Técnicas Actuales de Programación 2020/2021
Examen Convocatoria Extraordinaria (25.06.2021)

Responda al examen con un proyecto que lleve su nombre (si usa Netbeans elija Java with Ant -> Java Application). Cada ejercicio deberá resolverlo en un package diferente (elija como su espacio de nombres edu.upvehu.tap.<usted>).

1- Jerarquía de clases (puntuación 4)

Una tienda de componentes electrónicos necesita una aplicación Java para gestionar sus existencias. Como arranque, se plantea escribir el código de unas pocas clases con una definición muy limitada para las mismas. Se pretende lo siguiente:

- *Estructura*: La clase raíz de los productos a la venta se llamará **ComponenteElectronico** y de entrada sólo contará con el precio y el número de existencias. Por debajo tendremos las clases **Analogico** y **Digital**, donde la primera tiene a su vez dos hijas: **Resistencia** y **Condensador**, y la segunda tres: **Microprocesador**, **Combinacional** y **Memoria**. Los objetos de las dos primeras quedarán respectivamente definidos por los valores de sus ohmios y faradios, mientras que todos los objetos de tipo **Digital** tendrán como característica únicamente un identificador (**String**).

- *Getters/setters*: todos los campos estarán encapsulados. Los precios y existencias podrán consultarse y alterarse, mientras que el resto de valores sólo podrán ser consultados.

- *Instanciación*: Deberán escribirse los constructores que permitan instanciar objetos de todas las clases más profundas aportando sus valores específicos, en dos versiones: con y sin aportación de precio y existencias (si no se aportan su valor será cero). No se deberán poder generar objetos de las clases que tienen hijas.

- *Sobreescrituras*: se reescribirán los métodos **toString()** y **equals()** en todas aquellas clases que tenga sentido.

- *Interfaces*: Se contempla también que las resistencias sean comparables entre sí (por sus ohmios), así como los condensadores (por sus faradios), para lo cual implementarán el interfaz **Comparable** aplicable genéricamente sobre sí mismas

- *Comprobación (colecciones):* Para comprobar el funcionamiento, se escribirá una clase con un **main()** que generará dos pequeños conjuntos ordenados de resistencias y condensadores con valores aleatorios, y una lista, también reducida, conteniendo todo tipo de objetos de las subclases de Digital con identificadores enteros aleatorios, y posteriormente mostrará todo ello en pantalla.

2- Una clase/Un algoritmo (puntuación 6)

Dado un texto que versa sobre una temática concreta, podemos clasificar las palabras como más o menos significativas dentro de dicha temática comparando su frecuencia de aparición en el texto frente a su frecuencia de aparición en el lenguaje en general (en la práctica en un texto muy amplio que verse sobre la mayor cantidad de temas posibles).

Vamos a realizar una clase con métodos estáticos que nos permitan realizar esta tarea frente a dos ficheros concretos en una rutina "main(.)" de prueba.

Los ficheros con los textos a utilizar, codificados en utf-8 y consistentes en palabras separadas por espacios en blanco, se encuentran en http://gtts.ehu.es/Mikel/tmp/macrotexto.txt y http://gtts.ehu.es/Mikel/tmp/minitexto.txt

Para ello necesitaremos

• Una función **histogram(.)** que recoja como parámetros la URL del fichero de texto y su codificación, y devolverá un Map (el histograma), cuyas claves sean las palabras del texto y sus valores el correspondiente número de apariciones de cada palabra.

> *Si no sabe trabajar con URLs puede descargar los archivos y hacer el método para trabajar con ficheros locales. Si por el contrario sí sabe hacerlo y tiene tiempo de sobra, puede escribir las dos versiones, pero ¡ojo!, no sea redundante.*

• Una función **normalize(.)** que recibe un histograma de cuentas y devuelve el histograma de frecuencias (normalizado frente a la cuanta total). Es decir, Las claves serán las mismas palabras y los valores las frecuencias de aparición correspondientes (valor original/suma de todos los valores originales).

Con ayuda de estas dos funciones podremos escribir la que nos interesa:

• **relevantWords(.)** que tome como parámetros dos URLs (o dos nombres de ficheros) y, ayudado de las rutinas anteiores, devuelva una lista de objetos (palabra-relevancia) ordenados según su "relevancia", valor que obedece a la siguiente fórmula.

$$relevancia(w) = \frac{frecuencia_{histograma2}(w)}{\max\{1e-7, frecuencia_{histograma1}(w)\}} \qquad \forall w \in Texto2$$

Finalmente, en un main() llamaremos a este último método con los ficheros indicados y mostraremos el resultado por pantalla.