La implementación de cada ejercicio debe realizarse usando el lenguaje JAVA.

Ejercicio 1: Viajeros

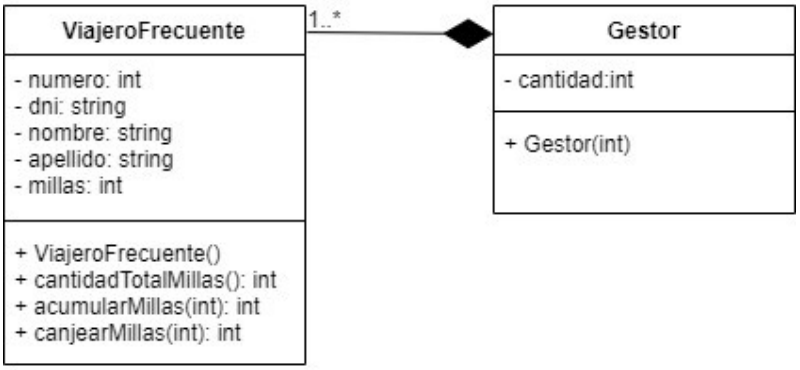
Objetivos:

- Repasar conceptos del paradigma orientado a objetos
- Conocer el lenguaje y el entorno de desarrollos implementando una aplicación sencilla.

Descripción General

En una aerolínea ofrece una promoción a sus viajeros frecuentes que consiste en acumular millas por los viajes que realizan, pudiendo luego recibir beneficios a través del canje de millas.

Usted trabaja en el área de sistemas de la aerolínea y le han solicitado la implementación de un sistema capaz de gestionar la promoción. Respetando el siguiente diseño de clase:



Descripción de los métodos de la clase ViajeroFrecuente:
a- El constructor.
b- "cantidadTotaldeMillas", retorna la cantidad de millas acumuladas.
c- "acumularMillas", recibe como parámetro la cantidad de millas recorridas, las suma en el atributo correspondiente y retorna el valor del atributo actualizado.
d- "canjearMillas", recibe como parámetro la cantidad de millas a canjear. Para canjear las millas debe verificarse que la cantidad a canjear sea menor o igual a la cantidad de millas acumuladas, caso contrario mostrar un mensaje de error en la operación. Retorna el valor de la cantidad de millas acumuladas.

Implemente:

1- **Las clases dadas en el diagrama. La clase gestor se basa en un arreglo.**
2- **Un programa que presente un menú con las siguientes funcionalidades:**
   a) **Cargar viajero**: registrar un nuevo viajero.
   b) **Mostrar viajero**: dado el número de un viajero mostrar todos sus datos (use el método toString).
   c) **Consultar Cantidad de Millas**: se ingresa un dni y si corresponde a un viajero registrado, retorna la cantidad de millas del viajero.
   d) **Acumular Millas**: se **ingresa** un dni y cantidad de millas, si corresponde a un viajero registrado se incrementa las millas acumuladas del viajero en la cantidad de millas dadas.
   e) **Canjear Millas**: se ingresa un dni y cantidad de millas, si corresponde a un viajero registrado se decrementa las millas acumuladas del viajero en la cantidad de millas dadas.
   f) **Mejor viajero**: Mostrar los datos del/ los viajero/s con mayor cantidad de millas.

Ejercicio 2: Gestión deportistas

Objetivo: Implementar una aplicación que use interface.

Descripción General

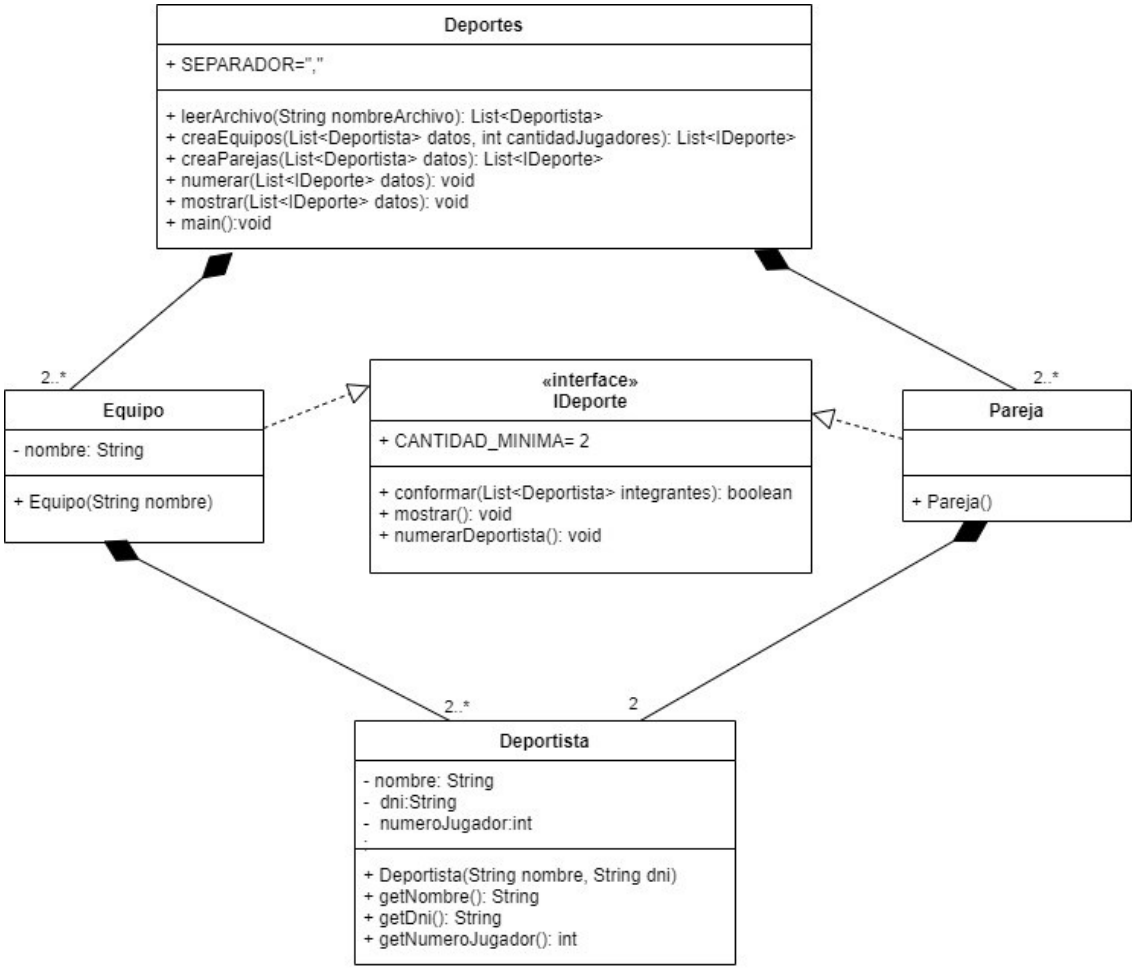
Un club deportivo ha organizado un campeonato de futsal y un campeonato de pin pon. Requiere de una aplicación que le permita administrar los datos de los deportistas inscriptos. Los datos de los inscriptos se han almacenado en archivos csv separados por coma, los de Futsal se han registrado en el archivo *inscriptosFutbol.csv* y los datos de los inscriptos para pin pon en el archivo *inscriptosPinPon.csv*. En ambos casos los datos registrados son nombre y DNI del deportista.

Los equipos de futsal tienen un nombre que los identifica, mientras que las parejas se identifican por los nombres de los deportistas que la conforman.

Tanto los equipos que participan en futsal como las parejas de pin pon se generan conforme el orden que se presenta en el archivo con los respectivos datos.

Le han solicitado al grupo de desarrollo donde usted trabaja una aplicación que genere automáticamente los equipos correspondientes. Además, la aplicación debe asignar un número a cada deportista que conforma el equipo (1…5) o pareja (1 y 2) según corresponda. Finalmente mostrar los datos (nombre y datos de cada deportista que lo integra) de los equipos y parejas conformados.

El analista ha desarrollado el siguiente diagrama de clases:



Descripción de los métodos de la clase Deporte.

- **leerArchivo**: Recibe como parámetro el nombre del archivo .csv que contiene los datos (nombre y DNI) de los deportistas. A partir de ellos crea las instancias correspondientes, las almacena en una lista y retorna dicha lista. (Se utiliza para los dos archivos)
- **creaEquipos**: Recibe como parámetro una lista con instancias de la clase Deportista y la cantidad de jugadores que conforman un equipo. De la lista toma la cantidad de jugadores para conformar un equipo, crea una instancia de Equipo y la almacena en una lista. Repite este proceso mientras haya jugadores para conformar un equipo. Retorna la lista con los equipos creados. **Nota**: Un jugador solo puede estar en un equipo.
- **creaParejas**: Recibe como parámetro una lista con instancias de la clase Deportista, de esta lista toma de a dos jugadores para conformar una pareja, crea una instancia de Pareja y la almacena en una lista. Repite este proceso mientras haya jugadores para conformar una pareja. Retorna la lista con las parejas creadas. **Nota**: Un jugador solo puede estar en una pareja.
- **numerar**: Numera cada integrante de un equipo o de una pareja.
- **mostrar**: Muestra los datos de cada equipo o de cada pareja.

Parte de algunos métodos ha sido implementado, usted deberá completarlo de forma tal que responda a los requerimientos del cliente y a lo propuesto por el equipo (se proporciona el archivo Deportes.java)

Ejercicio 3: Optimizar la gestión de deportistas

Objetivo: Incluir excepciones en la aplicación que gestiona los deportistas (ejercicio anterior).

A través de excepciones definidas por el programador, maneje las siguientes situaciones:

a. El nombre del deportista es vacío.
b. El DNI del deportista es vacío.
c. La cantidad de deportistas no son suficientes para conformar un equipo o una pareja, según corresponda.

Ejercicio 4: Viajeros

Considerando la clase Viajero trabajada en el ejericio 1 se pide que cree una aplicación que responda a los siguientes requerimientos:

1- Implemente una clase gestor basada en una lista.
2- Desarrolle una aplicación que usando Streams proporcione las siguientes funcionalidades:
    a. Agregar viajeros al gestor.
    b. Mostrar los datos de los viajeros ordenados por la cantidad de millas.
    c. Mostrar los nombres de los viajeros que tienen más de 200 millas.
    d. Obtener el viajero que tiene más millas.

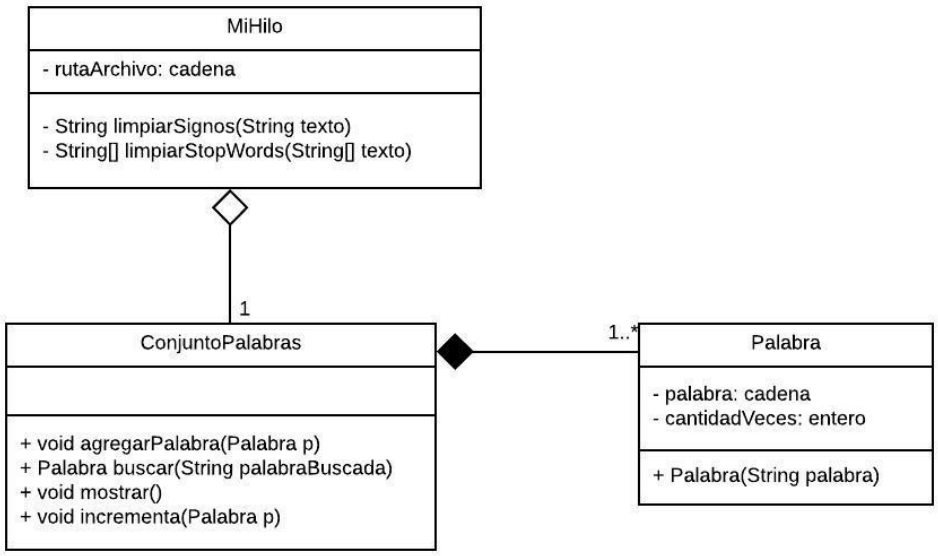
Ejercicio 5: Procesamiento de texto

Objetivo: Implementar un programa que agilice el procesamiento de texto a través del procesamiento en paralelo y concurrente.

Descripción: En ciencia de datos, para el procesamiento de texto, es necesario contar la cantidad de veces que cada palabra aparece en el texto. Esto requiere que previamente el texto sea limpiado, en primer lugar, eliminando caracteres que no son de utilidad como signos de

puntuación, comillas, etc., y posteriormente eliminando del texto las “stopwords”, es decir, aquellas palabras que no tienen valor para el análisis, tales como preposiciones, artículos y otras.

Diagrama de clases (incompleto)



Se pide: Implementar un programa que procese de forma paralela dos archivos de texto (.txt) y muestre la cantidad de veces que aparece cada palabra considerando los dos textos.

1. Genere dos instancias de la clase *MiHilo.*
2. Cada instancia de la clase *MiHilo* debe procesar un archivo de texto.
   2.1. Limpiar el texto.
   2.2. Registrar cada palabra y la cantidad veces que aparece en una misma instancia de la clase *ConjuntoPalabras*.
3. Mostrar cada palabra y la cantidad de veces que aparece, considerando los dos textos.

Ejemplo

Texto1.txt

Como cuesta, subir la cuesta.
Si la cuesta no fuera cuesta, no costaría.

Texto2.txt

La cuesta no costaría si no fuera cuesta.
Pero como es cuesta, cuesta mucho.

Salida del programa

Palabra: como cantidad: 2
Palabra: cuesta cantidad: 8
Palabra: subir cantidad: 1
Palabra: no cantidad: 4
Palabra: si cantidad: 2
Palabra: fuera cantidad: 2
Palabra: pero cantidad: 1
Palabra: es cantidad: 1
Palabra: mucho cantidad: 1

Información de interés para resolver el ejercicio:

- Diapositivas de la cátedra.
- Métodos toLowerCase, split, replace de la clase String.
- https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html
- https://docs.oracle.com/javase/tutorial/essential/io/buffers.html