# Gestión de notas de exámenes

## 1. Introducción

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 y la respuesta correcta.

## 2. Ficheros de la aplicación

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.

1. El fichero *`questions.txt`* describe el examen y contiene una línea por cada pregunta del mismo. 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:

```
choice1.0   a
choice1.0   b
gap   0.5   stuff
gap   0.5   computer
value 1.5   12.5
value 1.5   100.0

...
```

2. El fichero *`answers.gz`* (comprimido en formato gz) recoge, en diversas líneas, todas las respuestas de todos los estudiantes; esto es, cada línea corresponde a las respuestas 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
```

## 3. Objetivo

El programa deberá cargar las preguntas del examen, cargar las respuestas de los estudiantes y con ello tiene que calcular la nota de 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 al examen.
- 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 será 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 será la puntuación de la pregunta en el caso de que la respuesta esté en el rango [respuesta correcta– 0,1 ... respuesta correcta +0,1] (es decir, se permite una desviación de 0,1 sobre el valor correcto). Si no lo está, será 0.
  - Para preguntas de rellenar-hueco (`gap`): la nota será la puntuación de la pregunta en el caso de que la respuesta coincida con la palabra correcta. Será 0 en caso contrario.

El resultado del proceso de asignación de notas es una **lista de notas** de los estudiantes. Cada nota (**StudentMark**) estará compuesta por la identificación del estudiante más el valor correspondiente a su nota final.

## 4. Algoritmo para el cálculo de notas

- La clase ExamMarker contine un metodo mark() para calcular las notas de todos los estudiantes. Esto es, este método itera por la lista de exámenes generando una instancia de la clase StudentMark para cada uno.
- La clase StudentExam será responsable de calcular la nota individual de cada estudiante sumando las puntuaciones de cada respuesta.
- Finalmente, la clase Question tiene la responsabilidad de generar la nota de cada pregunta.

Considera los siguientes fragmentos de código:

```
class ExamMarker {
   void mark() {
      for (StudentExam se: studentExams) {
        this.marks.add(new StudentMark ( se.getId(), se.mark(questions)));
      }
    }
...
}
```

```
class StudentExam
      private String id = ...

      StudentMark mark(List<Question> questions) {
         double value = 0.0;
         for(int i = 0; i < questions.size(); i++) {
            Question q = questions.get( i );
            String answer = answers.get( i );
            value += q.mark( answer );
          }
         return value;
       }
}

class Question {
      public abstract double mark(String answer);
}
```

## 5. Escritura de notas a un fichero

Las notas finales pueden ser escritas en un fichero de texto (marks.txt) en orden ascendente de identificador de estudiante. El format de cada línea en este fichero de salida debe ser:

student_id-->nota

y contendrá exáctamente tantas lines como respuestas, un estudiante por línea.

## 6. Guía para desarrollar la aplicación

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 examen de un estudiante (*StudentExam*).
3. Crea una clase para representar cada *nota* de un estudiante (*StudentMark*).
4. Crea la clase *ExamMarker*. Tendrá la lista de preguntas (*Question*) recogidas del fichero *questions.txt*, la lista de respuestas *(StudentExam)* de los alumnos, y la lista de notas (*StudentMark*) de los estudiantes. Esta última lista se generará a través de una operación después de haber cargado las preguntas y las entregas (exámenes) de los estudiantes.
5. Realiza en la clase *ExamMarker* las siguientes operaciones:
   - Cargar fichero con las preguntas del examen (*questions.txt*)
   - Cargar fichero con las respuestas de los alumnos (*answers.gz*)
   - Calcular las notas de los estudiantes, generando la lista de notas (método mark)
   - Devolver la lista de preguntas
   - Devolver la lista de respuestas
   - Devolver la lista de notas
   - Devolver la nota correspondiente al identificador de un alumno
   - Guardar las notas de todos los alumnos en el fichero *results.txt*
   - Guardar las notas de todos los alumnos en el fichero comprimido *results.gz*
   - Ordenar la lista de notas por identificador (orden natural)
   - Ordenar la lista de notas por nota y, a igual nota, por identificador
   - Ordenar lista de exámenes por identificador

Para la realización de estas operaciones:

6. 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 comprimido en una lista de String.
7. Crea las clases oportunas para analizar las líneas (lista de String) y crear los objetos concretos de las clases Question y StudentExam.
8. Implementa en *ExamMarker* un método *mark* que creará una lista de notas *(StudentMark)*.
9. Crea las clases oportunas para serializar las notas antes de guardarlas en el fichero.

## 7. Validaciones

Se deben validar parámetros en todos los métodos públicos. Se lanzará una `IllegalArgumentException`, con un mensaje apropiado, cuando un método recibe un parámetro inválido. Se consideran los siguientes casos:

- Todos los parámetros deben ser distintos de null, incluidas las listas.
- Los Strings, además deben ser distintos de blanco.
- En el constructor de Question, no pueden ser negativos, ni el peso, ni el número de pregunta.
- En el constructor de StudentExam, el identificador debe ser de longitud 6

## 8. Manejo de Excepciones

**1. Errores del parser**

El fichero de preguntas (questions.txt) podría tener algún error de formato:

- líneas en blanco
- números incorrectos
- tipos de preguntas desconocidas
- más o menos campos de los esperados

Haz que el programa sea capaz de ignorar estas líneas con formato inválido de forma que se continúe analizando el resto de líneas**.** Además, el error concreto y el número de línea en la que se ha producido éste, han de grabarse en un fichero de log con el siguiente formato:

```
INVALID LINE <line-number>: <message>
```

Se supone que el fichero de respuestas (answers.txt) es correcto: no hay errores de sintaxis y para todos los alumnos, el número de respuestas coincide con el número de preguntas del examen.

**2. Errores de usuario**

2.1. Respuestas del mismo alumno: si en el proceso de carga de las respuestas de los alumnos se encuentra un identificador de alumno que ya se ha cargado previamente, el programa debe generar el mensaje informativo "`Entrega repetida del alumno <identifier> `".
2.2. Buscar nota de un alumno que no existe: gestiona la posible excepción cuando se intenta buscar la nota de un alumno que no existe. En este caso, únicamente se deberá generar el mensaje informativo "`Nota no encontrada o estudiante desconocido`".

2.3 Fichero no encontrado: en este caso el programa debe generar el mensaje informativo `"Fichero no encontrado"`.

En estos tres casos, el programa debe imprimir `ERROR: <mensaje> Ejecute de nuevo`. Además, el programa **debe finalizar**.

3. **Errores de programación y del sistema**
   Si se produce alguno de estos errores se deberá mostrar el mensaje informativo al usuario "`ERROR IRRECUPERABLE: <mensaje>`, grabar el mensaje y la pila de ejecución en el fichero de log. Además, el programa debe finalizar.

## 9. Ordenación

La lista de notas debe ser ordenada de tres formas diferentes: por orden natural (identificación del estudiante), por nota ascendente y por nota descendente; en el caso de igual nota, por identificación. Añade los métodos necesarios para ordenar por los tres criterios. Añade estos tres casos al simulador, de forma que se vea el resultado por consola.

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`.

## 10. Pruebas

Prueba los siguientes métodos:

- método parse de la clase Question
- método loadQuestion de ExamMarker
- método mark de la clase Question
- método serialize de la clase ExamMarks

**Notas:**

- Para facilitar las operaciones del *comparator* con atributos de tipo *double,* usa el método *compareTo* de la clase *Double*.
- Se necesita obtener un mínimo en estos apartados para aprobar el examen:
  - Polimorfismo
  - Manejo de excepciones
  - Pruebas unitarias.