

AIR ROUTE FINDER

Manual del programador

Apablaza Karen | Programación Orientada a Objetos | 11-03-2022

Contenido

[Introducción 2](#_Toc97892754)

[Estructura, Patrones de diseño 2](#_Toc97892755)

[Paquete Lógica 4](#_Toc97892756)

[Clase logicaAeropuerto y LogicaVuelo 4](#_Toc97892757)

[Clase LogicaTipoBusqueda 4](#_Toc97892758)

[Paquete dato.conexion 6](#_Toc97892759)

[Paquete dato.vo 6](#_Toc97892760)

[Paquete dato.dao 7](#_Toc97892761)

[Paquete vista 7](#_Toc97892762)

[Clase panelMenu 7](#_Toc97892763)

[Clase panelbusquedaeconomico 8](#_Toc97892764)

[Clase panelbusquedaMenosHoras 8](#_Toc97892765)

[Clase panelbusquedamenosescalas 8](#_Toc97892766)

[Clase panelcargaraeropuertos 9](#_Toc97892767)

[Clase panelcargarvuelos 9](#_Toc97892768)

[Paquete controlador 10](#_Toc97892769)

[Clase aplicación 10](#_Toc97892770)

[Clase coordinador 10](#_Toc97892771)

[Diagrama de Clases 10](#_Toc97892772)

[Lote de Pruebas 11](#_Toc97892773)

[Conclusiones: Errores y futuras ampliaciones para la aplicación 13](#_Toc97892774)

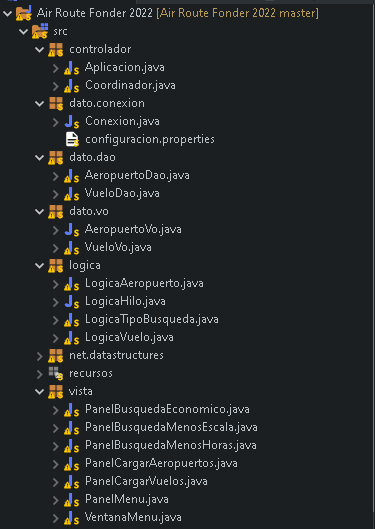
# Introducción

Esta aplicación (Air Route Finder) permite buscar un vuelo disponible entre dos localidades para una fecha determinada, ingresados por el usuario. Además, cuenta con distintas opciones de búsqueda. Al presionar el botón “buscar vuelo” se mostrará primero la opción elegida y también los otros dos tipos de búsqueda ya realizados, para esto se llevó a cabo la implementación de hilos.

Entre los diferentes tipos de búsqueda se encuentran:

* Más económico: Devuelve el vuelo con el precio más bajo sin importar el tiempo ni las escalas.
* Menos horas de vuelo: Devuelve el vuelo con el menor tiempo de viaje hacia su destino.
* Menos escalas: Devuelve el vuelo que hace menos paradas durante su trayect0

## Estructura, Patrones de diseño

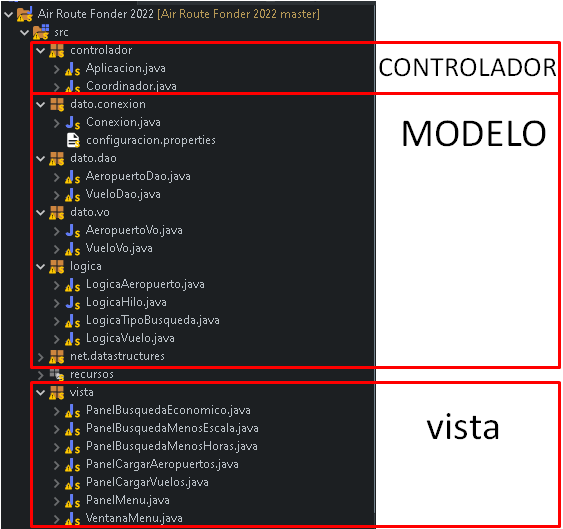


Para esta aplicación utilice 4 patrones: MVC, DAO, VO, SINGLETON.

Se aplica el MVC permitiendo independizar la lógica y la parte visual de la aplicación usando para eso un controlador que administra los procesos sirviendo como puente entre estos.

Para la Base de datos se usa MYSQL donde se aplica el patrón singleton, el Scritp se adjunta junto con el documento, donde se creará la base de datos y las tablas Vuelo y Aeropuerto con el cual se realizarán todas las operaciones CRUD (agregar, modificar, eliminar, buscar).

La agrupación de clases de la aplicación es la siguiente:



Como vemos tenemos la parte visual compuesta por paneles, la lógica se establece en el modelo el cual se compone por los paquetes “lógica” y “dato. Conexión” y aplique los patrones DAO y VO por medio de los paquetes “dato.dao” y “dato.vo” y por último el controlador se define en la clase Coordinador, encargada de establecer todas las relaciones de la aplicación, la clase Aplicación prepara las instancias iniciales del sistema y posteriormente lo ejecuta.

# Paquete Lógica

## Clase logicaaeropuerto y LogicaVuelo

Estas clases permiten realizar las operaciones asociadas a la carga de datos, desde ella realizamos las validaciones y llamamos a las operaciones CRUD de la aplicación.

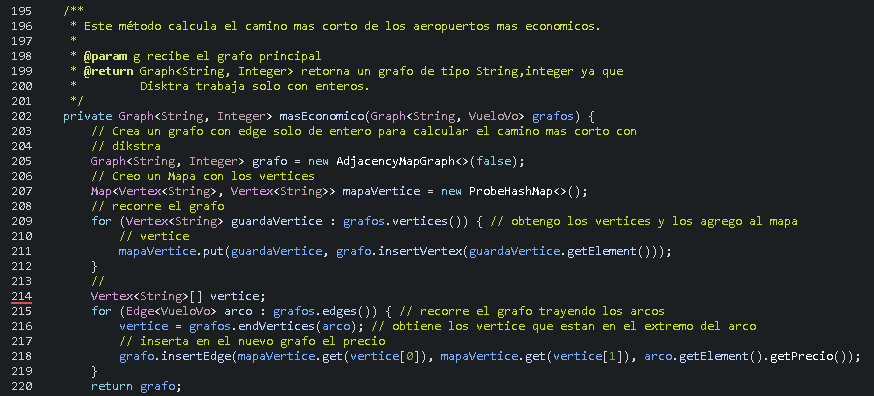
Aquí será donde se creen métodos para dichos procesos, por ejemplo, el método **validarRegistroVuelo** que se encuentra en la clase logicaVuelo,determina si los datos son correctos y permite registrar el vuelo en el Dao.

## CLASE LOGICATIPOBUSQUEDA

Esta clase será la encarga de realizar los caminos más cortos con el algoritmo de DIKSTRA, devolviendo así, una Lista de los tipos de búsqueda con todos aquellos vértices que lo conforman.

Se crearán dos hilos, uno para el económico y otro para las menos horas, lo bueno de hacer esto es que calculará los 2 tipos de búsqueda al mismo tiempo, por lo tanto, el usuario verá primero la opción que eligió y también las otras dos opciones.

Tipo de BÚSQUEDA: mÁS ECONÓMICO



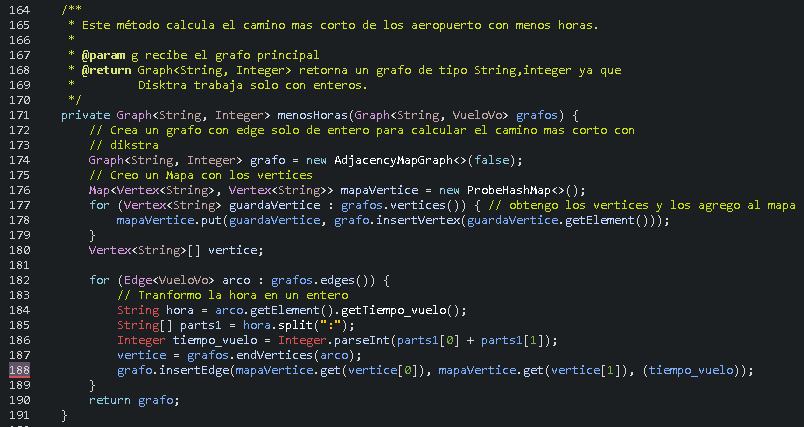
En este método se crea un grafo no dirigido de tipo <String, Integer>, ¿porque Integer y no VueloVo? Dikstra solo usa para calcular el camino más corto el tipo <Object, Integer>.

En la línea 207 se crea un mapa que almacenara los vértices del nuevo grafo que se acaba de crear. En las líneas 209 a la 212 se recorre el grafo recibido por parámetro y se va agregando al mapa el vértice del grafo como clave, y como valor el vértice del nuevo grafo.

En la línea 113 a la 118 se recorre los arcos del grafo que se recibió por parámetro y se busca los vértices (línea 216), luego se inserta en el nuevo grafo los vértices y el precio.

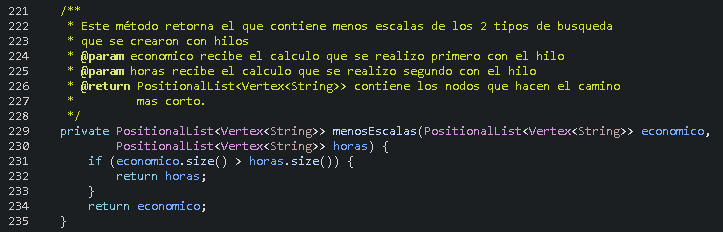
Tipo de Búsqueda: menos horas de vuelo

Este método es muy similar al masEconómico pero con la diferencia de que, en lugar de insertar solo el precio como peso del arco, se inserta las horas de vuelo (como enteros).

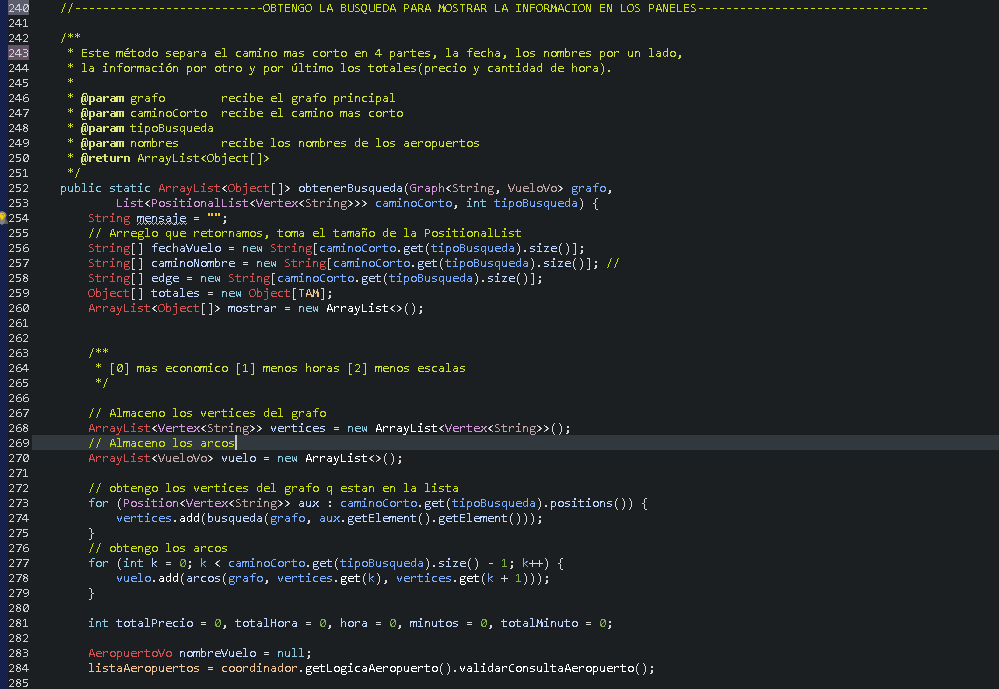


tiPO DE BÚSQUEDA: MENOS ESCALAS

Este método recibe por parámetro el económico y menos horas que se realizaron con hilos y pregunta cuál de las dos listas es de menor tamaño.



Mostrar datos



En este método que se encuentra en la logicaTipoBusqueda, se obtiene los distintos datos que se necesitan mostrar en pantalla.

Estos datos son:

* La fecha del vuelo
* Los nombres de los vuelos
* Los pesos de los arcos
* Los totales (Precio y cantidad de horas)

# Paquete dato.conexion

En esta clase se utiliza un archivo de propiedades para obtener los datos de la conexión a la base de datos, para que no se tenga que utilizar el código si cambian los datos de acceso. En esta clase se utilizó el patrón Singleton para obtener la conexión a la base de datos y cuando ya se finalice el programa se cierra la base de datos.

# Paquete dato.vo

En este paquete se encuentran las clases “AeropuertoVo” y “VueloVo” se aplicó el patrón Value Object o VO (Anteriormente conocidas como DTO Data Transfer Objects), en él se representa las entidades (Tablas) de la base de datos, la tabla **Aeropuerto** tiene los campos **id**, **abreviación, nombre** y la tabla **Vuelo** tiene los campos **id,** **numero de vuelo,** **fecha, aeropuerto origen, aeropuerto destino, precio, tiempo de vuelo, demora** entonces nuestra clase Vo tendrá estos mismos atributos y de esta manera se transporta un objeto aeropuerto o vuelo con todos estos valores por medio de los métodos **set** y **get** de cada atributo.

Este patrón me facilito mucho el transporte de la información, evitando que se envíen gran cantidad de parámetros a un método cuando quiero hacer un registro o actualización, también en caso de que se modifique la tabla de la BD, nuestra clase deberá ser modificada así como los métodos que obtienen la información, mas no los métodos que la transportan.

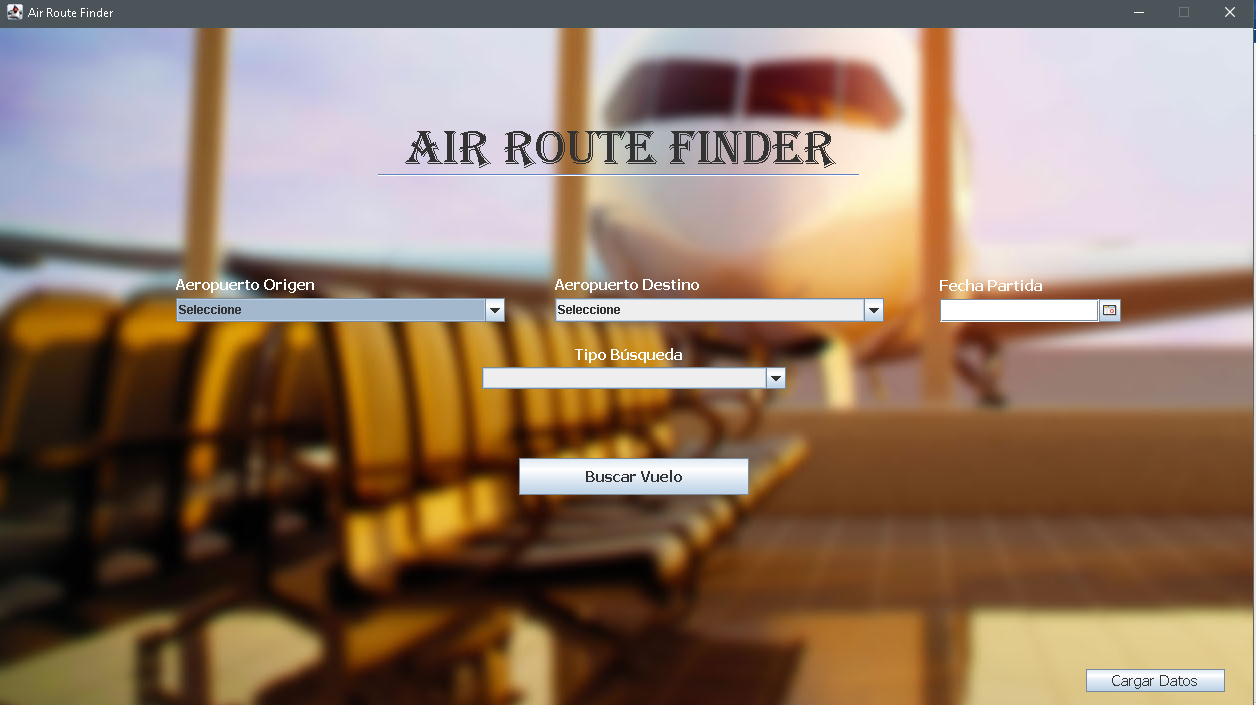
# Paquete dato.dao

En este paquete se encuentra las clases “AeropuertoDao“ y “VueloDao”, se aplicó el patrón Data Access Object o Dao, esté patrón consiste en centralizar los procesos de acceso a la base de datos evitando inconsistencias y posibles problemas cuando esto se realiza a lo largo de la aplicación. Con este patrón independizamos la lógica de negocio de la lógica de acceso a datos obteniendo mayor organización y flexibilidad en la aplicación.

# Paquete vista

En este paquete se crea la parte visual de la aplicación, en las vistas contiene paneles o interfaces graficas de usuario, mediante las cuales represento todo el modelo permitiendo la interacción entre la aplicación y el cliente.

## Clase panelMenu



En esta clase representa el panel inicial, posee 3 comboBox, dos de aeropuertos y uno de tipos de búsqueda, un dateChooser donde selecciona la fecha y dos botones que darán inicio a los eventos principales de la aplicación.

Se comunica con la lógica mediante la clase coordinador y desde esta se cargan los otros paneles de la aplicación.

Cuando se presiona el botón buscar vuelo, se busca primero todos los vuelos que hallan para esa fecha ingresada, si existen vuelos por medio del coordinador se accede al método camino más corto que se encuentra en la clase LogicaTipoBusqueda, ahí se crea dos hilos para que realice los dos tipos de búsqueda económico y menos hora al mismo tiempo, por último, se añade el de menos escalas a la lista de caminos.

Una vez calculado los caminos de búsqueda se muestra a continuación las siguientes pantallas:

## clase panelbusquedaeconomico

En esta clase se mostrará los datos del camino más corto de la opción de tipo de búsqueda “Económico”, recibirá como parámetro la lista del hilo uno.

## clase panelbusquedaMenosHoras

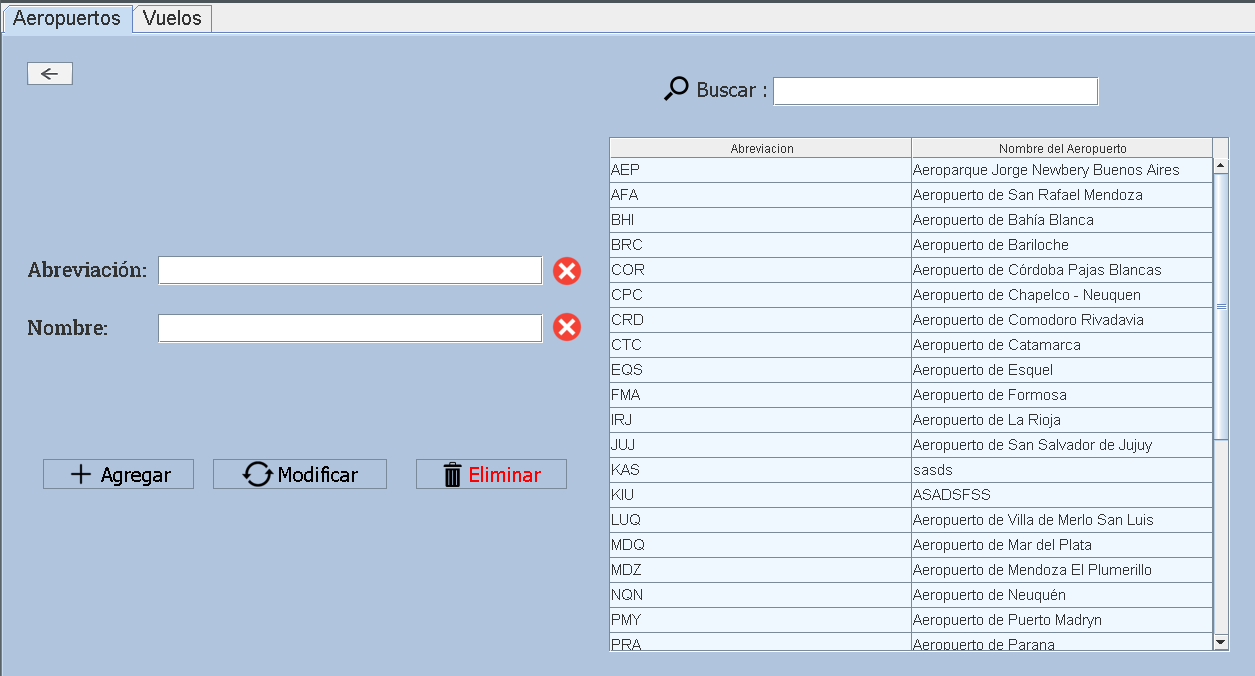
En esta clase se mostrará los datos del camino más corto de la opción de tipo de búsqueda “Menos Horas”, recibirá como parámetro la lista del hilo dos.

## clase panelbusquedamenosescalas

En esta clase se mostrará los datos del camino más corto de la opción de tipo de búsqueda “Menos escalas”, recibirá como parámetro la lista tres donde se calculó cuál de las dos listas (económico, menos horas) es de menor tamaño.

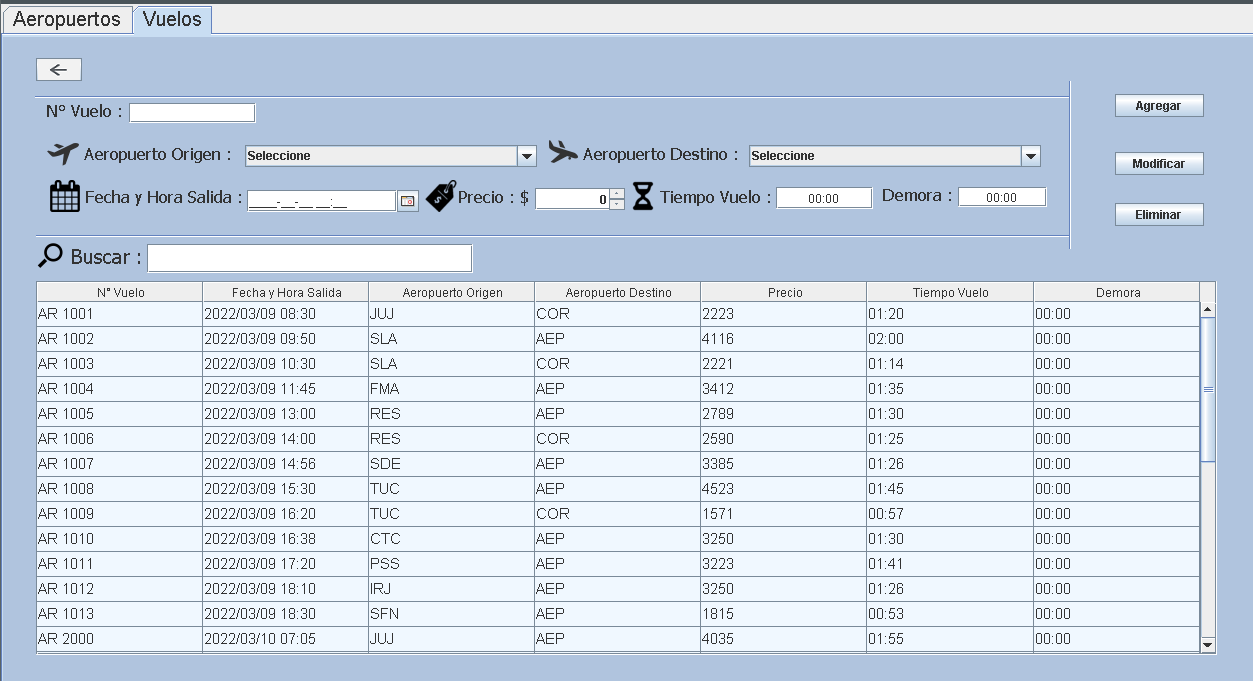
En el PanelMenu cuando se presiona el botón Cargar Datos se mostrará los siguientes paneles:

## clase panelcargaraeropuertos



En esta clase permite el ingreso de información a la Base de Datos, en ella se evidencian todos los campos de la tabla Aeropuerto y presenta un medio gráfico para las clases **AeropuertoVo** y **AeropuertoDao**, al ejecutar el evento de agregar, modificar, eliminar y buscar se establece comunicación entre el coordinador y la clase LogicaAeropuerto donde se encuentran todas las validaciones y casos para el ingreso de información.

## clase panelcargarvuelos



En esta clase también es el ingreso de información a la Base de Datos, en ella se evidencian todos los campos de la tabla Vuelo y presenta un medio gráfico para las clases **VueloVo** y **VueloDao**, al ejecutar el evento de agregar, modificar, eliminar y buscar se establece comunicación entre el coordinador y la clase LogicaVuelo donde se encuentran todas las validaciones y casos para el ingreso de información.

# Paquete controlador

Esta parte del patrón MVC es la que define la lógica de administración de la aplicación, establece la conexión entre la vista y el modelo.

## Clase aplicación

Esta clase contiene el método main que ejecuta la aplicación, el método hace un llamado al método **iniciar** el cual crea las instancias de las clases paneles y las clases lógica estableciendo las relaciones con la clase **Coordinador.**

A cada instancia de las clases se les envía una instancia de la clase Coordinador, y a la instancia de Coordinador se le envía cada instancia de las clases, esto por medio de los métodos set y get estableciendo las relaciones necesarias y por último se usa la instancia de la clase VentanaMenu para cargarla en pantalla.

## clase coordinador

Esta clase contiene toda la lógica de relaciones en el aplicativo, es el puente entre el modelo y las vistas.

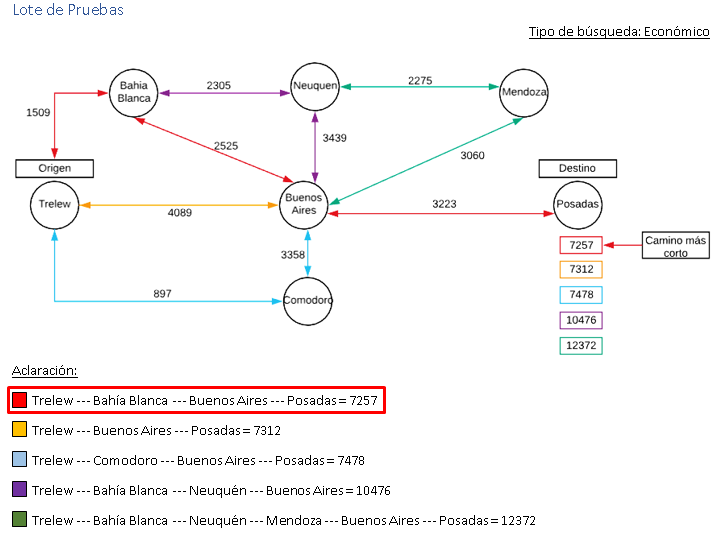
Puede contener instancias locales tanto de clases de la vista como de clases de la lógica, estas instancias tienen sus respectivos métodos set y get permitiendo el flujo de llamados de la aplicación.

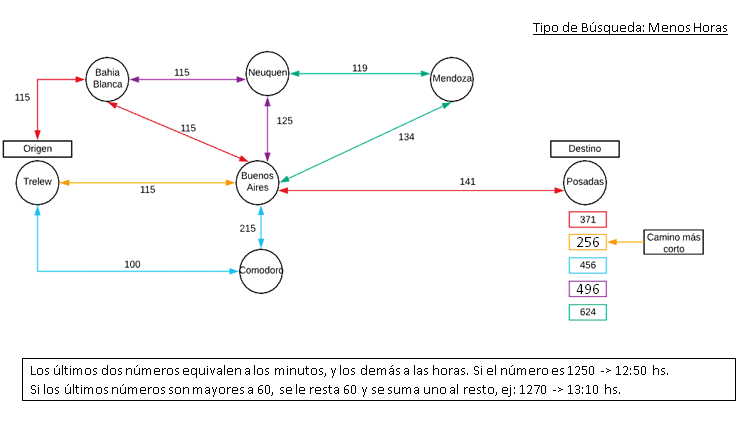
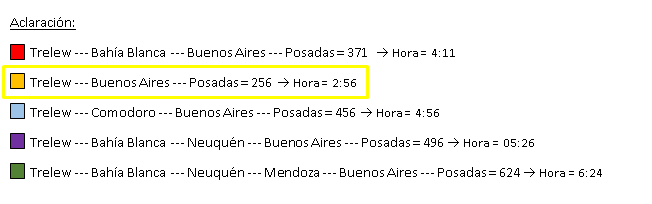
Cuando se desea registrar un Aeropuerto desde la clase **PanelCargaAeropuertos** se hace un llamado al método **registrarAeropuerto()** de la clase **Coordinadoor,** posteriormente esta clase llama al método **validarRegistroAeropuerto()** de la clase **logicaAeropuerto** y esta a su vez realiza las validaciones correspondientes para determinar si se llama o no al método **registrarAeropuerto()** de la clase **AeropuertoDao().**

# Diagrama de Clases

Se adjuntará una imagen junto con el documento para que se vea mejor, ya que cuando lo pego aquí se pone las letras chiquitas y no se entienden.

# Lote de Pruebas

Se mostrará a continuación, el cálculo que hacen los hilos, cuando se ejecutan los tipos de búsquedas:



# Conclusiones: Errores y futuras ampliaciones para la aplicación

* Uno de los cambios que se podrían hacer es que a la hora de buscar un vuelo en una determinada fecha y no halla vuelos, mostrar en pantalla las fechas que existe este vuelo ingresado por el usuario.
* Esta aplicación por ahora solo permite cargar un vuelo aeropuerto origen y aeropuerto destino en una fecha, a la hora de cargar otro mismo para la misma fecha, pero distinta hora, se rompe ya que en el método del camino corto la comparación de los nodos es la misma, no se tiene en cuenta la fecha aún.
* Se podría realizar otro método de tipo de búsqueda, buscando el de menos demora.
* Se podría agregar seguridad a la hora de cargar Datos (Aeropuertos, vuelos) ya que eso, lo haría un administrador, por ende, no cualquiera podría ingresar, ya que estaría protegido con contraseña.

# Herramientas

Se adjunta enlace de git, donde en el proceso de la creación de la aplicación fui guardando los pasos, para no perder el trabajo por fallas en mi compu.

<https://github.com/Karen-96/Air-Route-Fonder-2022>