EL TIPO 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 clases avanzarán a partir de lo realizado en las anteriores.

A continuación, se propone un conjunto de ejercicios relacionados con los vuelos de un aeropuerto:

1. Importe el proyecto Aeropuerto.
2. Implemente en el paquete fp.aeropuerto directamente (sin interfaz) el tipo:

VUELO

Atributos:

```
private String destino;
private Double precio;
private Integer numPlazas;
private Integer numPasajeros;
private String codigo;
private LocalDateTime fechaHora;
private Duration duracion;
private Integer numPlazasPrimera;
private Double recargoPrecioPrimera;
```

Métodos:

- Un constructor con parámetros a partir de cada uno de los atributos.
- Un constructor a partir de String (*vea en el punto 3 el formato de la cadena de entrada*)
- Los respectivos métodos getters (consultores)y añadir un método getFecha()
- Restricciones:
  - Ningún atributo puede ser nulo
  - El precio es mayor que cero
  - La duración es mayor que cero
  - El recargo es mayor o igual que cero.
  - El número de pasajeros es mayor que cero y menor o igual que el número de plazas.
  - El número de plazas de primera es mayor o igual que cero y menor que el número de plazas.
- El criterio de igualdad: Dos Vuelos son iguales si tienen el mismo destino y la misma fecha.
- El criterio de ordenación: Por la fecha de salida y igualdad de fecha por el destino.
- La representación como cadena: todos los atributos.

3. En el paquete fp.aeropuerto.test implemente un TestVuelo01.

1. Cree 7 vuelos a partir de las siguientes cadenas:
   1. Madrid;107;150;150;RYR-80;01/06/2020-15:28;78;15;21.4
   2. Madrid;107;150;150;RYR-290;01/06/2020-12:22;78;15;32.1
   3. Valencia;50.6;180;150;VLG-400;01/06/2020-14:03;45;18;15.18
   4. Valencia;55.6;180;180;VLG-300;02/06/2020-13:44;45;18;27.8
   5. Barcelona;100.79;300;250;IBE-840;02/06/2020-21:20;65;30;50.4
   6. Barcelona;90;300;250;IBE-104;02/06/2020-22:08;65;30;36
   7. Oviedo;90;260;222;VLG-780;03/06/2020-13:45;80;26;27
2. Cree una lista vacía "misVuelos" para almacenar objetos tipo Vuelo.
3. Añada a la lista los 5 primeros vuelos creados en el punto 1.
4. Visualice el número de vuelos que contiene la lista.
5. Visualice los vuelos de las posiciones 0 y 2.
6. Visualice de los vuelos 2 y 4 sólo la fecha de salida.
7. Visualice una sublista con los tres vuelos desde la posición 1 a la 3
8. Añada un nuevo segundo vuelo (en la posición 1) con el vuelo número 6 del punto 1
9. Añada un nuevo primer vuelo con el vuelo número 7 del punto 1
10. Compare si son iguales el vuelo de la posición 2 y de la 4.
11. Compruebe que vuelo es menor ¿el 3 o el 5?

4. Implemente directamente (sin interfaz) el tipo:

AEROPUERTO

Atributos:

```
private String nombre;
private List<Vuelo> vuelos;
```

Métodos:

1. Un constructor con un único parámetro para el nombre del aeropuerto. No olvide pensar sobre el segundo atribibuto.
2. Un método añadeVuelos que recibe como parámetro una lista de vuelos y carga el atributo vuelos.
3. Los respectivos métodos getters.
4. La representación como cadena con el nombre de aeropuerto seguido de una flecha y entre corchetes del número de vuelos. Por ejemplo: Sevilla (San Pablo) -> [64]
5. El criterio de igualdad por el nombre

5. Implemente en el paquete fp.aeropuerto la clase FactoriaVuelos1 con un único método leerVuelos:

El método *leerVuelos* debe devolver una lista de vuelos a partir del nombre de un archivo con vuelos. Este método se apoya en el constructor a partir de String del tipo Vuelo.

El formato del registro de archivo es:
"destino; precio; número de plazas; número de pasajeros; código del vuelo; fecha y hora de salida; duración (en minutos); número de plazas de primera clase; recargo en el importe de primera clase"
Un ejemplo de su primera línea es:
"Madrid;107;150;150;RYR-80;01/06/2020-15:28;78;15;21.4 "
Nota. - Coincide con el del constructor a partir de String.

6. Implemente en el paquete fp.aeropuerto la clase FactoriaVuelos2 con los métodos leerVuelos y stringAVuelo [o parsearVuelo]:

a) El método *leerVuelos* debe devolver una lista de vuelos a partir del nombre de un archivo con vuelos. Este método se apoya en el método del siguiente apartado.
b) El método *stringAVuelo* [o parsearVuelo], recibe una String con un registro de vuelos y devuelve un objeto Vuelo
El formato del registro es el del punto anterior.

7. Implemente en el paquete fp.aeropuerto.test una clase *TestAeropuerto01*

El test debe:
a) crear un objeto aeropuerto *miAeropuerto* con el nombre "Sevilla (San Pablo)"
b) Añadir al aeropuerto los vuelos del archivo "*data/vuelos.csv*" usando las dos implementaciones de Factoría y compruebe que los resultados son el mismo.
c) Visualizar los vuelos del aeropuerto uno debajo de otro.

8. Implemente en el paquete fp.aeropuerto.test una clase TestVuelos02

El test es el siguiente:
a) Cargue los vuelos del archivo "*data/vuelos.csv*" e insérteselos en una lista de vuelos ***listVuelos.*** Recuerde que tiene dos factorías para hacerlo.
b) Visualizar cuantos vuelos hay en ***listVuelos***.
c) Inserte todos los vuelos en un conjunto de vuelos ***conjVuelosA*** (Set<Vuelo>) creado con HashSet
d) Visualizar cuantos vuelos hay en el conjunto ***conjVuelosA***. Compruebe que hay menos vuelos que los que se visualizan en el apartado b). ¿A qué se debe?
e) Cargue un conjunto ***conjVuelosB,*** filtrando los vuelos con 150 pasajeros o menos.
f) Visualizar cuantos vuelos hay en el conjunto ***conjVuelosB***
g) Cargue un conjunto ***conjVuelosC,*** filtrando los vuelos con un precio de 75 euros o menos.
h) Visualizar cuantos vuelos hay en el conjunto ***conjVuelosC***
i) Cree un conjunto ***conjBackup*** a partir de ***conjVuelosB*** (*hacer un duplicado*)

j) Visualice cuantos elementos tiene el conjunto ***intersección de B y C***. (*esta operación modifica conjVuelosB, por eso antes se ha hecho un backup de dicho conjunto*)
k) Limpie ***conjVuelosB*** *y cargue los vuelos desde* ***conjBackup*** *(hemos restituido el conjunto B)*
l) Visualice el número de vuelos del conjunto B-C (*la diferencia de B y C*).
m) Cree un conjunto ordenado ***conjOrdVuelos*** (TreeSet).
n) Cargue en ***conjOrdVuelos*** la lista de vuelos ***listVuelos***.
o) Visualizar cuantos vuelos hay en el conjunto ***conjOrdVuelos***.(*compruebe que son los mismo que los del* ***conjVuelosA)***
p) Visualice los vuelos del conjunto ***conjOrdVuelos*** uno detrás de otro y observe su orden.

9. Implemente en el paquete fp.aeropuerto.test una clase TestVuelos03

El test es el siguiente:

a) Cargue Leer los vuelos del archivo "*data/vuelos.csv*" e insertarlos en una lista de vuelos ***listVuelos.***
b) Baraje la lista de vuelos ***listVuelos***
c) Visualice los vuelos de ***listVuelos*** uno debajo del otro.
d) Ordene ***listVuelos*** por el orden natural.
e) Visualice los vuelos de ***listVuelos*** uno debajo del otro.
f) Visualice el máximo de los vuelos de ***listVuelos*** y compruebe que debe coincidir con el último de los visualizados en el apartado d)

10. Implemente en el tipo Aeropuerto los siguientes métodos:

a) ***númeroDeVuelosPorCompañia***(): Que devuelva un Map/Diccionario que haga corresponder el número de vuelos de cada compañía. La compañía son los tres primeros caracteres del código de vuelo.
Vaya al punto 11 y haga el test que se le pide. Después siga en este punto.
b) ***códigosVuelosPorDestino***(): Que devuelva un Map/Diccionario que a cada destino le haga corresponder una lista con los códigos de los vuelos a ese destino.
Vaya al punto 11 y haga el test que se le pide. Después siga en este punto.
c) ***promedioPreciosPorFecha***(): [*Ejercicio de examen*] Que devuelva un Map/Diccionario que a cada fecha le haga corresponder el promedio de precios (sin incluir recargo de primera clase) de los vuelos de la fecha de que se trate.
Ayuda: Este es un mapa de los que en Python se denominaban complejos. Primero se genera un Map con los precios de cada fecha. Posteriormente se genera un nuevo diccionario cuya clave son las fechas y los valores se calculan como el promedio de los precios del diccionario anterior. En Java no hay método sum para sumar los elementos de una lista, por lo que deberá hacer un método privado y estático *sumaPrecios*, que recibiendo una lista de precios devuelva la suma.

11. Implemente en el paquete fp.aeropuerto.test una clase TestAeropuerto02:

1. En el método main cree un bloque ***try-catch*** en el que capture (catch) la excepción "Exception e" y, en su caso, visualice la traza de la pila de métodos por donde ha pasado el programa "printStackTrace". A partir de este momento todas las sentencias del test se realizarán dentro del bloque try.

2. Cree un Aeropuerto.
3. Añada al aeropuerto los vuelos del archivo "*data/vuelos.csv*" usando cualquiera de las dos factorías de Vuelo.
4. En la clase TestAeropuerto cree un método privado, estático y void ***testNúmeroDeVuelosPorCompañia***, que recibe un aeropuerto y visualiza uno debajo de otro el mapa que devuelve ***númeroDeVuelosPorCompañia.*** *Recuerde que para recorrer un mapa puede usar el método entrySet() que devuelve los pares clave-valor en u tipo Entry.*
5. Invoque desde el método main al método test del apartado anterior.
6. *Realice los apartados 4 y 5 para* ***códigosVuelosPorDestino***
7. *Realice los apartados 4 y 5 para* ***promedioPreciosPorFecha***

12. Implemente en el tipo Aeropuerto los siguientes métodos (Interfaz Comparator):

a) ***ordenaVuelosPorOrdenNatural:***
   - Que devuelva una lista de los vuelos ordenados por el orden natural de tipo Vuelo (recuerde que existe la clase de utilidad Collections).
   - Añada al TestAeropuerto02 lo necesario para probar el método.

b) ***ordenaVuelosPorFechaSalida***:
   - Que devuelva una lista de los vuelos ordenados por la fecha de salida.
   - Añada al TestAeropuerto02 lo necesario para probar el método.

c) ***ordenaVuelosPorDestinoYFechaSalida***:
   - Que devuelva una lista de los vuelos ordenados por el destino y en caso de empate por la fecha de salida.
   - Añada al TestAeropuerto02 lo necesario para probar el método.

d) ***ordenaVuelosPorNroPasajerosAlReves***:
   - Que devuelva una lista de los vuelos ordenados de mayor a menor número de pasajeros.
   - Añada al TestAeropuerto02 lo necesario para probar el método.

e) ***ordenaVuelosPorDuraciónYMayorNumeroPasajeros***:
   - Que devuelva una lista de los vuelos ordenados de por la duración y a igualdad duración de mayor a menor número de pasajeros.
   - Añada al TestAeropuerto02 lo necesario para probar el método.

13. Implemente en el tipo Aeropuerto los siguientes métodos (Stream -fáciles-):

a) getNúmeroVuelosADestino
Dada la denominación de un destino devuelve el número de vuelos a ese destino.
Modifique TestAeropuerto02 para probar el método

b) getNúmPasajerosADestino
Dada la denominación de un destino devuelve el número total de pasajeros a ese destino.
Modifique TestAeropuerto02 para probar el método

c) getRecaudacionADestino

Dada la denominación de un destino devuelve la recaudación de los vuelos a ese destino (sin tener en cuenta el recargo de primera).
Modifique TestAeropuerto02 para probar el método

d) getPromedioPasajerosDeUnaFecha
Dada una fecha devuelve el promedio de pasajeros de dicha fecha. Si no hubiera vuelo debe devolver cero
Modifique TestAeropuerto02 para probar el método con el 2/6/2020 y también en el día de hoy.

e) getCodigoPrimerVueloADestinoConPlazasLibres
Dado un destino devuelve el código del primer vuelo con plazas libres a ese destino. Si no hubiese vuelo debe lanzar la excepción "NoSuchElementException"
Modifique TestAeropuerto02 para probar el método con Málaga y con Dos Hermanas.

f) existeVueloPrecioMenor
Dado un precio, existe algún vuelo por debajo de ese precio
Modifique TestAeropuerto01 para probar el método

g) todosVuelosPorcentajeMayor
Dado un porcentaje de ocupación ¿Están todos los vuelos por encima de ese porcentaje?
Modifique TestAeropuerto01 para probar el método

h) getVueloMenorOcupacion
¿Cuál es el vuelo con menor ocupación? Si no hubiera vuelo debe devolver null.
Modifique TestAeropuerto01 para probar el método

14. Implemente en el tipo RedDeAeropuertos los siguientes métodos (Stream -fáciles-):

a) getNúmeroVuelosADestino
Dada la denominación de un destino devuelve el número de vuelos a ese destino de la red de aeropuertos.
Descomente la correspondiente línea en TestRedAeropuertos para probar el método

b) getPromedioPlazasVuelosCompletos
Devuelve el promedio de plazas de los vuelos completos de la red de aeropuertos. Si no es posible obtener el promedio devuelve -1.
Descomente la correspondiente línea en TestRedAeropuertos para probar el método

15. Implemente en el tipo Aeropuerto los siguientes métodos (Stream -medianos-):

a) ***getListaTresVuelosMasBaratos***
- Devuelve una lista con los tres vuelos más baratos.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

b) ***getListaNVuelosMayorDuracion***
- Dado un parámetro n devuelve una lista con los n vuelos de mayor duración
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

c) ***getListaVuelosOrdenadosFechaYNumPasajeros***
- Devuelve una lista con los vuelos ordenados por fecha y a igualdad de fecha, ordenador por número de Pasajeros
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

d) ***getConjuntoOrdenadoDestinos***
- Devuelve un conjunto ordenado con los destinos de todos los vuelos
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

e) ***getMapListaVuelosPorDestinos***
- Devuelve un Map que a cada destino le haga corresponder su lista de vuelos
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

f) ***getMapSetVuelosPorFecha***
- Devuelve un Map que a cada fecha le haga corresponder un conjunto con sus vuelos
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

g) ***getMapSetOrdenadoVuelosPorFecha***
- Devuelve un Map que a cada fecha le haga corresponder un conjunto ordenado, de sus vuelos por número de pasajeros-
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

h) ***getMapNumVuelosPorFecha***
- Devuelve un Map que con el número de vuelos (de tipo Integer) de cada día.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

i) ***getMapPrecioMedioPorDestino***
- Devuelve un Map que permita conocer el precio medio que cuesta volar a cada destino
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

16. Implemente en el tipo Aeropuerto los siguientes métodos (Stream -Complejos-):

a) ***getMapVuelosCompletosMásBaratoPorDestino***
- Devuelve un Map que a cada destino le haga corresponder el vuelo más barato. Si no hubiese vuelo para algún destino devolverá null para dicho destino.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

b) ***getMapCódigoVuelosCompletosMásBaratoPorDestino***
- Devuelve un Map que a cada destino le haga corresponder el código del vuelo más barato. Si no hubiese vuelo para algún destino devolverá null para dicho destino.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

c) ***getMapOrdenadoNumPasajerosPorDuraciónDeCompañia***
- Dada una compañía devuelve un SortedMap que relacione cada duración con el total de pasajeros a esa duración de los vuelos de la compañía dada como parámetro. Las duraciones deben estar ordenados en orden inverso.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

d) ***getMapPorcentajeVuelosPorDestino***
- Devuelve un Map que haga corresponder a cada destino el porcentaje de vuelos, respecto al total de vuelos del aeropuerto, que van al respectivo destino.
  Nota. - Vea primero si hay vuelos paa clacular el porcentaje, sino hubiese vuelos devuelve null.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

e) ***getSegundaCompañíaConMenosVuelos***

- Devuelve la segunda compañía con menos número de vuelos.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

17. Implemente en el tipo Aeropuerto los siguientes métodos (Stream -Complejos-):

a) ***getDestinoConMásVuelos***
- Devuelve el destino con más número de vuelos. Si no hubiese vuelos devolverá null.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

b) ***getFechaConMásDestinosDiferentes***
- Devuelve la fecha en que han salido más vuelos a *diferentes* destinos. Si no hubiese vuelos para algún destino lanzará la excepción NoSuchElementException.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

c) ***getPromedioMásCaroYMásBaratoPorDestino***
- Devuelve un SortedMap que hace corresponder a cada destino el promedio del precio más caro y el más barato al correspondiente destino. El SortedMap debe estar ordenado en el orden alfabético de los destinos.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

d) **getCódigosVuelosMasDuraciónPorCompañiaADestinoQueComienzaPor**
- Dado una cadena de texto y un número entero “n”, devuelve una lista con los códigos de los “n” vuelos de mayor duración de cada compañía. Los vuelos que considerar deben tener como destino aquellos cuyo nombre comienza por la cadena dada como primer parámetro.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método

e) **getCódigosVueloMásBaratoYMásCaroPorCompañía**
- Devuelve un Map que a cada compañía le hace corresponder una lista con dos elementos: el código del vuelo más caro y el código del más barato de la compañía de que se trate.
- Descomente la línea correspondiente en el TestAeropuerto03 para probar el método