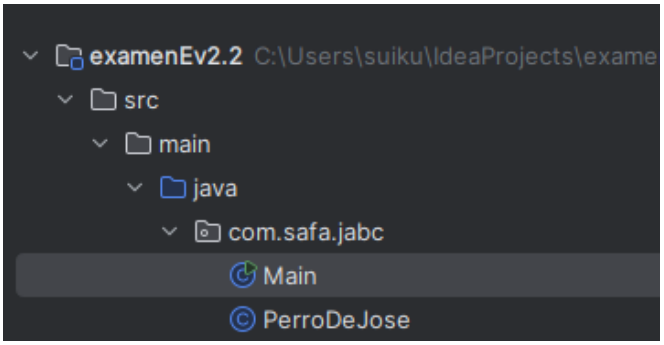
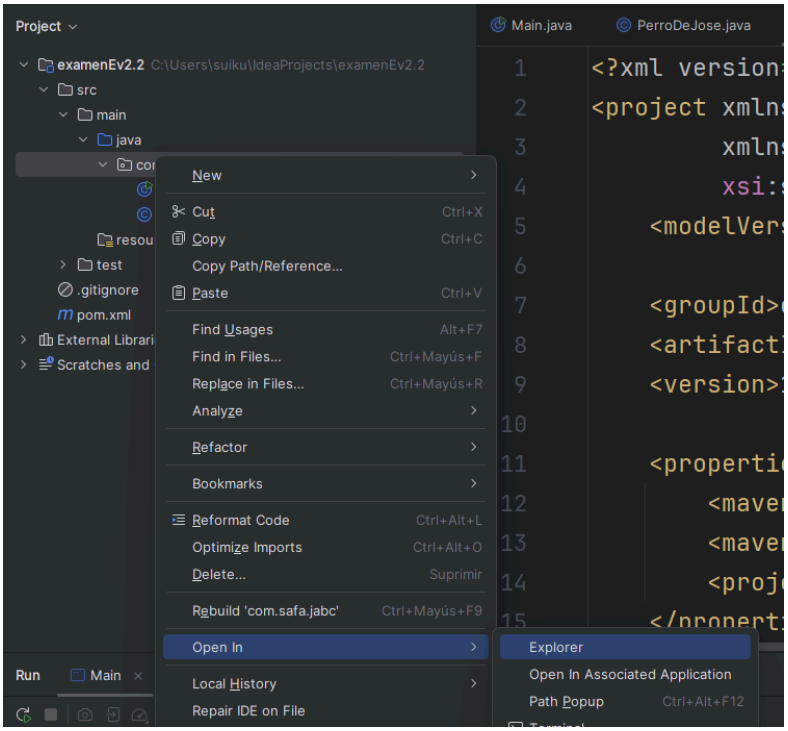
Programación: Recuperación Segunda Evaluación (1)

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.

Por último, se recuerda que el examen constará de una parte escrita y esta parte (para desarrollar en el ordenador). Para hacer media de ambos exámenes ambos deberán estar aprobados (nota de 5 o más). Si no, la nota del examen será la nota mínima de ambas partes, siendo esa la nota de la segunda evaluación.

Ejercicio 1: Clasificación por tamaño (1.5 puntos)

Utilizando la librería adjunta, se solicita que en una clase UtilidadesExamen se cree un método estático **clasificaPorTamanyo**. Este método devolverá un mapa cuyas claves serán de tipo TamanyoRazaPerro y sus valores será la lista de nombres de razas de perro (String) de dicho tamaño ordenada alfabéticamente.

Para recuperar la lista de razas de perros, utilizad **SoporteJoey**, llamando al método necesario.

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

Ejercicio 2: Validaciones de código (2.5 puntos)

En una aplicación de compras se pueden utilizar códigos de descuentos sobre productos. Antes de comprobar en base de datos si el código existe y ha sido utilizado o no, se quiere hacer una primera criba para ver si cumple el formato esperado.

Los código siempre tendrán el formato siguiente: **AAA-111AAA**

Es decir, 3 caracteres, un guion, 3 dígitos cualesquiera y 3 caracteres.

Además, los caracteres han de cumplir las siguientes condiciones:

1) Los caracteres válidos pueden ser tanto en mayúsculas como minúsculas.
2) Su versión en mayúsculas ha de estar contenida en el siguiente array (utilízalo en tu código y con el tipo definido)

```
static List<Character> caracteresValidos =
Arrays.asList('A', 'Z', 'F', 'Q', 'N', 'P', 'T', 'D', 'V',
'X', 'J', 'B');
```

3) El primer carácter del código está vinculado al último, el segundo al penúltimo y el tercero al antepenúltimo, de la siguiente manera:

   Si uno de los caracteres es el segundo del array, el otro ha de ser el segundo empezando desde atrás. Por ejemplo, Si un carácter es la F (3ª posición), el carácter relacionado ha de ser la X (3ª posición desde atrás, o antepenúltima).

   Ejemplos de códigos válidos:

   ADB-141**AnB**
   ZqJ-320**ZVJ**
   XxB-587**aFF**

Visto esto, en la clase UtilidadesExamen, crea un método **public static boolean validaCodigo(String codigo)**, que devuelva si el código que se pasa como entrada es correcto o no.

Para ello se solicita que utilices expresiones regulares (con Pattern y Matcher) como base de la resolución del ejercicio.

En el método main de la clase Main deberá llamarse a este método e imprimirse el resultado. Se dan unos ejemplos, de los cuales todos deberán devolver false menos el segundo.

```
System.out.println(validaCodigo("ABc-123Vxs"));
System.out.println(validaCodigo("AfQ-123vXB"));
System.out.println(validaCodigo("aBCF-123VxS"));
System.out.println(validaCodigo("ABC-1VXs"));
System.out.println(validaCodigo("ABc112VXS"));
System.out.println(validaCodigo("112VxS"));
```

Ejercicio 3: Mapa con valoraciones de opciones (1.5 puntos)

De entre las posibles opciones de cita, estas tenían unos comentarios a favor (pro) y otros en contra. Se requiere un método **public static Map<Integer, String> ejercicioOpciones()** que devuelva un mapa tal que contendrá 4 entradas:

1. Con la clave 1: Tendrá aquella valoración a favor con el número mínimo de caracteres de entre todas las valoraciones a favor con número impar de caracteres.
2. Con la clave 2: Tendrá aquella valoración a favor con el número máximo de caracteres de entre todas las valoraciones a favor con número par de caracteres.
3. Con la clave -1: Tendrá aquella valoración en contra con el número mínimo de caracteres de entre todas las valoraciones en contra con número impar de caracteres.

4. Con la clave -2: Tendrá aquella valoración en contra con el número máximo de caracteres de entre todas las valoraciones en contra con número par de caracteres.

Utiliza SoporteJoey para recuperar las diferentes opciones de cita y recupera de cada una los comentarios a favor y comentarios en contra para recuperar este

Imprime desde el método main el resultado devuelto por el método.

Ejercicio 4: Perretes famosos (4.5 puntos)

Se desea analizar una lista de perros que han alcanzado cierta fama histórica o mediática. Para ellos se solicita la creación de lo siguiente:.

1) Clase PerroFamoso (1 punto)

Se requiere una clase **PerroFamoso** que extenderá la clase Perro (de la librería). Tendrá los siguiente atributos adicionales: **lugarOrigen** (String) y **motivoFama**(String), de los 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 el nombre del perro, el nombre de la raza y su lugar de origen entre paréntesis. Los objetos de esta clase deberán ordenarse por defecto por orden alfabético del nombre propio del perro.

2) Creación de lista ordenada de perros famosos (1 punto)

Se solicita un método estático **crearPerros** (en UtilidadesExamen) que cree y devuelva una **lista** con cada uno de los siguientes perros famosos:

nombrePropio	**raza**	**lugarOrigen**	**motivoFama**	**fechaNacimiento**
Hachiko	Akita Inu	Tokio (Japón)	Esperó a su dueño en la estación hasta su muerte	1923-11-10
Snoopy	Beagle	Santa Rosa (EE.UU.)	Personaje del cómic ‘Peanuts’	1950-10-02
Gordo	Mastín	León (España)	Perro rescatista de montaña	2008-01-01
Barry	Mastín	Alpes suizos	Salvó más de 40 vidas como perro de rescate	1800-01-01
Rex	Pastor Alemán	Viena (Austria)	Protagonista de la serie ‘Rex, un policía diferente’	1994-01-01

Chips	Pastor Alemán	Nueva York (EE.UU.)	Héroe de guerra condecorado en la Segunda Guerra Mundial	1940-01-01
Stubby	Bulldog	Connecticut (EE.UU.)	Perro héroe en la Primera Guerra Mundial	1916-01-01
Bruiser	Chihuahua	EE.UU.	Compañero de Elle en 'Una rubia muy legal'	2000-01-01
Smoky	Yorkshire Terrier	Papúa Nueva Guinea	Perro mensajero en la Segunda Guerra Mundial	1943-01-01

NOTA: No creéis un objeto RazaPerro para dar de alta cada instancia de PerroFamoso. 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) Implementa los siguientes métodos utilizando streams (2.5 puntos)

Partiendo de la lista de perros creada en el apartado anterior (y pasándola como parámetro), crea los siguientes métodos:

a) **public static List<String> nombresDeRaza(List<PerroFamoso> listaPerros, String raza)**
Devuelve los nombres propios de todos los perros cuya raza contiene ese texto (ignora mayúsculas).

b) **public static Map<String, Long> contarPorRaza(List<PerroFamoso> listaPerros)**
Devuelve un Map que agrupe cuántos perros hay por raza.

c) **public static List<PerroFamoso> nacidosAntesDe(List<PerroFamoso> listaPerros, int anyo)**
Devuelve una lista con los perros cuya fechaNacimiento sea anterior al año indicado.

d) **public static long contarPosterioresA(List<PerroFamoso> listaPerros, int anyo)**
Devuelve el número total de perros famosos nacidos a partir de un año dado (inclusive).

e) **public static List<PerroFamoso> ordenadosLimitados(List<PerroFamoso> listaPerros, String texto, int n)**
Filtra el listado de perros por aquellos que su motivoFama contenga el texto que se pasa de entrada (ignorando mayúsculas y minúsculas), ordénalos por nombre de raza y quédate con los n primeros.

Haz una llamada a cada uno de estos métodos desde el método main de Main.

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