RESCATE PROGRAMACIÓN IV

Informe Presentado a

Ángel Augusto Zapata

Carolina Jiménez Gómez

Karen Stefanny Lopez Segura

Universidad Tecnológica de Pereira

Octubre 05 del 2015

*import java.io.BufferedReader;*

*import java.io.BufferedWriter;*

*import java.io.FileReader;*

*import java.io.FileWriter;*

*import java.io.IOException;*

*import java.io.Writer;*

*import java.util.ArrayList;*

*import java.util.Collections;*

*import java.util.HashMap;*

*import java.util.StringTokenizer;*

*import java.util.regex.Matcher;*

*import java.util.regex.Pattern;*

*class Spelling {*

*private final HashMap<String, Integer> nWords = new HashMap<String, Integer>();*

*public Spelling(String file) throws IOException {*

*BufferedReader in = new BufferedReader(new FileReader(file));//*lee el archivo

*Pattern p = Pattern.compile("[\\wñáéíóú:;.,]+");// se define la exprecion regular que se va a utilizar, en este caso todas las palabras que se pueden formar en el idioma español*

*for(String temp = ""; temp != null; temp = in.readLine()){*

*Matcher m = p.matcher(temp.**toLowerCase());//pasa a minuscula las palabras que recibe.*

*//mientras la palabra haga parte del patron al hashmap agregue el numero de coincidencias.*

*while(m.find())* *nWords.put((temp = m.group()), nWords.containsKey(temp) ? nWords.get(temp) + 1 : 1);*

*}*

*in.close();//*cierra el archivo

*}*

*private final ArrayList<String> edits(String word) {*

*ArrayList<String> result = new ArrayList<String>();*

*for(int i=0; i < word.length(); ++i) result.add(word.substring(0, i) + word.substring(i+1));*

*for(int i=0; i < word.length()-1; ++i) result.add(word.substring(0, i) + word.substring(i+1, i+2) + word.substring(i, i+1) + word.substring(i+2));*

*for(int i=0; i < word.length(); ++i) for(char c='a'; c <= 'z'| c <='á' | c <='é' | c <='í' | c <='ó' | c <='ú' ; ++c) result.add(word.substring(0, i) + String.valueOf(c) + word.substring(i+1));*

*for(int i=0; i <= word.length(); ++i) for(char c='a'; c <= 'z'| c <='á' | c <='é' | c <='í' | c <='ó' | c <='ú' ; ++c) result.add(word.substring(0, i) + String.valueOf(c) + word.substring(i));*

*return result;*

*}*

*public final String correct(String word) {*

*if(nWords.containsKey(word)) return word;//si la palabra se encuentra bien (es decir forma parte del big.txt) se devuelve la palabra*

*ArrayList<String> list = edits(word);//se crea una lista con las palabras editadas a su correccion*

*HashMap<Integer, String> candidates = new HashMap<Integer, String>();*

*for(String s : list) if(nWords.containsKey(s)) candidates.put(nWords.get(s),s);*

*if(candidates.size() > 0) return candidates.get(Collections.max(candidates.keySet()));*

*for(String s : list) for(String w : edits(s)) if(nWords.containsKey(w)) candidates.put(nWords.get(w),w);*

*return candidates.size() > 0 ? candidates.get(Collections.max(candidates.keySet())) : "error:la palabra no es valida";*

*}*

*public static void main(String args[]) throws java.io.IOException {//IOException,*

*BufferedReader in = new BufferedReader(new FileReader(**"archivo.txt")); //lee el archivo que se va a corregir.*

*BufferedWriter out=new BufferedWriter(new FileWriter("correccion.txt"));//crea un archivo .txt llamado correccion y crea un objeto de escritura sobre ese archivo para poder escribir en el.*

*for(String temp = ""; temp != null; temp = in.readLine()){//lee línea por línea*

*StringTokenizer palabra = new StringTokenizer(temp);//*separa palabras

*//* bucle por todas las palabras

*while(palabra.hasMoreTokens()){//*mientras hayan palabras en el archivo

*out.write((new Spelling("big.txt")).correct(palabra.nextToken()));//*escribe en el archivo "correccion.txt" *las palabras que va corrigiendo*

*out.write(" ");//*separa las palabras

*out.flush();//guarda los cambios realizados en el archivo*

*}*

*out.newLine();//crea un salto de linea*

*out.flush();*

*}*

*out.close();//cierra el archivo.*

*}*

*}*

**EXPLICACIÓN DEL CÓDIGO**

El código anterior lee un archivo de texto(*"archivo.txt"*), crea un archivo de texto(“correccion.txt”) donde se va a guardar el texto ingresado pero corregirlo, empieza a corregir el texto ingresado linea por linea, palabra por palabra y al mismo tiempo va guardando las lineas corregidas con sus correspondientes espacios entre palabras y saltos de linea.

La corrección de las palabras se realizan de las siguiente manera:

1. Se lee la palabra y la va pasando a minúscula

2. Comprueba si la palabra se encuentra bien escrita buscándola en las palabras utilizadas en texto de referencia (big.txt), si la encuentra devuelve la palabra y finaliza el proceso, de lo contrario continúa.

3. Busca y guarda en una lista el numero de coincidencias de la palabra, es decir las posibles correcciones.

4. Elige la palabra de la lista que mas se repite en el texto de referencia (big.txt) y devuelve la palabra corregida.

CÓMO CONSTRUIR EL ARCHIVO “big.txt” PARA RECONOCER PALABRAS EN ESPAÑOL

Para que se reconozcan palabras en español los únicos cambios que debemos hacerle al archivo es agregar diferentes libros en este idioma, puesto que el programa busca dentro del big.txt todas las ocurrencias de una palabra, y palabras parecidas, y devuelve la que aparezca más veces en el texto, por esto si le enviamos una palabra que no existe en español y no la encuentra en el archivo retorna dicha palabra o una excepción.

Nosotras utilizamos cuatro libros diferentes: Cien Años de Soledad, La Cándida Eréndira, un diccionario de español y Don Quijote de la Mancha.

Entre más palabras contenga el archivo big.txt más posibilidades encontrará el programa de corregir una palabra.

PRUEBAS QUE SE LLEVARON A CABO

Las pruebas que realizamos fueron enviarle un archivo con palabras mal escritas, sin tíldes, otras con tildes y otros errores, y miramos que las corrigiera correctamente en un archivo de salida.

Por ejemplo cada vez que le enviamos ace nos lo corrige con hace, y parce nos lo corrige por pase (puesto que la palabra parce no aparece muchas veces en los libros que utilizamos), comunicasion nos lo corrige a comunicación y así sucesivamente.

CAMBIOS HECHOS AL PROGRAMA

Los cambios que le hicimos al programa fueron mínimos, simplemente hicimos que reconociera las tildes, la “ñ” y los signos de puntuación agregándole al Patern.compile esto: *[\\wñáéíóú*ü*:;.,]+* .

También en la función edits(String word) en los dos últimos ciclos le pusimos que reconociera las letras con tíldes, puesto que el programa anterior únicamente iba desde la “a” hasta la “z” esto lo logramos agregándole lo siguiente al ciclo for:

for(char c='a'; c <= 'z'| c <='á' | c <='é' | c <='í' | c <='ó' | c <='ú'|c <= 'ü' ; ++c)

En la función correct(String word) el único cambio que le hicimos es que si no encuentra candidatos para reemplazar la palabra entonces ya no la devuelve si no que retorna un mensaje diciendo: "error:la palabra no es valida", aunque esto no es muy conveniente puesto que como nosotros escribimos las correcciones realizadas en otro archivo aparecerá este mensaje en vez de aparecer la palabra sin corregir.

**CONTENEDORES**

**Class Dictionary<K,V>**

Es la abstracción de cualquier clase, como la Hashtable, en el cuál cada llave tiene asociado un valor, donde no se permiten elementos nulos.

**Interface Map<K,V>**

La Interface Map (java.io.Map) en Java, nos permite representar una estructura de datos para almacenar pares “key/value”; de tal manera a cada llave se le asocia un valor. Los Maps en java tienen implimentada por debajo toda la teoría de las estructuras de datos de los Árboles. Un mapa no contiene llaves iguales.

**Clase HashMap<K,V>**

Es una hash table basada en la implementación de la interface Map. Permite todas las operaciones que se pueden hacer con Map. Los elementos que inserta en el map no tendrán un orden específico. No aceptan claves duplicadas ni valores nulos.

Utiliza todos los métodos descritos en la interface Map<K,V>.

**Clase HashTable<K,V>**

Es una hash table implementada sobre la interface Map<K,V>.

Utiliza todos los métodos descritos en la interface Map<K,V>.

**Clase LinkedHashMap<K,V>**

Es una hash table y una lista enlazada sobre la implementación de la interface Map<K,V>.

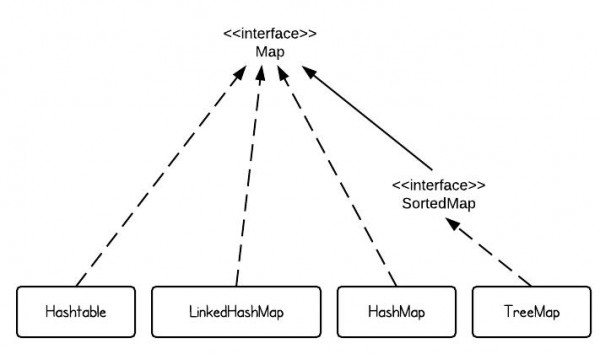
Utiliza todos los métodos descritos en la interface Map<K,V>.

**Clase TreeMap<K,V>**

Es un árbol Rojo-Negro basado en la implementación [NavigableMap](http://docs.oracle.com/javase/7/docs/api/java/util/NavigableMap.html), Map y SortedMap. El orden de las llaves es de menor a mayor.

**Diferencias entre HashMap, TreeMap, LinkedHashMap**

|  |  |  |  |
| --- | --- | --- | --- |
|  | HashMap | TreeMap | LinkedHashMap |
| Orden | No se garantiza el orden | De menor a mayor (las llaves) | Tal cual como se va insertando |
| Get/put/remove  Complejidad | O(1) | O(log(n)) | O(1) |
| Interfaces | Map | NavigableMap  Map  SortedMap | Map |
| Null  values/key | Se permiten | Solo en values | Se permiten |
| Implementación | buckets | Red-Black Tree | Double-linked buckets |



**Interface Queue<E>**

Es una estructura de datos que permite almacenar información de forma que el primer elemento que se ingresa a la cola es el primer elemento que se saca de ella “FIFO” (first to in, first to out). Todos los elementos que se insertan en la cola se ponen al final.

Está implementada sobre la interface Collection<E>.

Esta interface posee métodos para encolar, desencolar, consultar el primer elemento de la cola, entre otras, donde posee dos versiones de cada una de estas, una donde lanza excepciones si ocurre un fallo y la otra que retorna false o un valor en específico si ocurre esto.

Interface Deque<E>

El nombre “Deque” significa "double ended queue". Esta interfaz implementa pilas (stacks) y colas (queues) al mismo tiempo.

Interface PriorityQueueu<E>

Está implementada sobre las interfaces Collection<E>, Queue<E> e iterator<E>. Utiliza un comparador para ordenar los elementos por su prioridad. Inserta los elementos de menor a mayor.

**Interface List<E>**

Una lista es una secuencia de elementos dispuesto en un cierto orden, en la que cada elemento tiene como mucho un predecesor y un sucesor.

Es una colección (Collection<E>) ordenada. Permite tener elementos duplicados.

Un [ListIterator](http://www.w3api.com/wiki/Java:ListIterator) es un iterador [Java](http://www.manualweb.net/tutorial-java/) que nos permite recorrer una lista de elementos en varias direcciones, bien hacia delante o bien hacía atrás.

Podemos recorrerla normalmente con hasnext() o de atrás hacia adelante con previous().

**Clase ArrayList<E>**

Difieren de los Array en que no se accede a ella por medio de índices si no por medio de punteros que recorren la lista. Tiene un puntero que siempre apunta a la primera posición de la lista, y todos los elementos de la lista apuntan al siguiente, por lo que para acceder a cierto elemento debemos recorrer toda la lista. El último elemento apunta a NULL. Cuando no hay elementos en la lista, first apunta a NULL. Mientras que en un *array* los elementos están contiguos en la memoria, en una lista los elementos están dispersos.

Es una implementación de la interface List, y por ende utiliza todos los métodos de esta última.

Permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja de que el numero de elementos que almacena, lo hace de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los Arrays.

Esta clase provee métodos para manipular el tamaño del Array que se utiliza internamente para guardar en la lista.

**Clase LinkedList<E>**

Lista doblemente enlazada. La diferencia con ArrayList es que en esta los elementos tienen un puntero que señala el elemento siguiente y el anterior a él, por consiguiente, se puede acceder a los elementos de la lista recorriéndola desde el último elemento hacia el primero o al revés.

Es una implementación de la interface List y Deque, utiliza todos los métodos de estas interfaces.

**Clase Vector<E>**

Es una estructura de datos que permite almacenar un conjunto de datos del mismo tipo. Accedemos a sus elementos por medio de sus índices.

Difiere de los Arrays en que el tamaño de los vectores puede crecer.

**Class Stack<E>**

Implementa la interface List<E>, Collections<E>, entre otros.

Es una estructura de datos que permite almacenar información de forma que el último elemento que se ingresa a la cola es el primer elemento que se saca de ella “LIFO” (last to in, first to out). Todos los elementos que se insertan en la cola se ponen al comienzo de esta.

Sus operaciones básicas son push (inserta en la pila) y pop (retira elemento de la pila).

Interface Set<E>

Es una colección que no contiene elementos duplicados. Es una abstracción de conjuntos. Algunas implementaciones no permiten elementos nulos, y otros tienen restricciones en los elementos que se van a contener.

EXPRESIONES REGULARES

Las expresiones regulares en java definen un patrón de búsqueda para cadenas de caracteres (String) , que se utilizan para saber si una cadena coincide o contiene el patrón dado por la expresión regular y que número de veces y para comprobar que una cadena tiene una determinada estructura.

Las expresiones regulares son usualmente utilizadas para comprobar si una cadena ingresada cumple con un patrón necesario para tener un orden o sentido, por ejemplo una fecha de nacimiento en un formato dd/mm/aaaa, para comprobar que un correo electrónico es válido, para hacer búsquedas o para sustituir ocurrencias.

Uso de expresiones regulares:

El paquete que se utiliza es java.util.regex, está formado por dos clases, la clase Matcher y la clase Pattern y por una excepción, PatternSyntaxException.

La clase Matcher: Esta clase compara el String y la expresión regular. Contienen el método matches(CharSequence input) que recibe como parámetro el String a validar y devuelve true si coincide con el patrón. El método find() indica si el String contienen el patrón.

La clase Pattern: Un objeto de esta clase representa la expresión regular. Contiene el método compile(String regex) que recibe como parámetro la expresión regular y devuelve un objeto de la clase Pattern.

Las expresiones regulares que se pueden definir en Matcher se escriben como normalmente se escribiría una expresión regular y también se pueden escribir con los siguientes meta caracteres:

\\d : Dígito. Equivale a [0-9]

\\D : No dígito. Equivale a [^0-9]

\\s : Espacio en blanco.

\\S : No espacio en blanco.

\\w : Una letra mayúscula o minúscula, un dígito o el carácter \_ equivale a [a-zA-Z0-9\_]

\\W : Equivale a [^\w]

\\b : Límite de una palabra.

Además el símbolo “^” indica el principio del String, el símbolo “$” indica el final del String y el simbolo “.” indica cualquier carácter.

Ejemplo de aplicación de expresiones regulares en java:

Comprobar si el String cadena contiene “abc”

Pattern pat = Pattern.compile(".\*abc.\*");

Matcher mat = pat.matcher(cadena);

if (mat.matches()) {

System.out.println("SI");

} else {

System.out.println("NO");

}

PAQUETE JAVA.IO (Stream)

Aquí se encuentran varias clases de entrada y salida, que permiten manejar caracteres, bytes, objetos, archivos, etc.; estos soportan diferentes tipos de fuentes y destinos como el teclado, la consola, ficheros, páginas web.

Stream estandar:

- Entrada estándar - habitualmente el teclado

- Salida estándar - habitualmente la consola

- Salida de error - habitualmente la consola

En Java se accede a la E/S estándar a través de campos estáticos de la clase java.lang.System:

* System.in: Esta clase es para el flujo de bytes de entrada e instancia de la clase InputStream. Sus metodos son:
  + read(): permite leer un byte de la entrada como entero
  + skip(n): ignora n bytes de la entrada
  + available(): número de bytes disponibles para leer en la entrada
* System.out: Esta clase es para el flujo de bytes de salida e instancia de la clase PrintStream. Sus metodos para impresión de datos son:
* print(), println()
* flush() vacía el buffer de salida escribiendo su contenido

-System.err: Implementa la salida de error y su funcionamiento es similar a System.out

Clasificación de Stream:

-Representación de la información:

•Flujos de bytes: clases InputStream y OutputStream

•Flujos de caracteres: clases Reader y Writer

Se puede pasar de un flujo de bytes a uno de caracteres con InputStreamReader y OutputStreamWriter

-Propósito:

•Entrada: InputStream y Reader.

•Salida: OutputStream y Writer.

•Lectura/Escritura: RandomAccessFile.

-Transformación de los datos

•Realizan algún tipo de procesamiento sobre los datos (p.e. buffering, conversiones, filtrados): BuffuredReader y BufferedWriter.

Las clases java.io.InputStream y java.io.OutputStream:

Son clases abstractas que sirven para leer ficheros byte a byte.

-La clase java.io.InputStream lee bytes (8 bits), tiene una clase deriva principal que es java.io.FileInputStream que permite leer archivos(FileInputStream(File file)) y String(InputStream(String nombre) ).

La clase java.io.InputStream escribe bytes (8 bits), tiene una clase deriva principal que es java.io.FileOutputStream que permite escribir en archivos(FileOutputStream(File file)) y en String(java.io.FileOutputStream(String nombre) ).

Las clases java.io.Reader y java.io.Writer:

Estas clases abstractas trabajan al igual que el InputStream y OutputStream con la excepción de que lector y escritor se basan en caracteres.

-La clase java.io.Reader tiene una clase deriva prinsipal que es java.io.FileReader que permire leer archivos( FileReader(File file) ) y String(FileReader(String nombre) ).

-La clase java.io.Writer tiene una clase deriva prinsipal que es java.io.FileWriter que permite escribir archivos(FileWriter(File file)) y String(FileWriter (String nombre)).

Las clases BuffuredReader y BufferedWriter:

Son clases derivadas de java.io.Reader y java.io.Writer que proporcionan más rapidez, al leer y escribir grandes cantidades de caracteres.

-La clase BuffuredReader realiza una gestión más eficiente del fichero, leyendo muchos caracteres de golpe para que las llamadas a los métodos 'read()' sean más rápidas.

-La clase BufferedWriter permite escribir texto en un Outputstream, utilizando un buffer para proporcionar una escritura eficiente de caracteres, arrays y strings.

Otras clases de java.io:

-La clase java.io.PrintWriter esta derivada de Writer, es para escribir caracteres en un fichero, proporcionando una serie de métodos que facilitan tareas habituales.

-La clase java.io.File sirve para referirse a ficheros y directorios en el sistema de ficheros del ordenador.

Un objeto de la clase java.io.File se puede construir a partir del nombre o ruta completa, o a partir del directorio en el que se encuentra; esta clase contiene muchos metodos para saber el estado del fichero.

**WEBGRAFÍA**

* <http://jarroba.com/map-en-java-con-ejemplos/>
* <http://docs.oracle.com/javase/7/docs/api/java/util/Map.html>
* <http://jarroba.com/arraylist-en-java-ejemplos/>
* <http://programacion.jias.es/2011/10/colecciones-genericos-en-java/>