Sesión 11. Resolver un examen final

Se necesita desarrollar una aplicación para leer los resultados de exámenes, asignar calificaciones y mostrarlas. Un examen está compuesto de varias preguntas de diferentes tipos: elección, rellenar-hueco y dar un valor.

- Una pregunta de **elección** (`choice`) tiene varias respuestas posibles (a,b,c,d) pero solo una es correcta.
- Una pregunta de **rellenar-hueco** (gap) es una pregunta con un hueco dentro de una frase. El estudiante debe rellenarlo con la palabra correcta.
- Una pregunta de **valor** (`value`) es aquella en la cual el estudiante realiza un cálculo y proporciona un resultado numérico (`double`).

Cada pregunta tiene un número indicando la posición de esa pregunta en el examen y una puntuación en la nota final del examen.

Se proporciona información en dos ficheros de texto. Un fichero tiene las preguntas del examen. El otro fichero tiene las respuestas de los estudiantes a las preguntas del examen.

El fichero *`exam.model.txt`* contiene una línea por cada pregunta de examen. Cada línea tiene tres campos: el tipo de pregunta, puntuación de la pregunta y la respuesta correcta. Los campos están separados por el carácter "\t". Nótese que el número de la pregunta vendrá dado por el número de línea en la que viene definida.

<tipo de pregunta>\t<puntuación>\t<respuesta correcta>

Un ejemplo de este tipo de fichero sería:

```
choice      1.0   a
choice      1.0   b
gap   0.5   stuff
gap   0.5   computer
value 1.5   12.5
value 1.5   100.0

...
```

El fichero *`answers.txt.gz`* (comprimido en formato gz) contiene una línea por cada entrega de un estudiante **(StudentExam)**. Cada línea tiene el mismo número de campos. El primero es la identificación del estudiante (String) seguido de las respuestas a cada pregunta del examen. Habrá siempre tantas respuestas por estudiante como preguntas haya en el examen y siempre en el mismo orden (el primer campo después de la identificación es la respuesta a la primera pregunta y así sucesivamente). El carácter "\t" separa los campos.

<identification>\t<respuesta a pregunta 1>\t<res… 2>\t<res… 3>\t<…>

El siguiente es un ejemplo de formato del fichero de respuestas:

```
20215 a     b     stuff       thread      13.5  80
20214 b     c     thing       computer    14.5  0
20213 c     d     matter      computer    12.5  10
```

El programa deberá cargar las preguntas, cargar las respuestas de los estudiantes y con ello tiene que calcular la nota a cada estudiante, creando una lista de notas, de acuerdo con las siguientes reglas:

- La nota final del estudiante es la suma de las notas de cada respuesta.
- Dependiendo del tipo de pregunta, se aplicarán los siguientes criterios para calcular la nota de una pregunta:
  - Para preguntas de elección (`choice`): si la respuesta es correcta, la nota es la puntuación de la pregunta. Si es incorrecta, la nota es el 20% de la puntuación en valor negativo (se resta el 20% de la puntuación).
  - Para preguntas de valor (`value`): la nota es la puntuación si la respuesta está en el rango [respuesta – 0,1 .. respuesta +0,1]. Cero si no lo está.
  - Para preguntas de rellenar-hueco (`gap`): la nota es la puntuación si la respuesta se rellena con las mismas palabras. Cero en caso contrario.

El resultado del proceso de asignación de notas es una lista de notas de los estudiantes. Cada nota (**StudentMark**) está compuesta de la identificación del estudiante más su valor de nota final. Más adelante, está lista será ordenada por nota y por identificación.

Desarrolla esta aplicación teniendo en cuenta los siguientes puntos:

1. Crea una jerarquía de clases para representar los tipos de Preguntas (Question).
2. Crea una clase para representar cada *entrega* de estudiante (StudentExam).
3. Crea una clase para representar cada nota de un estudiante (Student*Mark*).
4. Crea la clase *ExamMarker*. Tendrá la lista de preguntas recogidas del fichero exam.model.txt, la lista de respuestas *(StudentExam)* de los alumnos con las respuestas, y la lista de notas con las notas de los estudiantes. Esta lista se generará a través de una opción del usario después de haber cargado las preguntas y las entregas.
5. Realiza en la clase ExamMaker las siguientes operaciones:
   1. Cargar fichero con preguntas.
   2. Cargar fichero con exámenes
   3. Calcular notas de estudiantes (generando lista de notas). Método mark en ExamMaker.
   4. Devolver lista de preguntas
   5. Devolver lista de exámenes
   6. Devolver lista de notas
   7. Guardar notas en fichero markResults.txt
   8. Guardar notas en fichero comprimido.
   9. Cargar notas de fichero comprimido
   10. Ordenar lista de notas por uo (orden natural)
   11. Ordenar lista de notas por nota y a igual nota por uo
   12. Ordenar lista de exámenes por uo

1. Crea una clase `FileUtil` capaz de leer cada línea de un fichero de texto en una lista de String. Haz lo mismo para leer de un fichero zip en una lista de String.
2. Crea clases para procesar las líneas (lista de String) y crear los objetos concretos.
3. Implementa en ExamMaker un método *mark* que creará una lista de notas *(StudentMark)*.
4. Implementa las operaciones que faltan para cargar los datos en las listas y crear la lista con las notas.

5. La lista de notas puede ser ordenada de formas diferentes: por nota o por identificación del estudiante. Crea los Comparadores necesarios. Ordena por los dos criterios y muestra el resultado por consola.
6. Escribe la lista de *notas* en un fichero de texto llamado `results.txt`
7. Los ficheros podrían tener algún error de formato: líneas en blanco, números incorrectos, tipos de preguntas desconocidas o más o menos campos de los esperados. Haz el programa capaz de ignorar estas líneas con formato inválido**.**
8. Utiliza excepciones apropiadas para gestionar potenciales problemas con la gestión de los ficheros y otros tipos de errores inesperados.
9. Añade Junit para probar el método getMarkForAnswer() que calcula la nota en las preguntas y el método mark() que calcula la nota final (de la clase StudentExam).

Nota:

- Para facilitar las operaciones del *comparator* crea todos los atributos de tipo double usando la clase Double.
- Debes usar las colecciones de Java.
- Se proporcionan los siguientes ficheros:
  - exam.model.txt
  - answers.txt.gz
  - Una clase simplificada con un método main().
- Se necesita obtener un mínimo en estos apartados para aprobar el examen:
  - Manejo de excepciones
  - Polimorfismo
  - Pruebas unitarias.

Para ordenar una `Lista` se puede implementar el algoritmo de ordenación visto en clase o usar el método de ordenación sort de la clase `java.util.Collections` de la siguiente forma:

1.- Primero importar la clase Collections

```
import java.util.Collections;
```

2.- Llamar al método estático `sort` de la clase `Collections` en el método de ordenación correspondiente:

```
Collections.sort(lista,comparador);
```

Siendo lista en este caso la lista de elementos a ordenar y comparador el objeto comparador que implementa el criterio de ordenación. Este método ordena la lista que se le pasa según el método `compare` (T t1, T t2) del comparador especificado.