Corrector ortográfico implementado con distancias de Levenshtein

Miruna Andreea Gheata, Rafael Adrián Gil Cañestro

Resumen—En este documento se muestra la implementación de la práctica *Corrector ortográfico*. Se explicará la estructuración del proyecto, el patrón de programación empleado para el desarrollo, la solución propuesta, los distintos problemas encontrados durante el proceso, un manual de usuario y, finalmente, unas conclusiones.

Index Terms—Levenshtein, diccionario, corrector ortográfico, programación dinámica

1. Introducción

En este articulo se detallará por capítulos la implementación de un corrector ortográfico implementado en java usando como técnica las ***distancias de Levenshtein***.

En el capítulo 2 **se presentará el problema propuesto**, al igual que los requerimientos definidos que se tienen que cumplir.

En el capítulo 3 **se expondrá la solución propuesta**, mencionando a grandes rasgos las técnicas y patrones principales que se han adoptado para resolver el problema: el *patrón MVC*, el *DDD*, la *concurrencia*; y mencionando aspectos importantes de la implementación.

En el capítulo 4 **se especificarán las tecnologías utilizadas**, es decir, el entorno de programación, librerías destacadas, etc.

En el capítulo 5 se explicará la **implementación del Modelo**, separado en Dominio (la definición de los datos) e Infraestructura (las operaciones que se realizan sobre estos datos).

En el capítulo 6 se explicará la **implementación de la Vista**, que se ha separado en varias clases, cada una representando un elemento gráfico.

En el capítulo 7 se explicará la **implementación del Controlador**, explicando los métodos que contiene.

En el capítulo 8 se calculará el **Coste computacional del programa**.

El capítulo 9 corresponde a una **Guía de usuario** en la que se definen los pasos a seguir para poder ejecutar el programa y unos ejemplos de ejecución.

En el capítulo 10 se exponen los **Problemas encontrados** a lo largo de la implementación del programa.

2. Definición del problema

El corrector ortográfico consiste en un programa capaz de detectar las palabras ortográficamente incorrectas y de proporcionar palabras con las que corregir. Para que sean posibles ambas cosas será necesario disponer de un diccionario con el que poder consultar las distintas palabras que se analizan, tanto para ver si está bien escrita como para poder encontrar las palabras para sustituir.

Parece un problema no trivial y que además puede llegar a ser muy costoso tanto en tiempo como en memoria, ya que se deben comparar **todas las palabras del texto** con **todas las palabras del diccionario**.

Para poder encontrar las palabras con las que corregir se usará la **distancia de Levenshtein**, obteniendo aquellas palabras que más se parecen a la incorrecta.

3. Solución propuesta

La solución propuesta hace uso de distintos patrones, técnicas y formas de programación y de estructuración de proyetos.

Programación dinámica: método ascendente de resolución de problemas que se basa en la división del mismo en varios subproblemas, la resolución de ellos y combinar las subsoluciones para obtener la solución global.

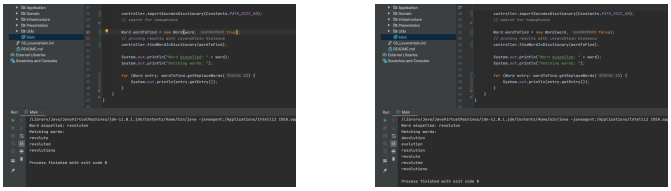
Puede ocurrir que la división del problema conduzca a un gran número de subproblemas idénticos. Si se resuelve cada uno de ellos sin tener en cuenta las posibles repeticiones, resulta un **algoritmo ineficiente**; en cambio si se resuelve cada ejemplar distinto una sola vez y se conserva el resultado, el algoritmo obtenido es mucho mejor. Esta es la idea de la programación dinámica: ***no calcular dos veces lo mismo*** y utilizar normalmente una tabla de resultados que se va rellenando a medida que se resuelven los subejemplares.

Distancia de Levenshtein: También conocida como ***distancia de edición***, es un algoritmo que analiza el coste de transformar una cadena de caracteres en otra. El coste se calcula a partir del número de operaciones que se deben realizar para que la cadena A sea convierta en la cadena B. Las operaciones posibles son las de ***inserción***, ***sustitución*** y ***eliminación***.

Se ha implementado un **corrector ortográfico interactivo** [2] mediante el cuál se podrán analizar ficheros de texto y escribir texto en el mismo editor, con la posibilidad de corregir todas las faltas de ortografía o de forma individual.

Debido a que se usa la distancia de Levenshtein hay que elegir un límite de distancia que se aceptará para poder proponer las palabras para corregir. En esta implementación se ha decidido que la máxima distancia será **4**. Además, sólo se propondrán las palabras de **2 distancias**, ya que se puede dar el caso de que por ejemplo no haya palabras con distancia 1 pero sí con distancia 2 y 3.

En cuanto a los idiomas posibles se ha decidido tener el corrector en **Español** y en **Inglés**. Para el diccionario en inglés se ha implementado el algoritmo de ***Soundex***, un algoritmo fonético que sirve para indexar cada palabra por sus sonidos, tal como son pronunciados en Inglés. El objetivo básico de este algoritmo es codificar de la misma forma los nombres con la misma pronunciación. Tal como se muestra en la figura 1, al utilizar el algoritmo se obtienen menos palabras, dado que es "más preciso" y sólo se muestran palabras que se parecen en la pronunciación.



(a) Con *Soundex* (b) Sin *Soundex*

Figura 1: Diferencia al utilizar el algoritmo *Soundex*. Se observa que en la 1a se obtienen menos palabras (distancia levenshtein 2)

La implementación del corrector ortográfico realizada permite:

1. **analizar ficheros enteros**, obteniendo las palabras incorrectas y teniendo la posibilidad de corregir el texto entero
2. **escribir en el editor** y que se marquen las palabras incorrectas de forma automática
3. **sustituir las palabras incorrectas** de 3 maneras:
   a) corrigiendo el texto entero
   b) eligiendo la palabra con la que sustituir desde la barra lateral
   c) pulsando en el editor encima de la palabra errónea y eligiendo una de la lista de propuestas que sale

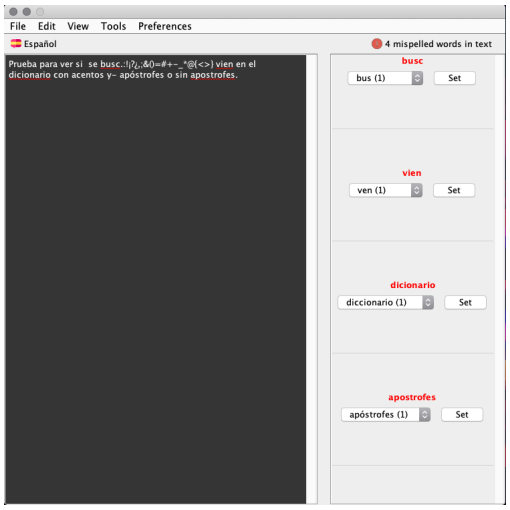


Figura 2: Corrector ortográfico implementado

Como **elementos extra** se han implementado:

1. Editor interactivo que a medida que vas escribiendo te **sugiere palabras**, de manera que al escribir te vaya completando la palabra (si por ejemplo se tiene escrito *man* te puede proponer *manzana* o *mandarina*)
2. *Find & Replace*, de manera que se puedan encontrar todas las ocurrencias de una palabra en el texto y con la posibilidad de sustituir una o todas las ocurrencias.
3. *Undo & Redo*, para poder rehacer/deshacer los cambios realizados al texto.

3.1. Patrón MVC y enfoque DDD

Este proyecto esta estructurado basándose en el diseño guiado por el dominio, en adelante DDD, ya que se complementa muy bien con MVC. El DDD no es una tecnología ni una metodología, es un enfoque que proporciona una visión estructural del sistema. DDD separa el *Modelo* en tres partes: ***Dominio, Aplicación e Infraestructura***.

La capa de ***Aplicación*** es responsable de coordinar la ***Infraestructura*** y la capa de ***Dominio*** para hacer una aplicación útil. Por lo general, la capa de *Aplicación* usaría la *Infraestructura* para obtener los datos, consultaría el *Dominio* para ver qué se debería hacer y luego volvería a usar la *Infraestructura* para realizar las operaciones pertinentes para obtener los resultados deseados. [3]. Por lo tanto,

- La capa de *Aplicación* se correspondería al *Controlador*
- la de *Dominio* e *Infraestructura* al *Modelo*
- la de *Presentación* a la *Vista*

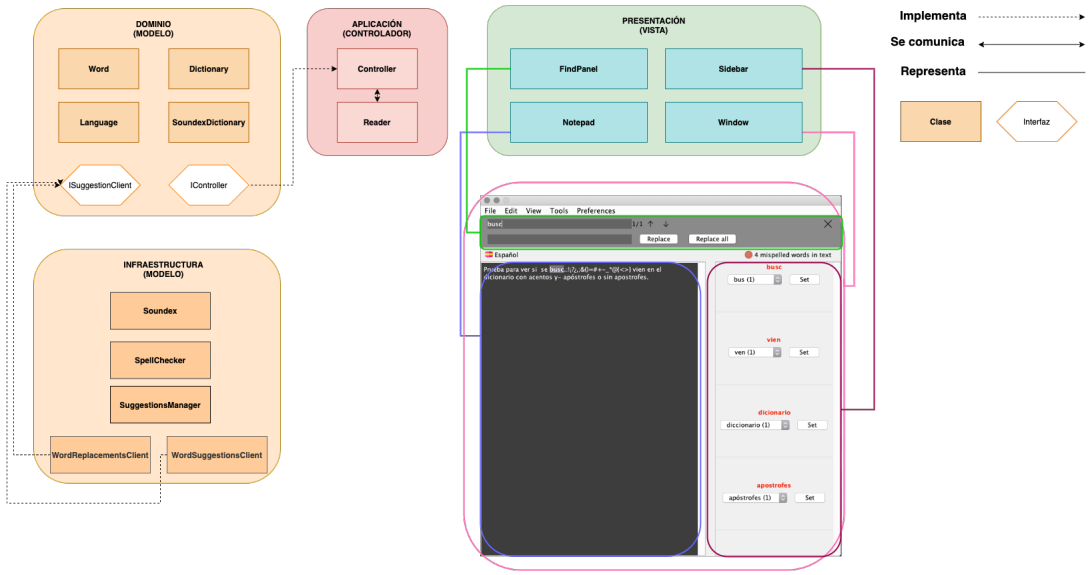


Figura 3: Estructura MVC del proyecto

3.2. Estructuración DDD

A nivel interno el programa está dividido en 5 carpetas (*Presentación*, *Aplicación*, *Dominio*, *Infraestructura* y *Utils*, en la que se guardan elementos que se usan en todo el proyecto) y un fichero Main.java.

3.3. Concurrencia

Con el fin de que el proceso sea más rápido y que además no se bloquee la interfaz grafica durante los cálculos, se han implementado varios cálculos mediante `Threads`. Estos cálculos son:

1. Revisión del texto para encontrar fallos ortográficos
2. Correción de un texto entero
3. Proceso de búsqueda de una palabra en el texto
4. Proceso de análisis de una palabra escrita en tiempo real

3.4. Aspectos a destacar

1. La máxima distancia de Levenshtein contemplada es la 4
2. Implementación de Find & Replace
3. Implementación de Undo & Redo
4. Se tiene una clase `Constantes` en la que se guardan todas las variables finales (nombres, títulos, dimensiones, etc.) a modo que sean visibles para todas las clases del proyecto. De esta manera se consigue centralizar todas las variables para que su manipulación sea más sencilla (creación, modificación, acceso, etc.).
5. Se ha "mecanizado" la creación del menú mediante la clase `MenuBuilder`: utilizando estructuras de datos se han mapeado todos los elementos de control (`ActionListeners`, `KeyStrokes`, iconos, elementos de submenú, etc.) para poder reducir el total de líneas de código necesarias para la implementación del menú. En concreto, **se ha conseguido comprimir 169 líneas de código en 47 líneas.**
6. Implementación del algoritmo ***Soundex*** para el diccionario en Inglés.
7. Implementación de un entorno interactivo que te sugiere palabras y te detecta los errores ortográficos a medida que se va escribiendo.

4. Tecnologías utilizadas

4.1. Lenguaje de programación utilizado

La implementación de este proyecto se ha hecho utilizando como lenguaje de programación Java.

4.2. Entorno de programación

La práctica se ha desarrollado en el IDE **IntelliJ IDEA**. **Cabe destacar que si se intenta ejecutar el proyecto el Netbeans no compilará debido a que IntelliJ utiliza carpetas extra (como la de .idea) para poder compilar el proyecto.** Por lo tanto, para ejecutar el proyecto se debe utilizar necesariamente IntelliJ.

Al igual que Netbeans, IntelliJ ofrece una ayuda a la hora de programar elementos gráficos, por lo que se ha facilitado todo el tema del diseño de las ventanas.

5. Implementación del Modelo

El *Modelo* es el elemento responsable de mantener la información con la que el sistema opera, en este caso los diccionarios, las estructuras de datos como puede ser la

palabra o el lenguaje, calcular la distancia entre las palabras y corregirlas. El Modelo también es el encargado de enviarle la información a la Presentación (vista), para que esta pinte por pantalla los botones, barra de navegación y el pad de notas.

Por otro lado, las peticiones de actualización de los datos llegan al Modelo desde el Controlador. Como ya se ha comentado las funciones del Modelo están divididas en dos partes. El Dominio se encarga mantener, enviar y modificar la información y estructuras.

Por el otro lado, la Infraestructura es la encargada de realizar las operaciones de cálculo de distancia, en este caso usamos la técnica de las distancias de Levenshtein, para encontrar la solución.

5.1. Dominio

Al ser el encargado de mantener y brindar la información, tiene que contener la información sobre los objetos que se van a usar y las interficies que usamos para facilitar los futuros cambios. Dentro del Dominio, por lo tanto, podemos encontrar las definiciones de las distintas estructuras de datos que se usarán para poder representar los elementos del problema, que son Word, Dictionary, tendremos también las variables que usarán las funciones del programa y una carpeta con las interficies que son IController y ISuggestionClient.

1. ***Dictionary*** Estructura que almacena todas las palabras de un idioma, para su consulta.
   - **`type`** Tipo de idioma del diccionario
   - **`Entries`** Todas las palabras que compone el diccionario
   - **`getType`**: Devuelve el idioma del diccionario.
   - **`getEntries`**: Devuelve un array con todas las palabras.
2. ***Language*** Estructura que almacena la información de un idioma, se usa para poder cambiar de diccionarios de forma sencilla.
   - **`name`** Nombre que identifica al idioma.
   - **`icon`** Icono que se usa en la vista para identificar el idioma en la vista.
   - **`getName`**: Devuelve el nombre del idioma.
   - **`getIcon`**: Devuelve el icono del idioma.
3. ***SoundexDictionary***‘ Diccionario que se usa en el caso de la corrección en Ingles, para relacionar palabras que se pronuncian igual.

   El constructor no tiene atributos, solo llama a la función `populateDict()`:
   - **`populateDict()`**: Completa el diccionario con parejas de palabras
   - **`getDict()`**: Devuelve el diccionario
4. ***Word*** La estructura básica del programa, almacena una palabra con la cuál se calcularán las respectivas distancias.

   Los atributos de la clase son:
   - **`distance`** Distancia entre la palabra original y por la que va a ser sustituida.(Este atributo no aparece en el caso de que sea la palabra que va a ser sustituida)
   - **`entry`** Almacena la palabra en forma de String.
   - **`Soundex`** Booleano que indica si pertenece o no al Inglés.

   Los métodos de la clase son:
   - **`String getEntry()`**: Devuelve la palabra
   - **`ArrayList<Word>getReplaceWords(int distance)`**: Devulve las sustituciones que están a una distancia igual a la que entra por parámetros
   - **`boolean isSoundexWord()`**: Devuelve si tiene se contempla su forma fonética
   - **`boolean replaceWordsInitialized()`**: Comprueba si la lista de sustitutos esta vacía
   - **`void addDistance(int distance)`**: Añade distancia a la palabra
   - **`void addReplaceWord(Word replaceWord)`**: Añade una palabra sustituta al conjunto total
   - **`void setMispelled(boolean mispelled)`**: Señala si una palabra esta mal escrita
   - **`boolean isSameWord(Word word)`**: Comprueba si las palabras son iguales
   - **`void setLine(int line)`**: Inicializa la linea en la que se encuentra la palabra.
   - **`void setPos(int pos)`**: Inicializa la posición en la línea

- **`void getPos(int pos)`**: Devuelve la posición en la línea.

Por otro lado, comentaremos una de las interfaces que usamos *ISuggestionClient*. En esta interfaz se encuentran las funciones que implementan los desplegables que aparecen cuando se sugiere una palabra o cuando se despliega la lista con palabras para reemplazar una incorrecta.

Encontramos:

- **`Point getPopupLocation`**: Devolverá el punto de la pantalla en el que debe aparecer el menú.
- **`void setSelectedText`**: Selecciona la palabra sustituta
- **`ArrayList<Word>get`**: Devolverá todas las opciones que irán en el menú.

5.2. Infraestructura

La infraestructura es la encargada de hacer los cálculos de las distancias entre palabras implementando la técnica de Levenshtein. Este comprobara que número de diferencias hay entre las palabras, siendo la distancia el número de discrepancias entre las palabras y las enviará para que sean mostradas. El la carpeta de *Infraestructura* encontramos 3 clases, *Soundex*, *SpellChecker* y *SuggestionsManager*, además de una carpeta *SuggestionsClients*, que contiene otras 2 clases, *WordSuggestionsClient* y *WordReplacementsClient*. Todas estas clases se explicarán a continuación.

1. ***Soundex*** Esta clase en la encargada de definir las palabras con misma forma fonética.
   Solo contiene un método:
   - **`String soundex(String s)`**: Método para convertir una palabra a su equivalente fonético.
2. ***SpellChecker*** Esta clase es la encargada de comprobar que una palabra este bien escrita o no y de calcular la distancia entre las palabras.
   Solo contiene un método:
   - **`int levenshtein(String a, String b)`**: Este método devuelve la distancia de Levenshtein que hay entre las dos palabras insertadas por parámetro.

   Este es el algoritmo usado:

```
a = a.toLowerCase();
 b = b.toLowerCase();
 int[] costs = new int[b.length() + 1];

 for (int j = 0; j < costs.length; j++)
     costs[j] = j;

 for (int i = 1; i <= a.length(); i++) {
     costs[0] = i;
     int nw = i - 1;

     for (int j = 1; j <= b.length(); j++) {
         int cj = Math.min(Math.min(costs[j], costs[j - 1]) + 1, a.charAt(i - 1) =
         nw = costs[j];
         costs[j] = cj;
     }
 }
 return costs[b.length()];
```

3. ***SuggestionsManager*** Esta clase es la encargada de crear, mostrar y actualizar la lista de sugerencias de palabras sustitutas.

   La clase tiene estos atributos:
   - `invoker`: Componente en el que se mostrará el desplegable con los elementos.
   - `client`: Encargado de realizar la sustitución en el componente.
   - `controller`: Controlador del programa

   Los métodos de la clase son estos:
   - **`<C extends JComponent>void decorate(C component, ISuggestionClient<C>suggestionClient)`**: Llama a todas las funciones de la clase, inicializando así todos sus elementos.
   - **`<C extends JComponent>void decorate(C component, ISuggestionClient<C>suggestionClient, Controller controller)`**: Llama a todas las funciones de la clase, inicializando así todos sus elementos.
   - **`void initPopup()`**: Inicializa el menú
   - **`void initSuggestionCompListener()`**: Inicializa los escuchadores
   - **`void update(DocumentEvent e)`** Actualiza la lista de elementos de la lista
   - **`void showPopup(ArrayList<String>suggestion`** Vuelve visible la lista seleccionada
   - **`void initInvokerKeyListeners()`**: Inicializa los escuchadores de las teclas
   - **`void initInvokerMouseListener()`**:Inicializa los escuchadores del ratón
   - **`void selectFromList(KeyEvent e)`**: Selecciona el elemento que se ha elegido mediante el ratón.
   - **`void selectFromList(MouseEvent e)`**: Selecciona el elemento que se ha elegido mediante el teclado.
   - **`void moveDown(KeyEvent keyEvent)`**: Desplaza la lista de sugerencias un elemento hacia abajo.
   - **`void moveUp(KeyEvent keyEvent)`**: Desplaza la lista de sugerencias un elemento hacia arriba.

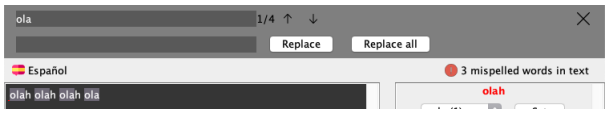


Figura 4: FindPanel

En la interfaz tenemos también las clases que se encargan de implementar los ActionListeners del `Notepad`. Dichas clases implementan la interfaz `ISuggestionClient`, y son:

- ***WordSuggestionsClient*** clase encargada de mostrar las sugerencias cuando esté habilitado; estas sugerencias aparecerán al lado de la palabra mientras se vayan escribiendo.

- ***WordReplacementsClient*** clase encargada de mostrar la lista de las palabras que pueden reemplazan a la palabra incorrecta seleccionada.

6. Implementación de la Vista

La *Vista* es la responsable de representar gráficamente el modelo. Como mínimo, tiene que ser capaz de tener un sistema de selección de archivos de texto para analizar y una manera corregir las palabras incorrectas. Con el fin de llevar esto a cabo, se ha dividido la representación de cada requisito en distintas clases:

1. ***FindPanel***: panel en el que se gestionan los elementos gráficos del ***Find & Replace*** [4]. Este panel contiene:

   - **FindTextPanel** panel en el cual se introduce la palabra que se desea encontrar en el texto. Debido a que puede haber más de una ocurrencia, se dispone de 2 botones mediante los cuales se puede ir saltando de ocurrencia en ocurrencia.

   - **ReplaceTextPanel** panel en el que se introduce la palabra con la que se desea reemplazar la palabra encontrada. Además de un recuadro de texto en el que introducir dicha palabra, hay dos botones para reemplazar: un botón para reemplazar una sola ocurrencia (se elige la deseada mediante los botones mencionados en el FindTextPanel), y un botón para reemplazar todas las ocurrencias.

2. ***Notepad***: panel con el cual se representa el editor de texto. Este panel contendrá el texto que se encuentre en el fichero seleccionado, o el texto que se vaya escribiendo. Dentro de este panel además se irán marcando las palabras ortográficamente incorrectas y las ocurrencias de las palabras que se estén buscando mediante el *Find & Replace*.
   Este panel tiene 2 `Listeners`:

   - Un `MouseListener`: Para detectar cuando se haya seleccionado una palabra del texto y en caso de que sea incorrecta, se mostrará una lista debajo de la palabra con los sugerimientos de reemplazo
   - Un `KeyListener`: Para detectar mientras se vayan escribiendo las palabras las faltas de ortografía. Las palabras escritas se analizarán siempre y cuando se haya pulsado una tecla con un símbolo especial (espacio, salto de línea, signos de puntuación, etc). Sin embargo, **cuando el símbolo requiere que se pulse el SHIFT o ALT o alguna otra tecla, como por ejemplo el ?, la palabra no se detectará.** (Ver apartado de Problemas encontrados para una explicación más extensa)

3. ***Sidebar***: panel en el cual se muestran las diferentes palabras incorrectas. Esta barra lateral se va actualizando a medida que se van detectando las palabras (analizando el texto entero, escribiendo las palabras o borrando las palabras). Para que el usuario tenga más control, se ha añadido una manera de poder corregir la palabra desde este mismo panel.

   Debajo de cada palabra incorrecta se muestra un desplegable con las distintas palabras con las que puede sustituir organizadas por las distancias de Levenshtein. Como bien se mencionó al principio, las palabras sugeridas para reemplazar pertenecen a sólo 2 distancias.

   Esto quiere decir que si la palabra no tiene palabras a una distancia 1, se mostrarán las palabras con distancias 2 y 3, y así succesivamente. Cabe recordar que la máxima distancia permitida es la 4. Si una palabra no tiene ninguna palabra para reemplazar se avisará.

4. ***Window***: ventana principal del programa que recoge todos los demas componentes. Dicha ventana contiene:

   a) el editor de texto

   b) los paneles de *Find & Replace*

   c) la barra lateral que contendrán las palabras incorrectas

   d) un Menú con distintas opciones. A continuación se explicarán las distintas pestañas y todas las opciones que contienen:

   - ***File***
     - ***New***: permite la edición del `Notepad`.
     - ***New from existing***: añade el contenido de un fichero al `Notepad`

y permite la edición del mismo. Cuando se carga el contenido, se analiza para encontrar los fallos ortográficos.

- ***Open***: abre un fichero y añade su contenido al `Notepad`, pero no permite su edición. Para ello se debe seleccionar *Edit file...* de la pestaña del menú *Tools*.

- ***Edit***
  - ***Undo***: deshace la acción anterior
  - ***Redo***: rehace la acción posterior
- ***View***
  - ***Hide mispelled words panel***: esconde la barra lateral
- ***Tools***
  - ***Edit file***: permite la edición del texto. Cuando se selecciona, se convierte en "Stop editing file". Cuando se pulsa en este estado, el texto se analiza para encontrar los fallos ortográficos.
  - ***Correct spelling***: analiza el texto para encontrar los fallos otrográficos. Las palabras incorrectas se añadirán a la barra lateral
  - ***Find***: muestra el panel para encontrar una palabra en el texto. En caso se que exista, se podrá navegar por las distintas ocurrencias de la misma.
  - ***Replace***: muestra el panel para reemplazar una palabra en el texto.
- ***Preferences***
  - ***Enable suggestions***: habilita las sugerencias de palabras mientras se va escribiendo en el editor.
  - ***Select language***: sirve para elegir el idioma del editor. Se dispone de 2 idiomas: Español e Inglés. Cuando se cambia el idioma se analiza el texto escrito en el editor para encontrar los fallos ortográficos.

e) Por último, contiene una barra en el que se muestra el idioma seleccionado en este momento y el estado del texto, es decir, si hay fallos ortográficos o si el texto es correcto.

Se han utilizado clases extra para el diseño de la interfaz. Dichas clases se encuentran dentro de la carpeta **`Utils`** y son las siguientes:

- ***DocumentSizeFilter***: sirve para restringir la cantidad de caracteres que se pueden tener en el editor.
- ***MenuBuilder***: sirve para automativar la construcción del menú. Contiene distintas estructuras:
  - `String [] MENU_ITEMS_ORDER`: sirve para saber cuál es el orden de los elementos del menú.
  - `ArrayList<String>IS_SUBMENU`: aquí se encuentran aquellos elementos del menú que con submenús. Es el caso de *Select language*.
  - `ArrayList<String>ADD_SEPARATION_AFTER`: sirve para saber después de qué elementos del menú hay que añadir un JSeparator.
  - `Map<String, ArrayList>MAP_MENU_ITEMS`: define la estructura de los menús y de los elementos que contendrá. Por ejemplo la definición del menú File es:

```
new AbstractMap.SimpleEntry<>(Constants.TEXT_FILE_MENU, new ArrayList<>()
        {
            add(Constants.TEXT_NEW_FILE_ITEM);
            add(Constants.TEXT_NEW_FROM_EXISTING_ITEM);
            add(Constants.TEXT_OPEN_FILE_ITEM);
        }
    }
)
```

  - `Map<String, String>KEYSTROKES`: contiene los *Keystrokes* de los elementos del menú.
  - `Map<String, String>MENU_ICONS`: contiene los iconos de los elementos del menú.
  - `Map<String, JMenuItem>MENU_ITEMS`: sirve para guardar los componentes JMenuItem de cada elemento del menú para que se puedan acceder a ellos.
  - `Map<String, ActionListener>MENU_ACTION`: contiene los ActionListener de cada elemento del menú.
  - `Map<String, Integer>KEYEVENTS`: contiene las teclas que se tiene que pulsar para que se acceda a un elemento del menú.
- ***UnderlineHighlightPainter***: clase mediante la cual se definen los HighlightPainters. Hay definidos dos: el que sirve para subrayar las palabras incorrectas y el que sirve para mostrar las palabras cuando se buscan mediante el *Find*.

7. IMPLEMENTACIÓN DEL CONTROLADOR

El *Controlador* es el elemento que monitoriza y controla las comunicaciones entre la Vista y el Modelo.

Los atributos que se encuentran en el controlador son:

1. `SoundexDictionary soundex` diccionario soundex al que se accede cuando el idioma seleccionado es el Inglés.

2. `static Dictionary dictionary` diccionario que se consulta para encontrar las palabras. Es static debido a que se debe utilizar en un contexto estático (el método getWords al que se accede a través de WordSuggestionsClient).

3. `static ArrayList<Word>mispelledWords` lista en las que se guardan todas las palabras ortográficamente incorrectas. Se va modificando a medida que se van añadiendo/corrigiendo palabras.

4. `boolean isSoundexDictionary` variable que indica si se trata de un diccionario Soundex o no. Sirve para saber de qué manera se deben encontrar las palabras para reemplazar las incorrectas.

5. `ExecutorService executor` hilo que ejecutará el análisis del texto, la búsqueda de una palabra en el texto, el proceso de análisis de una palabra escrita en tiempo real, y la correción del texto entero.

6. `static int distance` sirve para saber a qué distancia Levenshtein se están buscando las palabras.

7. `static boolean suggestionsEnabled` sirve para saber si están activadas las sugurencias de palabras al escribir en el editor.

8. `boolean suggestionUsed` sirve para saber si se ha corregido ya una palabra mediante una sugerencia, para no añadirla a las palabras incorrectas.

9. `boolean dictPopulated` sirve para saber si el diccionario soudnex ya está cargado.

10. `HashMap<Integer, Word>mispelledWordsCursorPos` se guardan las ultimas posiciones de las palabras incorrectas. Mediante ellas se va indexando el texto para poder corregirlas y saber exactamente dónde se encuentran.

11. `HashMap<String, Dictionary>languageDictionaries` se guardan los diccionarios cargados para cada idioma.

12. `HashMap<String, String>dictionaryPath` contiene las rutas a los diccionarios de cada idioma.

13. `HashMap<String, Language>availableLanguages` contiene la lista de idiomas disponibles en programa.

El controlador, además de los métodos de get y set, implementa la interfaz `IController` que tiene los siguientes métodos:

`void addMispelledWord(Word word)`: método mediante el cual se añade a la lista de `mispelledWords` una palabra incorrecta. Primero se comprueba que dicha palabra no existe ya, y si existe, si mira si está en una posición distinta del texto, ya que se puede dar el caso de que se escriba mal más de una vez.

`void addReplaceWord(Word wordToFind, String replaceWord, int distance)`: método que se llama al buscar una palabra en el diccionario para añadir la palabra `replaceWord` a la lista de palabras por las que se puede reemplazar la palabra `wordToFind`. Si la distancia a la que se encuentra está dentro del rango especificado, se añade a la lista.

`void checkText()`: analiza el texto entero para detectar los errores ortográficos.

`void correctSpellingFromText()`: reemplaza todas las palabras incorrectas por la primera palabra que encuentra para sustituir.

`void correctMispelledWord(Word mispelledWord, String correctedWord)`: método que se llama al pulsar el botón de `Replace` del panel `ReplaceTextPanel`. Se busca primero la palabra que se quiere sustituir, la sustituye, y a continuación se actualizan los indices del resto de palabar incorrectas para saber dónde se encuentran.

`void deleteMispelledWord(int idx)`: elimina la palabra incorrecta de la lista.

`boolean findWordInDictionary(Word wordToFind)`: busca la palabra que le pasan por parametro dentro del diccionario. Dependiendo del idioma, buscará en el diccionario normal o en el `SoundexDictionary`. La palabra se busca en el diccionario utilizando las distancias de levenshtein. Si se encuentra la palabra se devuelve un verdadero, pero si no se encuentra se van añadiendo las palabras con distancia menor o igual que la máxima a la lista de `replaceWords` de la palabra.

`Future<ArrayList<Integer>findWordInText(String wordToFind)`: método que se utiliza para encontrar una palabra en el texto. Devuelve una lista con los índices en los que se encuentra.

`String getFirstReplaceWord(Word word)`: devuelve la primera palabra de la lista de `replaceWords` de la palabra. Va en orden desde la mínima distancia hasta la máxima, en caso de que no se encuentre en la lista de distancia 1 se busca en la de distancia 2, y así sucesivamente.

`ArrayList<Language>getLanguages()` : analiza la carpeta `dicc/` y crea los elementos `Language`.

`static ArrayList<Word>getReplaceWords(String input)`: retorna la lista de palabras con las que se pueda sustituir la palabra que se pasa por parametro. Las palabras que se devuelven son aquellas que se corresponden a la distancia que se tiene guardada en la variable estática `distance`. Este método se llama cuando se pulsa encima de una palabra incorrecta para poder obtener la lista de palabras que se tiene que mostrar en el desplegable.

`static ArrayList<Word>getWords(String input)`: método que retorna una lista de palabras que empiezan por el input que se pasa por parametro. Se llama cuando están activados los sugerimientos.

`void replaceMispelledWordFromText(int idx, int lengthDifference)`: método que se llama cuando se substituye una palabra incorrecta por una propuesta. Dentro de este se elimina la palabra de la lista y se actualizan las posiciones de las otras.

`int replaceWord(int index, int lengthPreviousWord, String replacement)`: método que se llama cuando se pulsa el botón de `Replace` del panel `ReplaceTextPanel`. Sustituye la palabra que se encuentra en la posición `idx` (ya que puede haver varias) que se ha buscado por la nueva y actualiza los índices del resto de palabras incorrectas.

`void replaceWords(String old, String newWord)`: método que se llama cuando se pulsa el botón de `Replace All` del panel `ReplaceTextPanel`. Sustituye todas las ocurrencias de la palabra que se ha buscado por la nueva y actualiza los índices del resto de palabras incorrectas.

`void updateMispelledCursorEnds(int idx, int lengthDifference)`: método que sirve para actualizar las posiciones de las palabras incorrectas tras la modificación de una de estas palabras. Las nuevas posiciones serán las anteriores sumando el `lengthDifference`.

8. Coste computacional

El coste computacional del algoritmo de las distancias de Levenshtein es de

$$O(n * m) = O(n^2)$$

donde n es la longitud de la primera palabra y m es la longitud de la segunda palabra.

9. Guía de usuario

En esta guía se explicarán los requerimientos del proyecto, las funcionalidades del gestor de ficheros implementado, los pasos a seguir para poder utilizar el programa y, por último, se mostrarán varios ejemplos de ejecución.

9.1. Limitaciones del programa

La manipulación de las posiciones de las palabras incorrectas es muy compleja, sobretodo si se tiene en cuenta que es un sistema interactivo en el que se analiza a medida que se va escribiendo.

Por lo consecuente, la edición de las palabras ya marcadas como incorrectas no se hace de forma satisfactoria: si se edita una palabra incorrecta, es decir, se borra y se escribe en la misma posición, se irán añadiendo palabras incorrectas con cada letra que se pulse (debido a que detecta que el carácter que sigue es un espacio).

Por lo tanto, para un correcto funcionamiento **NO SE DEBEN SUSTITUIR LAS PALABRAS INCORRECTAS EDITANDO EL TEXTO EN EL EDITOR A MANO**, sino que se debe utilizar una de las tres maneras proporcionadas por el sistema:

1. desde la barra lateral
2. sustituyendo la palabra con *Find & Replace*
3. pulsando encima de la palabra y eligiendo una de las palabras del desplegable

9.2. Ejecución del programa

9.2.1. *Carga de texto en el editor*

Para empezar a ejecutar el programa primer se debe seleccionar el texto: o bien a través del menú abriendo un fichero `File >Open`, o bien empezando a escribir en el editor (se debe habilitar la escritura o bien seleccionado las opciones `File >New` o `File >New from existing` del menú o bien la opción `Tools >Edit File`).

9.2.2. *Análisis del texto*

Una vez cargado el texto aparecerán las palabras incorrectas en la barra lateral, además se quedar subrayadas en el panel con rojo.

9.2.3. *Sustitución de palabras incorrectas*

Para poder sustituir las palabras se puede:

1. hacer click encima de la palabra en el editor y seleccionar una palabra del desplegable
2. ir a la barra lateral y seleccionar una palabra del desplegable
3. mediante el panel de *Find & Replace*

9.2.4. *Corrección del texto*

Se realiza mediante la opción del menú `Tools>Correct Spelling`.

10. Problemas encontrados

10.1. Índices de las palabras incorrectas

El sistema interactivo ha resultado ser muy complicado a la hora de gestionar la posición de las palabras incorrectas, con el fin de saber dónde sustituirlas en el texto. No se ha conseguido tener un sistema 100 % fiable y correcto (ver las Limitaciones del proyecto [9.1]).

Figura 5: Ejemplo de ejecución corrigiendo un texto

10.2. Detección de palabras escritas antes que un carácter especial

La detección de las palabras en tiempo real se hace en el `KeyEvent KeyPressed`, por lo que cuando se escribe un carácter como por ejemplo ! o ? primero se debe pulsar SHIFT y luego se pulsa el carácter deseado. Esto complica la detección de la tecla que se ha pulsado, y por lo consecuente si se escribe por ejemplo olah? no se detectará como incorrecta.

11. Conclusión

Esta práctica no ha sido nada fácil debido a la dificultad de detectar las palabras incorrectas en tiempo real en todas las ocasiones. Aún así, se ha podido ver que situaciones que parecen muy costosas (encontrar una palabra en el diccionario) no lo son debido a la potencia de cálculo de los ordenadores de hoy en día.