

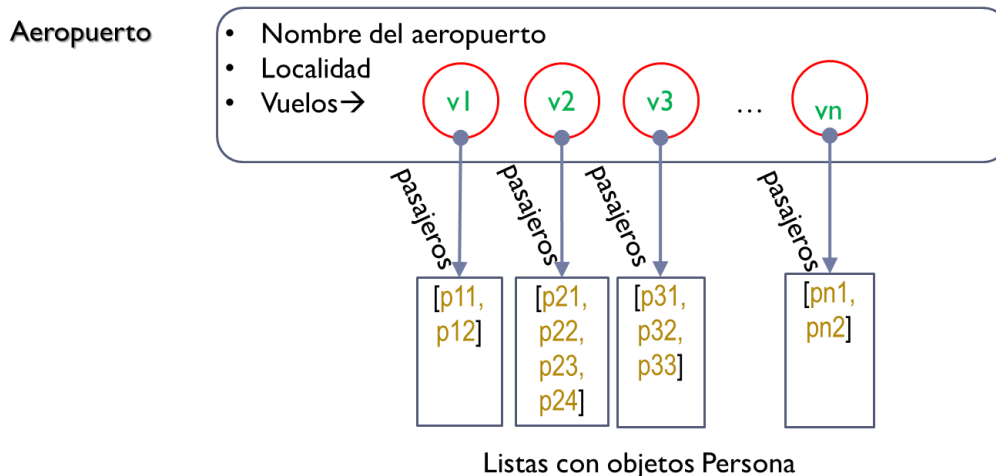


## EL TIPO PERSONA, VUELO y AEROPUERTO

### Consideraciones Iniciales

- Este es un enunciado que se irá incrementando a medida que avancemos en la teoría del 2º cuatrimestre del curso.
- Es **MUY IMPORTANTE** llevarlo al día, porque **numerosas cuestiones** avanzarán a partir de lo realizado en las anteriores, bien porque se ha resuelto en clase o se han dejado como tarea para casa.

### Esquema gráfico de la estructura de datos del proyecto



### EJERCICIOS:

1. Importe el proyecto: T03\_Aeropuerto. (ver documento “ExportareImportarUnProyecto”)
2. Vea que en el proyecto ya contiene los paquetes `fp.aeropuerto` y `fp.aeropuerto.test`
3. En `fp.aeropuerto` ya tiene implementado un tipo **Persona** con las siguientes características:

### PERSONA

#### Propiedades:

- `String dni`. Consultable.
- `String nombre`. Consultable.
- `String apellidos`. Consultable.

#### Métodos Constructores:

- Un constructor a partir de cada uno de los atributos (constructor canónico).
- Un constructor con apellidos y nombre en el que el dni quede con la cadena vacía (“”).

#### Restricciones:

- Ningún atributo puede ser nulo.

#### El criterio de igualdad:

- Dos personas son iguales si tienen el mismo dni, apellidos y nombre.

#### El criterio de ordenación:

- Por dni y, en caso de igualdad desempatan por los apellidos y igualdad de apellidos por el nombre.

#### La representación como cadena:



- Todos los atributos básicos con el nombre del tipo y separados por comas.

4. En `fp.aeropuerto,test` ya tiene implementado una clase ***TestPersona01*** que si la ejecuta debe obtener lo siguiente:

```
Test constructor-1 con todos los parámetros
Persona[dni=12345678A, nombre=Manolito, apellidos=Gafotas]
Construido correctamente!!

Test constructor-1 con todos los parámetros nulos
Ha habido un error al construir la persona
java.lang.IllegalArgumentException: fp.aeropuerto.Persona.<init>: el parámetro 1 es nulo

Test constructor-2 con apellidos y nombre
Persona[dni=, nombre=Gafotas, apellidos=Manolito]
Construido correctamente!!

Test constructor-2 con parámetros nulos
Ha habido un error al construir la persona
java.lang.IllegalArgumentException: fp.aeropuerto.Persona.<init>: el parámetro 2 es nulo
```

5. Implemente el Tipo Vuelo con los siguientes requisitos:

**VUELO**

**Propiedades:**

- String **código**. Consultable. Indica el código del vuelo.
- String **destino**. Consultable. Ciudad de destino del vuelo
- LocalDateTime **fechaHoraSalida**. Consultable. Fecha y hora de salida del vuelo.
- Duration **duración**. Consultable.
- Double **velocidad**. Consultable.
- Double **precio**. Consultable. Precio del billete.
- Integer **númeroPlazas**. Consultable. Número de plazas del vuelo.
- Boolean **conEscalas**. Consultable. Indica si el vuelo tiene, o no, escalas.
- List<Persona> **pasajeros**. Consultable. Lista con los pasajeros del vuelo.
- Integer **númeroPasajeros**. Consultable.
- Boolean **vueloCompleto**. Consultable. Indica si el vuelo está, o no, completo.
- Double **porcentajeOcupación**. Consultable.
- LocalDateTime **fechaHoraLlegada**. Consultable. Fecha y hora de llegada a destino.
- Compañía **compañía**. Se calcula a partir de los 3 primeros caracteres del código. Podrá tomar uno de los cuatro siguientes valores: IBE, LUF, RYA, VGL.

**Métodos Constructores:**

- Un constructor a partir de cada uno de los atributos (constructor canónico).

**Restricciones:**

- Ningún atributo puede ser nulo.
- El código debe tener 7 caracteres, los 3 primeros alfabéticos y los 3 último numéricos. El del centro serán un guion (-). "El formato del código debe ser aaa-nnn"
- El precio es mayor o igual que cero.
- El número de plazas debe ser mayor o igual que cero.
- La velocidad debe ser mayor o igual que cero.
- La duración es mayor que cero.
- El número de pasajeros es menor o igual que el número de plazas.

**El criterio de igualdad**

- Dos vuelos son iguales si tienen el mismo código, destino y la misma fecha de salida (sin la hora)

**El criterio de ordenación:**



- Por el código y desempatan por el destino y, en su caso, también por la fecha de salida (sin la hora)

**La representación como cadena:**

- El código, el destino, fecha y hora de salida, fecha y hora de llegada, número de plazas y número de pasajeros.

6. En `fp.aeropuerto,test` ya tiene implementado una clase **TestVuelo01** que si la ejecuta debe obtener lo siguiente:

```
Test constructor-1 con todos los parámetros
Vuelo [código=IBE-001, destino=Valencia, fechaHoraSalida=2023-08-01T09:00,
fechaHoraLlegada=2023-08-01T10:55 , Nro.pasajeros=3]
Construido correctamente!!

Test constructor-1 con algún parámetro nulo
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: el parámetro 1 es nulo

Test constructor-1 con el código malo
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El formato del código
debe ser aaa-nnn

Test constructor-1 con la duración negativa
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: La duración debe ser >0

Test constructor-1 con la velocidad negativa
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: La velocidad debe ser >=0

Test constructor-1 con el precio negativo
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El precio del billete
debe ser >=0

Test constructor-1 con las plazas negativas
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El número de plazas debe
ser >0

Test constructor-1 con las plazas menores que el número de pasajeros
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El número de pasajeros
debe ser <= número de plazas

Test constructor-2 con todos los parámetros
Vuelo [código=IBE-001, destino=Valencia, fechaHoraSalida=2023-08-01T09:00,
fechaHoraLlegada=2023-08-01T10:55 , Nro.pasajeros=0]
Construido correctamente!!

Test constructor-2 con algún parámetro nulo
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: el parámetro 1 es nulo

Test constructor-2 con el código malo
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El formato del código
debe ser aaa-nnn

Test constructor-2 con la duración negativa
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: La duración debe ser >0

Test constructor-2 con la velocidad negativa
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: La velocidad debe ser >=0

Test constructor-2 con el precio negativo
Ha habido un error al construir el vuelo
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El precio del billete
debe ser >=0

Test constructor-2 con las plazas negativas
Ha habido un error al construir el vuelo
```



```
java.lang.IllegalArgumentException: fp.aeropuerto.Vuelo.<init>: El número de plazas debe ser >0
```

## 7. Practicamos métodos de Collection y sus subinterfaces

Crear una clase **TestPersona02** para realizar operaciones de Collection.

1. Para ello primero creamos 5 personas con los datos:

```
("FRANCISCO MIGUEL","AARAB ORTIZ","12346678A");  
("ALBERTO","AGUILAR RALSTON","13457789B");  
("ALVARO","AGUILAR RALSTON","14568900C");  
("ADRIÁN","ALARCON MARTIN","15680011D");  
("PABLO","ALBA CONRADI","16791122E");
```

2. Crea 3 listas de pasajeros con los siguientes datos:

```
pasaj1 con las 3 primeras personas  
pasaj2 con las 2 últimas personas  
pasaj3 con las 3 personas que ocupan las posiciones impares
```

3. Visualiza el número de elementos de las 3 listas.
4. Añade a **pasaj1** los elementos de **pasaj3**
5. Visualiza el número de elementos de **pasaj1**.
6. Crea un conjunto de pasajeros **conj1**. y añádele los elementos de **pasaj1**.
7. Visualiza el número de elementos de **conj1**.
8. Añade a **pasaj2** los elementos de **conj1** desde **la posición 1**.
9. Visualiza el número de elementos de **pasaj2**.
10. Comprobar si el **primer pasajero** está en **pasaj2**
11. Eliminar el **primer pasajero** de **pasaj2**.
12. Visualiza el número de elementos de **pasaj2**

## 8. Experimento sobre la mutabilidad de las colecciones

1. Crear **TestPersona03** y copie del **TestPersona02**
  - La construcción de las 5 primeras personas.
  - La construcción de la lista **pasaj1**.
  - Añadir a **pasaj1** las 3 primeras personas.

2. Construya un **Vuelo "v"** con los siguientes datos:

```
("IBE-  
001","Valencia",LocalDateTime.of(2023,8,1,9,0),Duration.ofMinutes(115),890.5,150d,5,false,pasaj1);
```

3. Visualice el número de pasajeros del vuelo. (deben ser 3)
4. Añada el pasajero 4 a la lista **pasaj1**.
5. Visualice el número de pasajeros del vuelo. (deben ser 4). **¿Qué ha pasado sino hemos actuado sobre el vuelo?**



6. Añada la siguiente sentencia: `v.pasajeros().add(p5);`
7. Visualice el número de pasajeros del vuelo. (deben ser 5). **¿Hemos podido modificar un atributo privado mediante un método consultor? La respuesta es SI**  
Ello ha sido porque las **colecciones son mutables** y se manejan internamente a través de punteros que permite modificar el contenido de dichas colecciones. Salvo que se haga la Colección inmutable.  
**SOLUCIÓN:** En el **bloque etiquetado** del record introduzca la siguiente sentencia:  

```
pasajeros=List.copyOf(pasajeros);
```

  
Esta sentencia copia la lista pasajeros sobre ella misma, pero haciendo **inmutable** la lista.
8. Ejecute otra vez el test y comprobará que:
  - a) Aunque modifique pasaj1 la cuarta persona no afecta a los pasajeros del Vuelo ya creado
  - b) Que no permite modificar un atributo que ya se ha hecho privado.

9. Implemente el Tipo Aeropuerto con los siguientes requisitos:

**AEROPUERTO**

**Propiedades:**

- String **nombre**. Consultable. Indica el nombre del aeropuerto
- String **localidad**. Consultable. Indica la localidad donde se ubica el aeropuerto.
- List<Vuelo> **vuelos**. Consultable. Lista con los vuelos que salen del aeropuerto.
- Integer **número de vuelos**. Consultable.

**Métodos Constructores:**

- Un constructor a partir de cada uno de los atributos (constructor canónico).

**Restricciones:**

- Ningún atributo puede ser nulo.

**El criterio de igualdad**

- Dos aeropuertos son iguales si tienen el mismo nombre.

**El criterio de ordenación:**

- Por el nombre.

**La representación como cadena:**

- El nombre, un guion y entre paréntesis el número de vuelos.  
P.e: San Pablo – Sevilla (9)

10. Realice un test para probar el tipo Aeropuerto:

- 1) Crear TestAeropuerto01
  - Copie el vuelo “v” de test de la semana pasada (¿TestPersona03 o TestVuelo03?) y lo denomina v1. Construya otro vuelo v2.
  - Incorpore los dos vuelos a una lista vuelos.
  - Construya un aeropuerto con los siguientes datos:
    - Nombre: San Pablo
    - Localidad: Sevilla
    - Vuelos: la lista creada en el apartado anterior
- 2) Visualice todas sus propiedades
- 3) Visualice la representación como cadena.



## 11. Construir un aeropuerto leyendo los datos de los vuelos desde fichero.

Dada la siguiente estructura de un **fichero de vuelos** (se muestra la línea de cabecera y la primera línea con datos). Se recomienda ver el fichero en sí mismo:

```
codigo;Destino;fecha y hora;Duracion(en minutos);Velocidad;Precio;Número plazas;Escalas;Pasajeros
IBE-180;Las Palmas;01/06/2020-16:04;40;910.4;52.6;9;S;ALBERTO,AGUILAR RALSTON,13457789B#
ALBERTO,ARAGÓ CANOVAS,23457788L#GABRIEL MARÍA,YUSTE TEVAR,98047766B#ALEJANDRO,DOMINGUEZ
GALVÁN,79013338R# ALEJANDRO,REYES GARDUÑO,98004437J
```

Construir en el paquete `fp.aeropuerto` el método factoría: **FactoríaAeropuerto**, siguiendo la siguientes pautas:

- 1) El método público y estático **"leerAeropuerto"** tendrá 3 parámetros:

- Nombre, tipo String. Nombre del aeropuerto
- Localidad, tipo String. Localidad donde se ubica el aeropuerto
- Ruta, tipo String. Ruta del fichero del fichero de vuelos.

Devolverá un objeto Aeropuerto con el nombre y la localidad pasadas por parámetro y como lista de vuelos los vuelos leídos del fichero cuya ruta se pase como tercer parámetro.

**Nota.** Este método *Factoría* se diferencia de *FactoríaAnimal* en que no se devuelve la lista de vuelos que se lea, sino que dicha lista se usará al final del método para devolver un Aeropuerto usándola como tercer parámetro.

- 2) El método privado y estático **"parseaVuelo"** que, recibiendo una línea con los datos de un vuelo (observar que los campos están separados por ";") devolverá un objeto tipo Vuelo.

**Nota.** Este método necesitará invocar a un método auxiliar **"parseaPasajeros"**, también privado y estático que, recibiendo una cadena con los datos de los pasajeros de un vuelo separados por coma (,), devuelva una lista de tipo Persona con los pasajeros que contiene la cadena pasada por parámetro. **Tenga en cuenta** que un array de elementos tipo T se puede recorrer con un `for` → `for (T elemento : array)`

## 12. Realizar un TestAeropuerto02

- 1) Cree una carpeta `data` dentro del proyecto y copie en ella `"vuelos.csv"`
- 2) Dentro del método `main()`, cree un Aeropuerto `"miAeropuerto"` con el nombre `"San Pablo"`, localidad `"Sevilla"` y los vuelos se leen del fichero `"data/vuelos.csv"`. Invoque a un método denominado `"visualizaAeropuerto"` pasando como parámetro `"miAeropuerto"`
- 3) Cree un método privado, estático y de tipo void, `"visualizaAeropuerto"` que recibiendo como parámetro un objeto Aeropuerto visualice las siguientes tres líneas:
  - Objeto aeropuerto: datos del aeropuerto
  - Primer vuelo: datos del primer vuelo
  - Pasajeros del primer vuelo: datos de los pasajeros del primer vuelo

Resultado esperado

```
Objeto aeropuerto: San Pablo - Sevilla (65)
Primer vuelo: Vuelo [código=IBE-180, destino=Las Palmas, fechaHoraSalida=2020-06-01T16:04, fechaHoraLlegada=2020-06-01T16:44 , Nro.pasajeros=5]
Pasajeros del primer vuelo: [Persona[dni=13457789B, nombre=ALBERTO, apellidos=AGUILAR RALSTON], Persona[dni=23457788L, nombre=ALBERTO, apellidos=ARAGÓ CANOVAS], Persona[dni=98047766B, nombre=GABRIEL MARÍA, apellidos=YUSTE TEVAR], Persona[dni=79013338R, nombre=ALEJANDRO, apellidos=DOMINGUEZ GALVÁN], Persona[dni=98004437J, nombre=ALEJANDRO, apellidos=REYES GARDUÑO]]
```



13. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Recorrido Secuenciales)

- a) **existeVueloADestino(String destino)** que, recibiendo un destino como parámetro devuelva un boolean con el valor true o false según exista, o no, vuelo al destino dado.

Para probar cree un método estático en *TestAeropuerto02* que se llame `private static void testExisteVueloADestino(Aeropuerto a)` y que se invoque desde el main pasando como parámetro mi Aeropuerto. Invoque dos veces al método a probar con **Málaga** y con **Viena**

- b) **todosLosVuelosCuestanMenosQue(Double precio)** que, recibiendo un precio como parámetro devuelva un boolean con el valor true o false según todos los vuelos, o no, sean más barato que el parámetro dado.

Para probar cree en *TestAeropuerto02* el respectivo método `testTodosLosVuelosCuestanMenosQue` con la misma estructura del apartado a). Pruebe con **1000** y con **50**.

14. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Recorrido Secuenciales)

- a) **vueloMásDuración()** que, devuelva el vuelo que más duración ha tenido.

Para probar cree un método estático en *TestAeropuerto02* que se llame `private static void testVueloMásDuración(Aeropuerto a)` que visualice el código de vuelo y la duración, y que se invoque desde el main pasando como parámetro mi Aeropuerto.

15. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Map)

- a) **cuentaVuelosPorDestino()** que, devuelva un map con el número de vuelos a cada destino.

Para probar cree un método estático en *TestAeropuerto02* que se llame `private static void testCuentaVuelosPorDestino(Aeropuerto a)` que visualice el mapa, pareja por pareja (clave y valor) una debajo de otra, y que se invoque desde el main pasando como parámetro mi Aeropuerto.

- b) **distintosDestinosPorCompañía()** que, devuelva un map en que a cada compañía le asocie los distintos destinos a los que vuela.

Para probar cree un método estático en *TestAeropuerto02* que se llame `private static void testdistintosDestinosPorCompañía(Aeropuerto a)` que visualice el mapa, pareja por pareja (clave y valor) una debajo de otra, y que se invoque desde el main pasando como parámetro mi Aeropuerto.

- c) **distintosDestinosVuelosCompletosPorCompañía()** que, devuelva un map en que a cada compañía le asocie los distintos destinos a los que vuela en los que los vuelos están completos. Para probar cree un método estático en *TestAeropuerto02* que se llame `private static void testdistintosDestinosVuelosCompletosPorCompañía(Aeropuerto a)` que visualice el mapa, pareja por pareja (clave y valor) una debajo de otra, y que se invoque desde el main pasando como parámetro mi Aeropuerto.

16. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Collections)

- a) **vuelosPorOrdenNatural()** que devuelva los vuelos del aeropuerto por el orden natural de los vuelos. OJO! La lista de vuelos original no debe ser alterada.



Realice el respectivo test en el que se visualice, para cada vuelo, el código de vuelo, el destino y la fecha de salida. Debe comprobar que los vuelos se visualizan ordenados por esas propiedades y a igualdad de una de ellas se visualizan ordenadas por la siguiente. **¿Por qué ocurre eso?**

- b) **máximoVueloPorOrdenNatural()** que devuelva el vuelo del aeropuerto que sea el mayor según el orden natural de los vuelos.

Realice el respectivo test con la estructura ya conocida

**17. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Collections y la interfaz Comparator)**

- a) **vuelosPorInversoAlOrdenNatural()** que devuelva la lista de los vuelos del aeropuerto ordenada por el orden inverso al orden natural de tipo Vuelo. OJO! La lista de vuelos original no debe ser alterada.

Realice el respectivo test con la estructura ya conocida

- b) **vuelosPorPrecioYHoraSalida()** que devuelva la lista de los vuelos del aeropuerto ordenada por el precio y a igualdad de precio por hora de salida. OJO! La lista de vuelos original no debe ser alterada

Realice el respectivo test con la estructura ya conocida. Visualizando el código de vuelo y las propiedades relevantes para verificar el correcto funcionamiento.

- c) **vuelosPorDuraciónYMayorNroPasajeros()** que devuelva una lista de los vuelos del aeropuerto ordenada por la duración de los vuelos y a igualdad duración, de mayor a menor número de pasajeros. OJO! La lista de vuelos original no debe ser alterada

Realice el respectivo test con la estructura ya conocida. Visualizando el código de vuelo y las propiedades relevantes para verificar el correcto funcionamiento.

- d) **pasajerosDePrimerVueloPorNombreYDni()** que devuelva una lista con los pasajeros del primer vuelo del aeropuerto ordenada por el nombre de pila (sin apellidos) y, a igualdad de nombre, por dni. OJO! La lista de pasajeros original no debe ser alterada

Realice el respectivo test con la estructura ya conocida. Visualizando los datos de los pasajeros para verificar el correcto funcionamiento.

- e) **diferentesNombresDePasajerosPorOrdenAlfabéticoInverso()** que devuelva un conjunto ordenado con los nombre de todos los pasajeros de los vuelos del aeropuerto, ordenados alfabéticamente de mayor a menor (de la "z" a la "a". No obstante, los nombre con tildes van detrás de los que no tiene tilde).

Realice el respectivo test con la estructura ya conocida.

- f) **pasajerosDeTodosLosVuelosPorApellidosYNombre()** que devuelva una lista con los pasajeros de todos los vuelos del aeropuerto ordenada por apellidos y, a igualdad de apellidos, por nombre.





Realice el respectivo test con la estructura ya conocida. Visualizando los datos de los pasajeros para verificar el correcto funcionamiento.

- g) ***pasajerosSinRepetirDeTodosLosVuelosPorApellidosYNombre()*** que devuelva una lista con los pasajeros sin repetir de todos los vuelos del aeropuerto ordenada por apellidos y, a igualdad de apellidos, por nombre.

Realice el respectivo test con la estructura ya conocida. Visualizando los datos de los pasajeros para verificar el correcto funcionamiento

### 18. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -facilones)

- a) ***númeroVuelosADestino()*** que dada la denominación de un destino devuelve el número de vuelos a ese destino.

Realice el respectivo test con la estructura ya conocida para el destino Málaga.

- b) ***númeroPasajerosADestino()*** que dada la denominación de un destino devuelve la suma de pasajeros de todos los vuelos a ese destino.

Realice el respectivo test con la estructura ya conocida para el destino Barcelona.

- c) ***vueloMenorRecaudaciónVuelosCompleto()*** que devuelva el vuelo con la menor recaudación entre todos los vuelos completos. Si no hubiese vuelo debe lanzar la excepción “NoSuchElementException”

Realice el respectivo test con la estructura ya conocida.

- d) ***códigoDeAlgúnVueloADestinoConPlazasLibres()*** que dado un destino devuelve el código del algún vuelo con plazas libres a ese destino. Si no hubiese ningún vuelo debe devolver null.

Realice el respectivo test con la estructura ya conocida para el destino Málaga y para Cuenca

- e) ***existeVueloPrecioMenorQue()*** que dado un precio, existe algún vuelo por debajo de ese precio.

Realice el respectivo test con la estructura ya conocida para el precio 100 y precio 3.

### 19. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -facilones)

- a) ***promedioPreciosVuelosCompleto()***

Devuelve el promedio de los precios de los vuelos que estén completos. Si no hay vuelos devuelve 0.

Realice el respectivo test con la estructura ya conocida.

- b) ***sumaPreciosDistintosVuelosCompleto()***

Devuelve la suma de los precios distintos (no pueden repetirse) de los vuelos que estén completos.

Realice el respectivo test con la estructura ya conocida.



c) ***contarDistintosPasajeros()***

Devuelve el número de distintos pasajeros que vuelan desde el aeropuerto  
Realice el respectivo test con la estructura ya conocida.

d) ***distintosDNIPasajeroDeCompañia()*** que dada una compañía, devuelve un conjunto con los distintos dni's de los pasajeros de lo vuelos de la compañía dada.  
Realice el respectivo test con la estructura ya conocida para 'IBE'.

e) ***nombresOrdenadosDistintosPasajeroDeCompañia()*** que dada una compañía, devuelve un conjunto ordenado alfabéticamente (el orden natural de las String) con los distintos nombres de los pasajeros de lo vuelos de la compañía dada.  
Realice el respectivo test con la estructura ya conocida para 'VLG'.

20. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -facilones)

a) ***mapListaVuelosPorDestinos***

Devuelve un Map que a cada destino le haga corresponder su lista de vuelos  
Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

b) ***mapSetVuelosPorFechaLlegada***

Devuelve un Map que a cada fecha de llegada le haga corresponder un conjunto con sus vuelos.  
Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

c) ***mapSetOrdenadoVuelosPorFechaSalida***

Devuelve un Map que a cada fecha de salida le haga corresponder un conjunto de sus vuelos ordenados por número de pasajeros.  
Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

d) ***mapNumVuelosCompletoPorCompañia***

Devuelve un Map con el número de vuelos completos (de tipo Integer) por compañía.  
Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

e) ***mapPreciosOrdenadosPorDestino***

Devuelve un Map que asocie a cada destino un conjunto con los precios, ordenados de menor a mayor, de lo vuelos al destino de que se trate.  
Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.



21. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -medianos-)

a) ***mapPrecioMedioPorCompañia***

Devuelve un Map que permita conocer el precio medio que cuesta volar en cada compañía. El map debe estar ordenado alfabéticamente por las compañías.

Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

b) ***mapPrecioVuelosConEscalasMásBaratoPorDestino***

Devuelve un Map que indexe cada destino con el precio del vuelo con escala y más barato al destino de que se trate.

Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

c) ***mapSumaPlazasLibresPorHoraDeLlegada***

Dada una compañía, devuelve un mapa que indexe las horas de llegada con la suma de plazas libres. Las horas deben estar ordenadas en orden inverso -de mas tarde a más temprano-

Realice el respectivo test con la estructura ya conocida, para la compañía 'IBE', visualizando cada pareja clave valor en una línea.

22. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -complejos-)

a) ***destinoMayorNúmeroDeplazasLibres***

Devuelve el destino para el que hay más plazas libres.

Realice el respectivo **test**.

b) ***promediosDePasajerosPorFechasDeSalida***

Devuelve una lista en la que cada elemento es el promedio de pasajeros de cada fecha de Salida. La lista de promedios debe estar ordenada de menor a mayor fecha.

Realice el respectivo **test**.

c) ***promediosDePasajerosPorFechasDeSalida2***

Modifica el ejercicio anterior para que cada elemento de la lista que devuelve tenga la fecha y el promedio de pasajeros.

La lista de promedios debe estar ordenada de menor a mayor fecha.

Realice el respectivo **test**.

23. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -complejos-)

a) ***mapDestinosPorDuración***

Devuelve un mapa ordenado en orden inverso por duración y que relacione cada duración con el conjunto de los destinos de dicha duración, que también deben estar ordenados por orden inverso.



Realice el respectivo test con la estructura ya conocida, visualizando cada pareja clave valor en una línea.

b) ***mapPorcentajeVuelosPorDestino***

Dado un precio, devuelve un Map ordenado que le indexe a cada destino con precio superior al dado, el porcentaje de vuelos, respecto al total de vuelos que superan el precio dado que van al respectivo destino.

**Nota.**— Calcule y guarde primero el total de vuelos que cumplen el filtro y que permita calcular porcentajes, sino hubiese vuelos el método devuelve null.

Realice el respectivo **test**. Con precio de 60 y con 1000. Tenga en cuenta que si visualiza cada pareja clave-valor una debajo de otra debe contemplar que el resultado del método puede ser null.

c) ***compañíaConMayorSumaDePlazasLibres***

Devuelve la compañía en la que la suma de plazas libres entre todos sus vuelos es la mayor.

Realice el respectivo **test**.

d) ***segundoDíaConMenosVuelos***

Devuelve el segundo día de salida en que hay menos vuelos.

Realice el respectivo **test**.

e) ***mapPreciosSuperioresPorFechas***

Devuelve un SortedMap que a cada fecha de llegada le haga corresponder un conjunto con los precios que son mayores o iguales que el promedio entre el precio del vuelo más caro y el del más barato de los que salen en la correspondiente fecha.

Las fechas debe estar ordenadas desde más reciente a más antigua

Realice el respectivo **test**.

f) ***destinoConMayorDiferenciaDePrecio***

Devuelve el destino en el que la diferencia de precio entre el vuelo más caro y el más barato es mayor.

Realice el respectivo **test**.

**24. Realizar los siguientes métodos en el tipo Aeropuerto y en TestAeropuerto02. (Stream -complejos-)**

- a) ***fechaEnQueLleganMasConDistintoNombrePasajerosConDestinos*** (*Set<String> destinos*), que dado un conjunto con nombres de destinos, se trata de encontrar la fecha de llegada en la que hay más pasajeros con nombres distintos en los vuelos a alguno de los destinos dados.  
Realice el respectivo **test**.

- b) ***topNMediaPromediosPorDestino***(*Integer n*): dado un entero n, devuelve un SortedMap que tiene como clave los destino en orden alfabético inverso y como valor la media de los “n” precios más caros al destino de que se trate.  
Realice el respectivo **test**



- c) ***vuelosPorPasajero()***, [***Hacerlo con bucles***] devuelve un mapa que a cada pasajero le asocia una lista con los vuelos en los que ha volado  
Realice el respectivo ***test***.
- d) ***tiempoDeVuelosPorPasajero()***, devuelve un mapa que a cada pasajero le asocia la suma de duraciones de los vuelos realizados  
Realice el respectivo ***test***.
- e) ***dniPasajeroMásTiempoDeVueloPorPasajero()***, devuelve le pasajero que más tiempo ha volado.  
Realice el respectivo ***test***.