ESTRUCTURAS DE DATOS Y ALGORITMOS I

GRADO EN INGENIERÍA INFORMÁTICA (Segundo Curso, Primer Cuatrimestre)

Primer Examen Parcial (04/12/2014)

Práctica

Vamos a implementar un programa para la gestión de Alumnos matriculados en EDA-III (curso 14/15). Por cada alumno, además de su nombre, primer apellido y segundo apellido (si procede), vamos a gestionar las notas obtenidas durante las **n** pruebas teórico/prácticas realizadas durante el curso académico.

Los datos de los alumnos y calificaciones se encuentran en el archivo de entrada *alumnos.txt*, cuyo formato es el que se muestra a continuación

_____

Nombre Apellido1 Apellido2

$Nota_1$ $Nota_2$ – $Nota_4$

….

_____

La ausencia de nota en una prueba determinada (no se ha presentado) se denota con el símbolo – (en el ejemplo, no existe ninguna nota en la 3ª prueba). A efectos de cómputos se considerará como un 0…aunque en la estructura interna se deberá reflejar tal circunstancia con un valor **null**.

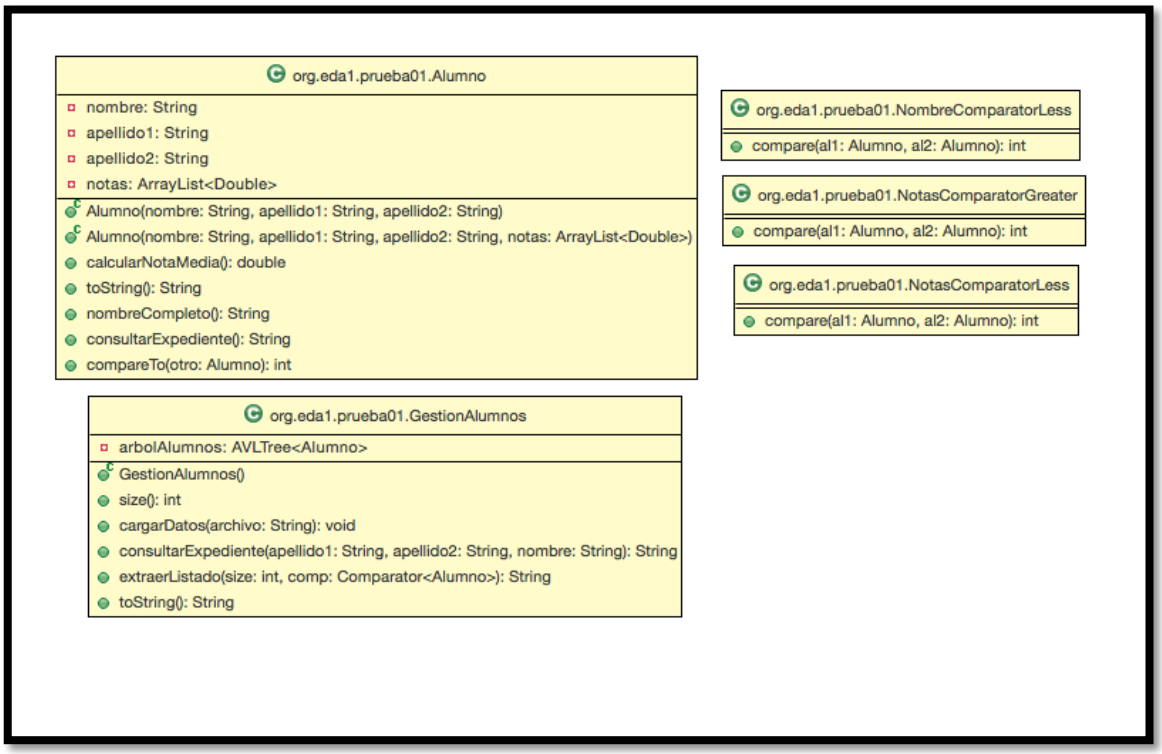
Aunque en el archivo de entrada el ORDEN es Nombre, Apellido1, Apellido2, en la práctica se suelen realizar las búsquedas por apellidos. Así pues, los datos deberán almacenarse en una estructura eficiente tipo AVL siguiendo como criterio de orden natural: APELLIDO1 → APELLIDO2 → NOMBRE.

Una vez cargados los datos en memoria, las operaciones de consulta más usuales que se podrán realizar son las siguientes:

- Listado de alumnos siguiendo el orden natural en el que se indique el nombre completo y la nota media. Por ejemplo:
  LOPEZ LOPEZ EMILIANO (5.00)
- Consultar expediente de un alumno (puede que no exista). Por ejemplo:
  LOPEZ LOPEZ EMILIANO --> [10.00, No presentado, 10.00, No Presentado]
- Listado de alumnos siguiendo el orden: NOMBRE → APELLIDO1 → APELLIDO2
- Listado de los k mejores expedientes (mejores notas medias).
- Listado de los k peores expedientes (peores notas medias).

Para mayor detalle sobre el formato de salida y criterios de implementación, véase el test (GestionAlumnosTestJUnit4) que se adjunta en el paquete org.eda1.prueba01.

Para la solución del problema, se propone el siguiente diseño de clases:



A continuación, destacamos algunos aspectos de entrega, localización de repositorios e implementación de la solución:

1) Repositorio de la asignatura:

http://gipp.ual.es:9090/svn/practicaseda2015

2) Repositorio del alumno

http://gipp.ual.es:9090/svn/apellido1apellido2nombre

3) Formateo de números decimales (tal y como se especifica en el test, sólo mostraremos las notas con dos decimales):

Cadena ←-String.*format*(Locale.*US*, "%.2f", valorDouble)

Por ejemplo, si valorDouble = 2.225, la salid sería la cadena “2.23”.

4) Entrega del Ejercicio

En el **Escritorio** del PC, deberá existir una carpeta con vuestro nombre, en el que se encontrarán las 5 clases solución debidamente implementadas (test verde).

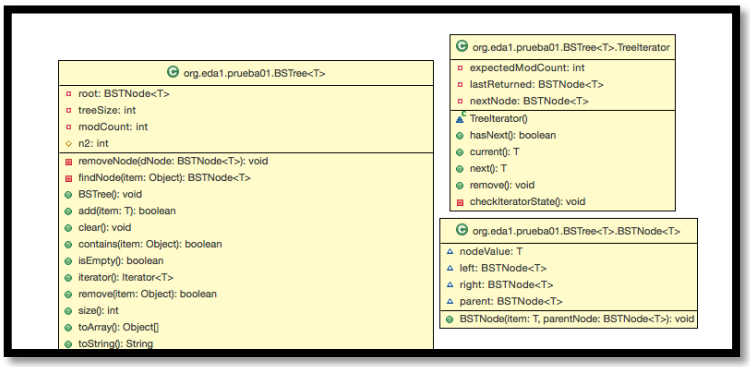
ESTRUCTURAS DE DATOS Y ALGORITMOS I

GRADO EN INGENIERÍA INFORMÁTICA (Segundo Curso, Primer Cuatrimestre)

Primer Examen Parcial (04/12/2014)

Teoría

Ejercicio 1) (8p) Dada la estructura BSTree<Integer> que se muestra a continuación, y partiendo del método *displayTree()* implementado en clase, vamos a implementar una versión del algoritmo (recursivo), al que llamaremos *displayTreeAsterisk(int) (versión pública y privada)*, que mostrará de la misma forma la misma estructura del árbol, pero sustituyendo el valor del nodo por el símbolo * en aquellos nodos que se encuentren en un determinado nivel especificado por el usuario.



Por ejemplo, dado el siguiente fragmento de código:

```
BSTree<Integer> arbol = new BSTree<Integer>();
arbol.add(50);
arbol.add(20);
arbol.add(60);
arbol.add(10);
arbol.add(25);
arbol.add(90);
System.out.println("displayTree: \n" + arbol.displayTree());
System.out.println("displayTreeAsterisk \n" + arbol.displayTreeAsterisk(1));
```

La salida por consola deberá ser:

```
displayTree:
          (90)[2]
     (60)[1]
(50)[0]
          (25)[2]
     (20)[1]
          (10)[2]
```

```
displayTreeAsterisk:
          (90)[2]
     (*)[1]
(50)[0]
          (25)[2]
     (*)[1]
          (10)[2]
```

En este caso, sustituimos por * los valores de los nodos contenidos en el nivel 1.

Para comprobar el correcto funcionamiento del método, y partiendo de la estructura del ejemplo, realice un seguimiento del método recursivo de forma exhaustiva, tal y como se ha realizado en clase.

Ejercicio 2) (2p) Dibuje, paso a paso e indicando el proceso de razonamiento, los árboles BSTree y AVLTree (inicialmente vacíos), que se generan a partir de la secuencia de datos de entrada que se especifica a continuación:

+ 22 + 20 + 18 + 16 + 14 + 2 + 4 + 6 + 8 + 10 + 12 - 16