PROGRAMACIÓN

Examen Práctico 3ª evaluación

11 de mayo de 2023

Nombre y Apellidos: _____

Grupo: _____

PROGRAMACIÓN Examen Práctico 3ª evaluación 11 de mayo de 2023 1
INSTRUCCIONES 2
Generales 2
Código 3
Ejercicio 1: Contador de palabras y letras en un fichero 4
Instrucciones 4
Pistas 5
Ejemplos de ejecución 6
Ejercicio 2: Zoológico 7
Instrucciones 7
Diagrama UML 8
Ejemplo de ejecución 9
Ejercicio 3: Apuestas Euromillones 11
Instrucciones 11
Pistas 12
Ejemplo de ejecución 13

INSTRUCCIONES

Generales

AVISO: El incumplimiento de alguna de estas instrucciones supondrá suspender el examen

1. El **móvil** deberá estar **en silencio y en modo avión encima de la mesa con la pantalla hacia abajo**.
2. Está prohibido llevar ningún **tipo de reloj digital, smartwatch o pulsera similar**. Tenéis la hora en el ordenador y la podría proyectar desde el ordenador del profesor si hiciese falta.
3. Aquellas personas que tengan el pelo largo, deberán recogérselo para tener **las orejas a la vista**. Huelga decir que no se podrá llevar ningún tipo de auricular.
4. Se recomienda encarecidamente traer **tapones para los oídos** por si hubiese ruido en el patio, para que os podáis aislar y concentrar en los exámenes. Con el calor que viene, cerrar las ventanas no va a ser posible.
5. Se debe **grabar la pantalla del ordenador**, del aula o portátil, durante la totalidad del examen. Para ello se recomienda utilizar el software OBS Studio. **Cada estudiante se debe responsabilizar** de custodiar la **grabación** en vídeo de su pantalla durante la realización de **todos los exámenes, teóricos y prácticos**. La grabación es responsabilidad del estudiante y no proporcionar el vídeo de la grabación supondrá suspender el examen.
6. **No se permite el acceso a internet**. Esto incluye, entre otras muchas cosas, que no se permite hacer **búsquedas en Google** ni acceder a **foros** como **StackOverflow** o similares. En caso de duda, se debe preguntar al profesor antes de abrir cualquier aplicación o web que salga del entorno del cuestionario del aula virtual.
   Las únicas excepciones son:
   1. consultar el API de java (javadoc)
   2. probar expresiones regulares en la página regex101

7. **No se permite** consultar apuntes, resúmenes de teoría, ejemplos ni ejercicios.
8. Debe mostrarse, al inicio de la grabación del examen, el **listado de extensiones instaladas en el IDE**.
9. Se recuerda la **prohibición** de usar ninguna **extensión en el IDE** que provea funcionalidad basada en **IA**, como *IntelliCode* o *GitHub Copilot*.
10. **No se permite** consultar apuntes, resúmenes de teoría, ejemplos ni ejercicios.

Código

AVISO: El incumplimiento de alguna de estas instrucciones puede suponer la pérdida de hasta 1'5 puntos

1. El código debe estar **bien indentado**, se recomienda utilizar la opción del IDE de *formatear documento*
2. Las variables y clases deben tener **nombres claros** que identifiquen lo que hacen
3. No debe haber *números mágicos* ni *cadenas de caracteres mágicas*: se debe utilizar variables para almacenar esos números o cadenas. Si pueden declararse como *final* y/o como *static*, se debe hacer.
4. No se debe *abusar de la autogeneración* de código por parte del IDE: aquellos métodos que no se usen, no deben implementarse.
5. Se debe **respetar la estructura de directorios/packages**. Cualquier clase auxiliar que se necesite crear, debe incluirse en el package correspondiente de cada ejercicio: *programacion.examen.ej1*, *programacion.examen.ej2* y *programacion.examen.ej3*. Se pueden crear sub-paquetes dentro de los proporcionados si se desea, aunque no lo creo necesario.
6. Algunos ficheros Java se han dejado a modo de pista de las clases que habría que implementar, pero están vacíos. Os puede ayudar borrar el fichero vacío y volver a crearlo para que el IDE os autogenere código.
7. El código debe compilar sin generar errores ni warnings.

Ejercicio 1: Contador de palabras y letras en un fichero

Queremos hacer un programa que cuente las palabras y las letras que hay en un fichero de texto y muestre las 6 palabras y las 6 letras con más ocurrencias. El programa *MainEjercicio1.java* debe leer el fichero recibido como primer argumento por el *main* (el fichero será *input1.txt* y la ruta será relativa, ejemplos de ejecución más abajo):

Instrucciones

1. Para evitar problemas de tildes y de la 'ñ', el texto en *input1.txt* está en inglés.
2. Cuenta el número de veces que aparece **cada palabra** usando un Map. Convierte las palabras a **minúsculas** (lower-case) para que, si aparece la palabra "The" y "the", se cuente como 2 ocurrencias.
3. Cuenta el número de veces que aparece **cada letra** usando un Map. Convierte las letras a **minúsculas** por el mismo motivo.
   Sólo hará falta contar las letras. Se ignorarán los espacios en blanco, los dígitos y los signos de puntuación, sólo se cuentan las letras del alfabeto.
4. **Imprime las 6 palabras** que más veces han aparecido en el fichero.
5. **Imprime las 6 letras** que más han aparecido en el fichero.
6. Se recomienda implementar la lectura, conteo de palabras e impresión de las 6 con más ocurrencias en el método
   void contarPalabras(String rutaFichero).
7. Se recomienda implementar la lectura, conteo de letras e impresión de las 6 con más ocurrencias en el método
   void contarLetras(String rutaFichero).
8. **Se deben capturar todas las excepciones**.

Pistas

1. Para convertir una palabra a minúsculas, mira el método String.toLowerCase()
2. Para convertir un caracter a minúsculas, mira el método Character.toLowerCase(char)
3. Para comprobar si un caracter es alfabético, mira el método Character.isAlphabetic(int)
4. Si tenemos un mapa cuya clave es un Character y el valor asociado un Integer, podríamos necesitar crear una Clase que represente cada entrada del mapa, cuyos atributos fuesen la letra y el contador. Si esos objetos los almacenamos en una lista, después podríamos ordenar la lista con el criterio que queramos.
5. Recuerda que para ordenar una lista podemos utilizar el método Collections.sort(lista)
6. Si queremos que una lista se ordene por el orden reverso al orden natural, podemos utilizar Colections.sort(lista, *Comparator.reverseOrder()*)
7. Podéis abrir el fichero input1.txt con el IDE y buscar letras o palabras para comprobar que vuestro programa está contando bien el número de ocurrencias

Ejemplos de ejecución

```
$ javac MainEjercicio1.java
$ java MainEjercicio1
No se ha recibido ningún argumento. Debe proporcionar la ruta relativa al fichero

$ java MainEjercicio1 FICHERO_QUE_NO_EXISTE
FileNotFoundException capturada: FICHERO_QUE_NO_EXISTE (No such file or directory)
No se ha podido abrir el fichero 'FICHERO_QUE_NO_EXISTE'
FileNotFoundException capturada: FICHERO_QUE_NO_EXISTE (No such file or directory)
No se ha podido abrir el fichero 'FICHERO_QUE_NO_EXISTE'

$ java MainEjercicio1 input1.txt
Imprimimos las 6 palabras con más ocurrencias:
1) the -> 6
2) lorem -> 4
3) ipsum -> 4
4) of -> 4
5) it -> 3
6) and -> 3
Imprimimos las 6 letras con más ocurrencias:
1) e -> 59
2) t -> 43
3) s -> 39
4) i -> 38
5) n -> 38
6) a -> 29
```

Ejercicio 2: Zoológico

Queremos diseñar una aplicación para gestionar un zoológico de animales.
El programa *MainEjercicio2.java* debe ejecutarse y mostrar una salida como la proporcionada más abajo. Para ello habrá que implementar todo el código que falta y corregir todos los posibles errores de compilación.

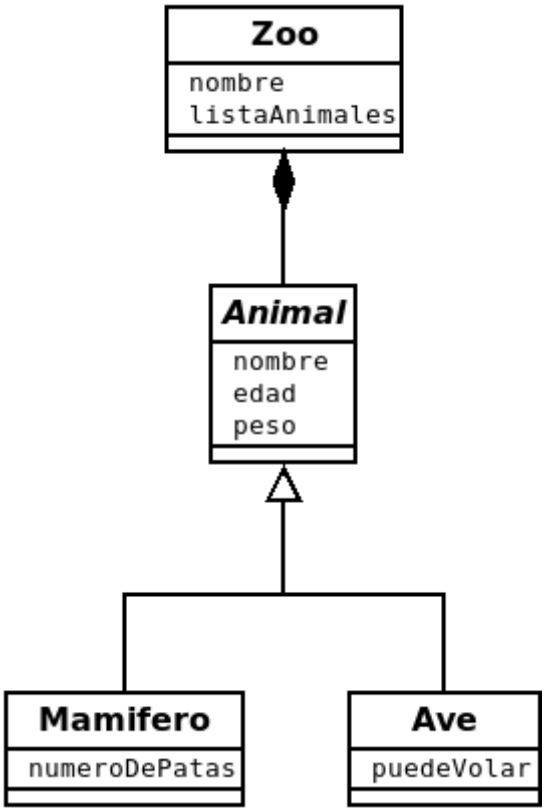
Instrucciones

1. Vamos a crear la clase Zoo que tenga como atributos el nombre del zoo y una lista de animales.
2. La clase abstracta Animal tendrá el nombre, la edad y el peso, y podrá ser de diferente tipo: mamífero o ave. El tipo lo reflejará con un atributo de tipo TipoAnimal, que será un enum.
3. Los **mamíferos**, además de todos los atributos heredados de Animal, tendrá como atributo especializado el **número de patas**.
4. Las **aves** tendrán como atributo especializado ***puedeVolar***, que reflejará si puede volar o no.
5. Los **constructores** de las clases Animal, Mamifero y Ave **lanzarán una excepción de tipo *ParametroInvalidoException***, con el mensaje correspondiente (ver ejemplo de ejecución), **en los siguientes casos**:
    1. El nombre está vacío
    2. La edad es menor que 0
    3. El peso es menor o igual que 0
    4. El número de patas es menor que 0
6. El **orden natural** de los animales se basa en comparar sus **nombres**. El resto de ordenaciones requerirán crear comparadores.
7. El método *toString() de la clase Ave* debe utilizar el **operador ternario** para imprimir "puede volar" o "NO puede volar".
8. **La clase Zoo**, además del método *toString()* que debe imprimir los animales por orden de inserción y cuya implementación se proporciona, **debe implementar los métodos**:
    1. *String toStringMamiferos()*, que imprimirá sólo aquellos animales cuyo tipo sea mamífero.

2. *String toStringAves()*, que imprimirá sólo aquellos animales cuyo tipo sea ave.
3. *String toStringOrdenEdad()*, que imprimirá todos los animales ordenados por su edad.
4. *String toStringOrdenPeso()*, que imprimirá todos los animales ordenados por su peso.
5. *String toStringOrdenNatural()*, que imprimirá todos los animales ordenados por su orden natural (basado en el nombre).

9. No se debe perder el orden de inserción de la lista de animales del zoo.
10. **Se deben capturar todas las excepciones**.

Diagrama UML



Ejemplo de ejecución

```
$ javac MainEjercicio2.java
$ java MainEjercicio2
Excepción capturada creando Ave: Nombre vacío
Excepción capturada creando Ave: Edad inválida: -1
Excepción capturada creando Mamífero: Peso inválido: 0.0
Excepción capturada creando Mamífero: Número de patas inválido:
-2

## ZOO: Zoo del Juan de la Cierva
# 4 animales:
MAMIFERO: cabra, 5 años, 8.50 kilos, 4 patas
MAMIFERO: ballena, 50 años, 35200.00 kilos, 0 patas
AVE: flamenco, 5 años, 6.30 kilos, puede volar
AVE: pingüino, 2 años, 9.50 kilos, NO puede volar

# Listado de Mamíferos:
MAMIFERO: cabra, 5 años, 8.50 kilos, 4 patas
MAMIFERO: ballena, 50 años, 35200.00 kilos, 0 patas

# Listado de Aves:
AVE: flamenco, 5 años, 6.30 kilos, puede volar
AVE: pingüino, 2 años, 9.50 kilos, NO puede volar

# Listado de animales por edad:
AVE: pingüino, 2 años, 9.50 kilos, NO puede volar
MAMIFERO: cabra, 5 años, 8.50 kilos, 4 patas
AVE: flamenco, 5 años, 6.30 kilos, puede volar
MAMIFERO: ballena, 50 años, 35200.00 kilos, 0 patas

# Listado de animales por peso:
AVE: flamenco, 5 años, 6.30 kilos, puede volar
MAMIFERO: cabra, 5 años, 8.50 kilos, 4 patas
```

```
AVE: pingüino, 2 años, 9.50 kilos, NO puede volar
MAMIFERO: ballena, 50 años, 35200.00 kilos, 0 patas

# Listado de animales por su orden natural:
MAMIFERO: ballena, 50 años, 35200.00 kilos, 0 patas
MAMIFERO: cabra, 5 años, 8.50 kilos, 4 patas
AVE: flamenco, 5 años, 6.30 kilos, puede volar
AVE: pingüino, 2 años, 9.50 kilos, NO puede volar

## ZOO: Zoo del Juan de la Cierva
# 4 animales:
MAMIFERO: cabra, 5 años, 8.50 kilos, 4 patas
MAMIFERO: ballena, 50 años, 35200.00 kilos, 0 patas
AVE: flamenco, 5 años, 6.30 kilos, puede volar
AVE: pingüino, 2 años, 9.50 kilos, NO puede volar
```

Ejercicio 3: Apuestas Euromillones

Queremos hacer una aplicación que registre apuestas de Euromillones.
El programa MainEjercicio3.java debe ejecutarse y mostrar una salida como la proporcionada más abajo. Para ello habrá que implementar todo el código que falta y corregir todos los posibles errores de compilación.
Además, cada vez que se genere una apuesta, se guardará en el fichero apuestas.txt.

Instrucciones

1. Cada **apuesta** se compone por **5 números diferentes**, entre el 1 y el 50, y por **2 estrellas diferentes**, que son números entre el 1 y el 12.
2. Cada jugada, independientemente del orden en el que se hayan introducido los números y las estrellas, mostrará los 5 números y las 2 estrellas **en orden ascendente**.
3. La clase *ApuestaEuromillones* tendrá los métodos:
    1. *void addNumero(int numero)*
    2. *void addEstrella(int numero)*
4. En caso de error, estos métodos lanzarán una excepción **no controlada/unchecked** con la descripción del error. Si no se sabe hacer una excepción del tipo solicitado, se valorará también, aunque con menor puntuación si se lanza una excepción normal.
5. Los posibles errores son:
    1. El número introducido no cumple las reglas de los números o de las estrellas de la apuesta de Euromillones
    2. El número o estrella introducido ya se había introducido anteriormente
    3. Ya se han introducido el máximo número de números: 5 para los números y 2 para las estrellas
6. Después de generar una apuesta completa, con 5 números y 2 estrellas, se escribirá la apuesta añadiendo una nueva línea al fichero *apuestas.txt*. Si le falta algún número a la apuesta, no se escribirá en el fichero. El método

boolean ApuestaEuromillones.escribirEnFichero() devolverá false si hay algún error o true si se escribe la línea en el fichero correctamente. Se espera que, en caso de error, se notifique imprimiendo algún mensaje por pantalla.

7. **Se deben capturar todas las excepciones**.

Pistas

1. Piensa en qué colección nos permite almacenar y acceder a los números con las características anteriores.

Ejemplo de ejecución

```
$ javac MainEjercicio3.java
$ java MainEjercicio3
Creando apuesta Euromillones...
Añadimos el número 50... OK
Añadimos el número 50... ERROR: El numero '50' ya se ha añadido
a la apuesta
Añadimos el número 10... OK
Añadimos el número -5... ERROR: El numero '-5' está fuera del
rango permitido [1,50]
Añadimos el número 51... ERROR: El numero '51' está fuera del
rango permitido [1,50]
Añadimos el número 0... ERROR: El numero '0' está fuera del
rango permitido [1,50]
Añadimos el número 7... OK
Añadimos el número 27... OK
Añadimos el número 43... OK
Añadimos el número 38... ERROR: La apuesta ya tiene 5 números
Añadimos la estrella -2... ERROR: La estrella '-2' está fuera
del rango permitido [1,12]
Añadimos la estrella 13... ERROR: La estrella '13' está fuera
del rango permitido [1,12]
Añadimos la estrella 12... OK
Añadimos la estrella 7... OK
Añadimos la estrella 1... ERROR: La apuesta ya tiene 2 estrellas
Apuesta Euromillones: Números: 7, 10, 27, 43, 50. Estrellas: 7,
12

$ cat apuestas.txt
Números: 7, 10, 27, 43, 50. Estrellas: 7, 12
```