

## ESTRUCTURAS DE DATOS Y ALGORITMOS I

### Grado en Ingeniería Informática (Segundo Curso, Primer Cuatrimestre)

### Primer Examen Parcial Grupo A. 21-11-2013

**Ejercicio Heap.** En teoría hemos estudiado los *heap* (montículos) máximos y mínimos, y en prácticas hemos hecho uso de dicha implementación en la clase `class Heap<T>`. En dicha implementación hemos utilizado un `ArrayList<T> theHeap`, que tiene la raíz en la posición 0. Para obtener un heap mínimo de enteros (`Integer`) hemos declarado un *Comparator* de la siguiente forma `Less<Integer> less = new Less<Integer>()`, y dicho *heap mínimo* lo hemos declarado de la siguiente manera `Heap<Integer> heap = new Heap<Integer>(less)`. En este ejercicio se pide implementar en Java una función *bottom-up* (`ArrayList<Integer> sumValuesOfBranches()`) que devuelva (en un `ArrayList` de `Integer`) la suma del valor de los nodos que tiene el *heap* en cada una de sus ramas.

**Ejercicio ABB AVL.** Uno de los mecanismos más sencillos que se utilizan en los sistemas de recuperación de información textual consiste en asociar a cada elemento de texto (documento) una **Tabla de Frecuencias**. En esta tabla aparecen de forma ordenada las palabras encontradas en el texto junto con su frecuencia de aparición (número de ocurrencias). El objetivo de este ejercicio es utilizar un **AVLTree** (ABB AVL) para establecer un mecanismo eficiente de asociación entre un conjunto de datos de entrada (cadenas de caracteres `String`) procedentes de un texto y su correspondiente tabla de frecuencias. Es decir, utilizar el ABB AVL como estructura de datos para almacenar la información de una Tabla de Frecuencias, donde en cada nodo vamos a almacenar pares (Palabra, Frecuencia).

org.eda1.examenParcialGrupoA.ejercicio02.WordFrequency
<ul style="list-style-type: none"><li>word: String</li><li>frequency: int</li></ul>
<ul style="list-style-type: none"><li>WordFrequency(w: String, f: int)</li><li>WordFrequency(w: String)</li><li>WordFrequency()</li><li>setWord(w: String): void</li><li>getWord(): String</li><li>setFrequency(f: int): void</li><li>getFrequency(): int</li><li>incrementFrequency(): void</li><li>compareTo(other: Object): int</li><li>equals(obj: Object): boolean</li><li>toString(): String</li></ul>

org.eda1.examenParcialGrupoA.ejercicio02.ProcessText
<ul style="list-style-type: none"><li>avlWords: AVLTree&lt;WordFrequency&gt;</li></ul>
<ul style="list-style-type: none"><li>ProcessText()</li><li>ProcessText(avlW: AVLTree&lt;WordFrequency&gt;)</li><li>loadFile(file: String): AVLTree&lt;WordFrequency&gt;</li><li>addWord(word: String): boolean</li><li>removeSpecialCharacters(input: String): String</li><li>getFrequencyOfWord(word: String): int</li><li>getWordsWithFrequency(frec: int): ArrayList&lt;String&gt;</li></ul>

Como material adicional se podrá utilizar la siguiente función, que elimina de un elemento del texto, todos los caracteres especiales.

```
public String removeSpecialCharacters(String input) {  
    String specialChart = ",.:;_()[ ]{ }<>*+ -/= %&$|'\"^?;!;0123456789";  
    String output = input;  
    for (int i = 0; i < specialChart.length(); i++) {  
        output = output.replace(String.valueOf(specialChart.charAt(i)), "");  
    }  
    return output.toLowerCase();  
}
```

Para cada ejercicio se proporciona el correspondiente test que se deberá de pasar correctamente.