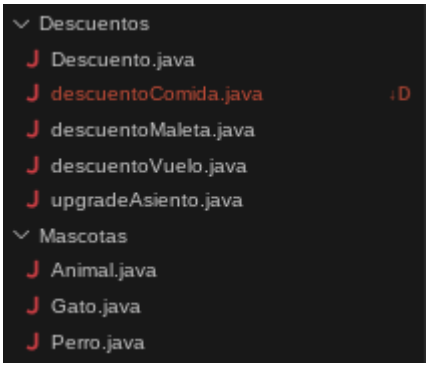



```java
public class Boleto implements Serializable {

    private static final long serialVersionUID = 1L;

    private static int cont = 0;
    private int id;

    // Atributos
    private String tipo;
    private Usuario user;
    private String status = "Pendiente";
    private String origen;
    private String destino;
    private boolean checkInRealizado = false;
    private ArrayList<ServiciosEspeciales> serviciosContratados = new ArrayList<>();
    private ArrayList<Animal> mascotas = new ArrayList<>();
    private int cantidadMascotasCabina = 0;
    private int cantidadMascotasBodega = 0;

    private float valor;

    private ArrayList<Maleta> equipaje = new ArrayList<>();
    private Asiento asiento;
    private Pasajero pasajero;


    // ...precios
    private float valorInicial;
    private float valorEquipaje;
    // precios...

    // Some atributos...
    private ArrayList<Descuento> descuentos = new ArrayList<>();
        private Vuelo vuelo;
```

Herencia: herramienta por la cual se nos permite crear una especie de jerarquía relacional y nos permite hacer un reuso de porciones de código que hayamos escrito previamente, permitiendo una

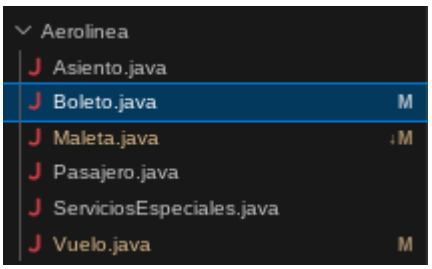
representación más fácil y clara de sistemas de la vida real en ambiente digital de nuestro programa permitiendo relaciones de poder y de familiaridad, un ejemplo claro de esto en nuestro código es el uso de clases abstractas, como Descuento y Animal, con las cuales creamos una jerarquía y reutilizamos código de Descuento para tener diversos tipos y distinciones entre clases con cierto parecido comportamental y distintivo.

Un ejemplo de esto son las clases Perro y Gato que heredan de la clase Animal, y las clases descuentoComida, descuentoAsiento, descuentoVuelo y descuentoMaleta heredan de la clase Descuento. Ambas son clases abstractas y como todas las clases core implementan la interfaz serializable y definen una variable serialVersionUiD que es heredada de la interfaz Serializable.




```
public class Gato extends Animal implements Serializable {
```

```
public class descuentoComida extends Descuento implements Serializable {
```




```
public class Boleto implements Serializable {

    private static final long serialVersionUID = 1L;
```

.

DIAGRAMAS

Diagrama de Clases (UML)

https://app.genmymodel.com/api/projects/_2HXnUHKAEe6A45N60iXn8g/diagrams/_2HXnU3KAEe6A45N60iXn8g/svg

este es el link hacia el esquema UML

FUNCIONALIDADES

1.Funcionalidad “Comprar vuelo”

Esta funcionalidad implica una interacción entre varias clases, incluyendo Usuario, Aeropuerto, Avión y Silla. El proceso inicia con la revisión de los aeropuertos existentes en la ubicación seleccionada por el usuario, se muestran los vuelos disponibles e información relacionada a estos. A continuación, el usuario elige uno de los vuelos disponibles. Si el vuelo tiene asientos disponibles, se procede a la selección del tipo de silla, que puede ser Primera Clase o Económica. Finalmente, se selecciona un número de silla entre las disponibles, se agrega el equipaje y se asigna un número de boleto al pasajero, el cual va relacionado a su equipaje y asiento. Además, en esta funcionalidad, se otorgan "créditos" o “millas” al usuario por cada vuelo que compre, en proporción al valor del boleto adquirido; cuanto más caro sea el boleto, más créditos se otorgarán. También, se debe tener en cuenta la wallet que almacena el dinero e información del usuario, así como la clase de cuenta asociada a dicho usuario en la cual se asignan los créditos .

2.Funcionalidad “Reasignar vuelo”

En esta opción, primero se llevará a cabo la reasignación de vuelo de un pasajero, y por lo tanto, la silla asignada se eliminará del vuelo anterior y se cambiará al nuevo vuelo seleccionado por el usuario. En primer lugar, se verificará la existencia del vuelo del que se solicita la reasignación y luego la del pasajero y la silla. Posteriormente, se mostrarán los vuelos disponibles hacia el mismo destino (esto cambiará en caso de que la hora del vuelo haya pasado). Se realizará el procedimiento de asignación revisando los tipos de sillas disponibles, dado que, si se realiza la reasignación, no se garantiza que el tipo de silla elegido sea disponible en el futuro; puede que no se pueda seleccionar el mismo tipo, o en caso de que se seleccione otro tipo, se cobrará el faltante para llegar al precio. Luego, se procederá a la elección de la silla y la respectiva asignación del asiento al cliente que lo solicita, pagando además un monto por cambiar el asiento En esta funcionalidad intervienen y hay interacciones entre las clases Usuario, Aeropuerto, Avión y Silla.

3. Funcionalidad “Cancelar vuelo”

Descripción General:

Este código en Java describe la funcionalidad de cancelar un vuelo para un usuario, este código permite a un usuario ver su historial de vuelos, seleccionar un vuelo específico para cancelarlo y realizar la cancelación si lo confirma. También proporciona una retroalimentación visual al usuario para informarle sobre el resultado de la cancelación.

Principales Características y Acciones:

- La función cancelarVuelo toma un objeto Usuario como parámetro, que representa al usuario que desea cancelar un vuelo.
- Se muestra una lista de los vuelos que el usuario ha reservado en el pasado. Esto se logra accediendo al historial de boletos del usuario mediante user.getHistorial().
- Se itera a través de los boletos en el historial y muestra información sobre cada uno de ellos. El usuario debe seleccionar el vuelo que desea cancelar ingresando el número correspondiente.

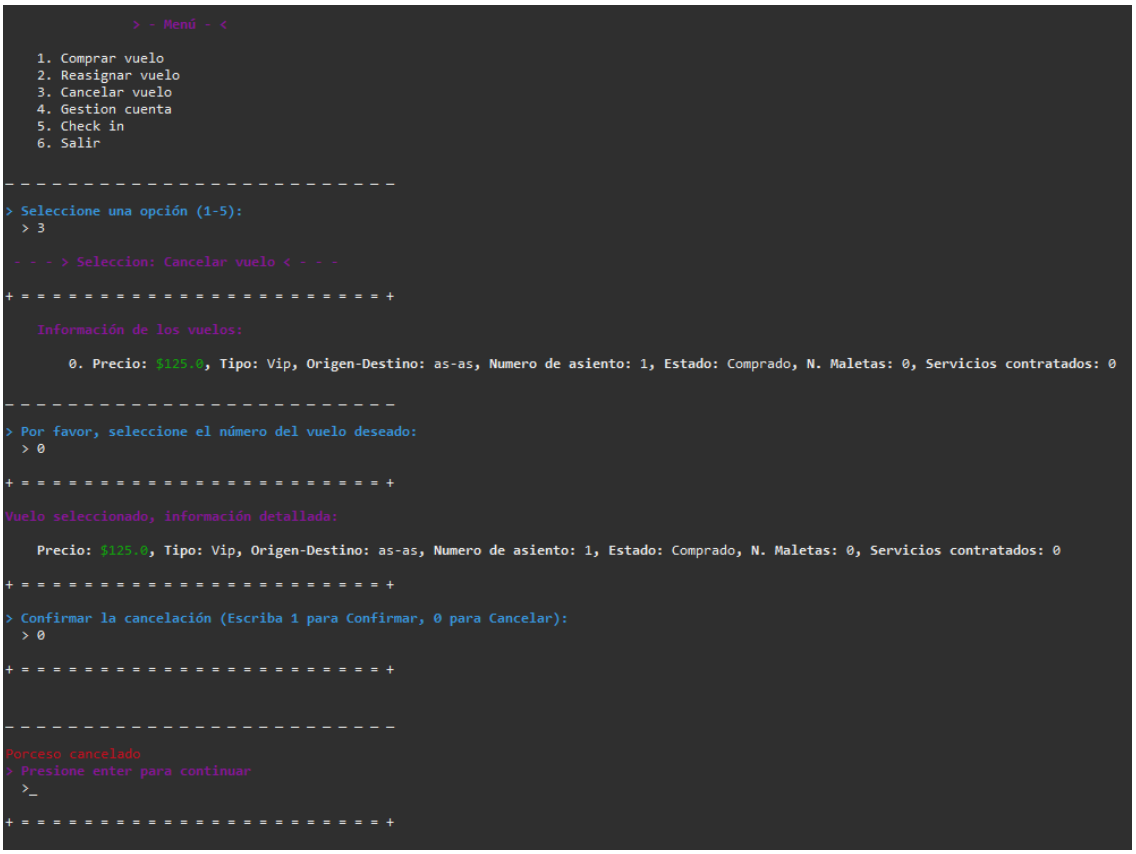
  Para esto se utilizan instancias de Boleto y Usuario, además de Asiento y Vuelo para mostrar la información

- Una vez que el usuario selecciona un vuelo, se muestra información detallada sobre ese vuelo.
- Se le pide al usuario que confirme la cancelación ingresando "1" para confirmar o "0" para cancelar.
- Si el usuario confirma la cancelación (ingresando "1"), se realizan las siguientes acciones:
  - El estado del boleto se cambia a "Cancelado".
  - El boleto se elimina de la lista de boletos del usuario.

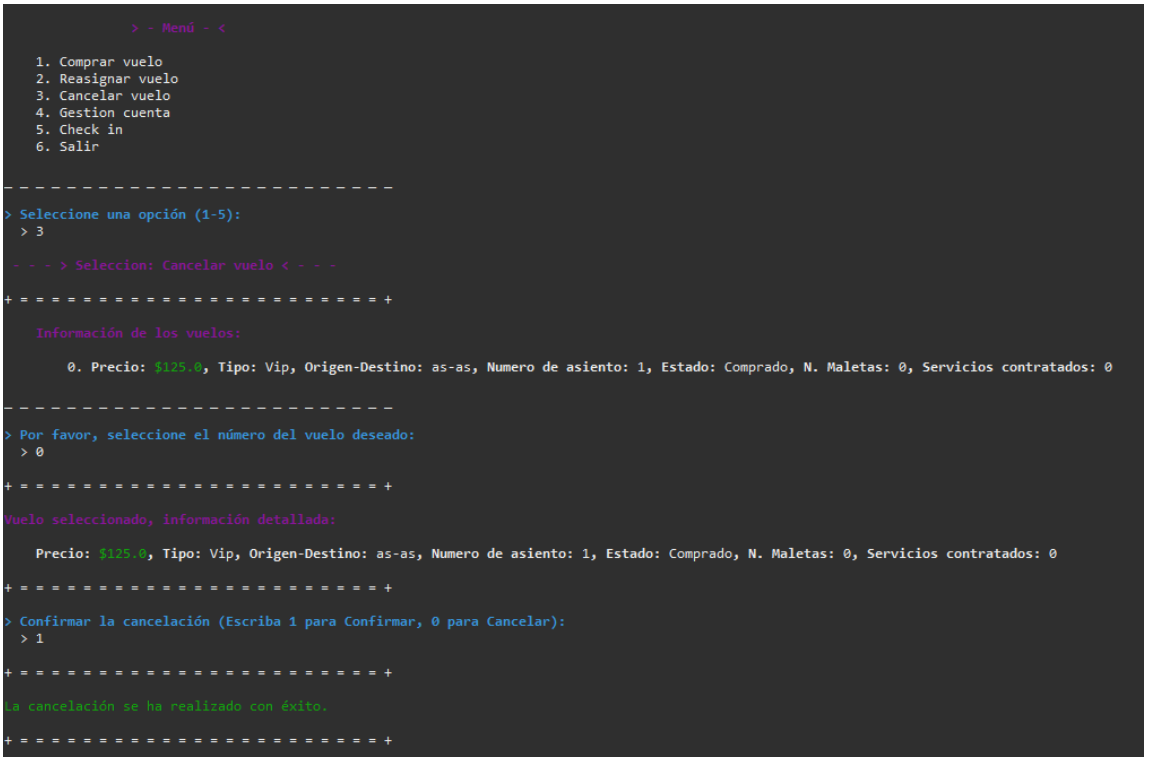
- El asiento asociado al boleto se desasigna, lo que significa que está disponible nuevamente para su reserva.
- Se muestra un mensaje al usuario confirmando que la cancelación se ha realizado con éxito.

Aquí se usan instancias de Boleto para establecerlo como cancelado, de Usuario para eliminar las millas ganadas por la compra del vuelo y el retorno de un porcentaje de lo que pagó por el vuelo y una instancia de Asiento para eliminarlo del boleto y establecerlo como disponible

Capturas de los resultados



Se selecciona la opción de cancelar vuelo, se elige el vuelo que se desea cancelar, no se confirmó la cancelación por lo que se aborta el proceso.



Se selecciona la opción de cancelar vuelo, se elige el vuelo que se desea cancelar, se confirmó la cancelación entonces se cancela el vuelo.

Se selecciona la opción de cancelar vuelo, se elige el vuelo que se desea cancelar, como ya fue cancelado no permite cancelarlo de nuevo.

4.Funcionalidad “Gestión cuenta”

La funcionalidad le permite al usuario ver información acerca de su cuenta, mirar el historial de vuelos comprados, hacer transferencias a la cuenta y sobre todo lo más importante tiene la funcionalidad canjear millas; dónde el usuario puede intercambiar millas por ciertos beneficios, por ejemplo puede volver un asiento económico en una VIP canjeando un descuento de asiento para las millas. También puede realizar cierto porcentaje de descuento al valor base del vuelo millas por un descuento de vuelo y lo mismo con el costo de equipaje.

5. Funcionalidad "Check-in"

Descripción General:

La funcionalidad de "Check-in" es una parte esencial de un sistema de gestión de vuelos que permite a los pasajeros confirmar su presencia en un vuelo específico antes de abordar la aeronave. El proceso de check-in es fundamental para garantizar una experiencia de vuelo fluida y cómoda para los pasajeros, y puede incluir varias acciones, como mejorar asientos y adquirir servicios especiales.

Principales Características y Acciones:

- **Mostrar la Lista de Vuelos:**

La función check-in comienza mostrando al usuario una lista de los vuelos anteriores que ha reservado. Esto se logra utilizando la lista de boletos almacenada en el objeto Usuario.

Aquí interviene objetos de la clase Usuario para obtener el historial de boletos y de la clase Boleto para obtener mostrar la información de cada boleto en el historial.

- **Seleccionar el Vuelo:**

El usuario selecciona el boleto correspondiente al vuelo deseado introduciendo un valor numérico entre los mostrados de su historial de vuelos reservados.

Aquí solo interviene un objeto de la clase Boleto para obtener el boleto del vuelo deseado.

- **Mostrar el vuelo seleccionado:**

El sistema muestra la información detallada del vuelo seleccionado.

Aquí interviene un objeto de la clase Boleto para mostrar la información de el objeto de la clase Vuelo y Asiento asociado a este.

- **Opciones de Check-in:**

Si el check-in no se ha realizado previamente (verificado mediante el estado del boleto), se muestra un menú con varias opciones al usuario.

El usuario elige entre realizar el check-in, mejorar el asiento o comprar servicios especiales.

Aquí interviene el objeto de Boleto para verificar si no se ha realizado el check in para poder proceder.

1. **Realizar Check-in:**

Si decide realizar el check-in, el sistema verifica si el check-in ya se ha realizado. Si no se ha realizado, el usuario confirma la operación, y el sistema actualiza el estado del boleto y marca el check-in como realizado.

Aquí interviene el objeto de Boleto para realizar la confirmación del check in.

2. **Mejorar Asiento:**

Si el usuario opta por mejorar su asiento, el sistema verifica que el asiento que posee sea de la clase económica, la función muestra una lista de asientos VIP disponibles. El usuario selecciona uno de estos asientos y se le pide confirmación para realizar la transacción, se actualiza el asiento en el boleto y se realiza el pago del sobrecosto, en caso de no ser de clase económica se le informa que no es posible mejorar ya que pertenece a VIP.

Aquí intervienen objetos de la clase Boleto para obtener el asiento y verificar que el tipo de asiento pueda ser mejorado, se usa el objeto de la clase Usuario para realizar el pago de la mejora.

3. **Comprar Servicios Especiales:**

Si el usuario opta por comprar servicios especiales, el sistema muestra las opciones disponibles, el usuario selecciona un servicio y se confirma la compra. El sistema actualiza la lista de servicios especiales contratados en el boleto y realiza el correspondiente cargo al usuario.

Las opciones disponibles son:

- **Comida a la carta**

Cuando el usuario elige comprar "Comida a la Carta" como servicio especial, el sistema verifica la disponibilidad de este servicio en el vuelo y le pide confirmación para realizar la transacción así añadiendo el servicio al boleto y realizando el cargo a la cuenta del usuario.

Intervienen objetos de ServiciosEspeciales para agregarlos al objeto de Boleto y realizar el pago de este servicio con el objeto de Usuario.

- **Viajar con mascotas**

Cuando el usuario decide viajar con una mascota, el sistema guía al usuario a través de la información necesaria sobre la mascota, como el nombre, la raza, el tamaño y el peso. Se verifica si la mascota cumple con las regulaciones de viaje en cabina o en la bodega, estas regulaciones dependen de la raza, peso, tamaño de la mascota y de la cantidad de mascotas ya registradas para viajar con el pasajero, si cumple las restricciones el usuario confirma la transacción y se agrega el servicios al boleto, la mascota a la lista de mascotas del boleto y se realiza el pago de este servicios.

Aquí intervienen objetos de las clases Perro y Gato para crear las mascotas que van a viajar, con estos objetos se verifican que el animal pueda viajar y con ServiciosEspeciales se agregan el respectivo servicio al objeto de Boleto así como la mascota y con Usuario se realiza el pago del servicio

- **Contratar un acompañante para un menor de edad**

Al contratar un "Acompañante para Menor de Edad", el sistema solicita la confirmación del usuario. Si se confirma, se agrega este servicio al boleto y se realiza el respectivo cargo al usuario.

Intervienen objetos de ServiciosEspeciales para agregarlos al objeto de Boleto y realizar el pago de este servicio con el objeto de Usuario.

- **Solicitar asistencia para pasajeros con necesidades especiales**

Al seleccionar la opción "Asistencia para Pasajero con Necesidades Especiales," el usuario confirma la solicitud de este servicio, que no tiene costo adicional y se agrega el servicio al boleto.

Intervienen objetos de ServiciosEspeciales para agregarlos al objeto de Boleto y realizar el pago de este servicio con el objeto de Usuario.

- **Reservar transporte terrestre**

Cuando el usuario opta por reservar "Transporte Terrestre," el sistema solicita la confirmación. Si se confirma, se agrega este servicio al boleto y se realiza el respectivo pago.

Intervienen objetos de ServiciosEspeciales para agregarlos al objeto de Boleto y realizar el pago de este servicio con el objeto de Usuario.

- **Ver Servicios Contratados**

El usuario tiene la opción de ver una lista de los servicios especiales contratados. Esto proporciona una visión general de los servicios adicionales que ha adquirido para el vuelo.

Interviene objetos de Boleto para obtener la lista de los ServiciosEspeciales contratados y mostrar el resumen de estos servicios así como los Animales ya sea Perro o Gato que se lleven en el viaje

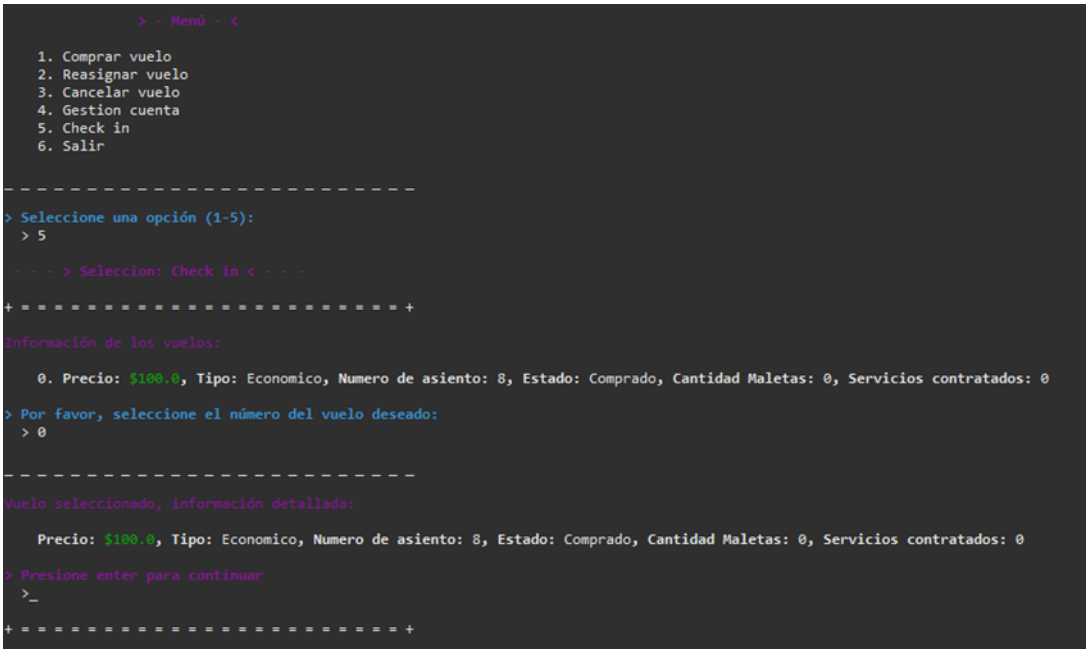
- **Volver al Menú Anterior:**

En cualquier momento, el usuario puede optar por volver al menú anterior.

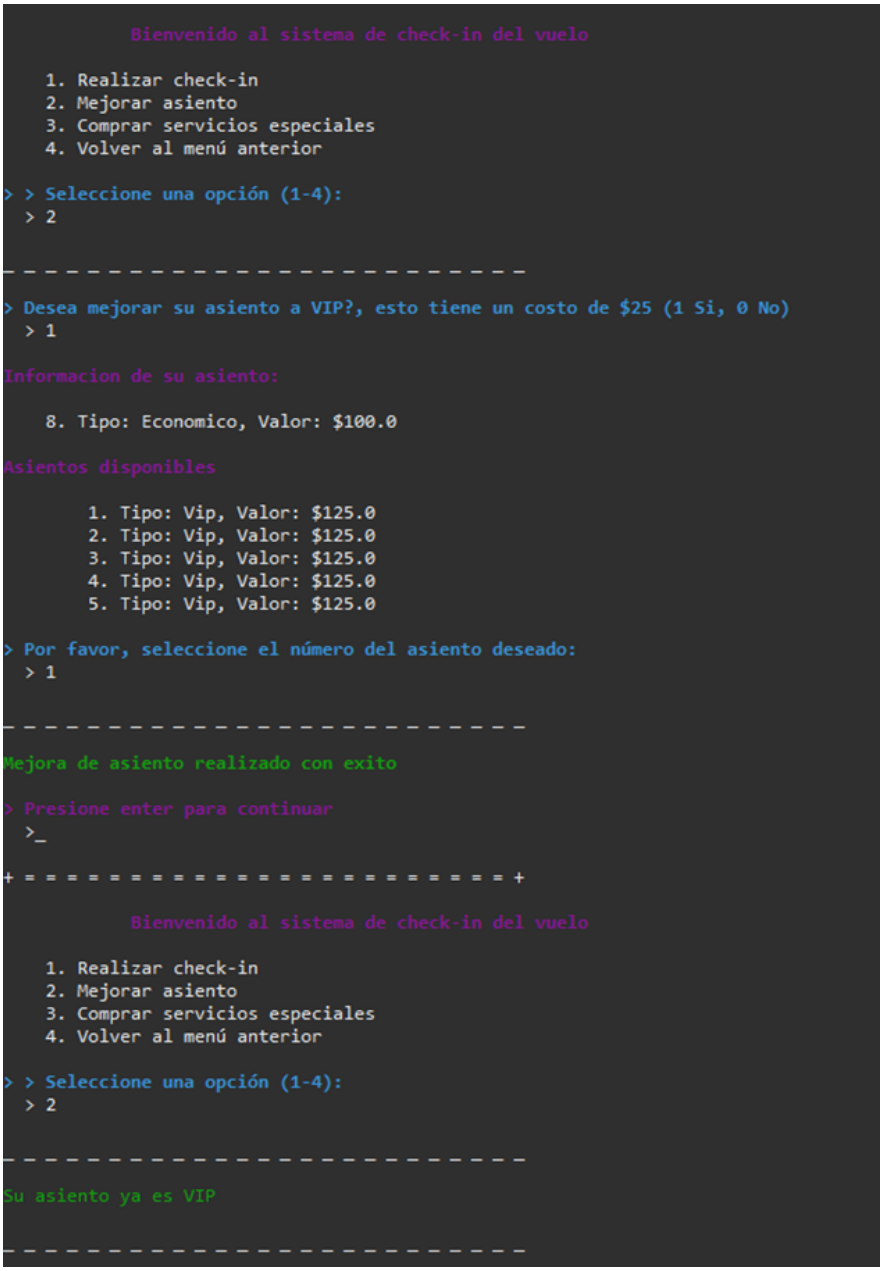
- **Realizar Check-in (Check-in Realizado):**

Si el check-in ya se ha realizado previamente para el vuelo, se muestra un mensaje que informa al usuario que el check-in ya ha sido completado.

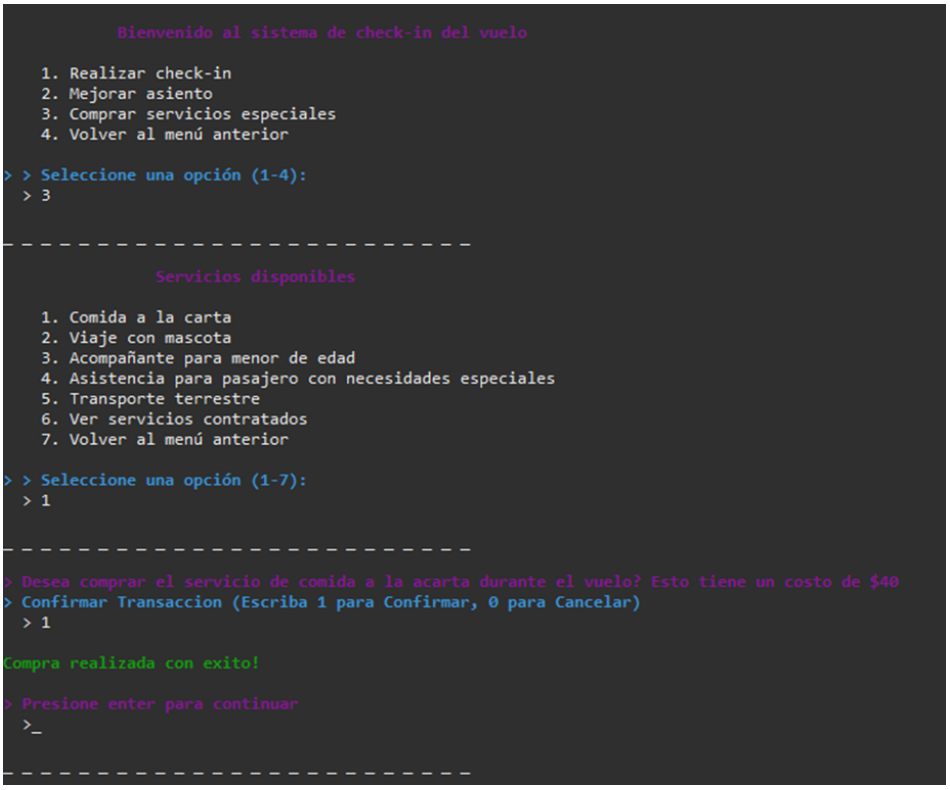
Capturas de los resultados



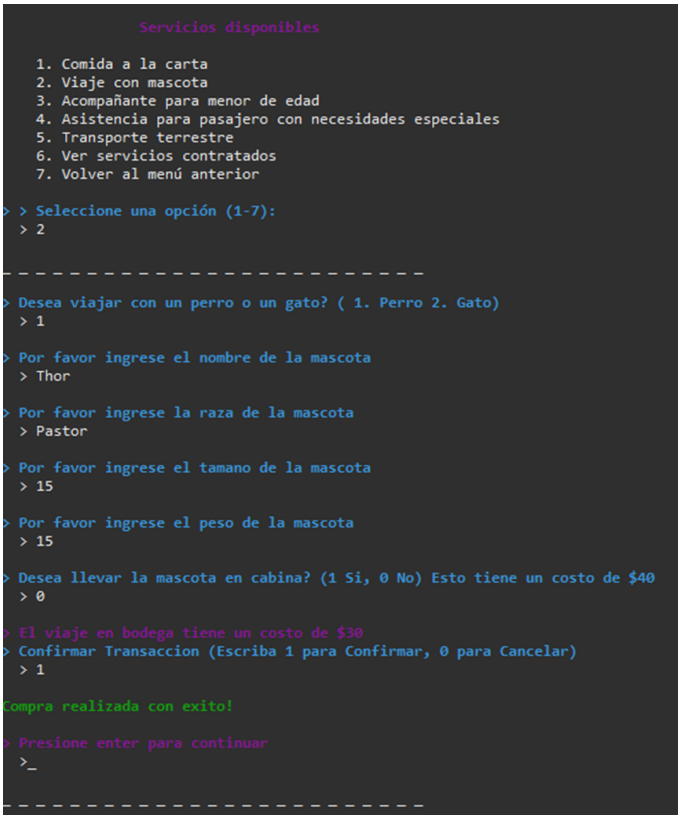
Se selecciona la opción Check-in, muestra los boletos que se tienen en el historial y se selecciona el deseado.



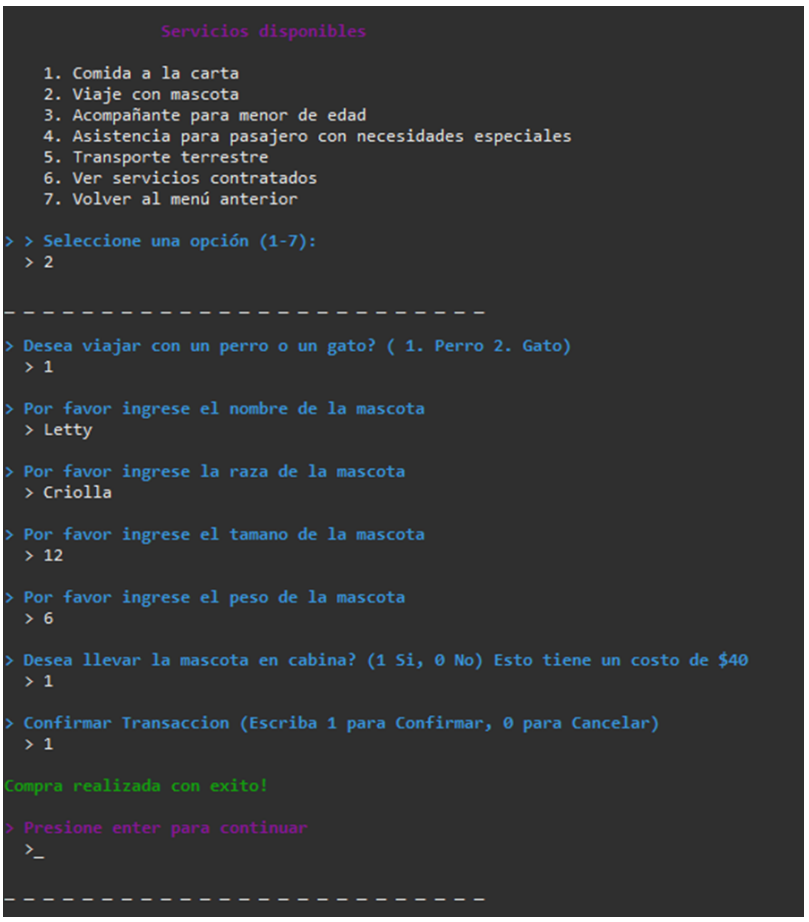
Se selecciona la opción Mejorar asiento, muestra la información del asiento actual y de los asientos disponibles para mejorar, de estos se selecciona el deseado y se realiza la transacción, si se vuelve a tratar de mejorar el asiento salta el mensaje que ya es VIP y por lo tanto no se puede mejorar más



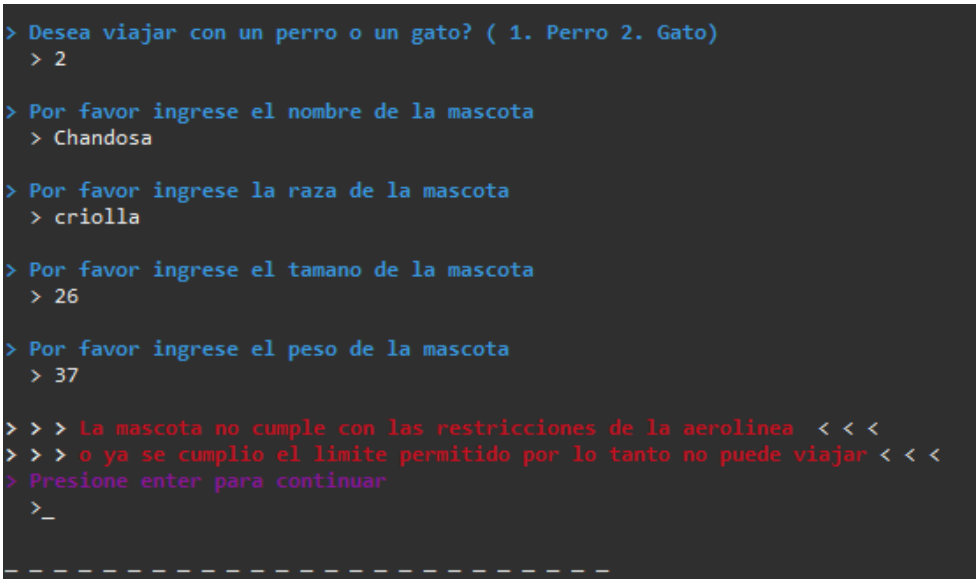
Se selecciona la opción de Comprar servicios especiales, se selecciona comida a la carta y se confirma la transacción.



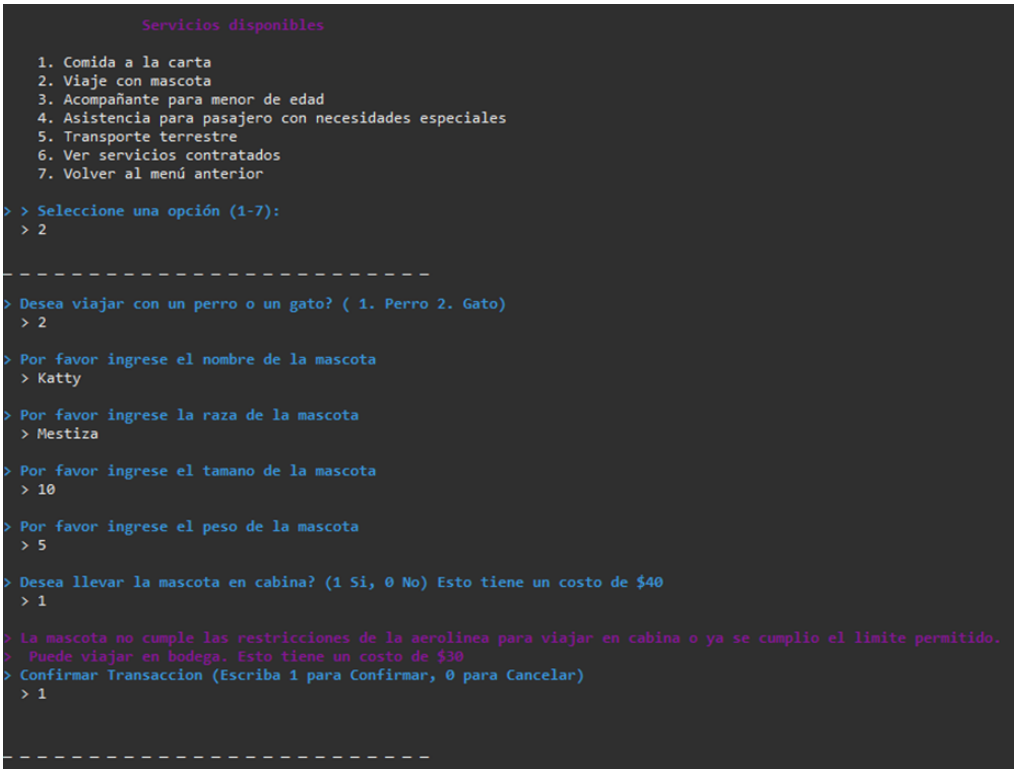
Se selecciona la opción Viajar con mascota se ingresan los datos del animal y se escoge que no se desea llevar en cabina, se verifica que cumpla con las restricciones y se confirma la compra.



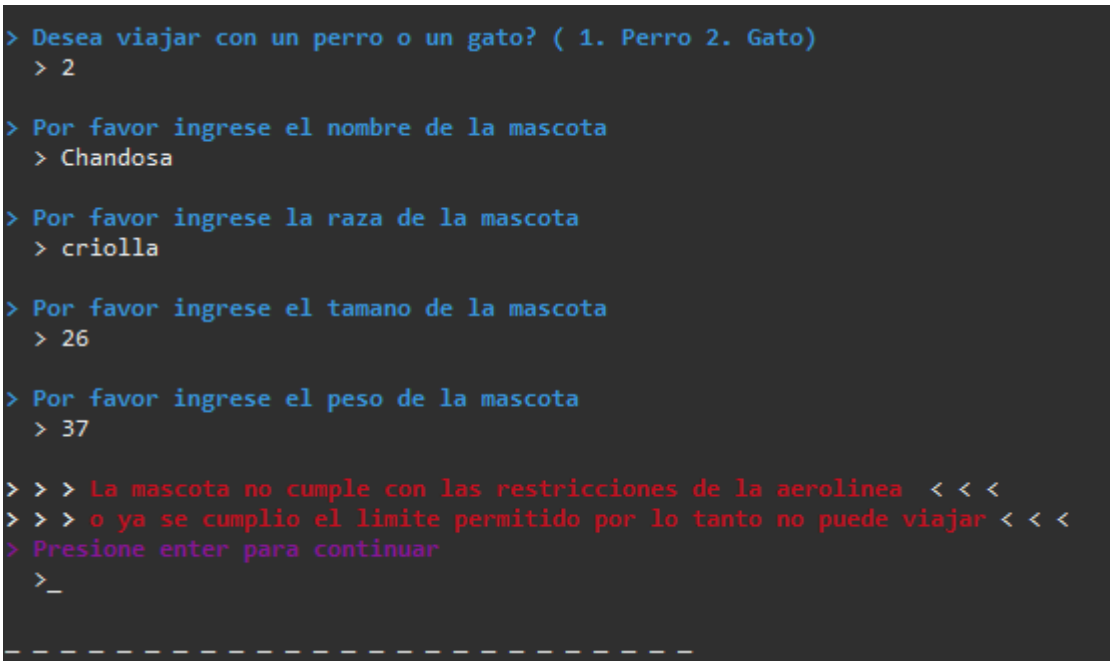
Se selecciona la opción Viajar con mascota se ingresan los datos del animal y se escoge que se desea llevar en cabina, se verifica que cumpla con las restricciones y se confirma la compra.



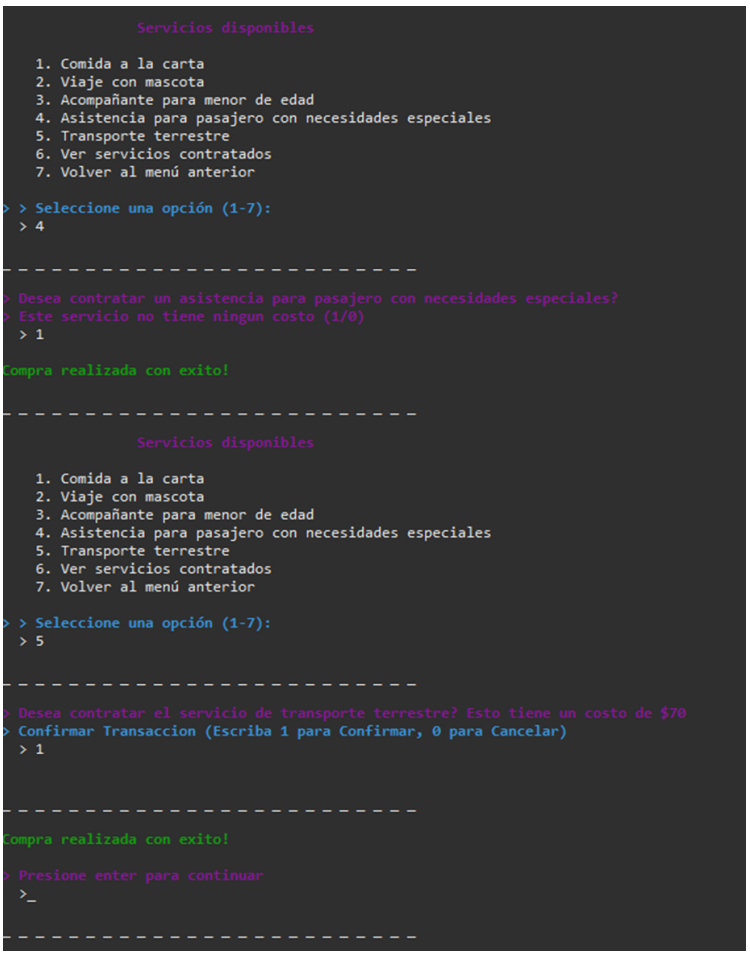
Se selecciona la opción Viajar con mascota se ingresan los datos del animal, pero como el peso excede el máximo permitido se cancela la operación.



Se selecciona la opción Viajar con mascota se ingresan los datos del animal y se escoge que se desea llevar en cabina, pero como ya hay una mascota en cabina no se puede por lo que se debe llevar en bodega, se verifica que cumpla con las restricciones y se confirma la compra.

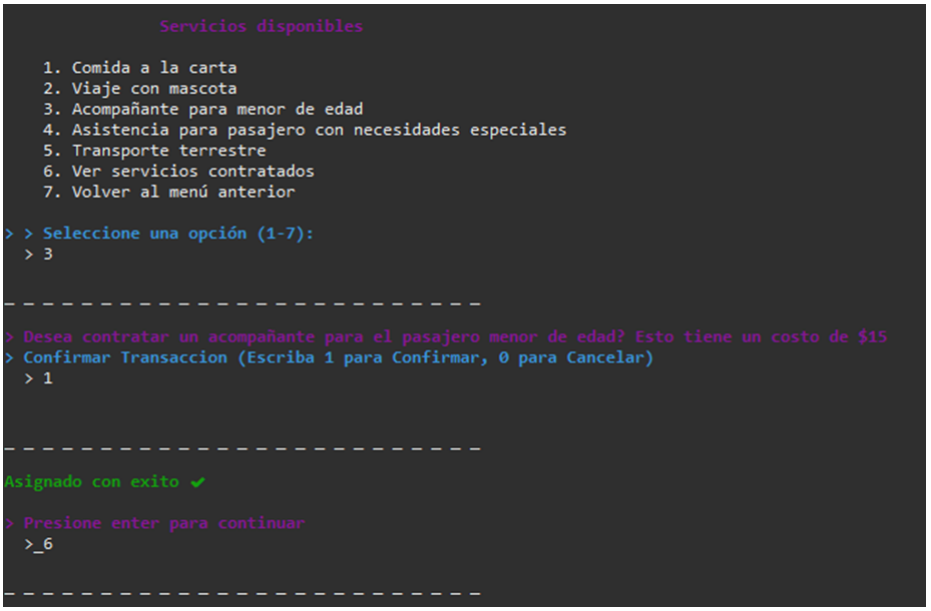


Se selecciona la opción Viajar con mascota se ingresan los datos del animal, pero como ya se ocupó el límite de 3 mascotas por viajes se cancela la operación .

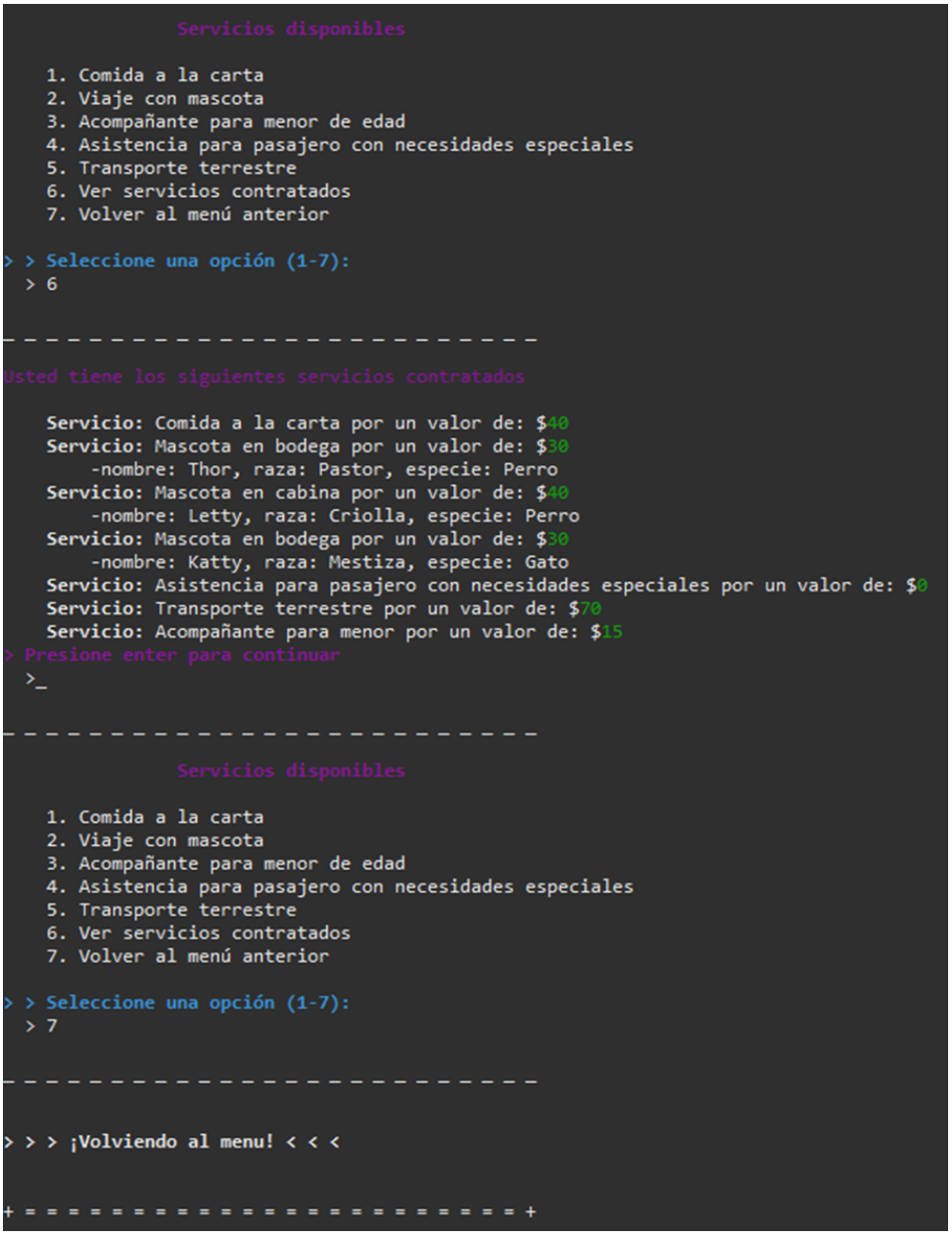


Se selecciona Asistencia para pasajero con necesidades especiales, se confirma la transacción y se realiza el cobro.

Se selecciona la opción de transporte terrestre, se confirma la transacción y se realiza el cobro.

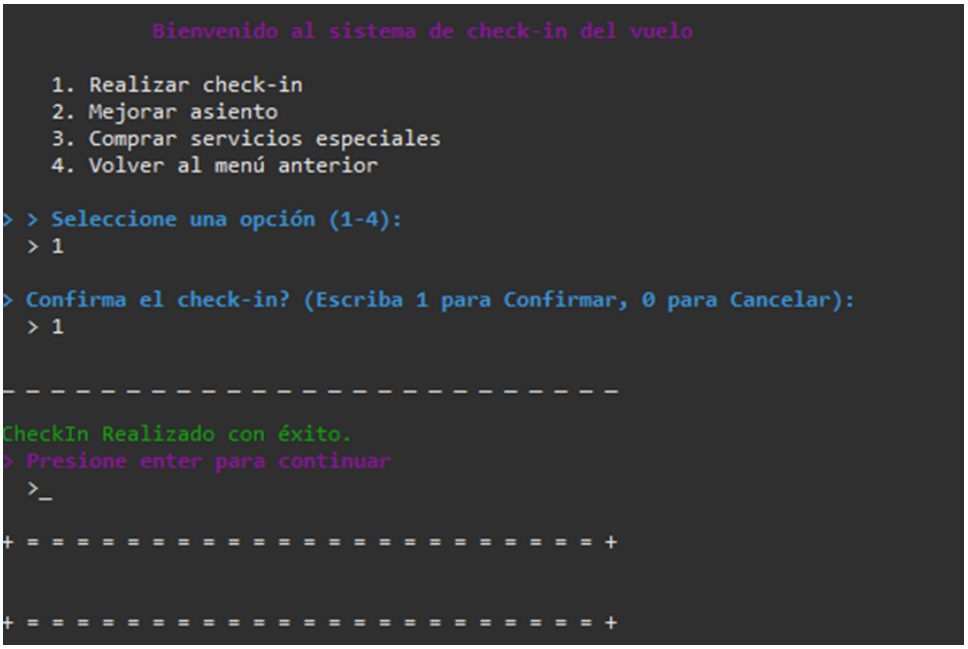


Se selecciona la opción de Acompañante para menor de edad, se confirma la transacción y se realiza el cobro.

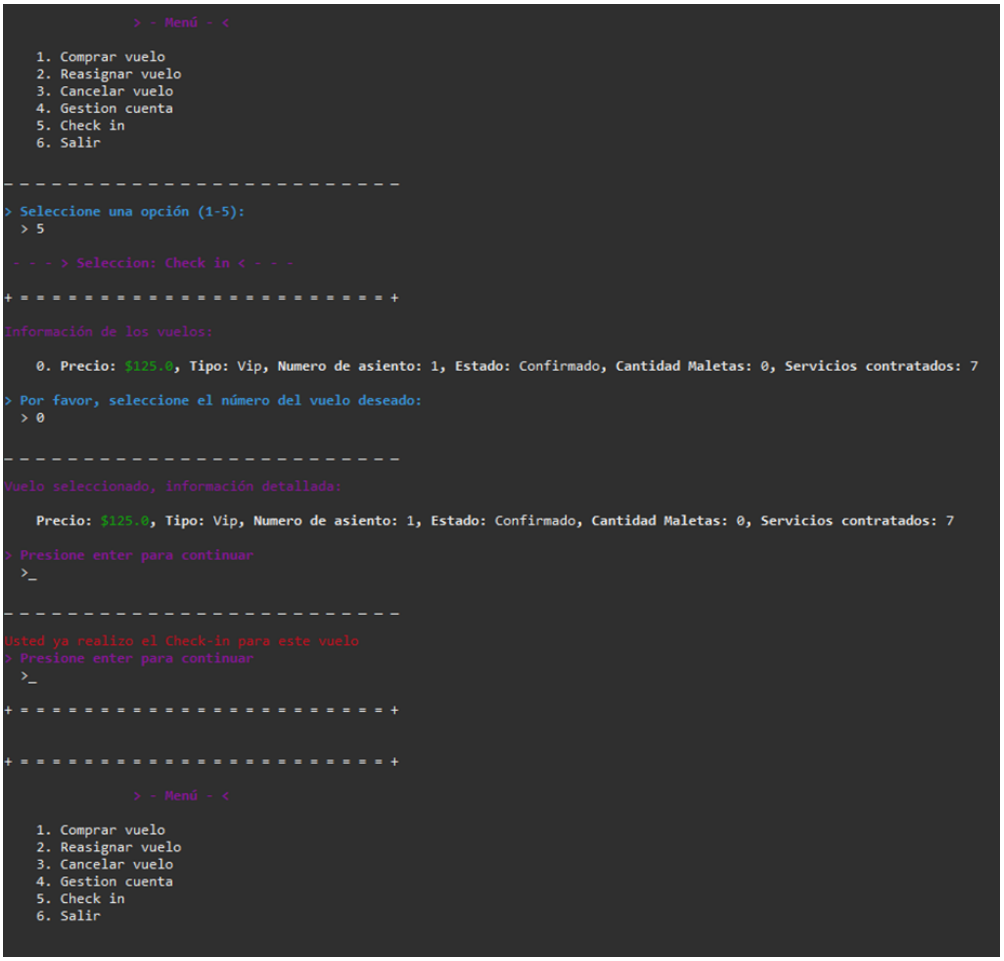


Se selecciona la opción Ver servicios contratados, se muestra la lista de los servicios y de las mascotas que se llevan en el viaje.

Se selecciona la opción volver al menú.



Se selecciona la opción Realizar check-in y se confirma el proceso.



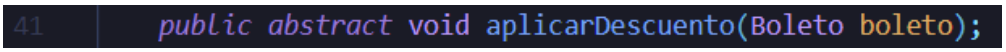
Se selecciona la opción Check-in y se selecciona el vuelo, pero como ya se realizó el check-in no se puede modificar ni agregar más servicios y retorna al menú principal.

IMPLEMENTACIÓN POO

Clases abstractas y métodos abstractos:

Clase abstracta Descuento, se encuentra dentro del paquete gestor Aplicación.Descuentos

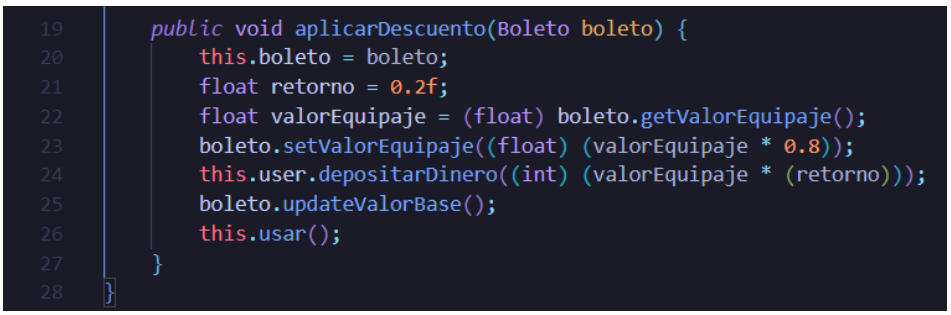
Tiene el método abstracto aplicarDescuento(Boleto boleto), que se encuentra en la línea 41.



Las clases descuentoMaleta, descuentoVuelo y upgradeAsiento heredan de Descuento

Cada subclase se ve obligada a implementar el método aplicarDescuento(Boleto boleto).

Implementación en subclase descuento Maleta, se encuentra en línea 19:



Implementación en subclase descuentoVuelo, se encuentra en línea 20:

```
20      public void aplicarDescuento(Boleto boleto) {
21          this.boleto = boleto;
22          float retorno = 0.2f;
23          float valorVuelo = (float) boleto.getValorInicial();
24          boleto.setValorInicial((float) (valorVuelo * 0.8));
25          this.user.depositarDinero((int) (valorVuelo * (retorno)));
26          boleto.updateValorBase();
27          this.usar();
28      }
```

Implementación en subclase upgradeVuelo, se encuentra en línea 20:

```
20      public void aplicarDescuento(Boleto boleto) {
21          this.boleto = boleto;
22          this.estado = "Usado";
23          this.usar();
24      }
```

Interfaces:

Interfaz RestriccionesMaleta, con las constantes static final peso, largo, ancho y alto (medidas fijas de la Maleta), la interfaz se encuentra en el paquete gestor Aplicación.Aerolínea. Tiene el método verificarRestricciones que se encuentra en la línea 9.

```
src > gestorAplicacion > Aerolinea > RestriccionesMaleta.java > RestriccionesMaleta
    You, 1 second ago | 2 authors (You and others)
1   package gestorAplicacion.Aerolinea;
2
    You, 1 second ago | 2 authors (Juan Carlos Largo and others)
3   public interface RestriccionesMaleta {
4       public int peso = 60;
5       public int largo = 250;
6       public int ancho = 250;
7       public int alto = 250;
8
9       public boolean verificarRestricciones();
10  }       You, 1 second ago • Uncommitted changes
```

La clase Maleta implementa la interfaz RestriccionesMaleta.

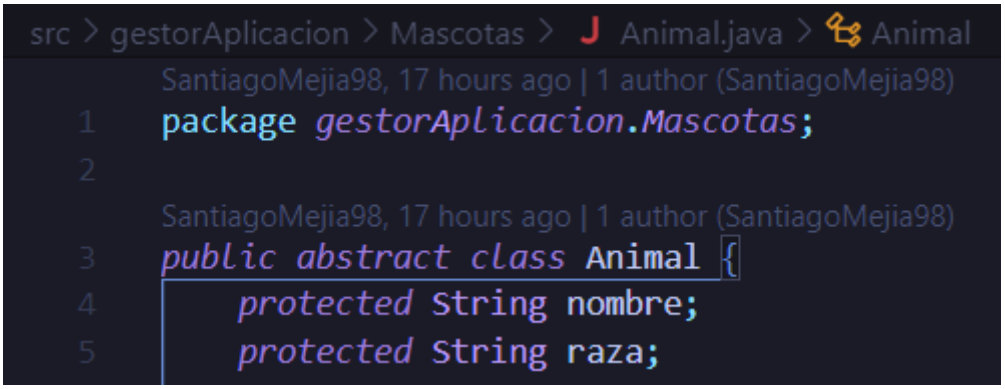
```
10  public class Maleta implements Serializable, RestriccionesMaleta {
```

En la línea 36, implementa el método abstracto verificarRestricciones:

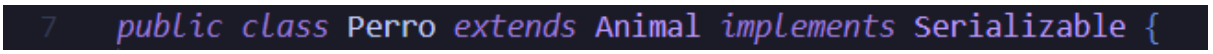
```
36      public boolean verificarRestricciones() {
37          if (this.peso <= RestriccionesMaleta.peso && this.ancho <= RestriccionesMaleta.ancho
38                  && this.alto <= RestriccionesMaleta.alto && this.largo <= RestriccionesMaleta.largo
39                  && this.peso <= RestriccionesMaleta.peso) {
40              return true;
41          } else {
42              return false;
43          }
44      }
```

Herencia:

La clase abstracta Animal tiene dos subclases que heredan de ella, se encuentra en el paquete gestor Aplicacion.Mascotas



La subclase Perro hereda los atributos e implementa los métodos de Animal:



La subclase Gato hereda los atributos e implementa los métodos de Animal:

Ligadura dinámica:

```
1435    private static void millasAsiento(Usuario user, Descuento descuento) {
1436
1437        Boleto boleto = selecBoleto(user);
1438        Asiento asiento = boleto.getAsiento();
1439        // se verifica que el asiento sea economico
1440        // si es vip ya no se puede mejorar
1441
1442        if (asiento.getTipo() == "Economico") {
1443
1444            // Hacer asiento vip
1445            ArrayList<Asiento> asientos = (boleto.getVuelo()).getAsientos();
1446
1447            printNegrita(colorTexto("Asientos disponibles", "morado"));
1448            salto();
1449            for (Asiento asientoTemp : asientos) {
1450                if (asientoTemp.getTipo().equals("Vip")) {
1451                    identacion(asientoTemp.getInfo(), 2);
1452                }
1453            }
1454
1455            salto();
1456            promptIn("Por favor, seleccione el número del asiento deseado: ");
1457            int indexAsiento = inputI();
1458
1459            // ... Cambiar y reasignar todo
1460            Asiento newAsiento = asientos.get(indexAsiento - 1);
1461            boleto.upgradeAsiento(asiento, newAsiento);
1462            descuento.aplicarDescuento(boleto);
```

Nótese que en este método millasAsiento de la clase App, se pasa un objeto de tipo descuento en la línea 1435, y proximamente en la línea 1462 se llama el metodo aplicarDescuento

```
public abstract class Descuento {
```

```
    public abstract void aplicarDescuento(Boleto boleto);
```

```
public class descuentoMaleta extends Descuento implements Serializable {
    private static final long serialVersionUID = 1L;

    public static int costoMillas = 20;
    public static int descuento = 20;

    public descuentoMaleta(Usuario user) {
        this.init(user);
        this.tipo = "Descuento de maleta";
    }

    public void aplicarDescuento(Boleto boleto) {
        this.boleto = boleto;
        float retorno = 0.2f;
        float valorEquipaje = (float) boleto.getValorEquipaje();
        boleto.setValorEquipaje((float) (valorEquipaje * 0.8));
        this.user.depositarDinero((int) (valorEquipaje * (retorno)));
        boleto.updateValorBase();
        this.usar();
    }
```

```
public class descuentoVuelo extends Descuento implements Serializable {
    private static final long serialVersionUID = 1L;

    public static int costoMillas = 20;
    public static int descuento = 20;

    public descuentoVuelo(Usuario user) {
        this.init(user);
        this.tipo = "Descuento Vuelo";
    }

    public void aplicarDescuento(Boleto boleto) {
        this.boleto = boleto;
        float retorno = 0.2f;
        float valorVuelo = (float) boleto.getValorInicial();
        boleto.setValorInicial((float) (valorVuelo * 0.8));
        this.user.depositarDinero((int) (valorVuelo * (retorno)));
        boleto.updateValorBase();
        this.usar();
    }
```

Según lo que se puede ver de las capturas anteriores, las únicas dos clases que heredan de la clase abstracta descuento son descuentovuelo y descuentoMaleta, y estas clases definen su propio método aplicar descuento

con esto como al meter el objeto en el método se le hace el casteo (Descuento) al objeto y al hacerle próximamente .aplicarDescuento() , se llama al método más especifico que serian dependiendo del caso, el método de descuento maleta o vuelo, demostrando la ligadura dinámica que se aplica aquí

Atributos y métodos de clase:

En la clase Boleto que se encuentra en el paquete gestor Aplicacion.Aerolínea hay un atributo de clase llamado cont en la línea 18:

```
18      private static int cont = 0;
```

En la clase Vuelo que encuentra en el paquete gestor Aplicación.Aerolínea hay un método de clase llamado generarHora() en la línea 49.

```
49      public static String generarHora() {
50          String[] horas = {
51                  "08:00 AM",
52                  "09:15 AM",
53                  "10:30 AM",
54                  "11:45 AM",
55                  "12:00 PM",
56                  "01:15 PM",
57                  "02:30 PM",
58                  "03:45 PM",
59                  "04:00 PM",
60                  "05:15 PM"
61          };
```

Uso de constante:

En la clase Perro que se encuentra en el paquete gestor Aplicacion.Mascotas se hace uso de las siguientes constantes:

```
15      private static final double PESO_MAXIMO_BODEGA = 30.0; // Peso máximo permitido en la bodega
16      private static final double TAMANO_MAXIMO_BODEGA = 30.0; // Tamaño máximo permitido en la bodega
17      private static final double PESO_MAXIMO_CABINA = 8.0; // Peso máximo permitido en la cabina
18      private static final double TAMANO_MAXIMO_CABINA = 20.0; // Tamaño máximo permitido en la cabina
```

Encapsulamiento:

La clase GestionUsuario que se encuentra en el paquete gestor Aplicación.Cuenta hace uso del 2 tipos de encapsulamiento en sus atributos (private y public):

```
9   public class GestionUsuario implements Serializable {
10
11      private static final long serialVersionUID = 1L;
12
13      public ArrayList<Usuario> Usuarios = new ArrayList<>();
14      public Usuario user = null;
15      public ArrayList<Maleta> inventarioMaletas = new ArrayList<>();
```

Los atributos de la clase abstractaDescuento utilizan el encapsulamiento protected:

```java
public abstract class Descuento {

    protected Boleto boleto;
    protected Usuario user;
    protected boolean usado = false;
    protected String tipo;
    protected String estado;
    protected Boolean guardado = false;
```

Sobrecarga de métodos:

En la clase Estática que se encuentra en el paquete uiMain se sobrecarga el método salto() sin parámetro y con parámetro int:

```java
    public static void salto() {
        System.out.print(s:"\n");
    }

    public static void salto(int n) {
        for (int i = 0; i < n; i++) {
            System.out.print(s:"\n");
        }
    }
```

Constructores:

Constructor de la clase Boleto, en la línea 68:

```java
    public Boleto(String origen, String destino, Usuario propietario, Vuelo vuelo) {
        cont++;
        this.origen = origen;
        this.destino = destino;
        this.user = propietario;
        this.vuelo = vuelo;
        this.pasajero = new Pasajero(propietario, this);
        this.id = cont;
    }
```

Constructor de la clase Vuelo, en la línea 35:

```java
public Vuelo(String origen, String destino, String aerolinea, String id, String tiempoSalida, String tiempoLlegada) {
    this.AEROLINEA = aerolinea;
    this.ID = id;
    this.horarioSalida = tiempoSalida;
    this.horarioLlegada = tiempoLlegada;
    this.DESTINO = destino;
    this.ORIGEN = origen;
}
```

Manejo de referencias this para desambiguar y this():

En la clase Maleta el método verificarRestricciones hace uso de this para evitar confusión entre los atributos de la propia clase y los de la interfaz que implementa.

```java
    public boolean verificarRestricciones() {
        if (this.peso <= RestriccionesMaleta.peso && this.ancho <= RestriccionesMaleta.ancho
                && this.alto <= RestriccionesMaleta.alto && this.largo <= RestriccionesMaleta.largo
                && this.peso <= RestriccionesMaleta.peso) {
            return true;
        } else {
            return false;
        }
    }
```

Implementación de un caso de enumeración:

Se implementó el enum Servicios Especiales que se encuentra en el paquete gestor Aplicacion.Aerolínea:

src > gestorAplicacion > Aerolinea > ServiciosEspeciales.java > ServiciosEspeciales

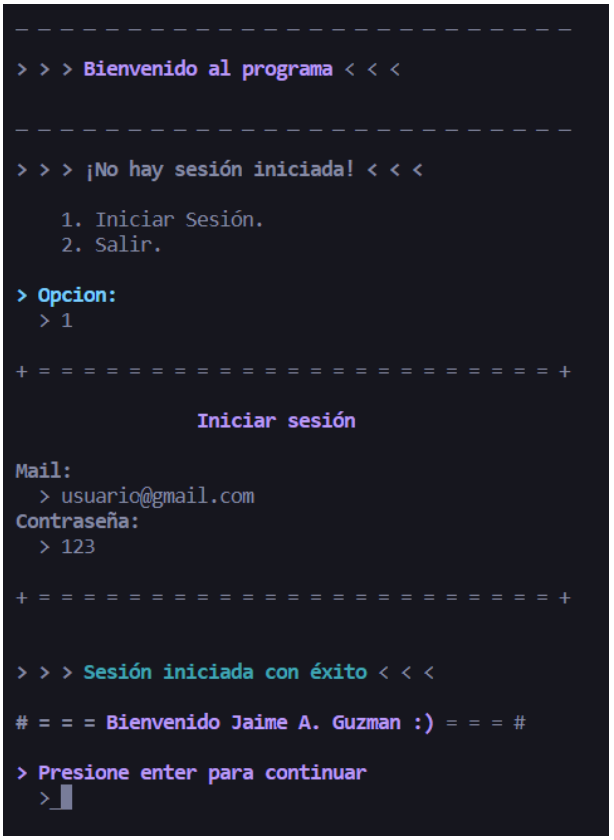
SantiagoMejia98, 6 hours ago | 2 authors (SantiagoMejia98 and others)

```java
package gestorAplicacion.Aerolinea;

public enum ServiciosEspeciales {
    COMIDA_A_LA_CARTA(servicio:"Comida a la carta", precio:40),
    MASCOTA_EN_CABINA(servicio:"Mascota en cabina", precio:40),
    MASCOTA_EN_BODEGA(servicio:"Mascota en bodega", precio:30),
    ACOMPANANTE_PARA_MENOR(servicio:"Acompañante para menor", precio:15),
    ASISTENCIA_NECESIDADES_ESPECIALES(servicio:"Asistencia para pasajero con necesidades especiales", precio:0),
    TRANSPORTE_TERRESTRE(servicio:"Transporte terrestre", precio:70);

    private String servicio;
    private int precio;

    ServiciosEspeciales(String servicio, int precio) {
        this.servicio = servicio;
        this.precio = precio;
    }
```

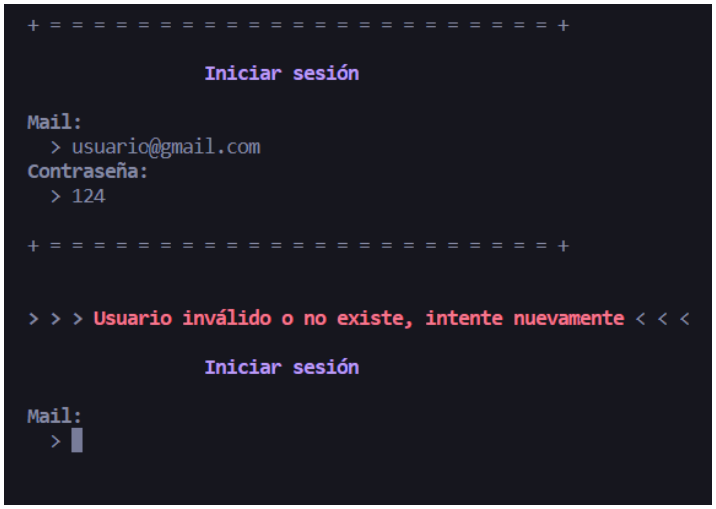
MANUAL DE USUARIO

Inicio de sesión:

- Cuando se corre el programa se da la bienvenida y se requiere iniciar sesión, para esto se escribe 1 en el espacio de Opción, se da enter.
- Se pide el Mail, el cual es usuario@gmail.com y la Contraseña es **123**, se da enter.
- La sesión se inició con éxito, se saluda al nombre de usuario y se pide presionar enter para continuar, se da enter.

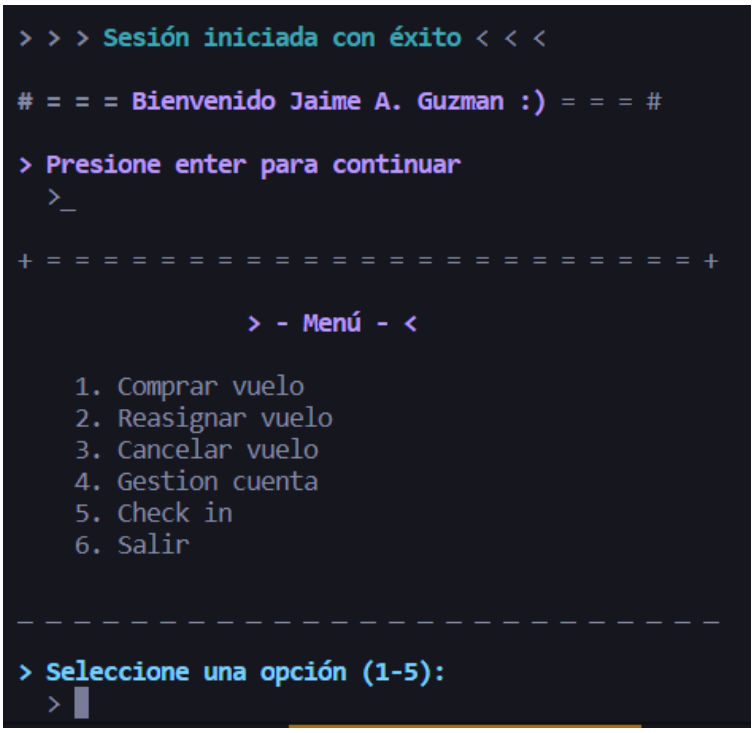


- Si se ponen los datos incorrectos se imprimirá el mensaje en rojo “Usuario inválido o no existe, intente nuevamente”, por lo que debe volver a ingresar el Mail y Contraseña de manera correcta.



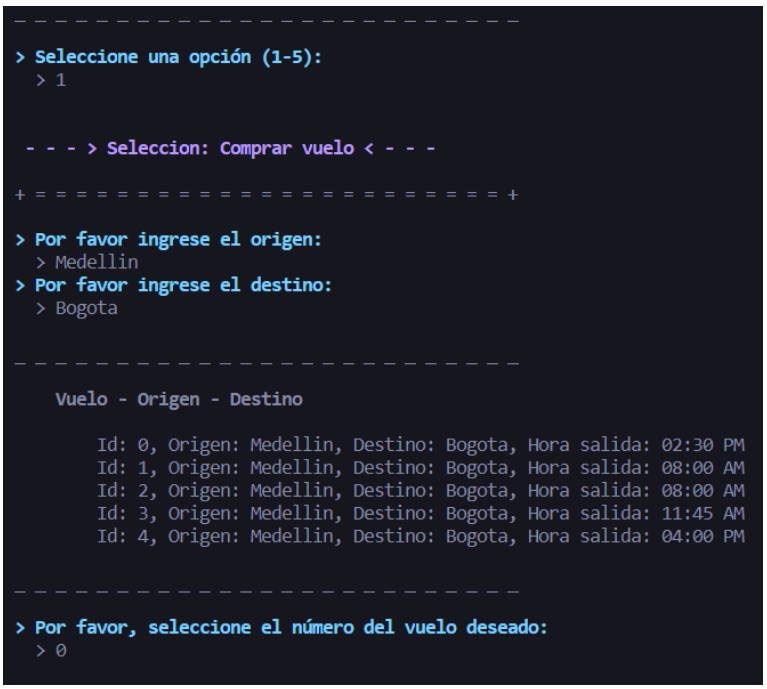
Opciones del Menú:

- Después de presionar enter para continuar, se despliegan las opciones del Menú, escoja la funcionalidad que desea realizar escribiendo el número asignado en la parte de >Seleccione una opción (1-5).



Opción 1. Comprar vuelo:

- Si se escribe 1, se selecciona la funcionalidad Comprar vuelo, se pide ingresar la ciudad de origen y de destino, escribir en su espacio correspondiente.
- Luego se despliega las opciones de vuelos cada uno con su Id, para seleccionar el vuelo deseado debe escribir el número de Id del vuelo que quiere (en este caso se seleccionó el vuelo con Id: 0 con Hora salida: 2:30 PM).



- Luego se despliega las opciones de vuelos cada uno con su Id, para seleccionar el vuelo deseado debe escribir el número de Id del vuelo que quiere (en este caso se seleccionó el vuelo con Id: 0 con Hora salida: 2:30 PM).

- Al seleccionar el vuelo, se muestran los asientos disponibles que son de dos tipos Vip y Económico; escriba el número de asiento que desea comprar (en este caso se escogió el asiento número 1).
- Se imprime el mensaje de previsualización del precio con el valor del asiento escogido, luego se pide presionar enter para continuar.
- Seguidamente, se pregunta si se desea añadir equipaje, escriba 1 para Sí ó 0 para No.