Curso 2015-2016
Estructura de Datos y Algoritmia
Práctica 1 (14 de septiembre)

lsi Departamento de Lenguajes y Sistemas Informáticos

Objetivo

Esta primera práctica nos servirá para procesar información de ficheros de texto (entrada-salida) para nuestros diferentes objetivos a lo largo del curso. En particular, queremos construir un diccionario para traducir inglés a otras lenguas, y para ellos iremos revisando y utilizando diferentes estructuras de datos en las siguientes prácticas que nos servirán para almacenar la información necesaria. Además, con estos diccionarios podremos realizar diferentes aplicaciones.

En particular:

1. Implementación y uso de clases para gestionar la entrada y salida de información a partir de ficheros de texto.
2. Implementación de estructuras de datos para el almacenamiento interno de la información leída desde fichero.
3. Implementación de aplicaciones concretas con las clases definidas.

Plazo de entrega: desde el jueves 1 de octubre hasta el **miércoles 7 de octubre de 2015**.
Más información al final de este documento.

1. Entrada-salida de información desde ficheros de texto

El modelo *Diccionario* se define como un conjunto de datos dinámico, es decir que soporta las operaciones básicas de inserción y borrado de elementos y en el que la operación más característica es la de búsqueda. Un modelo de estas características se puede implementar con diferentes estructuras de datos, como por ejemplo los tipos lineales (vectores y listas) o los tipos no lineales (árboles de búsqueda o tablas hash). Si se puede establecer una relación de orden total entre sus elementos, observaremos

que las operaciones de diccionario que se realizan con las diferentes estructuras de datos tendrán diferente coste [1].

Los diccionarios con los que trabajaremos en estas prácticas son de traducción de inglés a varias lenguas, con el objetivo de realizar traducciones de todas ellas simultáneamente o por parejas. Una forma de optimizar la búsqueda en los diccionarios es mantenerlos de forma ordenada. Crearemos diccionarios con diferentes estructuras de datos que en algunos casos nos permitirán mantener esta ordenación independientemente de las operaciones que realicemos y en otros esta ordenación la tendremos que mantener nosotros.

Además de los diccionarios, también dispondremos de textos sencillos para traducir en algunas aplicaciones. Ambos tipos de información se leerán de ficheros de texto.

La información asociada a los diccionarios será la siguiente:

- número de lenguas a traducir en el diccionario
- iniciales de las lenguas (por ejemplo, E para español, F para francés, etc)
- en cada una de las líneas
  - palabras o expresiones en las diferentes lenguas, separadas por " * "; el orden de las lenguas es el mismo que el indicado en la segunda línea del fichero;
  - es posible que para una palabra dada haya varias opciones (irán separadas por "/", como mucho habrá 5 acepciones). La lengua origen (inglés) aparecerá sólo con una acepción (es la lengua referencia para hacer la traducción). Sin embargo, puede aparecer de nuevo en el fichero la misma palabra, o cuando se aplique alguna operación de inserción. En este caso habrá que añadir las traducciones nuevas a las ya almacenadas (si no están repetidas);
  - es posible que no haya traducción para alguna palabra en alguna lengua;

Como hemos dicho, se trabajará con dos tipos de ficheros de texto, uno para almacenar diccionarios y otro para almacenar textos.

Por ejemplo, un fichero de texto para el diccionario podría ser el siguiente:

```
3
E F P
allow * permitir * permettre/autoriser * permitir/dar/conceder
```

[1] este tipo de cuestiones las iremos estudiando y discutiendo a lo largo de todo el curso, sobre todo en clases de teoría.

```
at * a * chez * em/a
are * hay * sont * és/somos/sois/são
aunt * tía * tante *
after * después * après * depois
arrive * llegar * arriver * chegar
avenue * avenida * *
advertisement * publicidad * publicité * anúncio
something * algo * quelque chose * alguma coisa
everybody * todo el mundo * tout le monde * todo mundo/todos
 * * * alergia
```

Como se puede observar en el ejemplo, es posible que el fichero no muestre la lengua origen (primera palabra) de forma ordenada, y además puede haber traducciones en alguna lengua que no existan. Si se trata de la lengua por la que se indexa el índice, entonces no se almacena en el diccionario. Es decir, en el ejemplo anterior la última línea no se almacenaría.

La información de fichero donde se describe el diccionario se leerá automáticamente y se almacenará en estructuras de datos adecuadas para su posterior procesamiento. En concreto, para esta práctica se usará para la construcción del diccionario un array de `Palabra`.

La clase `DiccA` en esta primera práctica se definirá con las variables de instancia:

- `private int nlenguas`;
- `private char[] lenguas;` //E español, P portugués, F francés, por ejemplo, y pueden estar en cualquier orden
- `private Palabra[] dicc;`
- `private Palabra[] diccord;` //almacena el diccionario ordenado por la lengua origen

y los siguientes métodos de instancia:

- `public DiccA()`: inicializa las variables de instancia a sus valores por defecto, y -1 para `nlenguas`;
- `public void leeDiccA(String f)`: se leerá el diccionario desde un fichero de texto (que habrá que abrir y cerrar, y cuyo nombre le habremos pasado por parámetro). Se irán leyendo todas las líneas del fichero y se irá incorporando toda la información en las variables correspondientes definidas en la clase. En

el caso particular de la variable de instancia `dicc` el array de `Palabra` que se obtiene estará en el mismo orden que se lee el fichero, y en el caso de la variable `diccord` el array deberá estar ordenado por la lengua origen. Para el caso de los arrays `dicc` y `diccord` inicialmente se puede crear un array de dimensión 10, que hay que ir redimensionando de 10 en 10 conforme sea necesario. Este método no propaga excepciones, por lo tanto cualquier excepción que aparezca se tiene que tratar en el propio método, y dicho tratamiento será mostrar por pantalla el objeto `Exception`;

- `public boolean insertaPalabra(Palabra p)`: inserta una nueva palabra (con sus traducciones) en el diccionario, de forma que en la variable `dicc` lo almacenará en la última posición, y en la variable `diccord` de forma ordenada. Hay que redimensionar el array en caso de necesidad (de 10 en 10). En caso de que la cadena de la lengua origen ya exista, sólo tiene que añadir a la palabra las nuevas acepciones de las traducciones; el método devolverá `true` si realiza la inserción de una nueva palabra o acepción, y `false` en caso contrario;

- `public boolean borraPalabra(String s)`: borra la palabra de los dos diccionarios cuya cadena de la lengua origen coincide con `s` en las dos variables, `dicc` y `diccord`. Además, deberá adelantar una posición en el array todos los objetos de tipo `Palabra` posteriores al elemento borrado. Si no encuentra la cadena devuelve `false`, en caso contrario `true`;

- `public int busqueda(String s)`: busca entre las cadenas de la lengua origen de `dicc` la cadena `s` pasada como parámetro, y devuelve el número de comparaciones que realiza para encontrarla; si la cadena no existe devuelve -1;

- `public int busquedaOptima(String s)`: busca de forma óptima entre las palabras del diccionario la cadena `s` pasada como parámetro y devuelve el número de comparaciones que realiza para encontrarla; si la cadena no existe devuelve -1;

- `public String traduce1(String s, char l)`: busca entre las palabras de la lengua origen la cadena `s` pasada como parámetro y devuelve la primera opción de traducción de la cadena en la lengua `l`; por defecto el método devuelve `null`;

- `public String traduce2(String s, char l)`: busca entre las palabras de la lengua origen la cadena `s` pasada como parámetro y devuelve todas las traducciones de la cadena en la lengua `l` como un String (concatenando los resultados separados por "/"); por defecto el método devuelve `null`;

- `public void muestraDiccA(int i)`: muestra todas las palabras del diccionario con todas las traducciones con el formato indicado para el método `escribeInfo` de la clase `Palabra`; si `i` es cero muestra el diccionario original en el mismo orden leído en fichero, y si es 1 muestra el diccionario ordenado;

- `public void muestraDiccA(int i, int j)`: muestra solo las j primeras líneas del diccionario con todas las traducciones con el formato indicado para el método `escribeInfo` de la clase `Palabra`; si `i` es cero muestra el diccionario original en el mismo orden leído en fichero, y si es 1 muestra el diccionario ordenado;

- `public void muestraDiccA(int i, int j, char l)`: muestra solo las j primeras líneas del diccionario con la traducción de la lengua "l" (si no existe esa lengua en el diccionario muestra solo la lengua origen) con todas las traducciones con el formato indicado para el método `escribeInfo` de la clase `Palabra`. Si `i` es cero muestra el diccionario original en el mismo orden leído en fichero, y si es 1 muestra el diccionario ordenado. Por ejemplo, la salida de la llamada `muestraDiccA(0,3,'F')` sería:

  ```
  at:chez
  are:sont
  aunt:tante
  ```

La clase `Palabra` que hay que implementar contendrá:

- las siguientes variables de instancia:
  - `private String origen;` //palabra de la lengua de origen (inglés)
  - `private Traduccion[] trad;` //almacenará las diferentes opciones de traducción de cada lengua, implementación libre
- y los siguientes métodos de instancia:
  - `public Palabra(String p, int nl)`: `p` es la palabra origen y `nl` es el número total de lenguas a traducir, que deber ser mayor que 0, en otro caso tomará el valor 3. Los datos que no se pasan por parámetro se inicializan a sus valores por defecto;
  - `public int setTrad(Traduccion t, char l)`: almacena la traducción `t` en la primera posición libre de `trad` si dicha lengua no tiene ya una traducción almacenada y hay posiciones libres. Si ya hay una traducción anterior, se debe modificar para que sea la unión de ambas traducciones (la anterior y la nueva). El método devuelve la posición de `trad` que ocupa la traducción almacenada o modificada, o -1 por defecto.

- `public String getOrigen()`: devuelve la cadena de la lengua origen del objeto;
- `public String getTraduccion(char l)`: devuelve una cadena que contiene la primera acepción de las traducciones posibles en la lengua `l`. Por defecto el método devuelve `null`.
- `public String getTraducciones(char l)`: devuelve una cadena que concatena todas las acepciones de las traducciones posibles en la lengua `l`, separadas por "/". Por defecto el método devuelve `null`.
- `public boolean agregaAcepcion(String s, char l)`: añade la nueva acepción `s` al final de las traducciones de la lengua `l` si esta nueva acepción no existía y el número de acepciones es inferior al máximo permitido. El método devuelve `true` si puede realizar la operación, y `false` en caso contrario.
- `public void escribeInfo()`: muestra por pantalla las traducciones asociadas a la cadena origen; esta información se presentará en el mismo orden establecido en el diccionario leído de fichero de texto. El formato consiste en presentar la palabra origen seguida de ":", y a continuación cada nueva lengua (formada a su vez por cadenas o expresiones separadas por "/") con el mismo identificador.

  Por ejemplo, la salida para la palabra ``at'' en el ejemplo del enunciado será:

  ```
  at:a:chez:em/a
  ```

  la de ``allow'' será:

  ```
  allow:permitir:permettre/autoriser:permitir/dar/conceder
  ```

  y la de ``avenue'' será:

  ```
  avenue:avenida::
  ```

Hay que recordar que vosotros debéis definir la clase `Traduccion` para poder almacenar, leer, escribir y gestionar todo lo referente a las diferentes opciones de traducción de cada lengua. Para ello, es posible que tengáis que añadir métodos relacionados con esta clase en algunas de las clases que se han definido en este enunciado.

2. Aplicación de traducción

En esta aplicación vamos a definir un sencillo traductor *palabra* a *palabra*. Para evitar problemas de ambigüedad, utilizaremos la simplificación de considerar una única traducción por cada palabra. Para ello, utilizaremos solo la primera acepción de la lengua a traducir. Aunque el diccionario contenga varias traducciones, presentaremos la traducción entre las dos primeras, la origen del diccionario, y la primera de las opciones de traducción [2]. Por tanto el fichero de texto a traducir será un texto en inglés. Nótese que este tipo de traducción no conduce en la mayoría de los casos a traducciones correctas; en general, en el proceso de traducción intervienen no sólo la información léxica sino otras fuentes de conocimiento como las sintácticas, semánticas y pragmáticas [3].

Para ello, implementa una clase denominada `Bilingue` que recibirá como parámetros de entrada tres argumentos:

- `arg1` representa el fichero con la información del diccionario;
- `arg2` representa el fichero con un texto a traducir;
- `arg3` representa la lengua a la que se va a traducir el texto (un char).

En la clase `Bilingue` se ejecutará la aplicación, que consiste en diseñar e implementar un sencillo traductor palabra a palabra entre dos lenguas, lo que implica segmentar el texto leído para obtener las palabras.

Se implementará un método `main` que:

- detectará los parámetros de la aplicación (3 en total);
- abrirá los ficheros de texto con el diccionario (primer argumento) que lee y almacena, el texto a traducir (segundo argumento), que lee, procesa y traduce; en caso de que el carácter de la lengua que se pase no exista (tercer argumento), por ejemplo se pase una `‘‘S’’` y el diccionario no almacene dicha lengua, se mostrará por pantalla el mensaje `‘‘No existe traduccion para esta lengua’’`;

[2] Las palabras se buscarán en el diccionario independientemente de si están en mayúscula o minúscula, y la traducción que se presenta es con el formato que aparezca en el diccionario.

[3] La Sintaxis analiza la relación existente entre los distintos símbolos o signos del lenguaje. La Semántica estudia la relación entre los signos y su significado. La Pragmática es la disciplina que se ocupa de la relación entre los signos del lenguaje y los contextos o circunstancias en que las personas usan tales signos.

- mostrará por pantalla la traducción del texto y, a continuación, el porcentaje de éxito en la traduccción (sin decimales).

Por ejemplo, ante la frase en inglés ‘‘`The cat is red`’’, se esperaría como salida la frase en español ‘‘`El gato es rojo`’’ si todas las palabras están almacenadas en el diccionario en la lengua origen, y todas ellas tienen al menos una traducción asociada. Si alguna palabra no aparece, por ejemplo ‘‘`cat`’’ en el primer caso (o exista, pero no tiene asociada ninguna traducción), la salida sería ‘‘`El - es rojo`’’.

Además, la aplicación nos mostrará el porcentaje de traducción realizado.

En resumen, en el primer caso la salida sería:

```
El gato es rojo
```
100%

En el segundo caso:

```
El - es rojo
```
75%

Si en el texto aparecen signos de puntuación, se respetarán en la salida en el mismo orden. Por ejemplo, ante el siguiente texto:

```
The weather was very bad all the day, but in the night was better.  So,
I went to the party without the umbrella.
```

La traducción al español sería:

```
El tiempo era muy malo todo el día, pero en el noche era mejor.  Entonces,
yo fui a el fiesta sin el paraguas.
```
100%

4. Documentación

Documenta en el mismo código (con un comentario antes de cada variable o método) **como mínimo** los siguientes elementos, con una breve descripción (o más extensa, dependiendo de las aclaraciones añadidas en cada método que se describe a continuación):

- las variables y métodos de instancia (o clase) añadidos por tí, justificando su necesidad;
- las variables y métodos de instancia (o clase) de la clase `Traduccion`;

- la aplicación;
- los siguientes métodos de instancia:
  - de la clase `DiccA`:
    - `public void leeDiccA(String f);` aclarando cómo se ha construído la variable `diccord`
    - `public int busquedaOptima(String s);` en este caso debes describir con detalle qué hace el algoritmo de búsqueda, cuáles son las condiciones para poder aplicarlo y qué coste tiene el uso de dicho algoritmo en función del tamaño del diccionario que se aplique;
  - de la clase `Palabra`:
    - `public String getTraducciones(int i);` en este caso hay que describir cómo se genera la cadena de salida a partir de los datos del problema;
    - `public boolean agregaAcepcion(String s, char l);` describiendo con detalle el algoritmo que se aplica y cómo se manipulan las estructuras de datos que intervienen;

Apéndice 1: lectura de fichero de texto

La lectura de un fichero de texto en Java, tal y como se vio en la asignatura de Programación II, se puede realizar de la siguiente manera:

- Abrir el fichero utilizando la clase FileReader y después leerlo utilizando la clase BufferedReader para leer por líneas. Por ejemplo:

```
import java.io.*;
// estas serian las variables de instancia
FileReader fichero=null;
BufferedReader lectura=null;

try{
// esto podria ir al metodo abre
 fichero=new FileReader("myfile.txt");
 lectura=new BufferedReader(fichero);
// esto podria ir al metodo lee
 String linea = lectura.readLine();
// bucle lectura hasta final de fichero
 while(linea!=null) {
  linea=lectura.readLine(); //procesar linea
 }
}catch(IOException e){
 System.err.println("Error con myfile.txt");
 System.exit(0);
}
// esto podria ir al metodo cerrar
 try{
  if (fichero!=null)
   fichero.close();
  if (lectura!=null)
   lectura.close();
 }catch(IOException ex){
   System.out.println(ex);
 }
```

Apéndice 2: Segmentación de una cadena en distintos elementos

Una vez tenemos una línea del fichero en una variable de tipo `String`, en los ejemplos la variable `linea`, hay que segmentarla para obtener de ella las distintas traducciones que necesitamos para construir un objeto de tipo **Palabra**.

La segmentación la podemos realizar utilizando el método `split` de la clase `String` de Java que ya se vio en Programación II:

- Este método devuelve un array de `Strings` a partir de uno dado usando el separador que se especifique. Por ejemplo, supongamos que estamos leyendo el diccionario y tenemos la variable `linea` de tipo `String` que contiene la cadena

  ```
  allow * permitir * permettre/autoriser * permitir/dar/conceder
  ```

  de la cual queremos obtener por separado las diferentes traducciones que contiene:

  ```
  allow permitir permettre/autoriser permitir/dar/conceder
  ```

  Para ello primero debemos indicar el separador y después segmentar usando el método `split`.

  Al método `split` le podemos indicar que el separador pueden ser distintos caracteres utilizando una expresión regular, que indicamos de la siguiente manera [4]:

  ```
  //indicamos el separador de campos
   String separador = "[ ]*\\*[ ]*";
  ```

  Para la aplicación, debemos leer un fichero de texto donde pueden aparecer signos de puntuación. Por ejemplo, supongamos que estamos leyendo la siguiente cadena:

  ```
  Yes. What happens to you?
  ```

  En este caso queremos obtener por separado las palabras que contiene [5]:

[4] obsérvese que esta expresión regular representa que el separador es el carácter "*" que puede ir acompañado o no (por delante y/o por detrás) de varios espacios en blanco. Esto viene representado también por el símbolo "*".

[5] en la aplicación deberemos ver la forma de recuperar los símbolos que hemos utilizado para segmentar las palabras y poder ponerlos en la traducción que se realice.

```
Yes What happens to you
```

Ahora, el separador lo podemos definir de la siguiente manera:

```
String separador = "[,:;?!\\.() ]+";
```

Los caracteres que aparecen entre corchetes son los caracteres que consideramos que son separadores de palabras, mientras que el `+` que aparece a continuación significa que puede aparecer uno o más de estos caracteres (como es el caso cuando aparece una coma o un punto, que a continuación hay un espacio en blanco)

```
//segmentamos
 String[] elems = linea.split(separador);
// En el caso que elems sea una linea del
// texto, cada elemento de elems tendra
// almacenada una palabra segmentada
```

Normas generales

Entrega de la parte práctica:

- Lugar de entrega: servidor de prácticas del DLSI, dirección `http://pracdlsi.dlsi.ua.es`
- Plazo de entrega: desde el jueves 1 de octubre hasta el **miércoles 7 de octubre**.
- Se debe entregar la práctica en un fichero comprimido con todos los ficheros `.java` creados y ningún directorio de la siguiente manera

  ```
  tar cvfz practica1.tgz *.java
  ```

- No se admitirán entregas de prácticas por otros medios que no sean a través del servidor de prácticas.
- El usuario y contraseña para entregar prácticas es el mismo que se utiliza en el Campus Virtual.
- La práctica se puede entregar varias veces, pero sólo se corregirá la última entregada.
- Los programas deben poder ser compilados sin errores con el compilador de Java existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla.
- Los ficheros fuente deber estar adecuadamente documentados usando comentarios en el propio código, sin utilizar en ningún caso acentos ni caracteres especiales.
- Es imprescindible que se respeten estrictamente los formatos de salida indicados ya que la corrección se realizará de forma automática.
- Al principio de cada fichero entregado (primera línea) debe aparecer el DNI y nombre del autor de la práctica (dentro de un comentario) tal como aparece en el Campus Virtual (pero sin acentos ni caracteres especiales).

  Ejemplo:
  `DNI 23433224 MUÑOZ PICÓ, ANDRÉS` ⇒ NO
  `DNI 23433224 MUNOZ PICO, ANDRES` ⇒ SI

Sobre la evaluación en general:

- La práctica debe ser un trabajo original de la persona que entrega; en caso de detectarse indicios de copia de una o más entregas se suspenderá la práctica con un 0 a todos los implicados.
- La influencia de la nota de esta práctica sobre la nota final de la asignatura se detallan en las transparencias de presentación de la misma.
- La nota del apartado de documentación supone el 10% de la nota de la práctica.

Probar la práctica

- En el Campus Virtual se publicará un corrector de la práctica con una sola prueba (se recomienda realizar pruebas más exhaustivas).
- El corrector viene en un archivo comprimido llamado `correctorP1.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar: `tar xfvz correctorP1.tgz`.

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica1-prueba`: dentro de este directorio están los ficheros
    - `p01.dic`: fichero de texto con el diccionario usado en esta prueba;
    - `p01.java`: programa en Java con un método main que realiza una serie de pruebas sobre la práctica.
    - `p01.txt`: fichero de texto con la salida correcta a las pruebas realizadas en `p01.java`.
- Una vez que tenemos esto, debemos copiar nuestros ficheros fuente (sólo aquellos con extensión .java) al mismo directorio donde está el fichero `corrige.sh`.
- Sólo nos queda ejecutar el *shell-script*. Primero debemos darle permisos de ejecución. Para ello ejecutar:

```
chmod +x corrige.sh
corrige.sh
```