**MANUAL TÉCNICO**

**ANALIZADOR DE TEXTOS**

**POR:** JUAN FELIPE LÓPEZ VERGARA

**MATERIA:** Lógica III

**PROFESOR**: Roberto Flórez

**FECHA:** 1 de noviembre de 2014

UNIVERSIDAD DE ANTIOQUIA

DEPARTAMENTO INGENIERÍA DE SISTEMAS

MEDELLÍN

2014

Contenido

[1. INTRODUCCIÓN 5](#_Toc403854329)

[2. ALCANCE 5](#_Toc403854330)

[3. OBJETIVOS 5](#_Toc403854331)

[3.2. Objetivo general 5](#_Toc403854332)

[3.3. Objetivos específicos 5](#_Toc403854333)

[4. CONTENIDO 5](#_Toc403854334)

[4.1 . Definición del problema 5](#_Toc403854335)

[1. Obtener las preposiciones 5](#_Toc403854336)

[2. Obtener palabras tildadas 5](#_Toc403854337)

[3. Eliminar Nodo 6](#_Toc403854338)

[4. Orden Inverso 6](#_Toc403854339)

[5. Sustituir palabra 6](#_Toc403854340)

[6. Ordenar Alfabéticamente 6](#_Toc403854341)

[7. El texto leído 6](#_Toc403854342)

[4.1.1. Análisis del problema 6](#_Toc403854343)

[1. Obtener las preposiciones 6](#_Toc403854344)

[2. Obtener palabras tildadas 6](#_Toc403854345)

[3. Eliminar Nodo 7](#_Toc403854346)

[4. Orden Inverso 7](#_Toc403854347)

[5. Sustituir palabra 7](#_Toc403854348)

[6. Ordenar Alfabéticamente 7](#_Toc403854349)

[7. El texto leído 8](#_Toc403854350)

[4.2. Clases 8](#_Toc403854351)

[4.2.1. Clase AnalizadorTextos.java 8](#_Toc403854352)

[4.2.2. Clase NodoD 8](#_Toc403854353)

[4.2.2.1. Variables globales 8](#_Toc403854354)

[ String palabra 8](#_Toc403854355)

[ Int cont 8](#_Toc403854356)

[ NodoD siguiente 8](#_Toc403854357)

[ NodoD anterior 8](#_Toc403854358)

[4.2.2.2. Contructores 9](#_Toc403854359)

[ Constructor vacío 9](#_Toc403854360)

[ Constructor con la palabra 9](#_Toc403854361)

[ Constructor con la palabra y el contador 9](#_Toc403854362)

[4.2.2.3. Asignaciones y retornos 9](#_Toc403854363)

[ asignaCont 9](#_Toc403854364)

[ retornaCont 9](#_Toc403854365)

[ asignaPalabra 10](#_Toc403854366)

[ retornaPalabra 10](#_Toc403854367)

[ asignaSiguiente 10](#_Toc403854368)

[ asignaAnterior 10](#_Toc403854369)

[ retornaSiguiente 10](#_Toc403854370)

[ retornaAnterior 11](#_Toc403854371)

[4.2.3. Clase ListaDCabeza 11](#_Toc403854372)

[4.2.3.1. Variables 11](#_Toc403854373)

[ Variables de tipo NodoD 11](#_Toc403854374)

[ Vectores 11](#_Toc403854375)

[ Variable de tipo String 11](#_Toc403854376)

[4.2.3.2. Métodos 12](#_Toc403854377)

[ Métodos de asignación 12](#_Toc403854378)

[ Métodos de retorno 12](#_Toc403854379)

[ Constructor 12](#_Toc403854380)

[ Método esVacia() 12](#_Toc403854381)

[ Método limpiarLista() 13](#_Toc403854382)

[ Método insetarPalabra(String x) 13](#_Toc403854383)

[ Método mostrarLista() 14](#_Toc403854384)

[ Método mostrarPreposiciones() 14](#_Toc403854385)

[ Método buscar(String palabra) 15](#_Toc403854386)

[ Método eliminarPalabra(String palabra) 16](#_Toc403854387)

[ Método textoInverso() 16](#_Toc403854388)

[ Método sustuirPalabra(String reemplazar, String con) 16](#_Toc403854389)

[ Método insertarPalabra (String x, int i) 17](#_Toc403854390)

[ Método contarPalabras() 17](#_Toc403854391)

[ Método ordenAlfabetico() 19](#_Toc403854392)

[ Método mostrarTildadas() 20](#_Toc403854393)

[4.2.4. Clase Ventana 21](#_Toc403854394)

[4.2.4.1. Variables globales 21](#_Toc403854395)

[ Variables para el manejo de texto: 21](#_Toc403854396)

[ Lista ligada: 21](#_Toc403854397)

[4.2.4.2. Constructor: 21](#_Toc403854398)

[4.2.4.3. Métodos 21](#_Toc403854399)

[ Método botonSalirActionPerformed 21](#_Toc403854400)

[ Método botonLimpiarActionPerformed 21](#_Toc403854401)

[ Método botonGuardarActionPerformed 22](#_Toc403854402)

[ Método menuItemMostrarPreposicionesActionPerformed 22](#_Toc403854403)

[ Método menuItemEliminarPalabraActionPerformed 22](#_Toc403854404)

[ Método menuItemMostrarInvertidoActionPerformed 23](#_Toc403854405)

[ Método menuItemReemplazarPalabraActionPerformed 23](#_Toc403854406)

[ Método menuItemMostrarTildadasActionPerformed 24](#_Toc403854407)

[ Método menuItemOrdenAlfabeticoActionPerformed 24](#_Toc403854408)

[ Método botonExaminarActionPerformed 25](#_Toc403854409)

[5. CONCLUSIONES 25](#_Toc403854410)

[6. LISTA DE ILUSTRACIONES 26](#_Toc403854411)

# INTRODUCCIÓN

La finalidad de este manual técnico es proporcionar al lector la lógica con la que fue desarrollada la aplicación. Este manual no pretende ser guía de aprendizaje para el lector, sino documentar el proceso de desarrollo de la aplicación y proceso para la solución del problema general presentado para la creación del sistema.

# ALCANCE

Este manual va dirigido a un público con conocimientos técnicos sobre la programación orientada a objetos, en él se describen las clases usadas para la solución del problema y los métodos con su respectivo funcionamiento, el lenguaje bajo el que fue creado es java, usando la interfaz Netbeans IDE 8.0.1.

# OBJETIVOS

## Objetivo general

Elaborar un programa que de solución al problema presentado a continuación procurando continuamente que sea lo más amable posible con el usuario.

## Objetivos específicos

* Diseñar el programa con una interfaz gráfica sencilla y legible.
* Elaborar un algoritmo estructurado y con una lógica entendible para el público con conocimientos técnicos en el tema.

# CONTENIDO

# . **Definición del problema**

Elaborar un programa que lea un archivo de texto y almacene cada palabra en un nodo de una lista doblemente ligada con nodo cabeza. El programa debe incluir un menú con las siguientes opciones:

1. Obtener las preposiciones**:** dada la lista creada a partir del texto leído, crear una nueva lista ligada con las preposiciones, sin repetir, que tiene la lista del texto, además de mostrar en pantalla la lista de las preposiciones y cuántas veces aparece cada una.
2. Obtener palabras tildadas**:** crear una nueva lista ligada con las palabras tildadas que contiene la lista del texto, mostrar en pantalla dichas palabras y cuántas veces está cada una de ellas.
3. Eliminar Nodo**:** eliminar el nodo que se encuentra antes de una palabra dada, el usuario debe indicar la palabra.
4. Orden Inverso**:** mostrar los datos de la lista del texto en orden inverso.
5. Sustituir palabra**:** reemplazar una palabra por otra todas las veces que se encuentre en el texto, ambas palabras deben ser indicadas por el usuario.
6. Ordenar Alfabéticamente**:** organizar las palabras de la lista del texto en orden alfabético. Los resultados se deben ver reflejados en la misma lista. Además, mostrar todas las palabras y el número de veces que se halla cada una de ellas.
7. El texto leídodebe mostrarse en pantalla permanentemente.

### 4.1.1. Análisis del problema

Es importante resaltar que los archivos con extensión (.txt) se guardan con diferentes codificaciones, la que funciona correctamente en el programa es la codificación ANSI.

Se requiere leer un archivo de texto y almacenar cada una de las palabras en un nodo dentro dentro de una lista doblemente ligada con nodo cabeza, para lo cual se debe realizar la lectura del archivo y luego almacenar todo su contenido en un área de texto desde la cual se lee nuevamente y se llevan sólo las palabras a la lista ligada, por lo cual se debe hacer una limpieza del texto y eliminar todos los símbolos especiales, signos de puntuación y demás.

1. Obtener las preposiciones**:** se debe recorrer la lista y comparar la palabra del nodo actual con todas las del vector donde están almacenadas todas las preposiciones y cuando la encuentra simplemente la lleva a una nueva lista ligada y en un contador en la posición correspondiente a dicha preposición aumentar en uno cada que la encuentre para saber el número de repeticiones, cabe anotar que si dicho contador es diferente de cero la palabra no se lleva a la nueva lista ligada. Por último se lleva la nueva lista ligada a un string junto con las preposiciones y la cantidad de veces que se repiten cada una y se retorna.

Las preposiciones usadas son las que define la RAE que son: “*a, ante, bajo, cabe, con, contra, de, desde, durante, en, entre, hacia, hasta, mediante, para, por, según, sin, so, sobre, tras, versus* y *vía”*

1. Obtener palabras tildadas**:** se debe recorrer la lista y haciendo uso del método matches sobre la palabra actual se define si es o no una palabra tildada o no y se lleva a una nueva lista que al final simplemente se retorna haciendo uso de un string.
2. Eliminar Nodo**:** El usuario proporciona una palabra que está situada justo luego de la palabra que se va a eliminar, hay que tener en cuenta que si la palabra no está en el texto debe retornar diciendo que la palabra no existe, por otro lado se debe tener en cuenta que se ingrese sólo una palabra. Además se debe tener en cuenta que si la palabra ingresada es la primera de la lista no habrá nada adelante que se pueda eliminar, también es importante aclarar que se pide que el usuario ingrese la palabra, no la posición y tampoco se especifica qué hacer cuando la palabra de referencia ingresada por el usuario está varias veces dentro de la lista, por lo cual se eliminará sólo la palabra que esté antes de la primer aparición de la palabra de referencia y el resto se dejarán intactas, para ello basta con hacer un método que recorra la lista comparando la palabra dada por el usuario con la palabra del nodo actual y si la encuentra simplemente se realizan las acciones necesarias para desconectar el nodo anterior a esta, luego se retorna un string con el resultado de la operación sobre la lista.
3. Orden Inverso**:** basta con recorrer la lista desde el último nodo hasta el primero e ir agregando cada palabra a un string, cuando se llegue a la cabeza se retorna dicho string con el resultado, cabe anotar que no se pide la creación de una nueva lista, simplemente imprimir las palabras en orden inverso.
4. Sustituir palabra**:** el usuario debe ingresar las dos palabras, tanto la palabra que se va a sustituir como la palabra por la cual será sustituida, por lo cual se debe garantizar el ingreso de una sola palabra. Para hacer el reemplazo basta con recorrer la lista buscando la palabra que se va a reemplazar y cuando se encuentra se cambia por la palabra de sustitución, cabe anotar que el método para buscar la palabra hace la búsqueda hasta que su resultado sea nulo, es decir hasta que la palabra no aparezca en la lista, luego de esto retorna el resultado de la operación. Por otro lado si la palabra que se quiere reemplazar no existe aparece un mensaje diciendo esto.
5. Ordenar Alfabéticamente**:** usando dos auxiliares se recorre toda la lista comparando la primer palabra con todas las restantes y dependiendo del resultado de la comparación se intercambian o se dejan en su posición actual dentro de la misma, al terminar de comparar la primer palabra con todas la demás la primer palabra estará en su posición correspondiente dentro de la lista, ahora se compara la segunda palabra con todas las que hay a su derecha y así sucesivamente se hace con todas hasta que todas queden en orden alfabético, finalmente, retorna el resultado de la operación.
6. El texto leído**:** para ello se hace uso de un área de texto en la cual simplemente se imprime el texto leído.

## Clases

### Clase AnalizadorTextos.java

Esta es la clase principal del programa la cual crea la ventana.

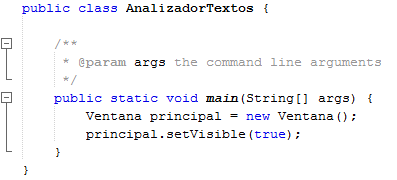


Ilustración 1. Clase AnalizadorTextos .java

### Clase NodoD

Esta es la clase que contiene toda la configuración del tipo de nodo que se va a usar dentro de la lista que es el nodo doble, lo que significa que contiene un apuntador al siguiente nodo y un apuntador al anterior, además de una palabra y hay un constructor especial que permite además añadir al nodo un contador usado para contar la repetición de las palabras.

#### Variables globales

* String palabra**:** contiene la palabra que será almacenada en el nodo.
* Int cont**:** es un entero usado para almacenar la cantidad de veces que se repite la palabra en la lista.
* NodoD siguiente**:** se usa para determinar el siguiente nodo.
* NodoD anterior**:** se usa para determinar el anterior nodo.

#### Contructores

* Constructor vacío**:** permite crear un nodo vacío.



Ilustración 2. Constructor vacío

* Constructor con la palabra**:** permite crear un nodo y asignarle una palabra y sus campos de siguiente y anterior se asignan nulos.

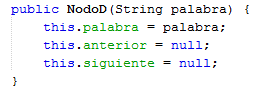


Ilustración 3. Constructor con la palabra

* Constructor con la palabra y el contador**:** este nodo en particular sólo se usa para contar la palabra almacenada dentro del mismo haciendo uso del entero que lo acompaña, al nodo se le asigna la palabra y el contador.

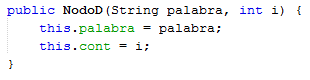


Ilustración 4. Constructor con el contador y la palabra

#### Asignaciones y retornos

Se usan para asignar y retornar cada una de las variables al nodo y se muestran a continuación.

* asignaCont**:** este método asigna el contador al nodo, recibe como parámetro de entrada un int cont y lo asigna a la variable global cont, no retorna nada.



Ilustración 5. Método asignaCont

* retornaCont**:** este método retorna el contador del nodo y no recibe nada.



Ilustración 6. Método retornaCont

* asignaPalabra**:** este método asigna la palabra al nodo, recibe como parámetro de entrada un String palabra y lo asigna a la variable global palabra, no retorna nada.



Ilustración 7. Método asignaPalabra

* retornaPalabra**:** este método no recibe ningún parámetro de entrada y retorna la palabra que contiene el nodo.



Ilustración 8. Método retornaPalabra

* asignaSiguiente**:** recibe un NodoD siguiente y lo asigna a la variable global siguiente y no retorna nada.



Ilustración 9. Método asignaSiguiente

* asignaAnterior**:** recibe un parámetro de entrada de tipo NodoD anterior y lo asigna a la variable anterior, no retorna nada.



Ilustración 10. Método asignaAnterior

* retornaSiguiente**:** método que retorna el valor de la variable global siguiente, no recibe parámetro de entrada.

****

Ilustración 11. Método retornaSiguiente

* retornaAnterior**:** método que retorna el valor de la variable global anterior, no recibe parámetro de entrada.



Ilustración 12. Método retornaAnterior

### Clase ListaDCabeza

Esta clase contiene todas las funcionalidades y métodos que hacen posible realizar las modificaciones requeridas por el programa, a continuación se describen las variables, métodos y demás.

#### Variables

* Variables de tipo NodoD**:** las variables de tipo NodoD son: aux, cabeza, primero y ultimo, los cuales se usan dentro de los métodos para hacer los recorridos en las lista.

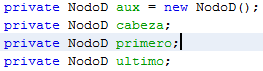


Ilustración 13. Variables de tipo NodoD

* Vectores**:** esta clase cuenta con dos vectores uno de tipo String para almacenar las preposiciones y uno de tipo int que se usa para contar la cantidad de veces que se repiten las preposiciones dentro de la lista.

****

Ilustración 14. Vectores

* Variable de tipo String**:** es una variable llamada textoMostrar que se usa dentro de los métodos para almacenar el resultado de las operaciones y retornarlo.

#### Métodos

* Métodos de asignación**:** asignan los valores correspondientes a las variables de tipo NodoD cabeza, primero y último, reciben una variable de tipo NodoD y no retornan nada.

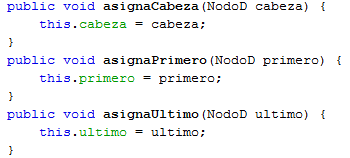


Ilustración 15. Métodos de asignación

* Métodos de retorno**:** retornan los valores de las variables de tipo NodoD y no reciben ningún parámetro.

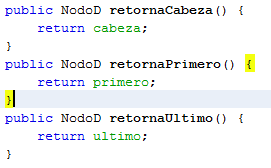


Ilustración 16. Métodos de retorno

* Constructor**:** instancia el nodo cabeza y asigna los valores correspondientes al vector de las preposiciones.

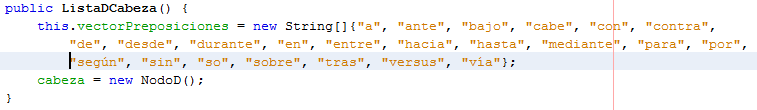


Ilustración 17. Constructor ListaDCabeza

* Método esVacia() **:** determina si la lista de palabras está vacía, retorna un valor booleano dependiendo del resultado de la operación y no tiene parámetro de entrada, si el valor de primero es igual a null entonces la lista es vacía.

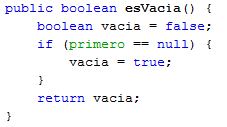


Ilustración 18. Método esVacia

* Método limpiarLista() **:** asigna al nodo primero de la lista de palabras el valor de null y no recibe ni retorna nada.



Ilustración 19. Método limpiarLista

* Método insetarPalabra(String x)**:** este método recibe una palabra crea un nuevo NodoD y le asigna el string x, luego determina si la lista es vacía y hace el nodo nuevo igual al primero y al último de la lista y a la cabeza le asigna como siguiente nodo el primero. Si la lista no es vacía hace el nodo nuevo igual al siguiente del ultimo y hace el último igual al nuevo. Este método no retorna nada.

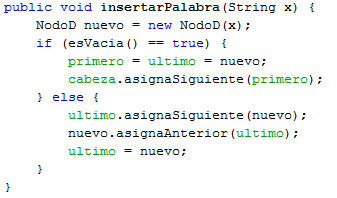


Ilustración 20. Menú insertarPalabra

* Método mostrarLista()**:** se usa un nodo auxiliar para recorrer la lista, mientras este es diferente de null se retorna la palabra de dicho nodo y se lleva al String textoMostrar y finalmente se retorna su valor.

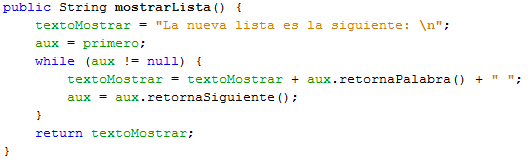


Ilustración 21. Método mostrarLista

* Método mostrarPreposiciones()**:** no recibe nada, crea una nueva ListaDCabeza que será la lista de preposiciones, asigna el tamaño del vector cont y hace un nodo aux igual al primer nodo de la lista de palabras y mientras este sea diferente de null se crea un contador inicialiazado en cero y mientras este contador sea menor que el tamaño del vector de preposiciones se compara la palabra del nodo aux con todas las palabras del vector y en el momento que coincide con alguna simplemente se lleva la palabra a la nueva lista de preposiciones y el vector cont aumenta en uno la posición correspondiente a la preposición encontrada, si este contador es diferente de cero entonces la palabra no se lleva a la lista de preposiciones y lo único que se hace es aumentar en uno la posición del vector cont correspondiente a la posición de la preposición en el vector de preposiciones.

Si la lista de preposiciones es vacía retorna un mensaje para informarlo, sino se lleva la lista al String textoMostrar, se hace el nodo aux igual al primero de la lista de preposiciones y mientras este sea diferente de null se crea un entero i=0, mientras este sea menor que la longitud del vector preposiciones se avanza comparando la palabra del nodo aux, cuando ambas palabras coinciden la palabra es llevada al String textoMostrar además de llevar también el contador de las repeticiones y finalmente se retorna el resultado de la operación.



Ilustración 22. Método mostrarPreposiciones

* Método buscar(String palabra)**:** recibe la palabra que se va a buscar, se crea un nodo y se hace igual al primero de la lista, luego mientras la palabra del nodo sea diferente de la palabra de búsqueda y el nodo sea diferente de null se avanza a través de la lista y finalmente se retorna el uno NodoD con la posición de la palabra dentro de la lista.

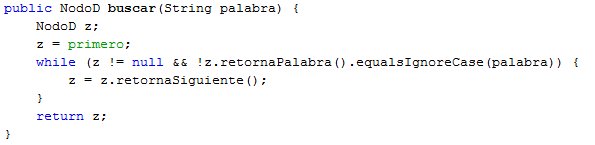


Ilustración 23. Método buscar

* Método eliminarPalabra(String palabra)**:** recibe la palabra que está situada luego de la palabra que se va a eliminar y se busca haciendo uso del método buscar, cuando se encuentra se desconecta el nodo anterior a este y se retorna el resultado de la operación sobre la lista usando un String, si la palabra no está en la lista se retorna un mensaje informando que la palabra de referencia no está, cabe anotar que no se especifica la posición de la palabra, por lo cual el método sólo usa como referencia la primer palabra encontrada que coincida con la palabra de referencia.

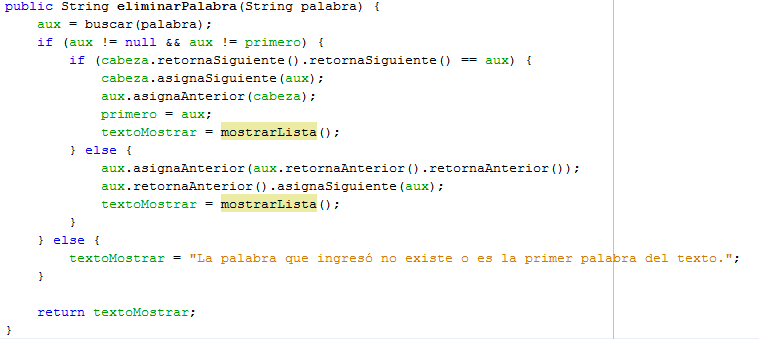


Ilustración 24. Método eliminarPalabra

* Método textoInverso()**:** no recibe parámetro de entrada, se hace un nodo aux igual al ultimo de la lista y se recorre hasta que sea igual a null haciendo uso del método retornaAnterior() y almacenando el resultado en un String que al final del recorrido se retorna.

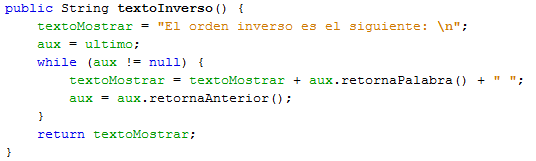


Ilustración 25. Método textoInverso

* Método sustuirPalabra(String reemplazar, String con)**:** se reciben dos palabras la que se va a reemplazar y la palabra de reemplazo, se crea un nodo y se hace igual al resultado de la búsqueda de la palabra que se va a reemplazar, si la palabra no está en la lista retorna un string informando de esto, de lo contrario al nodo le asigna la palabra de reemplazo y nuevamente busca en la lista la palabra a reemplazar y así sucesivamente hasta que el resultado de la búsqueda sea null y finalmente retorna el string con el resultado de la operación sobre la lista.

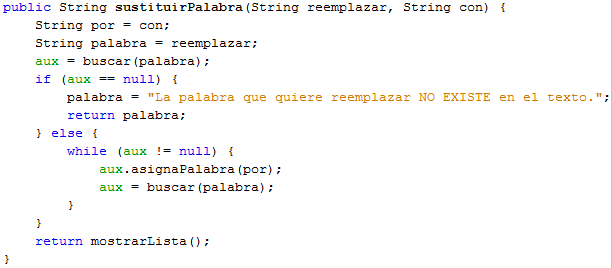


Ilustración 26. Método sustituirPalabra

* Método insertarPalabra (String x, int i)**:** este método recibe una palabra y un entero crea un nuevo NodoD y le asigna el string x y el entero i, luego determina si la lista es vacía y hace el nodo nuevo igual al primero y al último de la lista y a la cabeza le asigna como siguiente nodo el primero. Si la lista no es vacía hace el nodo nuevo igual al siguiente del ultimo y hace el último igual al nuevo. Este método no retorna nada.

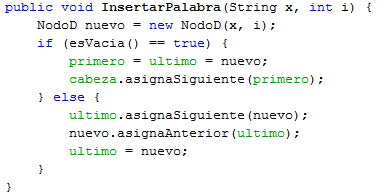


Ilustración 277. Método insertarPalabra con contador

* Método contarPalabras()**:** no recibe nada, se crea una lista “z” y se usan dos nodos uno llamado aux para recorrer la lista principal y uno llamado “y” para recorrer la lista “z”, luego se inserta la palabra contenida en el nodo aux a la lista “z” usando el **Método insertarPalabra (String x, int i)** y se aumenta su parámetro cont en 1, seguidamente se hace el nodo “y” igual al primero de la lista “z”, se crea un entero cont=0 y se hace aux igual al siguiente de aux, ahora mientras el nodo aux es diferente de null y el nodo “y” es diferente de null se comparan las palabras de ambos nodos, si son iguales se aumenta el contador cont, luego se aumenta el parámetro cont de “y” en uno y se hace “y” igual a null. De lo contrario simplemente se avanza con el nodo “y” para comparar con el resto de palabras de la lista “z”.

Si el contador cont es igual a cero significa que la palabra contenida en el nodo aux no está en la lista “z” y se debe agregar, aux avanza al siguiente nodo en la lista principal, se igual el contador cont a cero nuevamente y se hace el nodo “y” nuevamente igual al primer nodo de la lista “z” para realizar nuevamente el proceso.

Finalmente, se hace el nodo aux igual al primer nodo de la lista “z” y mientras este es diferente de null en un string se almacenan tanto la palabra como su contador correspondiente y se retorna dicho string.

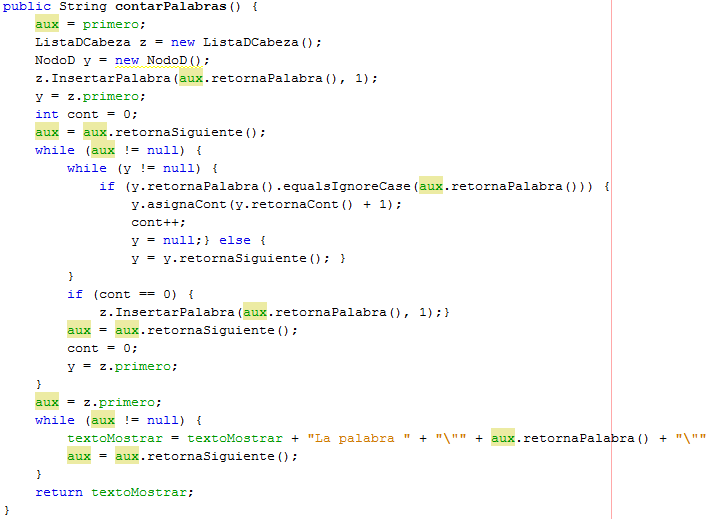


Ilustración 28. Método contarPalabras

* Método ordenAlfabetico()**:** no recibe parámetro de entrada, usa dos nodos para recorrer la lista, se hace el nodo aux igual al primero y mientras sea diferente de null se hace el nodo comparar igual al siguiente de aux y mientras este sea diferente de null se comparan ambas palabras haciendo uso del método compareTo, el cual retorna un valor igual a cero si son iguales y en este caso simplemente se hace el nodo comparar igual al siguiente comparando nuevamente, si el resultado de la operación compareTo es mayor a cero se intercambian las palabras y se avanza con el nodo comparar, por otro lado si el resultado del compareTo es menor que cero simplemente se avanza con el nodo comparar, finalmente se retorna la lista de palabras y las repeticiones de cada una de estas haciendo uso de un String.

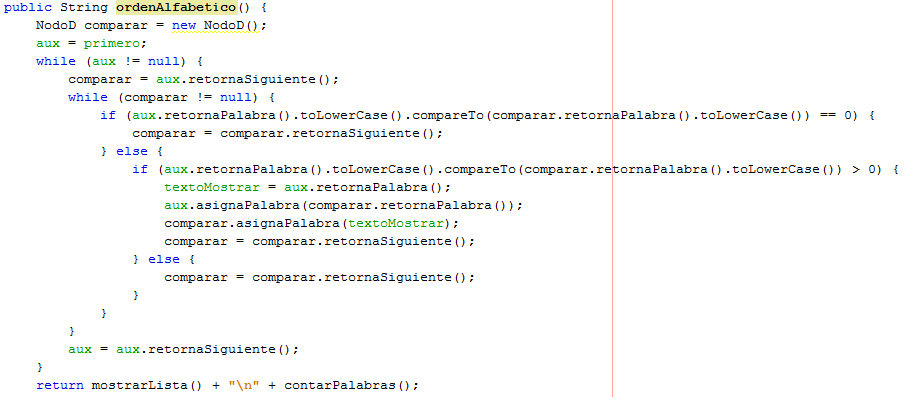


Ilustración 299. Método ordenAlfabetico

* Método mostrarTildadas()**:** no recibe nada, se hace un nodo aux igual al primer nodo de la lista, se crea una nueva lista para las palabrasTildadas y mientras aux es diferente de null se revisa si la palabra que contiene el nodo aux tiene tildes usando el método matches, si las tiene la palabra se inserta en la lista de palabrasTildadas, cuando el recorrido de aux termina se pregunta si la listaTildadas es vacía y se retorna el resultado de la operación, si no es vacía se lleva a un String la lista palabrasTildadas y se retorna.

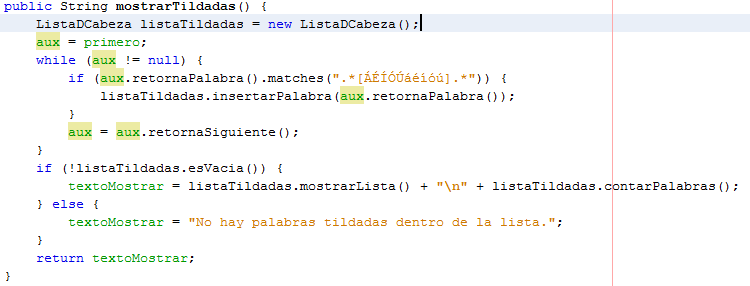


Ilustración 30. Método mostrarTildadas

### Clase Ventana

Contiene todos los métodos que relacionan la interfaz de usuario o formulario con las operaciones que se deben realizar sobre el texto.

#### Variables globales

* Variables para el manejo de texto:se usan dos variables una de tipo String llamada palabra y otra de tipo StringTokenizer llamada tkn que se usará para separar los token del texto.



Ilustración 31. Variables para manejar texto

* Lista ligada:se crea una lista ligada llamada listaPalabras que va a contener cada uno de los token hallados en el texto.

#### Constructor:

Inicializa las componentes de la ventana, la centra en la pantalla, le da el título y deshabilita el redimensionamiento de la misma.

#### Métodos

* Método botonSalirActionPerformed**:** este método elimina la ventana y no retorna nada, recibe una variable de tipo java.awt.event.ActionEvent.



Ilustración 32. Método botonSalirActionPerformed

* Método botonLimpiarActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, limpia la lista ligada, las áreas de texto y habilita los botones Examinar y Guardar del formulario.

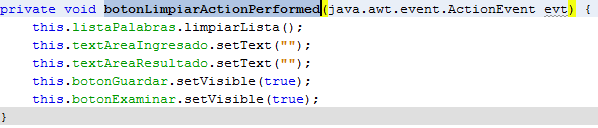


Ilustración 33. Método botonLimpiarActionPerformed

* Método botonGuardarActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, si el área de texto ingresado es vacía asigna un mensaje en el área de texto de resultado diciendo que no hay ningún texto, de lo contrario inicializa el StringTokenizer y le asigna el texto del área de texto ingresado, para separar los token se usan los símbolos " \":.,;{[^}]+\*~¿¡'?=)(/&%#!|°¬$´-", se almacena el primer token o palabra en un string y este a su vez se inserta a la lista ligada listaPalabras, ahora mientras hayan tokens se va a llevar el token al String palabra y esta se va a insertar en la listaPalabras. Luego en el área de texto resultado se muestra la lista haciendo uso del método mostrarLista y se ponen invisibles los botones Examinar y Guardar del formulario.

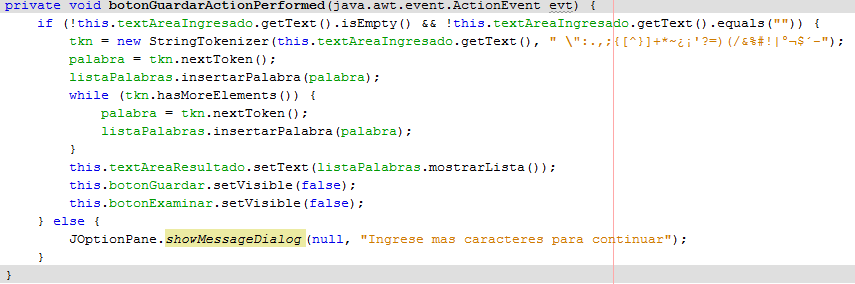


Ilustración 34. Método botonGuardarActionPerformed

* Método menuItemMostrarPreposicionesActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, verifica si la listaPalabras es vacía y muestra un mensaje, si no lo es al área de texto resultado lleva el resultado de aplicar el método mostrarPreposiciones a la listaPalabras.

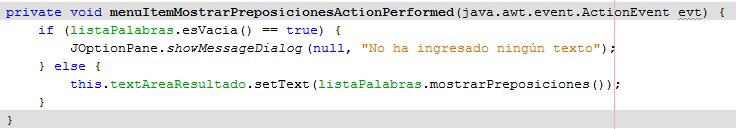


Ilustración 35. Método menuItemMostrarPreposicionesActionPerformed

* Método menuItemEliminarPalabraActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, verifica si la listaPalabras es vacía y muestra un mensaje, de lo contrario a la variable palabra lleva la palabra que el usuario ingresa en el ShowInputDialog, si la longitud de la palabra ingresada por el usuario es igual a cero lo informa con un mensaje, sino muestra en el área de resultado la operación eliminarPalabra llevada a cabo sobre la listaPalabras usando como parámetro de entrada el String palabra.

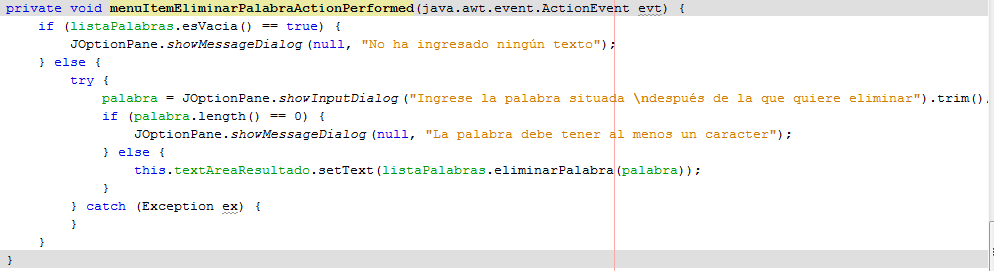


Ilustración 36. Método menuItemEliminarPalabraActionPerformed

* Método menuItemMostrarInvertidoActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, verifica si la listaPalabras es vacía y muestra un mensaje, de lo contrario en el área de texto resultado muestra el resultado de la operación textoInverso aplicada sobre la listaPalabras.

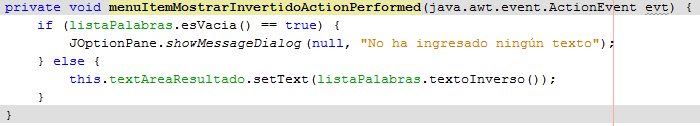


Ilustración 37. Método menuItemMostrarInvertidoActionPerformed

* Método menuItemReemplazarPalabraActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, verifica si la listaPalabras es vacía y muestra un mensaje, de lo contrario hace el String palabra igual al primer ShowInputDialog y el String por igual al segundo ShowInputDialog, luego verifica que la longitud de ambas palabras sea diferente de cero y a ambas les aplica el método matches para verifica que sólo contengan letras y números, de ser así lleva al área de texto resultado el resultado obtenido de aplicar el método sustituirPalabra a la listaPalabras usando como parámetro de entrada los string palabra y por.

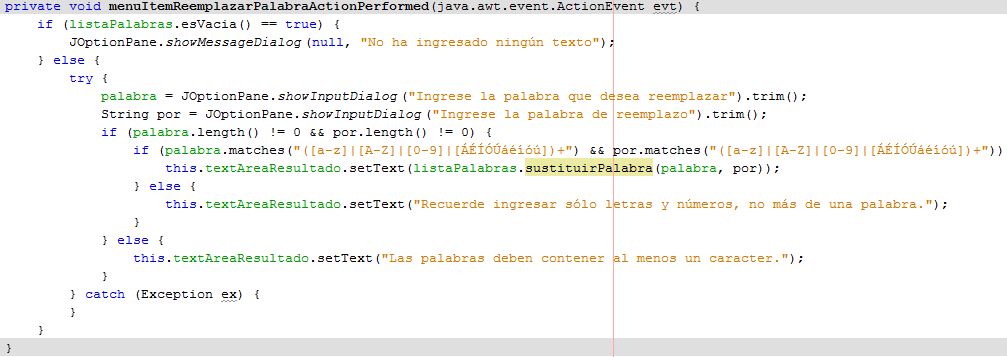


Ilustración 38. Método menuItemReemplazarPalabraActionPerformed

* Método menuItemMostrarTildadasActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, verifica si la listaPalabras es vacía y muestra un mensaje, de lo contrario aplica el método mostrarTildadas a listaPalabras y lo muestra en el área de texto resultado.

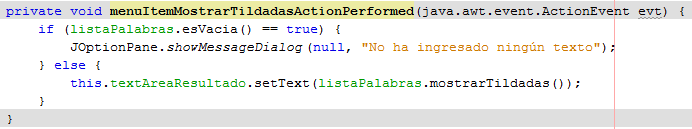


Ilustración 39. Método menuItemMostrarTildadasActionPerformed

* Método menuItemOrdenAlfabeticoActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, verifica si la listaPalabras es vacía y muestra un mensaje, de lo contrario aplica el método ordenAlfabetico a listaPalabras y lo muestra en el área de texto resultado.

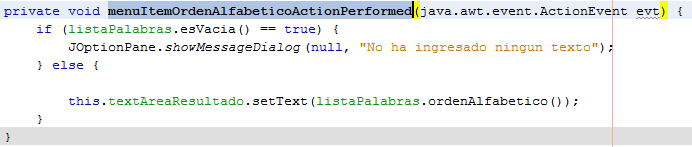


Ilustración 40. Método menuItemOrdenAlfabeticoActionPerformed

* Método botonExaminarActionPerformed**:** recibe una variable de tipo java.awt.event.ActionEvent, no retorna nada, se crea un JFileChooser y un entero opcion al cuál se lleva el valor de la opción escogida por el usuario, si este valor es correspondiente al de aprobación se lleva al String palabra la ruta del archivo escogido por el usuario, luego a un FileReader fr se lleva la ruta del archivo como parámetro de entrada usando el String palabra y el FileReader fr se lleva como parámetro de entrada a un BufferedReader bf, se crea un String texto y se inicializa en vacío el String palabra y mientras el resultado de llevar las líneas del texto leído al String texto es diferente de null en el String palabra se almacenan todas las líneas del texto y finalmente el String palabra se imprime en el área de texto resultado.

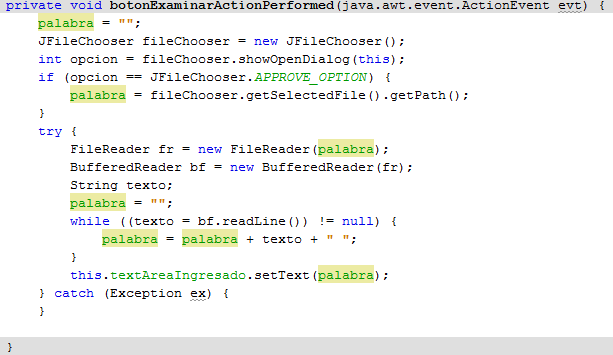


Ilustración 41. Método botonExaminarActionPerformed

# CONCLUSIONES

A la hora de hacer uso de información ingresada por el usuario es muy importante hacer uso del try catch para evitar las excepciones, por otra parte el trato de los datos de entrada es de suma importancia, se debe limpiar bien el texto dependiendo de las necesidades y evitar usar variables de más, si los métodos son muy largos se pueden dividir en varios e incluso hacer su llamado dentro de varios métodos que así lo puedan requerir.

# LISTA DE ILUSTRACIONES

[Ilustración 1. Clase AnalizadorTextos .java 8](#_Toc403854412)

[Ilustración 2. Constructor vacío 9](#_Toc403854413)

[Ilustración 3. Constructor con la palabra 9](#_Toc403854414)

[Ilustración 4. Constructor con el contador y la palabra 9](#_Toc403854415)

[Ilustración 5. Método asignaCont 9](#_Toc403854416)

[Ilustración 6. Método retornaCont 10](#_Toc403854417)

[Ilustración 7. Método asignaPalabra 10](#_Toc403854418)

[Ilustración 8. Método retornaPalabra 10](#_Toc403854419)

[Ilustración 9. Método asignaSiguiente 10](#_Toc403854420)

[Ilustración 10. Método asignaAnterior 10](#_Toc403854421)

[Ilustración 11. Método retornaSiguiente 11](#_Toc403854422)

[Ilustración 12. Método retornaAnterior 11](#_Toc403854423)

[Ilustración 13. Variables de tipo NodoD 11](#_Toc403854424)

[Ilustración 14. Vectores 11](#_Toc403854425)

[Ilustración 15. Métodos de asignación 12](#_Toc403854426)

[Ilustración 16. Métodos de retorno 12](#_Toc403854427)

[Ilustración 17. Constructor ListaDCabeza 12](#_Toc403854428)

[Ilustración 18. Método esVacia 13](#_Toc403854429)

[Ilustración 19. Método limpiarLista 13](#_Toc403854430)

[Ilustración 20. Menú insertarPalabra 13](#_Toc403854431)

[Ilustración 21. Método mostrarLista 14](#_Toc403854432)

[Ilustración 22. Método mostrarPreposiciones 15](#_Toc403854433)

[Ilustración 23. Método buscar 15](#_Toc403854434)

[Ilustración 24. Método eliminarPalabra 16](#_Toc403854435)

[Ilustración 25. Método textoInverso 16](#_Toc403854436)

[Ilustración 26. Método sustituirPalabra 17](#_Toc403854437)

[Ilustración 277. Método insertarPalabra con contador 17](#_Toc403854438)

[Ilustración 28. Método contarPalabras 19](#_Toc403854439)

[Ilustración 299. Método ordenAlfabetico 20](#_Toc403854440)

[Ilustración 30. Método mostrarTildadas 20](#_Toc403854441)

[Ilustración 31. Variables para manejar texto 21](#_Toc403854442)

[Ilustración 32. Método botonSalirActionPerformed 21](#_Toc403854443)

[Ilustración 33. Método botonLimpiarActionPerformed 21](#_Toc403854444)

[Ilustración 34. Método botonGuardarActionPerformed 22](#_Toc403854445)

[Ilustración 35. Método menuItemMostrarPreposicionesActionPerformed 22](#_Toc403854446)

[Ilustración 36. Método menuItemEliminarPalabraActionPerformed 23](#_Toc403854447)

[Ilustración 37. Método menuItemMostrarInvertidoActionPerformed 23](#_Toc403854448)

[Ilustración 38. Método menuItemReemplazarPalabraActionPerformed 24](#_Toc403854449)

[Ilustración 39. Método menuItemMostrarTildadasActionPerformed 24](#_Toc403854450)

[Ilustración 40. Método menuItemOrdenAlfabeticoActionPerformed 24](#_Toc403854451)

[Ilustración 41. Método botonExaminarActionPerformed 25](#_Toc403854452)