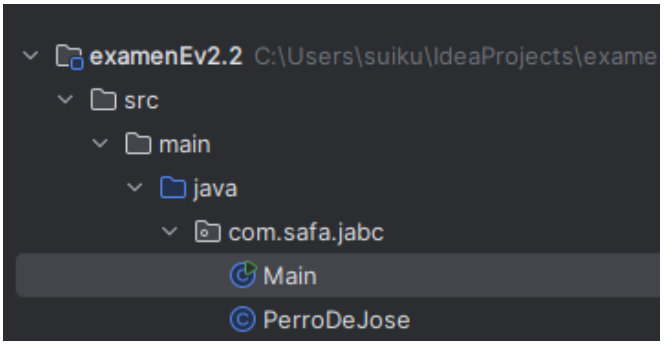
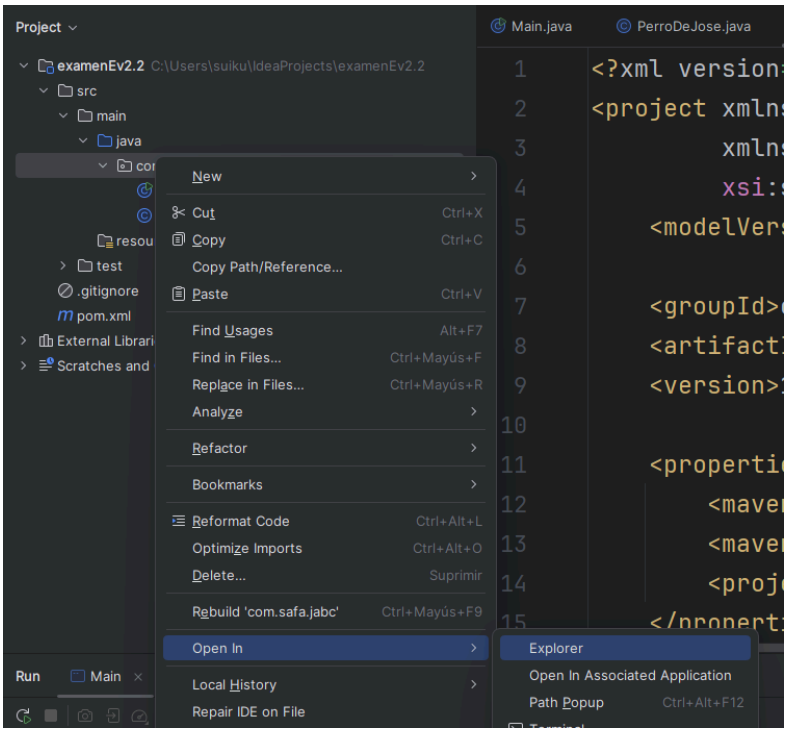
Examen de Programación: Segunda evaluación (2)

Para realizar este examen se generará una única clase Java de nombre Main. Tanto esta clase Main como el resto de clases que se necesiten se generarán en src>main>java y luego un paquete que cumpla la siguiente nomenclatura: **com.safa.<tus_iniciales>** Por ejemplo, en mi caso sería **com.safa.jabc**.



El contenido de este paquete será lo que se comprima en un fichero .ZIP y se adjunte como entrega en Moodle.



En cada ejercicio se indicará el **nombre exacto de las clases y los métodos** (ya que podrán probarse con clases ajenas a vuestro main también). Si la función tiene otro nombre (aunque sea el mismo cambiando mayúsculas o minúsculas), no se contabilizará el ejercicio. **Salvo que se indique lo contrario, ninguna función imprimirá nada por consola**.

Por otro lado, se recuerda que **el examen deberá permanecer en el equipo** (no una copia, no el comprimido, sino el original). En caso de tener dudas sobre posible trampa o copia, **si no se es posible acceder al histórico de cambios local del fichero, el examen se puntuará con 0**.

También, si se desea abandonar el aula para ir al baño, se deberá entregar antes el examen con lo que se tenga, dando así por finalizada la entrega.

Ejercicio 1: Filtrado y ordenación (2.5 puntos)

Jose Fuentes no quiere olvidarse de su próximo cumplemés con Elenita. Para ello ha estado sopesando diferentes opciones para celebrarlo (ha descartado el Popeyes porque sería algo habitual).

La lista de opciones la puedes recuperar utilizando la librería que se adjunta.

Para este ejercicio se solicita que se cree un método **public static void muestraOpcionesOrdenadas()** en el que se utilice stream() con expresiones lambda para el filtrado y referencias a métodos para la ordenación, tal que:

1) No devolverá nada, sino que los resultados los mostrarán por consola
2) Del listado devuelto por la librería nos quedaremos solo con las opciones que cuesten menos de 200€ y que no sean a más de 50 km
3) Los resultados del filtrado se ordenarán primero por la puntuación (lo que Jose cree de 0 a 10 que podría gustarle a Elenita) de forma descendente y, después (para desempate), por precio, de forma ascendente.
4) Para mostrar cada Opcion por pantalla, se tendrá que mostrar su puntuación y luego el resto de información por defecto de la Opcion. El formato que se mostrará por pantallá por cada Opcion será como se puede ver a continuación:

```
>> Puntuación: 7
----------
| Opción | Día de spa para dos
----------
    Descripción: Sesión de spa con jacuzzi, masajes y aromaterapia.
    Precio: 60.0€
    Distancia: 4.0km
```

Este método se llamará desde el método main de la clase Main, para pintar por consola el resultado.

Ejercicio 2: Mapa con las opciones (1 punto)

Dada la misma lista de Opciones de partida del ejercicio 1 (la de partida, no la filtrada y ordenada), se solicita un método **public static Map<Integer,List<String>> creaMapaOpciones()** que devolverá un Map ordenado por los valores de las claves, tal que las claves serán las puntuaciones de las opciones y los valores serán listados de los nombres en mayúsculas de las opciones que tengan esa puntuación.

En el método main de la clase Main deberá llamarse a este método e imprimirse el resultado.

Ejercicio 3: Validación con expresiones regulares (1.5 puntos)

Se quiere validar mediante un método **public static boolean esDocumentoValido(String nombreDocumento)** si los nombres de documentos legales que se le pasen cumplen o no el formato. Las reglas para que el formato sea válido son:

1. El identificador debe comenzar con las palabras "Ley", "Decreto", "Reglamento" u "Orden" (tal cual, con esa capitalización).
2. A continuación, debe haber un espacio y un número (de entre 1 y 4 dígitos). Dicho número no puede ir precedido de ceros (es decir 1 es válido, 01 no es válido).
3. Después, debe seguir una barra / y un año de 4 dígitos (que será entre 1975 y el año actual)
4. Opcionalmente, tras lo anterior, puede incluir un título después de la palabra "de".

Se recomienda el uso de expresiones regulares.

Una vez creado, llamad en el método main de Main al método tal que así:

```
System.out.println(esDocumentoValido("Ley 12/2024 de Protección de Datos"));

System.out.println(esDocumentoValido("Ley 0001/2025 de Ministerio de Igualdad"));

System.out.println(esDocumentoValido("Decreto 120/1999 de Seguridad Industrial"));

System.out.println(esDocumentoValido("Regla 15/2018"));

System.out.println(esDocumentoValido("Ley 21/1918 de Ministerio de Transportes"));

System.out.println(esDocumentoValido("Ley 5/23 de Consumo"));
```

El resultado, por consola, debería ser:

```
true
false
true
false
false
false
```

Ejercicio 4: De Paseo con los perros (5 puntos)

Jose Fuentes quiere ir con sus perros y sus hermanos de paseo por alguno de los parques de Sevilla. Los nombres de sus hermanos se pueden recuperar utilizando la librería proporcionada.

1) Clase PerroDeJose (1 puntos)

Se requiere una clase **PerroDeJose** que extenderá la clase Perro (de la librería). Tendrá un atributo adicional que será **paseadoPo**r (de tipo String), del que se requerirá su getter y setter. Se necesitará también un constructor de la clase y sobrescribir el método toString para que al final de la info de la clase padre pinte la info de paseadoPor, con el mismo formato. Ejemplo:

```
--------
Nombre: Chuli
>> Raza: Beagle
>> Tamaño: Mediano
>> Edad: Desconocida
>> Paseado por: Juanma
```

2) Creación de lista ordenada de perros de Jose (1 puntos)

Se solicita que se cree una **lista** con cada uno de los perros de Jose. Los perros de Jose son: **Chuli** (Beagle), **Cala** (Mastín) y **Lila** (Yorkshire Terrier). A la hora de crear cada PerroDeJose, tened en cuenta:

1. Los datos que desconozcáis pasadlos como null (como fechaNacimiento o paseadoPor)
2. No creéis un objeto RazaPerro para dar de alta cada instancia de PerroDeJose. Utilizad el método de la librería que recupera una instancia RazaPerro según el NombreRazaPerro (enum) que se le pase.

La lista debe ordenarse por el nombre de cada perro. Esta lista ordenada **debe mostrarse por consola** en el método main de Main.

3) Asignación de hermano que los pasea (1 punto)

Dada la lista de PerroDeJose del apartado anterior y la lista de hermanos de Jose que se puede recuperar con la librería. Se solicita:

1) Ordenad alfabéticamente (igual que se hizo con los perros) la lista de hermanos de Jose
2) Asignad a cada perro el hermano que lo va a pasear, tal que el primer perro de la lista será paseado por el primer hermano, el segundo perro por el segundo y el tercero por el tercero.

La lista de PerroDeJose, ya con el hermano que los pasea asignado, **debe mostrarse por consola** en el método main de Main.

4) Elección de parque (2 puntos)

Jose quiere ir a un parque con sus perros y hermanos, pero para que sus perros estén cómodos, quiere ir a un parque en el que las razas de los perros que Jose tiene (Mastín, Yorkshire Terrier y Beagle) sean mayoría.

Para ello, gracias a la librería tenemos una previsión de los perros que habrá en cada parque (sin contar los de Jose).

Este método getPerrosParques (de la librería) devuelve un Map cuyas claves son los nombres de los Parques que se están considerando. Los valores de este Map es otro Map en el que la clave es el enum NombreRazaPerro y el valor es el número de perros de esa raza que hay.

Se solicita la creación de un método **public static boolean sonMayoria(List<PerroDeJose> listaPerrosJose, Map<NombreRazaPerro,Integer> mapaParque)** que en base a si dada la lista de PerroDeJose (de la que se recomienda generar una lista con solo sus NombreRazaPerro) y un Map<NombreRazaPerro,Integer> de un parque concreto, vea si el número de perros de las razas de los perros de Jose (recordad que estos no están en el Map, habría que sumarlos) es mayoría en el parque o no. Si son mayoría, devuelve **true** y si no, **false.**

Una vez esté el método implementado, iterad sobre el mapa de parques (el que se obtiene con **getPerrosParques()**), para que se pinte por consola si se puede ir a ese parque o no.

El resultado debe ser este:

```
Ir a Parque Amate: No
Ir a Parque Miraflores: Sí
Ir a Parque del Alamillo: Sí
```

Anexo: librería joey-fountains.jar (y joey-fountains-javadoc.jar)

La librería joey-fountains.jar incluye las siguientes clases (con atributos):

Opcion
- String nombre
- String descripcion
- Double precio
- Double distancia
- Integer puntuacion
- Set<String> pros
- Set<String> contras

Perro
- String nombre
- RazaPerro raza
- LocalDate fechaNacimiento

RazaPerro
- NombreRazaPerro nombreRaza
- TamanyoRazaPerro tamanyoRaza

NOTA NombreRazaPerro y TamanyoRazaPerro son enums

Además, está la clase **SoporteJoey** (de tipo Singleton), cuya única instancia se recupera mediante SoporteJoey.getInstance()
Esta clase proporciona los métodos:

public List<Opcion> getOpciones(): devuelve la lista de opciones de Jose tiene está sopesando para su cumplemés.

public List<RazaPerro> getRazasPerros(): devuelve una lista con las RazaPerro definidas

public RazaPerro getRazaPerroByName(NombreRazaPerro nombre): devuelve una RazaPerro en base al valor del enum NombreRazaPerro que se le pase.

public Map<String, Map<NombreRazaPerro, Integer>> getPerrosParques(): recupera un mapa de Parques de Sevilla precargado, cuyo valor es un mapa de previsión de razas de perro y cantidad de la misma en dicho parque.

public List<String> getHermanosJose(): devuelve la lista con los nombres de los hermanos de Jose Fuentes.

La librería joey-fountains-javadoc incluye la información JavaDoc de la librería anterior.

Para instalar ambas en el repositorio maven (el pom bastaría como hemos visto hasta ahora), podría utilizarse (estando en la ruta pertinente) el comando:

```
mvn install:install-file -Dfile=joey-fountains.jar -DgroupId=com.joey.utils -DartifactId=joey-utils -Dversion=1.0 -Dpackaging=jar -Djavadoc=joey-fountains-javadoc.jar
```

Con esto se mostraría la información disponible de las clases y métodos de la librería joey-fountains.jar