Material de PRÁCTICAS

Resumen aproximado de la asignatura

Bloques de TEORÍA

- **Bloque 1:** el lenguaje Java
- **Bloque 2:** construcción tipos en POO
- **Bloque 3:** tipos contenedores y recorridos
- **Bloque 4:** programación en *streaming*

LABORATORIOS

- **Laboratorio 1:** Primeros pasos en Java
- **Laboratorio 2:** tipos inmutables (records) y lectura de ficheros
- **Laboratorio 3:** tipos mutables o tipos "clásicos"
- **Laboratorio 4:** factorías
- **Laboratorio 5:** tratamientos secuenciales "clásicos"
- **Laboratorio 6:** diccionarios I
- **Laboratorio 7:** diccionarios II
- **Laboratorio 8:** programación en streaming I
- **Laboratorio 9:** programación en streaming II
- **Laboratorio 10:** programación en streaming III

Primeros pasos en Java

Ejercicio: recorridos de listas

Este ejercicio puede hacerse en una clase de utilidad (clase con solamente métodos *static*) de nombre, por ejemplo, *RecorridosListas* y las funciones de testeo en otra clase de nombre *TestRecorridosListas.*

a) Construya una función o método tal que, dada una lista de números reales, devuelva el valor medio de aquellos que sean mayor que un parámetro dado como entrada.
   - A modo de comprobación o testeo, compruebe que si la lista es [21.5, 13.0, 19.0, 33.5, 11.0, 16.5] y el umbral es 15.0, entonces el resultado es 22.62.

```
//A modo de ayuda:
public static Double valorMedioMayorQue(List<Double> lista, Double umbral) { …}
```

b) Construya una función o método que, dada una lista de cadenas, devuelva la cadena que tenga un mayor número de caracteres. En caso de haber varias cadenas del mismo tamaño, devuelva la última en orden de aparición.
   - A modo de comprobación o testeo, compruebe que si la lista es [BRCA1, APOE, MTHFR, HBB, SOD1, CFTR], el resultado es MTHFR[1].

```
//A modo de ayuda:
public static String cadenaMasCaracteres(List<String> lista) {…}
```

c) Construya una función o método que, dada una lista de fechas – objetos del tipo LocalDate – y una fecha en concreto, devuelva otra lista con aquellas fechas que sean posteriores a la fecha dada como parámetro de entrada.
   - A modo de comprobación o testeo, compruebe que si la lista de entrada es [2023-02-02, 2022-12-05, 2023-01-10, 2022-11-15, 2023-02-06] y la fecha dada como parámetro es 2023-01-01, entonces devuelve [2023-02-02, 2023-01-10, 2023-02-06].
     - Nota: tenga en cuenta que el formato anterior es formato americano, es decir, el día 2022-12-5 se corresponde con el cinco de diciembre del año 2023.

```
//A modo de ayuda
public static List<LocalDate> fechasPosterioresA(List<LocalDate> lista, LocalDate
fecha){ …}
```

[1] *A modo de curiosidad, los nombres que hemos utilizado para hacer el testeo se corresponden con nombres de genes. Por ejemplo, BRCA1 y BRCA2 son genes relacionados con un mayor riesgo de padecer cáncer de mama y de ovario. APOE está asociado al riesgo de padecer la enfermedad de Alzheimer. HBB y MTHFR son genes que codifican la hemogoblina y una enzima relacionada con enfermedades del corazón, respectivamente. Por último, los genes SOD1 y CFTR son genes relacionados, por un lado, con la esclerosis múltiple y, por otro, con la fibrósis quística.*

Ejercicio: experimentos con cadenas

Este ejercicio puede hacerse en una clase de utilidad *ExperimentosCadenas* y las funciones de testeo en otra clase de nombre *Test ExperimentosCadenas.*

a) Construya una función o método que, dada una cadena con términos separados por un delimitador (podrá ser un punto y coma, una coma, un espacio en blanco), devuelva una lista de cadenas donde cada elemento es uno de los términos anteriores.
   - A modo de comprobación o testeo, compruebe que si la cadena es "Spain; Northern Spain; Tinta de Toro" entonces devuelve la lista de cadenas que devuelve es ["Spain", "Northern Spain , "Tinta de Toro"], siendo el delimitador el punto y coma.

```
//A modo de ayuda
public static List<String> troceaCadena1(String cadena, String delimitador){ …}
```

b) Construya una función o método que, dada una cadena con números reales separados por punto y coma, devuelva una lista compuesta por dichos números. (Tenga en cuenta que puede haber espacios en blanco antes o después de los punto y comas).
   - A modo de comprobación o testeo, si la cadena es "1.4; 27.9; -12.9;35.7; 5.8; 9.9", entonces la lista de números es [1.4, 27.9, -12.9, 35.7, 5.8, 9.9].

```
//A modo de ayuda
public static List<Double> troceaCadena2(String cadena){ …}
```

Ejercicio: experimentos con fechas (objetos del tipo LocalDate)

Este ejercicio puede hacerse en una clase de utilidad *ExperimentosFechas*.

a) Construya una función que se llame *parseoUnaFecha* que, reciba una cadena con el formato de este ejemplo concreto "20/02/2023" y devuelva un objeto del tipo *LocalDate*.
   - Utilice la barra como delimitador, transforme cada valor en un entero y utilice el método LocalDate.of(- , - , -) para construir la fecha asociada.

b) Repita la funcionalidad del método anterior de manera directa utilizando el método *parse* asociado al tipo *LocalDate*.
   - LocalDate.parse(--- , DateTimeFormatter(---))

c) Construya objetos fecha usando el método *parse* antes comentado con cadenas con los formatos especificado por los siguientes ejemplos:
   - "20-02-2023"
   - "02 - 20 - 2023"
   - 20:02:2023

Tipos inmutables y lectura de ficheros

Ejercicio: construcción de tipos inmutables (utilizando *records*)

Este ejercicio puede hacerse en un paquete que se llame de nombre poo.tipos. Por cada tipo, debe programar dicho tipo con *record* y hacer una clase de testeo en el que construya varios ejemplos concretos y se invoque algunas de sus propiedades.

a) Tipo **Usuario**:
   - Tipo inmutable con las siguientes propiedades:
     - identificador, de tipo String.
     - nombre, de tipo String.
     - apellidos, de tipo String.
     - edad, de tipo Integer.
     - esMayorEdad, de tipo Integer. Se calcula viendo si la edad es mayor que 18.
   - Restricciones: la edad tiene que ser una cantidad entre 0 y 130.
   - Representación como cadena: por defecto.
   - Criterio de igualdad: por defecto.
   - Orden natural: no tiene.

b) Tipo **MedicionPaciente**:
   - Tipo inmutable con las siguientes propiedades:
     - código de paciente, de tipo *String*.
     - colesterol, de tipo *Integer*.
     - triglicéridos, de tipo *Integer*.
     - hemoglobina, de tipo *Double*.
     - sexo, de tipo *enumerado* Genero {HOMBRE, MUJER}
     - fecha, de tipo *LocalDate*.
     - riesgoColesterol, de tipo enumerado FactorDeRiesgo {NORNAL, ALTO, MUYALTO}. Si está por debajo de 200 se considera normal, si está entre 201 y 240 se considera alto y por encima de 241 muy alto.
     - riesgoTrigliceridos, de tipo enumerado FactorDeRiesgo {NORNAL, ALTO, MUYALTO}. Si está por debajo de 150 se considera normal, entre 151 y 499 alto y por encima de 500 muy alto.
     - riesgoHemoglobina, de tipo Boolean. Devuelve falso para los hombres si está por debajo de 13 y para las mujeres por debajo de 12. En cualquier otro caso, devuelve cierto.
   - Restricciones:
     - El colesterol, los triglicéridos y la hemoglobina deben ser cantidades mayor estrictas que cero.
     - La fecha debe ser anterior o igual a la fecha actual.
   - Representación como cadena: por defecto.
   - Criterio de igualdad: por defecto.
   - Orden natural: no tiene.

c) Tipo **ViralogiaPaciente**:
   - Tipo inmutable con las siguientes propiedades:
     - fecha, de tipo *LocalDate*.
     - género, de tipo *enumerado* Genero {HOMBRE, MUJER}.

  - edad, de tipo *Integer*.
  - ciudad, de tipo *String*.
  - patógeno, de tipo *enumerado* {COVID, ZIKA, VIRUELA_MONO}.
- Restricciones:
  - La edad tiene que ser una cantidad entre 0 y 130.
- Representación como cadena: por defecto.
- Criterio de igualdad: por defecto.
- Orden natural: no tiene.

d) Tipo **Terremoto**:
- Tipo inmutable con las siguientes propiedades:
  - fecha, de tipo *LocalDate*.
  - latitud, de tipo *Double*.
  - longitud, de tipo *Double*.
  - ciudad, de tipo *String*.
  - intensidad, de tipo *Double*.
  - hemisferio, de tipo *enumerado* Hemisferio {NORTE, SUR, ECUADOR}. Se calcula a partir de la latitud.
- Restricciones:
  - La intensidad debe ser un valor del intervalo [0,10].
- Representación como cadena: por defecto.
- Criterio de igualdad: por defecto.
- Orden natural: no tiene.

Ejercicio: construcción de clases factoría para leer ficheros

En este ejercicio vamos a programar clases de utilidad que van a leer la información de varios ficheros, que se suministran con el enunciado, y que construirán listas de objetos de los tipos inmutables vistos en el ejercicio anterior.

Por cada clase factoría se debe programar una clase de testeo asociada en la que se compruebe que efectivamente se carga bien la información del fichero.

a) En primer lugar, una clase **FactoriaListas**, que construya listas de objetos a partir de la información contenida en ficheros. Concretamente:

   Método **leeGenes** que, dado el nombre de un fichero que contiene nombres de genes separados por comas, devuelve una lista de con los nombres de todos esos genes. Es decir, devuelve un objeto del tipo List<String>

   Fichero: *genes.txt*

```
ACTN1,APOE,BCL2,CDH1,CEBPA,CYP2D6, …
```

   Método **leeNumerosReales** que, dado el nombre de un fichero que contiene valores numéricos separados por comas – siendo el símbolo decimal el punto -, devuelve una lista con todos esos números. Es decir, devuelve un objeto del tipo List<Double>

Fichero: *genes.txt*

```
3.14, 1.23, 5.67, 8.91, …
```

b) Clase **FactoriaUsuario**: tiene un método principal que, dado un fichero con la estructura especificada, devuelve una lista de objetos *Usuario*.
Esta clase tendrá un método auxiliar que, dada una cadena con el formato asociado a una línea del fichero, devuelve un objeto del tipo *Usuario*.

Fichero: *usuarios.txt*

```
Identificador; nombre; apellidos; edad
id2345; Juan; Pérez Sánchez; 55
id4812; María; Gómez García; 39
…
```

c) Clase **FactoriaMedicionPaciente**: tiene un método principal que, dado un fichero con la estructura especificada, devuelve una lista de objetos *MedicionPaciente*.
Esta clase tendrá un método auxiliar que, dada una cadena con el formato asociado a una línea del fichero, devuelve un objeto del tipo *MedicionPaciente*.

Fichero: *estudio_pacientes.txt*

```
# cod_paciente, colesterol, trigliceridos, hemoglobina, sexo, fecha
15, 184, 119, 5.2, hombre, 12/09/2022
pac217, 175, 110, 4.1, mujer, 12/09/2022
id_2345, 210, 200, 8.5, hombre, 12/09/2022
pac4545, 303, 190, 8.5, mujer, 13/09/2022
id_1135, 179, 170, 4.5, hombre, 13/09/2022
```

d) Clase **FactoriaVirologiaPaciente**: tiene un método principal que, dado un fichero con la estructura especificada, devuelve una lista de objetos *VirologiaPaciente*.
Esta clase tendrá un método auxiliar que, dada una cadena con el formato asociado a una línea del fichero, devuelve un objeto del tipo *VirologiaPaciente*.

Fichero: *sistemaVigilancia.csv*

```
Fecha;Genero;Edad;Ciudad;Patogeno
04/01/2021;Male;80;Sevilla;COVID
04/01/2021;Female;74;Sevilla;COVID
04/01/2021;Female;14;Sevilla;COVID
…
```

e) Clase **FactoriaTerremoto**: tiene un método principal que, dado un fichero con la estructura especificada, devuelve una lista de objetos *Terremoto*.
Esta clase tendrá un método auxiliar que, dada una cadena con el formato asociado a una línea del fichero, devuelve un objeto del tipo *Terremoto*.

Fichero: *earthquake-utf8.csv*

```
fecha,lat,long,ciudad,richter
2017.07.31,37.42,31.37,konya,3.5
2017.07.31,36.9,27.56,mugla,3.3
2017.07.31,37,27.62,mugla,3.8
2017.07.30,37.39,31.37,konya,3.5
…
```

Tipos mutables o "clásicos"

Los siguientes tipos tienen **todos propiedades consultables y modificables** para sus propiedades básicas, además de uno o varios **constructores que se dejan a la elección del programador**. A continuación, proporcionamos su descripción en lenguaje natural.

a) El tipo **Libro** tiene dos propiedades título y autor de tipo String que representan el título y autor del libro, una propiedad númeroPáginas de tipo Integer que representa el número de páginas del libro, una propiedad fechaAdquisición de tipo LocalDate que representa la fecha de adquisición del libro, y una propiedad género del tipo enumerado Genero, que representa el género del libro y puede tomar los valores NOVELA, FICCIÓN y ENSAYO.

   El tipo Libro tiene una propiedad derivada año de tipo Integer que contiene el año de adquisición del libro. Dos libros son iguales si tienen el mismo título y autor. El número de páginas de un libro debe ser mayor que 0, y su fecha de adquisición debe ser posterior a 1900. El orden natural de los libros viene dado por el orden alfabético del título, y en caso de igualdad por el orden alfabético del autor.

   Construya una clase **FactoriaLibros** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo libro. Esta clase tendrá un método auxiliar que construya un objeto libro a partir de una cadena que siga el formato de cada línea del fichero.

   Fichero *libros.csv*

```
titulo,autor,paginas,fecha_adquisicion,genero
El Gran Gatsby,F. Scott Fitzgerald,180,2018-05-01,NOVELA
1984,George Orwell,328,2020-02-15,FICCIÓN
```

b) El tipo **Canción** tiene dos propiedades título y artista de tipo String que representan el título y autor de la canción, una propiedad duración de tipo Duration que representa la duración de la canción, una propiedad fechaLanzamiento que representa la fecha de lanzamiento al público de la canción, y una propiedad género del tipo enumerado Genero, que representa el género de la canción y puede tomar los valores POP, ROCK y FOLK.

   El tipo Canción tiene una propiedad derivada año de tipo Integer que contiene el año de lanzamiento de la canción. Dos canciones son iguales si tienen el mismo título y artista. La duración de una canción debe ser mayor que 0, y su fecha de lanzamiento debe ser posterior a 1950. El orden natural de las canciones viene dado por el orden alfabético del título, y en caso de igualdad por el orden alfabético del artista.

Construya una clase **FactoriaCanciones** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo canción. Esta clase tendrá un método auxiliar que construya un objeto canción a partir de una cadena que siga el formato de cada línea del fichero.

Fichero *canciones.csv*

```
titulo,autor,duracion,fecha_lanzamiento,genero
Shape of You,Ed Sheeran,3:54,2017-01-06,POP
Stairway to Heaven,Led Zeppelin,8:02,1971-11-05,ROCK
```

c) El tipo **Película** tiene una propiedad título de tipo String que representa el título de la película, una propiedad duración de tipo Duration que representa la duración de la película, una propiedad fechaEstreno de tipo LocalDate que representa la fecha de estreno de la película, una propiedad metraje del tipo enumerado TipoMetraje que puede tomar los valores CORTOMETRAJE y LARGOMETRAJE, y una propiedad valoración de tipo Double que representa la valoración de la película.

El tipo Película tiene una propiedad derivada metraje del tipo enumerado TipoMetraje, que puede tomar los valores CORTOMETRAJE y LARGOMETRAJE, según la duración de la película sea menor de 30 minutos o mayor o igual de 30 minutos, respectivamente. Dos películas son iguales si tienen el mismo título y fecha de estreno. La duración de una película debe ser mayor que 0, su fecha de estreno debe ser posterior a 1950 y su valoración debe estar comprendida entre 0 y 10. El orden natural de las películas viene dado por el orden alfabético del título, y en caso de igualdad por el orden cronológico de la fecha de estreno.

Construya una clase **FactoriaPeliculas** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo pelicula. Esta clase tendrá un método auxiliar que construya un objeto pelicula a partir de una cadena que siga el formato de cada línea del fichero.

Fichero *peliculas.csv*

```
Matrix|02:16:00|1999-03-31|LARGOMETRAJE|8.7
El rey león|01:28:00|1994-06-15|LARGOMETRAJE|8.5
```

Nota: observa el siguiente código, ejecútalo y observa qué ocurre. Investiga por tu cuenta el método *split* asociado al tipo *String*. ¿Por qué ocurre?

```
public class Prueba {
	public static void main(String[] args) {
		//
		String s = "15|25|12";
		System.out.println(Arrays.toString(s.split("|")));
		System.out.println(Arrays.toString(s.split("\\|")));
	}
}
```

d) El tipo **Vuelo** tiene dos propiedades origen y destino de tipo String que representan el origen y destino del vuelo, una propiedad fechaSalida de tipo LocalDate que representa la fecha de salida del vuelo, una propiedad numeroPasajeros de tipo Integer que representa el número de pasajeros del vuelo, y una propiedad compañía del tipo enumerado Compañía que representa la compañía que opera el vuelo y puede tomar los valores IBE, RYR y VLG.

El tipo Vuelo tiene una propiedad derivada código de tipo String que se obtiene concatenando la compañía y el día y mes de la fecha de salida, ambos con dos dígitos (por ejemplo, "IBE1503" para un vuelo de Iberia que sale el día 15 de marzo). Dos vuelos son iguales si tienen el mismo código y destino. El destino de un vuelo debe ser diferente del origen, la fecha de salida debe ser posterior al 1 de enero del año actual y el número de pasajeros debe ser mayor que 0. El orden natural de los vuelos viene dado por el orden cronológico de la fecha de salida, y en caso de igualdad por el destino.

Construya una clase **FactoriaVuelos** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo vuelos. Esta clase tendrá un método auxiliar que construya un objeto vuelo a partir de una cadena que siga el formato de cada línea del fichero.

Fichero *vuelos.csv*

```
Madrid|Barcelona|2023-03-09|120|RYR
Barcelona|Madrid|2023-03-10|100|RYR
Londres|París|2023-03-11|200|IBE
```

e) El tipo **Foto** tiene dos propiedades título y autor de tipo String que representan el título y autor de la foto, una propiedad fechaToma de tipo LocalDate que representa la fecha en que fue tomada la foto, una propiedad descargas de tipo Integer que representa el número de veces que la foto ha sido descargada, y una propiedad formato del tipo enumerado Formato que representa el formato de archivo de la foto y puede tomar los valores JPG, TIFF y RAW.

El tipo Foto tiene una propiedad derivada destacada de tipo Boolean que toma valor true si la foto tiene más de 1000 descargas, y false en caso contrario. Dos fotos son iguales si tienen el mismo título y autor. El número de descargas debe ser mayor o igual que 0 y la fecha de toma no puede ser posterior a la fecha de hoy. El orden natural de las fotos viene dado por el orden cronológico de la fecha de toma, y en caso de igualdad por el título.

Construya una clase **FactoriaFotos** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo foto. Esta clase tendrá un método auxiliar que construya un objeto foto a partir de una cadena que siga el formato de cada línea del fichero.

Fichero *fotos.csv*

```
"Montaña en la mañana"|"Pablo García"|2023-03-01|50|JPG
"Puesta de sol en la playa"|"María Pérez"|2023-03-02|40|TIFF
```

f) El tipo **Beca** tiene una propiedad código de tipo String que representa el código de la beca, una propiedad fechaComienzo de tipo LocaDate que representa la fecha de comienzo de la beca, una propiedad cuantía de tipo Double que representa el importe total de la beca, una propiedad duración de tipo Integer que representa la duración en meses de la beca, y una propiedad tipo del tipo enumerado TipoBeca que puede tomar los valores MOVILIDAD, TRABAJO e INVESTIGACIÓN.

El tipo Beca tiene una propiedad derivada cuantiaMensual de tipo Double que contiene el importe mensual de la beca. Dos becas son iguales si tienen el mismo código. La duración de la beca debe ser mayor o igual que 1 y la cuantía debe ser mayor o igual que 0. El orden natural de las becas viene dado por el orden cronológico de la fecha de comienzo, y en caso de igualdad por el código.

Construya una clase **FactoriaBecas** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo beca. Esta clase tendrá un método auxiliar que construya un objeto beca a partir de una cadena que siga el formato de cada línea del fichero.

Fichero *becas.csv*

```
C001,2022-09-15,1500.00,6,MOVILIDAD
C002,2023-01-02,2000.00,9,INVESTIGACIÓN
C003,2022-10-01,1000.00,12,TRABAJO
```

g) El tipo **Persona** tiene dos propiedades nombre y apellidos de tipo String que representan el nombre y apellidos de la persona, respectivamente, una propiedad dni de tipo Long que representa el número del DNI (sin letra) de la persona, una propiedad fechaNacimiento de tipo LocalDate que representa su fecha de nacimiento, y una propiedad género del tipo enumerado Genero que puede tomar los valores H y M

El tipo Persona tiene una propiedad derivada edad de tipo Integer que contiene la edad en años de la persona. Dos personas son iguales si tienen el mismo dni. La fecha de nacimiento de la persona debe ser igual o anterior a la fecha de hoy. El orden natural de las personas viene dado por el orden alfabético de los apellidos, en caso de igualdad por el orden alfabético del nombre, y en caso de igualdad por el orden creciente del DNI.

Construya una clase **FactoriaPersonas** que tenga un método que, dada una cadena con el nombre de un fichero – con el formato especificado –, devuelva una lista de objetos de tipo persona. Esta clase tendrá un método auxiliar que construya un objeto persona a partir de una cadena que siga el formato de cada línea del fichero.

Fichero *personas.csv*

```
nombre,apellidos,dni,fechaNacimiento,género
María,Pérez García,12345678M,1990-02-14,M
Juan,González López,23456789R,1985-06-21,H
```

Construcción de tipos agregados

h) Se proporciona la descripción del **tipo Biblioteca** que contiene una colección de libros.

Propiedades:
- nombre de la biblioteca, de tipo String, consultable y modificable.
- libros, de tipo List<Libros>, consultable,
- número de libros, de tipo Integer, consultable. Se calcula a partir de la lista de libros.

Constructores:
- C1: recibe como parámetro el nombre de la biblioteca.
- C2: recibe como parámetro el nombre de la biblioteca y una cadena con el nombre de un fichero, que se utilizará para cargar la información de los libros del fichero.

Representación como cadena:
- El nombre de la biblioteca.

Criterio de igualdad:
- Dos bibliotecas son iguales si coincide su nombre.

Criterio de orden natural:
- Según el nombre de la biblioteca.

Otras operaciones:
- void adquiereLibro(Libro libro): añade un libro a la biblioteca.
- void adquiereLibros(List<Libro> lista): añade una lista de libros.
- void adquiereLibro(Libro libro, Integer posicion): añade un libro a la biblioteca en la posición que indica el parámetro.
- void borraLibro(Libro libro): borra un libro de los registros de la biblioteca.
- List<Libro> catalogoSinRepeticion(): devuelva una lista con todos los libros sin que haya libro repetidos en dicha lista.
- List<Libro> catalogoOrdenadoSinRepeticion(): devuelve una lista ordenada de libros – según el orden natural asociado al tipo Libro – en la que no haya libros repetidos.

i) Se proporciona la descripción del tipo **ListaReproduccion** que contiene una lista de canciones.

Propiedades:
- nombre, de tipo String, consultable y modificable.
- canciones, de tipo List<Cancion>, consultable.
- número de canciones, de tipo Integer, consultable. Se calcula a partir de la propiedad anterior.

Constructores:
- C1: recibe un parámetro para indicar el nombre de la lista.
- C2: recibe un parámetro para indicar el nombre de la lista y una cadena con el nombre de un fichero, que se cargará en la lista de canciones.

Representación como cadena:
- El nombre de la lista.

Criterio de igualdad:
- Dos listas de reproducción son iguales si lo son todas sus propiedades básicas.

Otras operaciones:
- void aleatoriza(): cambia aleatoriamente el orden de las canciones en la lista. Utilice el método estático shuffle de la clase Collections.

- void incorpora(Cancion c): añade la canción al final de la lista de reproducción.
- void incorpora(List<Cancion> canciones): añade todas las canciones de la lista al final de la lista de reproducción.
- void eliminaPrimera(Cancion c): elimina la primera aparición de la canción en la lista de reproducción. Si la canción no pertenece a la lista, se lanza IllegalArgumentException.
- void eliminaUltima(Cancion c): elimina la última aparición de la canción en la lista de reproducción. Si la canción no pertenece a la lista, se lanza IllegalArgumentException.
- void eliminaTrozo(int primeraPosicion, int ultimaPosicion): elimina todas las canciones de la lista que van desde la posición primeraPosicion hasta la posición ultimaPosicion, ambas inclusive. Debe cumplirse que primeraPosicion sea mayor o igual que cero, ultimaPosicion sea menor que el número de canciones en la lista, y primeraPosicion sea menor o igual que ultimaPosicion; en caso contrario se lanza IllegalArgumentException.

Tratamientos secuenciales "clásicos" y diccionarios

Ampliación del tipo Biblioteca

Añada las siguientes propiedades al tipo Biblioteca que son propiedades derivadas que se programan mediante tratamientos secuenciales. (En este momento de la asignatura se programarán con bucles aunque, más adelante, se pueden programar mediante programación en streaming).

- Integer libroMasPaginas(): devuelve el título del libro que tiene mayor número de páginas.
- Set<Libro> librosAutor(String autor): devuelve los libros de un determinado autor.
- Libro libroMasReciente(String autor): devuelve el libro más reciente de un determinado autor.
- Boolean existeLibro(String autor): devuelve si la biblioteca tiene o no un libro del autor especificado.
- Map<Genero, List<Libro>> agrupaLibrosPorGenero: devuelve un diccionario o mapa donde las claves son los géneros de los libros y los valores son las listas de libros asociados a cada género.
- Map<String, Long> cuentaLibrosPorAutor: devuelve un diccionario o mapa que cuenta cuántos libros tiene cada autor de la biblioteca.

Ampliación del tipo ListaReproduccion

Añada las siguientes propiedades al tipo Biblioteca que son propiedades derivadas que se programan mediante tratamientos secuenciales. (En este momento de la asignatura se programarán con bucles aunque, más adelante, se pueden programar mediante programación en streaming).

- Integer getNumeroCancionesComienzanPorCadena(String cadena): cuenta el número de canciones que comienzan por la cadena dada como parámetro.
- Duration getDuracionTotal(): obtiene la duración total de la lista de reproducción.
- Duration getDuracionMedia(): obtiene la duración media de las canciones de la lista de reproducción.
- List<Cancion> getCancionesGenero(Genero genero): obtiene una lista con las canciones del género dado como parámetro.

- Boolean existeCancionConPalabraEnTitulo(String palabra): devuelve true si existe una canción cuyo título contiene la palabra dada como parámetro, y false en caso contrario.
- Boolean todasSuperioresA(Integer minutos): devuelve true si todas las canciones tienen una duración igual o superior a la dada como parámetro, y false en caso contrario.
- Cancion getCancionMayorDuracion(): obtiene la canción de mayor duración.
- Map<Integer, List<Cancion>> agrupaCancionesPorAño(): crea un Map que hace corresponder a cada año una lista con las canciones de ese año.
- Map<Integer, List<Cancion>> agrupaCancionesArtistaPorPalabras(String artista): crea un Map que hace corresponder a cada número de palabras una lista con las canciones del artista dado como parámetro cuyos títulos tienen ese número de palabras.
- Map<String, Long> getNumeroCancionesPorArtista(): crea un Map que relaciona cada artista con el número de canciones del artista que hay en la lista de reproducción.
- Map<Genero, Duration> getDuracionCancionesPorGenero(): crea un Map que relaciona cada género con la duración total de las canciones de ese género.

Ejercicio de vinos

Disponemos de un fichero con la información de las **valoraciones de distintos vinos** presentes en el mercado. Antes de comenzar, observe el fichero:

Fichero *wine_reviews.csv*

```
Country,Region,Points,Price,Grape
US,California,96,235.0,Cabernet Sauvignon
Spain,Northern Spain,96,110.0,Tinta de Toro
US,California,96,90.0,Sauvignon Blanc
US,Oregon,96,65.0,Pinot Noir
…
```

Se modeliza el problema mediante un tipo básico, que refleja la valoración de un vino en concreto, además de un tipo agregado, que modeliza todas las valoraciones de los vinos que están presentes en el fichero. Además, se utilizarán clases auxiliares como, por ejemplo, las factoría que carga la información del fichero o las clases propias del testeo de lo que se programa.

Detallamos cada tipo con más detalle:

- Tipo **Vino**:
  - Propiedades:
    - país, de tipo String, consultable.
    - región, de tipo String, consultable.
    - puntos, de tipo Integer, consultable y modificable. Valor entre 0 y 100.
    - precio, de tipo Double, consultable y modificable. Valor mayor o igual que 0.
    - uva, de tipo String, consultable.
  - Constructores;
    - un constructor que reciba como parámetro un valor para cada propiedad básica.

        - un constructor que reciba como parámetro el país, la región y el tipo de uva, asignando valores por defecto al resto de propiedades básicas.
    - Representación como cadena: mostrando todos los atributos
    - Criterio de igualdad: determinado por la igualdad de todas las propiedades básicas.
    - Criterio de ordenación natural: los vinos se compararán por su puntuación. A igualdad de puntuación por precio, y a igualdad de precio se usará el orden natural de las propiedades uva, país y región, en ese orden

- Clase **FactoriaVinos**: una clase que tenga un método que, dada una cadena con el nombre del fichero, devuelva una lista de objetos de tipo Vino.

- Tipo **ValoracionVinos**[2]:

    - Propiedades:
        - vinos, de tipo Set<Vino>. Consultable.
    - Constructor:
        - un constructor a partir de una colección de objetos Vino.
        - un constructor vacío o sin parámetros.
    - Representación como cadena: mostrando todas las propiedades básicas.
    - Criterio de igualdad: determinado por la igualdad de todas las propiedades básicas.

    - Operaciones: propiedades derivadas que se calculan usando propiedades de la API
        - añadir vino, añade un vino al conjunto de vinos.
        - añadir vinos, añade una colección de vinos al conjunto de vinos.
        - elimina vino, borra un vino del conjunto de vinos.
        - número de vinos, devuelve el número de vinos presentes en el conjunto.

    - Tratamientos secuenciales: propiedades derivadas que se calculan a través de un bucle.
        - calcularNumeroVinosPais: cuenta el número de vinos dado un país.
        - obtenerNombresPaises: devuelve una lista ordenada con el nombre de todos los países.
        - obtenerVinoMejorPuntuado: busca el vino con la puntuación más alta.
        - calcularUvasPorRegion: conjunto de uvas usadas en los vinos de una región dada.
        - todosVinosDePaisPrecioSuperior: recibe el nombre de un país y el valor de un precio, y comprueba si todos los vinos de dicho país tienen un precio superior al dado.
        - obtenerVinosRangoPuntos: calcula una colección de Vino solo con los vinos que estén valorados en un rango de puntos determinado por un valor mínimo y otro máximo. Ambos parámetros son enteros, y se debe comprobar que el valor mínimo es menor o igual que el valor máximo.
        - calcularMediaPrecioEnRegion: recibe el nombre de una región y devuelve la media del precio de los vinos de dicha región.
        - obtenerVinoMenorPrecio: busca el vino de menor precio.

[2] Este tipo va a tener dos implementaciones. En este momento utilizando bucles for para los tratamientos secuenciales y, más adelante, las mismas propiedades utilizando las propiedades asociadas al tipo Stream<Vino>. Se sugiere implementar el tipo a través de una interfaz. Es decir, una interfaz de nombre ValoracionVinos y una clase que la implemente de nombre ValoracionVinosBucles y otra ValoracionVinosStream. En cualquier caso, se puede también implementar utilizando dos clases llamando a una ValoracionVinos1 y a la otra ValoracionesVinos2.

- obtenerVinosDeRegion: recibe el nombre de una región y devuelve un conjunto ordenado con los vinos de dicha región. Dichos vinos deberán estar ordenados según el orden alfabético de la uva.
- contarVinosPorUva: diccionario con número de vinos por uva.
- calcularVinosPorPais: diccionario con una lista de vinos por cada país.
- calcularUvasPorPais: calcula un diccionario cuyas claves son los paises y los valores el conjunto de uvas usadas en los vinos de cada país.
- calcularCalidadPrecioPorRegionMayorDe: calcula un diccionario cuyas claves son las regiones y los valores son el número de vinos cuya relación calidad/precio supera un umbral dado.

Ejercicio de premios nobel

Disponemos de un fichero con la información de todos los **premios nobel** entregados a lo largo de la historia de dichos premios. Antes de comenzar, observe el fichero:

Fichero *nobel_prizes.csv*

Datos de *premios*

	A	B	C	D	E	F
1	year	category	firstname	surname	gender	birthyear
2	1901	physics	Wilhelm Conrad	Röntgen	male	1845
3	1902	physics	Hendrik Antoon	Lorentz	male	1853
4	1902	physics	Pieter	Zeeman	male	1865
5	1903	physics	Antoine Henri	Becquerel	male	1852
6	1903	physics	Pierre	Curie	male	1859
7	1903	physics	Marie	Curie née Sklodowska	female	1867
8	1911	chemistry	Marie	Curie née Sklodowska	female	1867
9	1904	physics	Lord Rayleigh	(John William Strutt)	male	1842
10	1905	physics	Philipp Eduard Anton	von Lenard	male	1862
11	1906	physics	Joseph John	Thomson	male	1856

Se modeliza el problema mediante un tipo básico, que refleja la información de cada premio en concreto, además de un tipo agregado, que modeliza todos los premios entregados y que están presentes en el fichero. Además, se utilizarán clases auxiliares como, por ejemplo, las factoría que carga la información del fichero o las clases propias del testeo de lo que se programa.

Detallamos cada tipo con más detalle:

- Tipo **Nobel**:
  - Propiedades:
    - año: atributo Integer con el año del premio. Consultable.
    - categoria: atributo String con la categoría del premio. Consultable.
    - nombre: atributo String con el nombre del premiado. Consultable y Modificable.

        - apellidos: atributo String con los apellidos del premiado. Consultable y Modificable.
        - genero: atributo de tipo Genero con el género del premiado. Crear previamente el tipo enumerado Genero con los valores MALE y FEMALE. Consultable y Modificable.
        - añoNacimiento: atributo entero con el año de nacimiento del premiado. Este año debe ser menor que el año del premio. Consultable y Modificable.
        - edadPremiado: edad del premiado en el momento que recibe el premio. Se estimará mediante la diferencia entre el año del premio y el año de nacimineto.
    - Constructores;
        - un constructor que reciba como parámetro un valor para cada propiedad básica.
        - un constructor que reciba como parámetro el año de entrega del premio y la categoría.
    - Representación como cadena: mostrando todos los atributos
    - Criterio de igualdad: usando los atributos año, categoria, nombre y apellidos para determinar la igualdad.
    - Método de criterio de orden natural: los premios se compararán por su año y categoría. A igualdad de las propiedades anteriores por nombre y apellidos.

- Clase **FactoriaPremios**: una clase que tenga un método que, dada una cadena con el nombre del fichero, devuelva una lista de objetos de tipo Premio.

- Tipo **Premios**[3]:

    - Propiedades:
        - premios, de tipo Set<Premio>. Consultable.
    - Constructor:
        - un constructor a partir de una colección de objetos Premio.
        - un constructor a partir del nombre de un fichero-
        - un constructor vacío o sin parámetros.
    - Representación como cadena: mostrando todas las propiedades básicas.
    - Criterio de igualdad: determinado por la igualdad de todas las propiedades básicas.

    - Operaciones: propiedades derivadas que se calculan usando propiedades de la API
        - añadir premio, añade un premio al conjunto de premios.
        - añadir premio s, añade una colección de premios al conjunto de premios.
        - elimina premio, borra un premio del conjunto de premios.
        - número de premios, devuelve el número de premios presentes en el conjunto.

    - Tratamientos secuenciales: propiedades derivadas que se calculan a través de un bucle.
        - public Collection<Premio> obtenerPremiosDeGenero(Genero genero): calcula un conjunto de Premio solo con los premios del género indicado como parámetro

[3] Este tipo va a tener dos implementaciones. En este momento utilizando bucles for para los tratamientos secuenciales y, más adelante, las mismas propiedades utilizando las propiedades asociadas al tipo Stream<Premio>. Se sugiere implementar el tipo a través de una interfaz. Es decir, una interfaz de nombre Premios y una clase que la implemente de nombre PremiosBucles y otra PremiosStream. En cualquier caso, se puede también implementar utilizando dos clases llamando a una Premios1 y a la otra Premios2.

- Integer calcularNumeroPremiadosMasJovenesDe(Integer edad): cuenta el número de premiados más jóvenes de la edad indicada mediante el parámetro.
- Map<Genero, Integer> calcularNumeroPremiosPorGenero(): calcula un diccionario cuyas claves sean el género y el valor el número de premiados de ese género.
- Map<Integer, List<Premio>> calcularPremiosPorEdad(): calcula un diccionario cuyas claves sean la edad y el valor los premios dados a esa edad.
- Map<String, Double> calcularMediaEdadPorCategoria(): calcula un diccionario cuyas claves son las categorías y los valores la media de edad de los premiados en esa categoría.
- Map<String, List<Integer>> calcularEdadesPorCategoria()

Programación en streaming

Ejercicio de vinos

Realice una segunda implementación del tipo ValoracionVinos utilizando propiedades del tipo Stream<Vino>. Es decir, la programación será en streaming y no se utilizará ningún bucle con un for.

Recordamos el enunciado de dicho tipo:

- Tipo **ValoracionVinos**[4]:
  - Propiedades:
    - vinos, de tipo Set<Vino>. Consultable.
  - Constructor:
    - un constructor a partir de una colección de objetos Vino.
    - un constructor vacío o sin parámetros.
  - Representación como cadena: mostrando todas las propiedades básicas.
  - Criterio de igualdad: determinado por la igualdad de todas las propiedades básicas.
  - Operaciones: propiedades derivadas que se calculan usando propiedades de la API
    - añadir vino, añade un vino al conjunto de vinos.
    - añadir vinos, añade una colección de vinos al conjunto de vinos.
    - elimina vino, borra un vino del conjunto de vinos.
    - número de vinos, devuelve el número de vinos presentes en el conjunto.
  - Tratamientos secuenciales: propiedades derivadas que se calculan a través de un bucle.
    - calcularNumeroVinosPais: cuenta el número de vinos dado un país.
    - obtenerNombresPaises: devuelve una lista ordenada con el nombre de todos los países.
    - obtenerVinoMejorPuntuado: busca el vino con la puntuación más alta.
    - calcularUvasPorRegion: conjunto de uvas usadas en los vinos de una región dada.

[4] El diseño completo se encuentra más arriba en la relación de problemas.

- todosVinosDePaisPrecioSuperior: recibe el nombre de un país y el valor de un precio, y comprueba si todos los vinos de dicho país tienen un precio superior al dado.
- obtenerVinosRangoPuntos: calcula una colección de Vino solo con los vinos que estén valorados en un rango de puntos determinado por un valor mínimo y otro máximo. Ambos parámetros son enteros, y se debe comprobar que el valor mínimo es menor o igual que el valor máximo.
- calcularMediaPrecioEnRegion: recibe el nombre de una región y devuelve la media del precio de los vinos de dicha región.
- obtenerVinoMenorPrecio: busca el vino de menor precio.
- obtenerVinosDeRegion: recibe el nombre de una región y devuelve un conjunto ordenado con los vinos de dicha región. Dichos vinos deberán estar ordenados según el orden alfabético de la uva.
- contarVinosPorUva: diccionario con número de vinos por uva.
- calcularVinosPorPais: diccionario con una lista de vinos por cada país.
- calcularUvasPorPais: calcula un diccionario cuyas claves son los paises y los valores el conjunto de uvas usadas en los vinos de cada país.
- calcularCalidadPrecioPorRegionMayorDe: calcula un diccionario cuyas claves son las regiones y los valores son el número de vinos cuya relación calidad/precio supera un umbral dado.

Ejercicio de premios nobel

Realice una segunda implementación del tipo Premios utilizando propiedades del tipo Stream<Premio>. Es decir, la programación será en streaming y no se utilizará ningún bucle con un for.

El tipo se definió en el apartado anterior.

Problemas adicionales