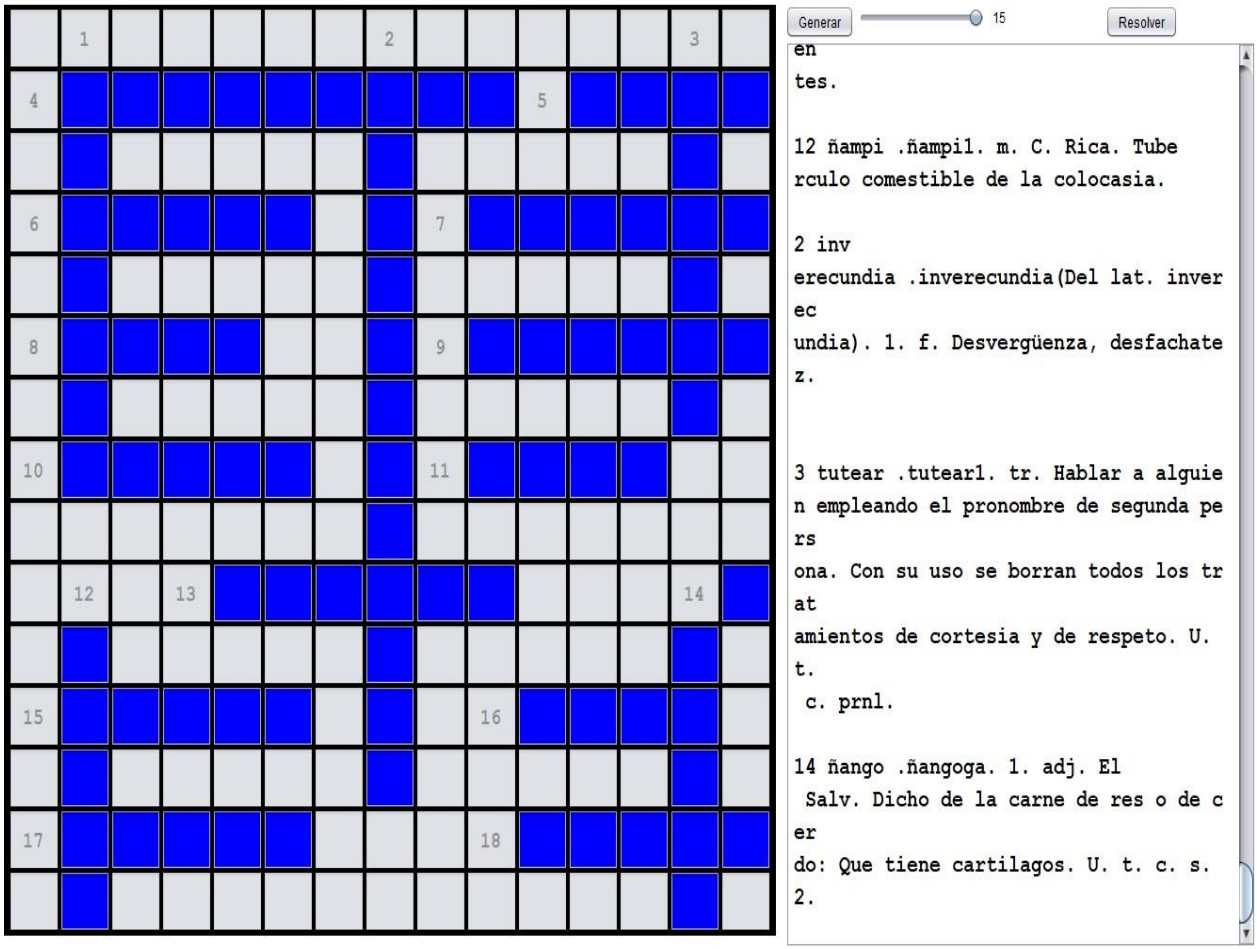
CRUCIGRAMA

En JavaFX Swing.



Un **crucigrama** es un juego o **pasatiempo** que consiste en **completar los huecos** de un dibujo con letras. Para descubrir qué letra debe escribirse en cada espacio, el crucigrama indica el **significado de las palabras** que deben leerse en sentido vertical y horizontal. La idea, por lo tanto, es que la plantilla del crucigrama ya completada presente una serie de palabras que puedan leerse en vertical y horizontal y que se cruzan entre sí.

Cómo completar un crucigrama

Para comenzar a completar un crucigrama, la persona debe leer las dos listas de definiciones que presenta el pasatiempo: una correspondiente al sentido vertical y otra para el sentido horizontal. La plantilla o dibujo se encuentra dividida en **casillas blancas** (donde hay que escribir las letras individuales) y **casillas negras** (que sirven para separar las palabras).

Así, al leer una definición y conocer la palabra, el participante tiene que ingresar al crucigrama, escribiendo una letra en cada casilla blanca del espacio correspondiente. De esta manera, poco a poco se irá completando el juego.

Historia de los crucigramas

El origen de los crucigramas **data del siglo I**. El primer crucigrama fue hallado en las ruinas de Pompeya y se conoce como cuadrado **sator**.

El Cuadrado de Sator, el primer crucigrama

Posteriormente, ya entrado el siglo XIX un hombre conocido como Hyperion publicó los famosos rompecabezas de doble diamante; se trataban de juegos de índole mental que se asemejaban mucho a los actuales crucigramas (con cuadraditos donde el usuario debía entrelazar las palabras hasta dar con la combinación correcta). Se cree que estas publicaciones se realizaron durante varios años.

Años más tarde surgiría un juego que se inspiraba profundamente en el cuadrado sator y que consistía en una rejilla de cuatro por cuatro con algunas pistas para que los usuarios pudieran dar con las respuestas; no obstante, este entretenimiento no tuvo mucho éxito.

Finalmente, Arthur Wynne, un periodista inglés publicó en el New York World el primer crucigrama, tal cual hoy lo conocemos. Se llamó "word-cross" aunque pocos años más tarde se acuñó el término por el que todavía conocemos a este entretenimiento.

Los elementos fundamentales de un crucigrama son: **la grilla** (celdas blancas con números pequeños que se encuentran asociados con las referencias) y **las referencias** (suelen ubicarse

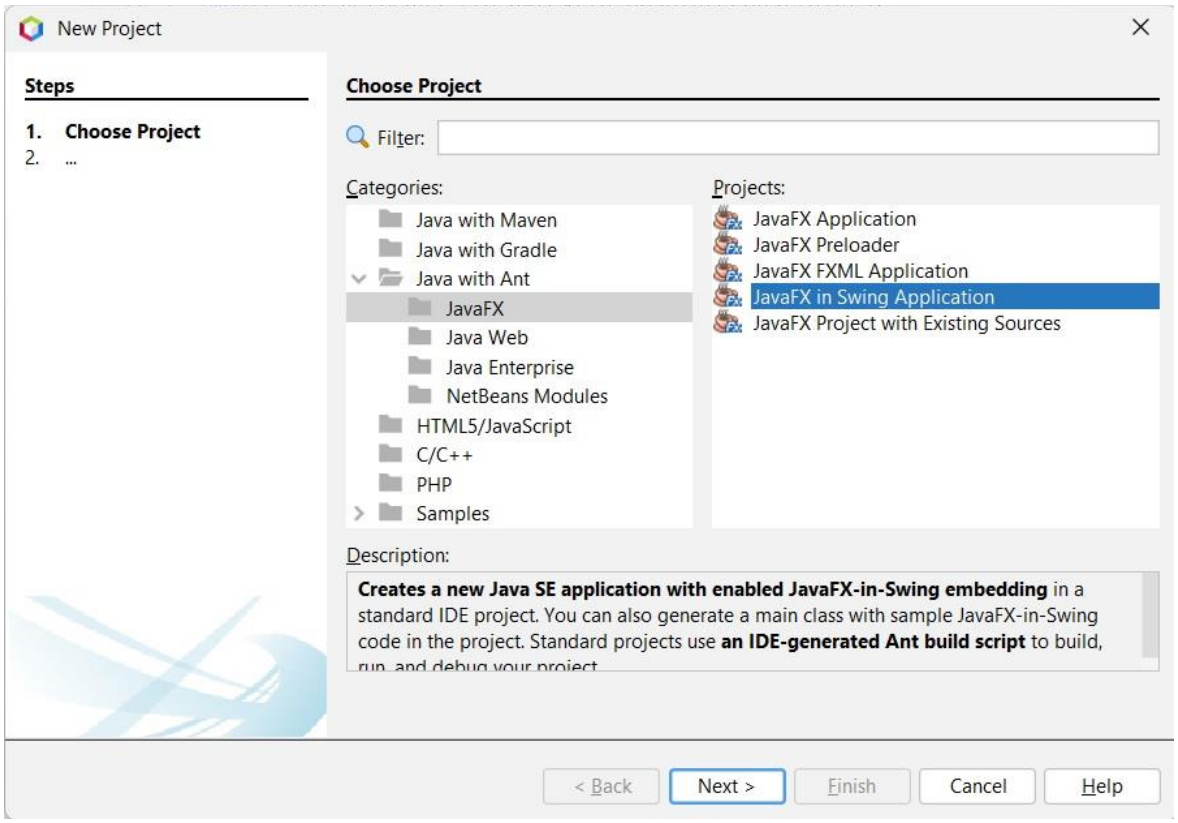
al pie de la página y constan de una serie de definiciones muy concisas que permiten al usuario vincular una palabra con el espacio del que dispone para completar la grilla).

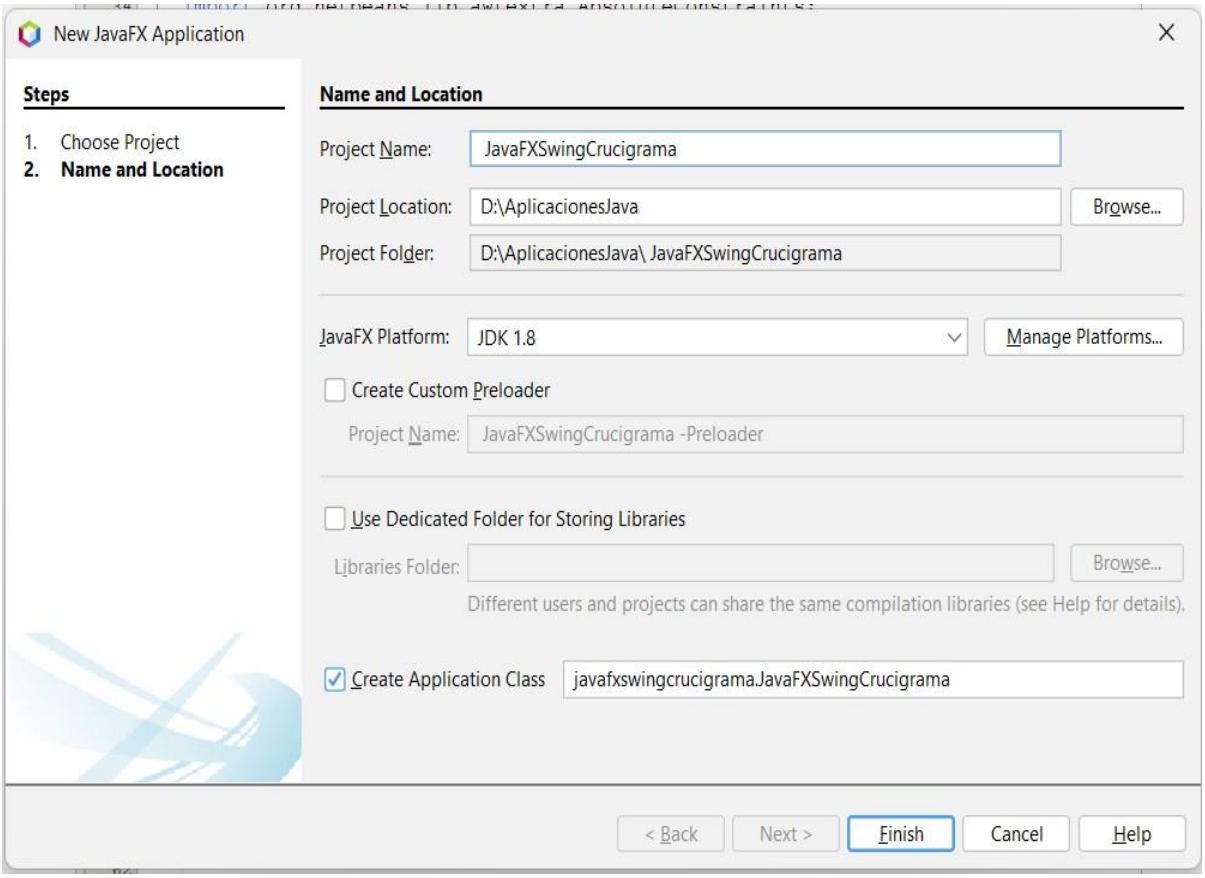
Es importante mencionar que las referencias se encuentran divididas en dos partes, las correspondientes a la grilla leída en sentido **horizontal**, y las que corresponden con la distribución vertical.

Las palabras se encuentran escogidas de tal modo que **aquellas casillas compartidas por dos palabras poseen una letra que es igual** para la posición de ambos términos, el horizontal y el vertical, de este modo, a medida que se va llenando la grilla resulta más sencillo encontrar las palabras que faltan.

Estos pasatiempos suelen recomendarse para aquellas personas que tienen facilidad para olvidar las cosas puesto que mantiene su mente despierta y los ayuda a asociar conceptos. Suelen publicarse de forma solitaria en revistas llamadas crucigramas o palabras cruzadas y también en periódicos y revistas; en algunos casos, estos **crucigramas son temáticos**.

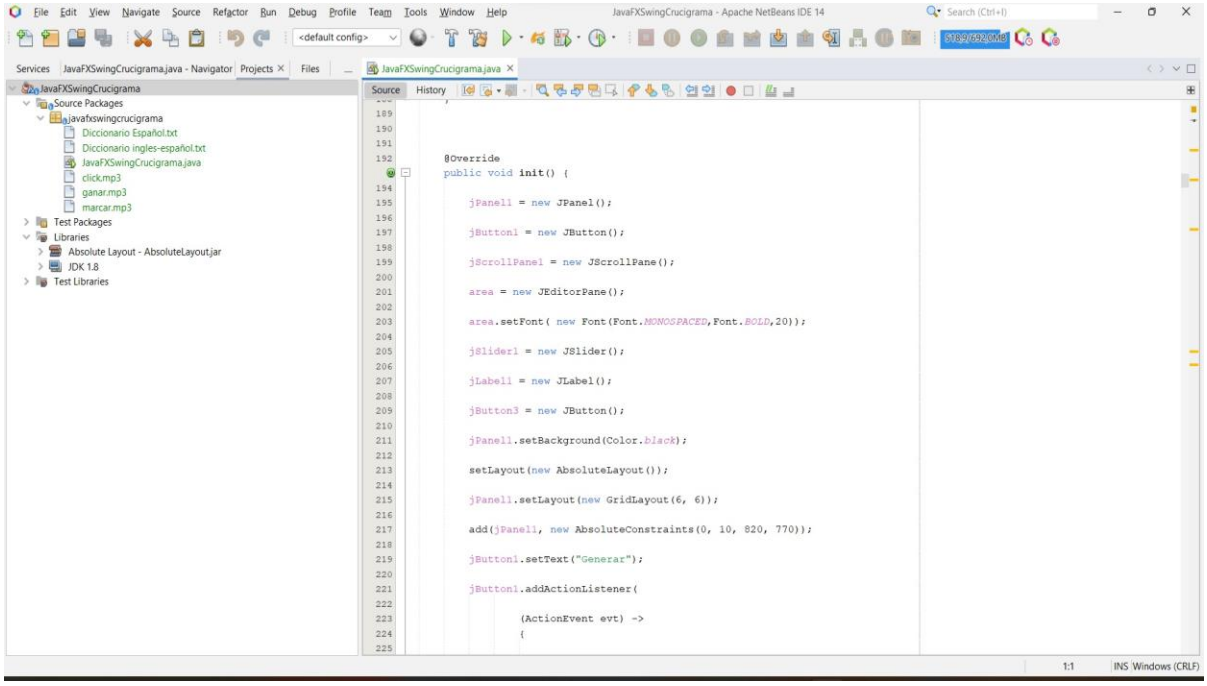
Para comenzar abrimos NetBeans y creamos nuestro proyecto JavaFX Swing.





Y agregamos a nuestro paquete los archivos multimedia que usaremos como efectos de sonido, en concreto los archivos de sonido “ganar.mp3”, “click.mp3”, “marcar.mp3”, y el archivo “diccionario.txt”, con el que construiremos los crucigramas con las palabras escogidas al azar desde el archivo en este ejemplo tenemos un “Diccionario Español.txt” y un “Diccionario inglés-español.txt”, pero puede usarse cualquier archivo de texto plano que contenga la estructura palabra: significado en cualquier idioma y sobre cualquier temática.

Código fuente:



/*

* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

* Click nbfs://nbhost/SystemFileSystem/Templates/javafx/FXSwingMain.java to edit this template

*/

package javafxswingcrucigrama;

Importamos las clases que vamos a necesitar

I

mport java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Random;
import java.util.Scanner;
import java.util.TimerTask;
import javafx.application.Platform;
import javafx.embed.swing.JFXPanel;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.Timer;
import javax.swing.UIManager;
import javax.swing.event.ChangeEvent;
```

```
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

/**
*
* @author Abel Gallart Bonome
*/

public class JavaFXSwingCrucigrama extends JApplet {

Declaramos las variables de la clase

    private static final int JFXPANEL_WIDTH_INT = 1400;

    private static final int JFXPANEL_HEIGHT_INT = 800;

    private static JFXPanel fxContainer;

    static JFrame frame;

    static JApplet applet;

    Media mediaganar,mediamarcar,mediaclick;

    boolean horizontal=true;

    JEditorPane area;

    JButton jButton1, jButton3;

    JLabel jLabel1;
```

```
JPanel jPanel1;

JScrollPane jScrollPane1;

JSlider jSlider1;

int l=0;

String [][]tabla;

Timer timer;

TimerTask rele;

Hashtable diccionario=new Hashtable();

ArrayList DICCIONARIO=new ArrayList();

Random random=new Random();

   ArrayList Da=new ArrayList();

   ArrayList Db=new ArrayList();

   ArrayList Dc=new ArrayList();

   ArrayList Dd=new ArrayList();

   ArrayList De=new ArrayList();
```

```
ArrayList Df=new ArrayList();

ArrayList Dg=new ArrayList();

ArrayList Dh=new ArrayList();

ArrayList Di=new ArrayList();

ArrayList Dj=new ArrayList();

ArrayList Dk=new ArrayList();

ArrayList Dl=new ArrayList();

ArrayList Dm=new ArrayList();

ArrayList Dn=new ArrayList();

ArrayList Dñ=new ArrayList();

ArrayList Do=new ArrayList();

ArrayList Dp=new ArrayList();

ArrayList Dq=new ArrayList();

ArrayList Dr=new ArrayList();

ArrayList Ds=new ArrayList();

ArrayList Dt=new ArrayList();
```

```
ArrayList Du=new ArrayList();

ArrayList Dv=new ArrayList();

ArrayList Dw=new ArrayList();

ArrayList Dx=new ArrayList();

ArrayList Dy=new ArrayList();

ArrayList Dz=new ArrayList();
```

En nuestro método main creamos la ventana y mostramos una instancia de nuestra clase.

```
public static void main(String[] args) {

    SwingUtilities.invokeLater(() ->
    {

       try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");

       }

       catch (Exception e) {e.printStackTrace();}
```

```
        frame = new JFrame("Crucigrama");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setUndecorated(true);

        applet = new JavaFXSwingCrucigrama();

        applet.init();

        frame.setContentPane(applet.getContentPane());

        frame.pack();

        frame.setLocationRelativeTo(null);

        frame.setVisible(true);

        applet.start();
    });
}
```

En el metodo init() se inicializan las variables y objetos necesarios tanto para gráfica como para la lógica.

```
@Override
public void init() {

    jPanel1 = new JPanel();
```

```
jButton1 = new JButton();

jScrollPane1 = new JScrollPane();

area = new JEditorPane();

area.setFont( new Font(Font.MONOSPACED,Font.BOLD,20));

jSlider1 = new JSlider();

jLabel1 = new JLabel();

jButton3 = new JButton();

jPanel1.setBackground(Color.black);

setLayout(new AbsoluteLayout());

jPanel1.setLayout(new GridLayout(6, 6));

add(jPanel1, new AbsoluteConstraints(0, 10, 820, 770));

jButton1.setText("Generar");
```

Creamos un botón “Generar”, y se declara el evento del clic sobre el botón, de modo que cuando esto ocurra nos lleve al método jButtonActionPerformed(), cada vez que el usuario accione este botón “Generar ” el hilo de ejecución ira al método y creara un nuevo crucigrama.

también se crea un área de texto que se usará más adelante para escribir las descripciones de los horizontales y verticales y las comunicaciones con el usuario, un Slider con el que podremos graduar el tamaño de las palabras que usaremos para generar nuestro crucigrama, y un Panel que contendrá el crucigrama.

```
jButton1.addActionListener(
```

```
        (ActionEvent evt) ->
        {

            jButton1ActionPerformed();

        }
    );



    add(jButton1, new AbsoluteConstraints(830, 10, -1, -1));

    area.setEditable(false);

    jScrollPane1.setViewportView(area);

    add(jScrollPane1, new AbsoluteConstraints(830, 40, 500, 750));

    jSlider1.setMaximum(15);

    jSlider1.setMinimum(4);
```

Aquí delimitamos un mínimo y un máximo del tamaño que el usuario podrá configurar para las palabras del crucigrama

```
    jSlider1.setPaintLabels(true);

    jSlider1.setPaintTicks(true);

    jSlider1.setSnapToTicks(true);
```

```
jSlider1.setToolTipText("");

jSlider1.addChangeListener(

        (ChangeEvent evt) ->
        {

            jSlider1StateChanged();

        }
);
```

Se declara el método a seguir cuando se realice una acción sobre la slider, el hilo de ejecución será lleva al método jSliderr1StateChanged()

```
add(jSlider1, new AbsoluteConstraints(910, 10, 130, -1));

add(jLabel1, new AbsoluteConstraints(1050, 10, 30, 20));

jButton3.setText("Resolver");

jButton3.addActionListener(

        (ActionEvent evt) ->
        {

            jButton3ActionPerformed();

        }
);
```

```
add(jButton3, new AbsoluteConstraints(1170, 10, -1, -1));
```

Se declara el evento sobre el botón "Resolver" y el hilo de ejecución será llevado al método jButton3ActionPerformed () , cada vez que el usuario accione el botón el programa ejecutará este método para resolver el crucigrama

```
Scanner scanner=null;

while(scanner==null){

   try {
      scanner =
            new Scanner(

                  new BufferedReader(

                        new FileReader(

                              new File("./Diccionario  Español.txt")
                        )
                  )
            );

      }

   catch (Exception ex) {ex.printStackTrace();}

}
```

Creamos un objeto scanner que leerá del fichero "Diccionario Español.tx" que está contenido el paquete, un texto simple sin formato con la estructura palabra-significado ,en este apartado del código podemos modificar el fichero del que scanner lee por otro diccionario en

cualquier otro idioma que utiliza el programa para construir el crucigrama. Este programa puede crear Crucigramas en cualquier idioma.

```
while (scanner.hasNextLine()) {
```

La clase scanner funciona como una enumeración que itera sobre los elementos de una lista, estos elementos son las líneas del documento "Diccionario Español.txt". Así que decimos que mientras scanner contenga líneas...

```
String aux = scanner.nextLine();
```

Guardamos en la variable "aux" el contenido de la línea del documento sobre el que vamos a trabajar.

```
aux=aux.replaceAll("ó",  "o");

aux=aux.replaceAll("á","a");

aux=aux.replaceAll("é","e");

aux=aux.replaceAll("í","i");

aux=aux.replaceAll("ú","u");

aux=aux.replaceAll("�","ñ");

aux=aux.replace("-",  " ");
```

Remplazamos algunos caracteres especiales que java interpreta de otro modo para dar formato al texto.

```
if (!aux.equals(""))

{
```

si no estamos trabajando sobre una linea vacia o de longitud 0 entonces...

```
String[] arr = aux.split(". ",2);
```

En el vector arr guardamos el resultado de aplicarle el método Split a nuestra variable aux que contiene la línea del documento sobre el cual estamos trabajando, el método Split nos

devuelve la línea dividida en 2 partes a partir del primer ". "que encuentre, así separamos la palabra del significado.

```
String  aux1="";

for(int i=1;i<arr.length;i++)

   aux1+=arr[i];

arr[0]=arr[0].replace("1",  "");

arr[0]=arr[0].replace("2",  "");

arr[0]=arr[0].replace("  ", "");
```

Damos un poco más de formato al archivo de texto y si el vector arr no está vacío usando una Tabla de Hash diccionario colocamos las parejas palabra-significado contenidas en la variable arr[0] y aux1 respectivamente.

```
if (arr.length!=0) diccionario.put(arr[0], aux1);

}
```

Haremos esto con cada línea del documento hasta la última de modo que en la tabla de hash diccionario tendremos todas las palabras del documento y sus significado, ordenadas.

```
}

Enumeration en=diccionario.keys();
```

Ahora iteramos sobre nuestra tabla de hash concretamente sobre las llaves las keys que son las palabras del del documento que hemos leído. s

```
while(en.hasMoreElements()){

   String aux=(String)en.nextElement();
```

Guardamos en la variable "aux" la palabra en curso.

```
   if (aux.startsWith("a")) Da.add(aux);
```

```
if (aux.startsWith("b")) Db.add(aux);

if (aux.startsWith("c")) Dc.add(aux);

if (aux.startsWith("d")) Dd.add(aux);

if (aux.startsWith("e")) De.add(aux);

if (aux.startsWith("f")) Df.add(aux);

if (aux.startsWith("g")) Dg.add(aux);

if (aux.startsWith("h")) Dh.add(aux);

if (aux.startsWith("i")) Di.add(aux);

if (aux.startsWith("j")) Dj.add(aux);

if (aux.startsWith("k")) Dk.add(aux);

if (aux.startsWith("l")) Dl.add(aux);

if (aux.startsWith("m")) Dm.add(aux);

if (aux.startsWith("n")) Dn.add(aux);

if (aux.startsWith("ñ")) Dñ.add(aux);

if (aux.startsWith("o")) Do.add(aux);
```

```
if (aux.startsWith("p")) Dp.add(aux);

if (aux.startsWith("q")) Dq.add(aux);

if (aux.startsWith("r")) Dr.add(aux);

if (aux.startsWith("s")) Ds.add(aux);

if (aux.startsWith("t")) Dt.add(aux);

if (aux.startsWith("u")) Du.add(aux);

if (aux.startsWith("v")) Dv.add(aux);

if (aux.startsWith("w")) Dw.add(aux);

if (aux.startsWith("x")) Dx.add(aux);

if (aux.startsWith("y")) Dy.add(aux);

if (aux.startsWith("z")) Dz.add(aux);
```

Y en este paso según con la letra que empiece “aux” que recordemos contiene la palabra sobre la que trabajamos, la agregamos al vector “ArrayList” correspondiente, por ejemplo, si empieza por “a” la agregamos al vector Da, si empieza por “b” la agregamos al vector Db , y así con cada una de las palabras que contiene la tabla de hash diccionario

```
}

DICCIONARIO.add(Da);
```

```
DICCIONARIO.add(Db);

DICCIONARIO.add(Dc);

DICCIONARIO.add(Dd);

DICCIONARIO.add(De);

DICCIONARIO.add(Df);

DICCIONARIO.add(Dg);

DICCIONARIO.add(Dh);

DICCIONARIO.add(Di);

DICCIONARIO.add(Dj);

DICCIONARIO.add(Dk);

DICCIONARIO.add(Dl);

DICCIONARIO.add(Dm);

DICCIONARIO.add(Dn);

DICCIONARIO.add(Dñ);

DICCIONARIO.add(Do);

DICCIONARIO.add(Dp);
```

```
DICCIONARIO.add(Dq);

DICCIONARIO.add(Dr);

DICCIONARIO.add(Ds);

DICCIONARIO.add(Dt);

DICCIONARIO.add(Du);

DICCIONARIO.add(Dv);

DICCIONARIO.add(Dw);

DICCIONARIO.add(Dx);

DICCIONARIO.add(Dy);

DICCIONARIO.add(Dz);
```

Entonces hasta aquí tenemos un vector DICCIONARIO que contiene 27 vectores Da,Db,Dc...Dz, que contienen las palabras del documento. Y una Tabla de Hash diccionario que contiene las parejas palabra, significado. Es el momento de tomar un descanso, tenemos ya el diccionario separado por palabras y significados y agrupados por orden alfabético. Yo cada vez que llego a esta parte me tomo un café.

```
fxContainer = new JFXPanel();

fxContainer.setPreferredSize(new Dimension(JFXPANEL_WIDTH_INT,
JFXPANEL_HEIGHT_INT));
```

```
        add(fxContainer, new
AbsoluteConstraints(0,0,JFXPANEL_WIDTH_INT,JFXPANEL_HEIGHT_INT));

        Platform.runLater(() -> {

          createScene();

        });
```

Terminamos de configurar las características gráficas y los archivos de audio y usaremos como efectos de sonido. Los archivos de audio se encuentran en la carpeta raíz del proyecto.

```
    File filemarcar=new File("./marcar.mp3");

    mediamarcar=new Media(filemarcar.toURI().toString());

    File fileclick=new File("./click.mp3");

    mediaclick=new Media(fileclick.toURI().toString());

    File fileganar=new File("./ganar.mp3");

    mediaganar=new Media(fileganar.toURI().toString());

    generar();
```

Llamamos al método "generar() "que creará nuestro crucigrama. Será una cuadricula de áreas de texto ,donde podremos introducir letra a letra del crucigrama, algunas textFields estarán inutilizadas y otras contendrá el índice de la palabra horizontal o vertical contenidas en el crucigrama.

Creamos un hilo de ejecución que en paralelo al hilo de ejecución del programa comprobará cada 5000 milisegundo o sea 5 segundo si hemos completado el crucigrama satisfactoriamente.

```
timer=new Timer(5000,  (e) -> {


        boolean correcto=true;
```

Tomamos una variable booleana correcto y la inicializamos en Verdadero.

```
       Component []comp=(Component[])jPanel1.getComponents();
```

Guardamos en el vector comp todas las áreas de texto “textFields” que contiene la cuadricula del crucigrama al aplicarle el método getComponents() a jPanel1 que es donde se encuentran contenidos con una distribución de Tabla filas por Columnas “GridLayout”.

```
        for (int i=1;i<comp.length;i++)
```

Recorremos todo el vector y si el componente está habilitado ...

```
          if (comp[i].isEnabled()


                &&


                !((JTextField)comp[i]).getText().equals(tabla[i/l][i%l]))
```

Ahora bien, esta parte requiere alguna aclaración matemática, la variable “l” contiene el valor de la slider que es la dimensión de la cuadricula del crucigrama o sea es el largo y el ancho, y “i” es la variable que marca la posición del botón en el vector de componentes gráficos, una lista, más adelante veremos que “tabla” es una matriz de caracteres con la solución al crucigrama, si dividimos la posición “i” entre “l” obtendremos la fila a la que pertenecería comp[i] en la matriz de solución tabla, y si tomamos el resto de la división “%” obtendremos la columna.

Aclarado esto continuamos...Si el área de texto ” comp[i]” no contiene en la posición “i” el mismo carácter que la tabla de solución ponemos a variable correcto en falso.

```
                correcto=false;
```

Esto se hace con cada una de las áreas de texto de modo que si al salir de este bucle la variable correcto continua con valor Verdadero es porque todas las áreas de texto coinciden con la tabla de solución y hemos resuelto el crucigrama.

```
        if (correcto){
                MediaPlayer efectoganar=new MediaPlayer(mediaganar);
```

```
                    efectoganar.setVolume(0.7);

                    efectoganar.play();
```

Lanzamos un efecto de sonido que indica al usuario que ha resuelto el crucigrama satisfactoriamente y se lanza también una ventana de dialogo que contiene un mensaje de texto con el mismo objetivo.

```
                    JOptionPane.showMessageDialog(frame,"Has completado el
crucigrama"

                            ,"Fin del juego",javax.swing.JOptionPane.OK_OPTION);
```

Una vez que el usuario confirme la ventana se genera un nuevo crucigrama a resolver.

```
                    generar();
```

Recordemos que esta rutina de comprobación es realizada cada 5 segundo por un hilo de ejecución en paralelo

```
            }

    });


      timer.start();
```

Damos inicio al hilo “time” de comprobación que hemos declarado anteriormente.

```
        }

   private void createScene() {

      StackPane root = new StackPane();

      fxContainer.setScene(new Scene(root));

   }
```

Se lanza la ventana y se despliegan los gráficos.

```
private void generar(){
```

Como ya se ha visto este método se encargará de crear nuestra cuadricula de áreas de texto que representará gráficamente el crucigrama, contenidas en jPanel1, y el crucigrama en una tabla de caracteres tabla[l][l] cuya dimensión es “l” en largo y ancho, tabla solución.

```
l=jSlider1.getValue();

jLabel1.setText(l+"");

jPanel1.removeAll();

jPanel1.setLayout(new GridLayout(l, l));

jPanel1.setSize(820,770);
```

Inicializamos el panel de trabajo y le damos tamaño y formato de tabla l X l y llamamos al método busca y le pasamos como parámetro l, este método encontrará palabras al azar dentro del diccionario y que formen el crucigrama y las colocara en la tabla solución .

```
busca(l);

int cont=1;
```

Inicializamos una variable “cont” ,un contador ,en valor 1

```
for(int f=0;f<l;f++)

   for(int c=0;c<l;c++)
```

{ recorremos una matriz cuadrada de dimensión “l” , tabla de solución tabla[f][c] en filas y en columnas , entonces f y c son sus coordenadas.

```
JTextField tf=new JTextField(" ");
```

```
tf.setFont(new Font(Font.MONOSPACED,Font.BOLD,18));

tf.setHorizontalAlignment(javax.swing.JTextField.CENTER);

jPanel1.add(tf);
```

Se crea el área de texto gráfica , se le da formato y se agrega al panel de trabajo

```
if (tabla[f][c].equals("*"))    tabla[f][c]=" ";
```

Si en la tabla de solución en la posición f,c hay un carácter “*”, lo sustituimos pon un espacio en blanco “ “.

```
if (tabla[f][c].equals("  ")||tabla[f][c].equals("$"))
```

y Si en la tabla de solución en la posición f,c hay un espacio en blanco “ ” o un carácter “$” decimos que el área de texto “tf” no estará habilitada. O lo que es lo mismo deshabilitamos todas las áreas de texto que estén marcadas en la matriz de solución como “ “ y “$“

```
tf.setEnabled(false);

else{
```

En otro caso visualizamos el área de texto y le damos formato con fondo azul a las áreas de texto que quedaran habilitadas para que el usuario pueda escribir las soluciones, en resumen, queda configurado gráficamente el crucigrama a partir de la tabla de solución .

```
tf.setVisible(true);

tf.setBackground(Color.BLUE);
```

Configuramos el evento sobre las áreas de texto presionar una tecla.

```
tf.addKeyListener(new KeyAdapter(){

@Override

public void keyPressed(KeyEvent evt) {
```

```
                jTextFieldKeyPressed(tf);}});

            tf.addKeyListener(new KeyAdapter() {

                @Override
```

Configuramos el evento sobre las áreas de texto liberar una tecla.

```
                public void keyReleased(KeyEvent evt) {

                    jTextFieldKeyReleased(tf);}});

            tf.addMouseListener(new MouseAdapter() {
```

Configuramos el evento sobre las áreas de texto, dar un clic sobre el componente gráfico.

```
                @Override

                public void mouseClicked(MouseEvent evt) {

                    jTextFieldMouseClicked(tf);}});

            }

            if (tabla[f][c].equals("$"))  {
                                tf.setText(cont+"");
```

```
                                    cont++;
                                   }
```

En este paso, si en la tabla de soluciones en la posición f,c hay una marca “$” significa que es el inicio de una palabra en el crucigrama tanto vertical como horizontal, entonces en esa área de texto colocamos el valor de la variable “cont” recordemos que esta inicializada en el valor “1” y aumentamos su valor en cada ciclo, de modo que al terminar quedaran indexadas cada palabra del crucigrama.

```
                }

jPanel1.validate();

this.validate();
```

Y hasta aquí las configuraciones gráficas y lógicas de nuestra cuadricula o crucigrama.

```
String text="Horizontal:\n";
```

Guardo en la variable “text” las palabras en horizontal de crucigrama, para lo que recorremos la tabla de soluciones saltando las filas pares.

```
for (int f=1;f<l;f+=2)

{
   String aux="";

   for(int c=0;c<l;c++)

   {
      if (tabla[f][c].equals("$")||f==1)
```

Si en la posición f,c hay una marca “$” , significa que comienza una palabra, o es la primera fila de la tabla

```
      {
         aux="";
```

```
if (f!=1) while(tabla[f][c].equals("$"))  c++;
```

Si la fila no es la primera, nos movemos por la fila "f" hasta la columna siguiente a la que este marcada con "$" , hasta el inicio de palabra .

```
String contH=((JTextField)jPanel1.getComponent(f*l+c-1)).getText();
```

Guardamos en contH el índice horizontal de la palabra que se encuentra en la posición f*l+c en la coordenada anterior que obtenemos al restar 1.

```
while(c<l&&(!tabla[f][c].equals("  ")&&!tabla[f][c].equals("$")))

{aux+=tabla[f][c];c++;}
```

Mientras en la tabla de soluciones no estén las marcas " " o "$", nos desplazamos por ella y vamos guardando en la variable acumulador "aux".

```
if (!aux.equals(""))
```

si la palabra no es vacía ""

```
{text+=contH+" "+aux+" ."+aux+diccionario.get(aux)+"\n\n";}
```

Guardamos en la variable acumulador "text", la cadena compuesta por la suma del índice horizontal, la palabra guardada en aux , ponemos una "," y concatenamos el significado de la palabra que obtenemos de la tabla de hash diccionario, le damos la llave o key , es la palabra guardada en "aux", y nos devuelve el significado de la palabra que hemos guardado previamente y un salto de línea por cada una.

```
    }

  }

}
```

Hasta aquí tenemos en el acumulador text las palabras y significados horizontales

Y ahora concatenaremos las palabras en Vertical.

```
text+="\n"+"Vertical:\n";
```

Recorremos la tabla de soluciones.

```
for (int f=1;f<l;f+=6)
{
  String aux="";
```

Inicializamos el acumulador aux en cada columna con el valor “” y recorremos la columna

```
for(int c=0;c<l;c++)

{
   if (tabla[c][f].equals("$"))
        {
          aux="";

          while(tabla[c][f].equals("$"))  c++;
```

Si la tabla está marcada con “$” nos desplazamos en vertical hasta la posición anterior al inicio de palabra y obtenemos el índice que guardaremos en la variable “contV” , básicamente repetimos el proceso seguido para las horizontales, pero desplazándonos por la matriz de solución, pero en dirección vertical.

```
          String contV=((JTextField)jPanel1.getComponent((c-1)*l+f)).getText();

          while(c<l&&(!tabla[c][f].equals("  ")&&!tabla[c][f].equals("$")))

          {aux+=tabla[c][f];c++;}
```

Nos desplazamos por la tabla de soluciones en vertical y concatenamos los caracteres en las áreas de testo y guardamos en aux la palabra.

```
          if (!aux.equals(""))

          {text+=contV+" "+aux+" ."+aux+diccionario.get(aux)+"\n\n";}
```

Comprobamos que no tenemos un resultado vacío, y concatenamos al acumulador “text” el índice en vertical, la palabra y el significado que obtenemos de la tabla de hash diccionario y un salto de linea.

```
        }
   }

}

String temp="";
```

```
for(int i=0;i<text.length()-40;i+=40)

  temp+=(String)text.subSequence(i, i+40)+"\n";
```

Dividimos el texto en líneas de 40 caracteres de longitud.

```
area.setText(temp);
```

Escribimos el resultado en el área de texto dispuesta en la ventana para comunicarse con el usuario.

En este punto del hilo de ejecución tenemos inicializados y definidos los componentes gráficos y lógicos del crucigrama, este proceso privado del programa ,"generar " es llamado por el hilos de ejecución al comienzo del lanzamiento de la ventana y cada y es llamado cada vez que acciona el usuario el botón "Generar".

```
}

private void jButton1ActionPerformed() {generar();}
```

Aquí , a este método se dirige el hilo de ejecución cuando es accionado el botón "generar" y de aquí se direcciona al método privado generar().

```
private void jSlider1StateChanged() {jLabel1.setText(jSlider1.getValue()+"");}
```

En este método, que es llamado cuando que acciona el slider , se actualiza el valor del texto en la etiqueta que muestra el valor de la dimensión de la matriz o tabla que tendrá el crucigrama, o lo que es lo mismo la longitud máxima que tendrán las palabras del crucigrama.

Recordando el código anterior es siguiente método definido es llamado a ejecución cuando el usuario acciona el botón "Resolver" .

```
private void jButton3ActionPerformed() {

    Component []comp=(Component[])jPanel1.getComponents();
```

Guardamos en un vector comp todas las áreas de texto de la matriz de componentes en el panel de trabajo, recordemos que podemos obtener la posición en la matriz de solución dividiendo la posición "i" en el vector lineal por la dimensión de la matriz ,tomamos la parte entera como filas y el resto de la división como columnas.

```
    for(int i=0;i<comp.length;i++)

      if (comp[i].isEnabled())
```

```
((JTextField)comp[i]).setText(tabla[i/l][i%l]);
```

Recorre el vector de componentes y compara el contenido en el área de texto con el contenido en la tabla de solución

```
}
```

```
private void jTextFieldMouseClicked(JTextField  tf) {
```

Este método es llamado por el hilo de ejecución cuando el usuario realiza una acción de clic sobre alguna de las casillas o areas de texto, se pintan de amarillo el fondo de las casillas que forman la palabra y se actualiza la orientación del cursor en horizontal o vertical alternativamente, si volvemos a dar clic sobre el área de texto la orientación del cursor de escritura cambia a vertical y se marcan en amarillo las casillas correspondientes que forman la palabra vertical

```
MediaPlayer efectomarcar=new MediaPlayer(mediamarcar);

efectomarcar.setVolume(0.7);

efectomarcar.play();
```

Lanzamos un efecto de audio que indica al usuario que se dado el clic y que se va a cambiar el sentido de escritura.

```
int index=0;
```

Se inicializa un contador index en 0 y actualizamos la variable booleana horizontal con su valor contrario lógico, alternamos su valor con cada clic en verdadero o falso, así cuando horizontal sea falso lo tomaremos como vertical el sentido de escritura

```
horizontal=!horizontal;

Component []comp=jPanel1.getComponents();

for(int i=0;i<comp.length;i++)
        {
           comp[i].setBackground(Color.white);
```

```
            if (comp[i].equals(tf)) index=i;

            if (comp[i].isEnabled()) comp[i].setBackground(Color.BLUE);
```

Recordemos que en la variable "tf" que recibe el método como argumento tenemos el componente sobre el que fue realizado el clic. Entonces recorremos el vector lineal de componentes ponemos el fondo de cada uno de color Blanco y si es el componente sobre el que se ha dado clic guardamos la posición en la variable index , y si el componente está habilitado para que el usuario escriba lo ponemos de color Azul.

```
}

    if (index+1<comp.length&&index-1>0&&horizontal

        &&

        (comp[index+1].isEnabled()||comp[index-1].isEnabled()))
```

Analicemos la siguiente condición: si la siguiente posición al index (que es la posición en el vector lineal sobre la cual se dio el clic) y la posición anterior se encuentran dentro de los límites del vector y la dirección es la horizontal.

```
    {
      while(index<comp.length&&comp[index].isEnabled())
```

Recorremos en dirección horizontal desde la posición index hasta el final de la palabra y pintamos de amarillo la casilla

```
      {
        comp[index].setBackground(Color.yellow);

        index++;

      }

      index--;

      while(index>0&&comp[index].isEnabled())
```

Recorremos en dirección horizontal desde la posición index hasta el principio de la palabra y pintamos de amarillo la casilla

{

comp[index].setBackground(Color.yellow);

index--;

}

return;

Finalizamos la ejecución del evento.

}

if (index+l<comp.length&&index-l>0&&!horizontal

En este caso si el componente siguiente y el anterior sobre el que ha dado el clic están dentro de los límites del vector lineal de componentes y el sentido es Vertical.

&&

(comp[index+l].isEnabled()||comp[index-l].isEnabled()))

Si el componente anterior o el siguiente están habilitados para escribir el usuario

{

while(index<comp.length&&comp[index].isEnabled())

{

comp[index].setBackground(Color.yellow);

index+=l;

Recorremos el vector de componentes desde la posición index hasta el final de la palabra y pintamos las casillas de amarillo

}

index-=l;

```
        while(index>0&&comp[index].isEnabled())

        {
            comp[index].setBackground(Color.yellow);

            índex-=l;
```

Recorremos el vector de componentes desde la posición index hasta el principio de la palabra y pintamos de amarillo

```
}

    }
```

Y hasta aquí tenemos el proceso para alternar sobre las casillas la dirección de escritura y señalizar las casillas de amarillo sobre las cuales escribiremos este método es llamado por el hilo de ejecución cada vez que se realiza pun evento del clic sobre alguna de las áreas de texto del panel de trabajo que representa gráficamente nuestro crucigrama.

```
}
```

Es importante distinguir ente tiempo de ejecución y compilación, pues en tiempo de compilación se inicializan todas las variables los objetos y componentes gráficos y se lanza la aplicación, y el tiempo de ejecución es todo lo que programamos para que ocurra una vez que ya está lanzada de aplicación de venta y durante todo el tiempo de vida de la aplicación, los eventos son un buen ejemplo de código que se ejecuta en tiempo de ejecución. A continuación vemos el proceso en tiempo de ejecución que se realizará cada vez que se presione una tecla sobre alguna de las áreas de texto de nuestra tabla de componentes.

```
private void jTextFieldKeyPressed(JTextField tf){

    MediaPlayer efectoclick=new MediaPlayer(mediaclick);

    efectoclick.setVolume(0.7);

    efectoclick.play();

    tf.setText("");
```

Lanzamos un efecto de sonido que indica que hemos tecleado un carácter, y limpiamos el área de texto sobre la que hemos dado el clic.

```
for(int f=1;f<l;f++)

  for(int c=1;c<l;c++)

  {   JTextField aux=(JTextField)jPanel1.getComponent(f*l+c);



  if(tabla[f][c].equals(aux.getText()))
```

Recorremos la tabla de solución y comparamos con el texto correspondiente en las casillas si coinciden con la solución pintamos el fondo de la casilla de color blanco.

```
      aux.setBackground(Color.white);

  else if  (!aux.getText().equals(" ")

        &&

        !aux.getText().equals(""))
```

Si no es la respuesta correcta y nos es un "" espacio vacío o en blanco entonces es porque es una respuesta incorrecta y la pintamos de color rojo

```
      aux.setBackground(Color.red);
```

Si finalmente es un espacio en blanco o vacío pintamos la casilla de color azul.

```
  else aux.setBackground(Color.BLUE);

  }

}
```

Este módulo es llamado por el hilo de ejecución cuando se realiza un evento de liberar la tecla aquí lo más importante es colocar el cursor o foco de escritura en la casilla que

corresponde a la siguiente letra de la palabra teniendo en cuenta la dirección de escritura seleccionada en el evento del clic.

```
private void jTextFieldKeyReleased(JTextField tf) {

  if (horizontal)tf.transferFocus();
```

Si el sentido es el horizontal transferimos el foco al siguiente componente llamando al método heredado de la superclase que pasara al siguiente componente habilitado en el orden de la secuencia natural.

```
  else {
```

Si el sentido es el vertical hay un poco más de trabajo, recorremos el vector lineal de componentes y...

```
        Component []comp=jPanel1.getComponents();

        for(int i=0;i+l<comp.length;i++)
        {
          if(comp[i].equals(tf))
          {
```

Si nos encontramos en la posición "i" del componente sobre el cual se ha liberado la tecla limpiamos el área de texto y transferimos el foco de escritura a la casilla que se encuentra en la posición "i" mas la dimensión de la tabla "l" o lo que es lo mismo, en la casilla siguiente en dirección vertical y hacia el final de la palabra sobre la que estamos escribiendo.

```
          ((JTextField)comp[i+l]).setText("  ");

          for (int j=0;j<l;j++)  comp[i+j].transferFocus();

          return;
```

Finalizamos la ejecución del método.

```
          }

        }

      }
}
```

Este método privado de la clase es el que busca las palabras que van a formar parte del crucigrama y se colocan en la tabla de solución, de forma horizontal y vertical de manera que se forma el crucigrama y se guarda en la tabla de solución, este método es llamado cada vez que se llama al método generar() como hemos definido anteriormente.

```
public void busca(int l){

  tabla=new String[l][l];
```

Creamos una tabla, una matriz cuadrada de dimensión l.

```
   String aux="";
```

Inicializamos la variable aux en "" una cadena vacía

```
   while(aux.length()<l )
   { mientras la longitud de la cadena auxiliar sea menos que l
      ArrayList
arraux=((ArrayList)(DICCIONARIO.get((int)(DICCIONARIO.size()*random.nextFloat()))))
;
```

Guardamos en la variable arraux un vector escogido al azar dentro del vector DICCIONARIO que contiene a su vez los vectores Da, Db,...Dz, que contienen las palabras agrupadas por la inicial de la palabra. De modo que en arraux tenemos al azar el Da o Db ...Dz .

```
      aux+=((String)(arraux.get((int)(arraux.size()*random.nextFloat()))))+"$";
```

Y guardaremos una palabra también escogida al azar dentro del vector Da, Db..o Dz le pondremos la marca "$" indicando que termina una palabra y concatenaremos con la siguiente hasta que la longitud de la variable aux sea menos que la dimensión de la matriz, en cullo caso se inicializa aux en un campo vacío y se repite la operación hasta que se encuentre la combinación de palabras que su longitud sumada entre sea igual a la dimensión de la matriz .

```
      if (aux.length()>l) aux="";

   }

   for(int f=0;f<l;f++) for(int c=0;c<l;c++) tabla[f][c]=" ";
```

Inicializamos la matriz cuadrada por filas y columnas y colocamos un espacio " "

```
for(int i=0;i<aux.length();i++) tabla[0][i]=aux.charAt(i)+"";
```

Nos desplazamos por la fila 0 y colocamos el contenido de la variable aux.

```
for(int i=0;i<l;i+=6){

    String s=(String)tabla[0][i];
```

Saltamos en la fila 0 a un paso de 6 y siendo menor que la dimensión y tomamos el carácter que se ha colocado en la posición "i" de la fila uno y lo guardamos en la variable "s", así garantizamos que este proceso de hará a lo sumo 2 veces ya que la dimensión l no puede ser mayor que 15 , y nos toca buscar una palabra que empiece con dicho carácter al azar dentro del vector de palabras Da, Db..Dz según corresponda y la colocaremos en sentido vertical dentro de la tabla de solución para ir construyendo el crucigrama de arriba hacia abajo. Este salto de 6 en 6 al recorrer las columnas nos limita la cantidad de combinaciones de palabras para rellenar la vertical porque si probamos todas las combinaciones el orden de operaciones rebasa el limite de la memoria virtual. Al igual que en el recorrido horizontal hacemos un salto de 2, así limitamos y espaciamos las combinaciones de palabras dentro de la matriz solución de modo que existan una cantidad suficiente de espacios en blanco en el crucigrama, y economizamos el numero de operaciones a realizar, esto se puede modificar pero con esta combinación conseguimos una buena distribución del crucigrama en la tabla de solución

```
    String aux2="";
```

Inicializamos una variable aux2 con un valor vacío, en ella guardaremos la la palabra en vertical que colocaremos en el crucigrama y mientras la longitud de aux2 sea distinta de la dimensión de la matriz de solución repetiremos la búsqueda.

```
while(aux2.length()!=l){

        do {
            aux2="";
```

Escogemos al azar una palabra dentro del vector de palabras del diccionario que comience con la letra contenida en la variable "s" que ha sido obtenida en los pasos anteriores.

```
            switch (s) {

                case "a":

                    aux2=""+Da.get((int)(Da.size()*random.nextFloat()));

                    break;
```

```
case "b":
    aux2=""+Db.get((int)(Db.size()*random.nextFloat()));
    break;
case "c":
    aux2=""+Dc.get((int)(Dc.size()*random.nextFloat()));
    break;
case "d":
    aux2=""+Dd.get((int)(Dd.size()*random.nextFloat()));
    break;
case "e":
    aux2=""+De.get((int)(De.size()*random.nextFloat()));
    break;
case "f":
    aux2=""+Df.get((int)(Df.size()*random.nextFloat()));
    break;
```

```
case "g":
    aux2=""+Dg.get((int)(Dg.size()*random.nextFloat()));
    break;
case "h":
    aux2=""+Dh.get((int)(Dh.size()*random.nextFloat()));
    break;
case "i":
    aux2=""+Di.get((int)(Di.size()*random.nextFloat()));
    break;
case "j":
    aux2=""+Dj.get((int)(Dj.size()*random.nextFloat()));
    break;
case "k":
    aux2=""+Dk.get((int)(Dk.size()*random.nextFloat()));
    break;
case "l":
```

```
    aux2=""+Dl.get((int)(Dl.size()*random.nextFloat()));
    break;
case "m":
    aux2=""+Dm.get((int)(Dm.size()*random.nextFloat()));
    break;
case "n":
    aux2=""+Dn.get((int)(Dn.size()*random.nextFloat()));
    break;
case "ñ":
    aux2=""+Dñ.get((int)(Dñ.size()*random.nextFloat()));
    break;
case "o":
    aux2=""+Do.get((int)(Do.size()*random.nextFloat()));
    break;
case "p":
```

```
    aux2=""+Dp.get((int)(Dp.size()*random.nextFloat()));
    break;
case "q":
    aux2=""+Dq.get((int)(Dq.size()*random.nextFloat()));
    break;
case "r":
    aux2=""+Dr.get((int)(Dr.size()*random.nextFloat()));
    break;
case "s":
    aux2=""+Ds.get((int)(Ds.size()*random.nextFloat()));
    break;
case "t":
    aux2=""+Dt.get((int)(Dt.size()*random.nextFloat()));
    break;
case "u":
    aux2=""+Du.get((int)(Du.size()*random.nextFloat()));
```

```
    break;
case "v":
    aux2=""+Dv.get((int)(Dv.size()*random.nextFloat()));
    break;
case "w":
    aux2=""+Dw.get((int)(Dw.size()*random.nextFloat()));
    break;
case "x":
    aux2=""+Dx.get((int)(Dx.size()*random.nextFloat()));
    break;
case "y":
    aux2=""+Dy.get((int)(Dy.size()*random.nextFloat()));
    break;
case "z":
    aux2=""+Dz.get((int)(Dz.size()*random.nextFloat()));
```

```
                    break;

                default:

                    break;
            }

            aux2+="*";
```

Colocamos al final de la palabra la marca “*” para indicar que finaliza una palabra en vertical y repetimos el proceso concatenando las palabras en vertical a la cadena aux2 mientras la longitud de la cadena sea mayor o igual que la dimensión de la matriz o sea una cadena vacía.

```
            } while(aux2.length()>=l&&!s.equals("  "));


            int ran=(int)(10*random.nextFloat());

            if (ran==0||ran==2||ran==4)  aux2+=" ";

            else if (ran==1||ran==3||ran==5) aux2+="  ";
```

En esta parte hacemos una discriminación al azar dándole valor a la variable “ran” con un valor del 0 al 10 , si el valor obtenido al azar es par en la variable aux2 concatenamos un espacio en blanco si es impar concatenamos dos espacios en blanco.

```
            String aux3="";


            while (aux2.length()+aux3.length()<l)
```

{ haremos esta operación mientras la longitud de la palabra contenida en la variable aux2 + longitud de la palabra contenida en aux3 sea menor que la dimensión

```
                ArrayList
arraux=((ArrayList)(DICCIONARIO.get((int)(DICCIONARIO.size()*random.nextFloat()))))
;
```

Guardamos en arraux in vector de diccionarios al azar Da, Db...o Dz.

String aux6=((String)(arraux.get((int)(arraux.size()*random.nextFloat()))));

Asignamos a la variable aux6 una palabra escogida al azar del diccionario de vectores Da, Db,..o Dz guardado en la variable arraux , escogido también al azar.

if (aux6.length()==1) aux6=" ";

Ahora comprobamos que la longitud de la variable aux6 sea igual a la dimensión en cuyo caso la inicializamos a un espacio vacío

De lo contrario tendremos en aux6 una palabra de longitud menor o mayor que la dimensión de la matriz en la dirección vertical.

int ran1=(int)(10*random.nextFloat());

if (ran1==0||ran1==2||ran1==4) aux3+="$"+aux6+"*";

else aux3+="$"+aux6+"* ";

Guardamos un valor al azar entre 0 y 10 en la variable ran1 si es par concatenamos a la variable aux3 que contiene la otra palabra en vertical la marca "$" para indicar la separación entre palabras y la marca "*" y si es impar con la marca "* " al final para indicar que ha terminado la combinación de palabras. Y este proceso se repetirá hasta que encontremos la combinación de palabras correcta tal que suma de las longitudes de cadena sean igual a la dimensión de la matriz de solución

}

Y repetimos la operación concatenando la palabra contenida en aux2 con la nueva palabra obtenida cuya combinación es la correcta.

aux2+=aux3;

}

Recorremos la columna de la posición "i" desde la fila 0 hasta la última y colocamos cada uno de los caracteres de la cadena contenida en aux2 en su respectivo lugar en la matriz de solución.

for(int x=0;x<l;x++) tabla[x][i]=aux2.charAt(x)+"";

} //vertircal

Y aquí termina la búsqueda de combinaciones por la vertical.

```
//////////////////////////////////////////////////////

for(int i=2;i<l;i+=2)

{   Comenzando en la posición 2 con saltos de 2 y mientras la posición "i" sea menor que la dimensión de la matriz de solución....

    String s=(String)tabla[i][0];

   Obtenemos el carácter en la columna 0 desplazándonos por las filas "i" como hemos apuntado de 2 en 2, al igual en los casos anteriores los saltos de 2 en 2 en la horizontal se hacen para trocar el número de combinaciones posibles y facilitar los espacios en blancos al crear el crucigrama no nos quede demasiado denso, como seguramente imaginas buscaremos ahora las combinaciones horizontales.

    String aux2="";

    while(aux2.length()!=l){

        do {
          aux2="";
          Escogemos el diccionario que corresponde con la primera letra de la fila "i"
          switch (s) {

            case "a":

              aux2=""+Da.get((int)(Da.size()*random.nextFloat()))+"*";

              break;

            case "b":

              aux2=""+Db.get((int)(Db.size()*random.nextFloat()))+"*";
```

```
    break;
case "c":
    aux2=""+Dc.get((int)(Dc.size()*random.nextFloat()))+"*";
    break;
case "d":
    aux2=""+Dd.get((int)(Dd.size()*random.nextFloat()))+"*";
    break;
case "e":
    aux2=""+De.get((int)(De.size()*random.nextFloat()))+"*";
    break;
case "f":
    aux2=""+Df.get((int)(Df.size()*random.nextFloat()))+"*";
    break;
case "g":
    aux2=""+Dg.get((int)(Dg.size()*random.nextFloat()))+"*";
    break;
```

```
case "h":
    aux2=""+Dh.get((int)(Dh.size()*random.nextFloat()))+"*";
    break;
case "i":
    aux2=""+Di.get((int)(Di.size()*random.nextFloat()))+"*";
    break;
case "j":
    aux2=""+Dj.get((int)(Dj.size()*random.nextFloat()))+"*";
    break;
case "k":
    aux2=""+Dk.get((int)(Dk.size()*random.nextFloat()))+"*";
    break;
case "l":
    aux2=""+Dl.get((int)(Dl.size()*random.nextFloat()))+"*";
    break;
```

```
case "m":
    aux2=""+Dm.get((int)(Dm.size()*random.nextFloat()))+"*";
    break;
case "n":
    aux2=""+Dn.get((int)(Dn.size()*random.nextFloat()))+"*";
    break;
case "ñ":
    aux2=""+Dñ.get((int)(Dñ.size()*random.nextFloat()))+"*";
    break;
case "o":
    aux2=""+Do.get((int)(Do.size()*random.nextFloat()))+"*";
    break;
case "p":
    aux2=""+Dp.get((int)(Dp.size()*random.nextFloat()))+"*";
    break;
case "q":
```

```
    aux2=""+Dq.get((int)(Dq.size()*random.nextFloat()))+"*";
    break;
case "r":
    aux2=""+Dr.get((int)(Dr.size()*random.nextFloat()))+"*";
    break;
case "s":
    aux2=""+Ds.get((int)(Ds.size()*random.nextFloat()))+"*";
    break;
case "t":
    aux2=""+Dt.get((int)(Dt.size()*random.nextFloat()))+"*";
    break;
case "u":
    aux2=""+Du.get((int)(Du.size()*random.nextFloat()))+"*";
    break;
case "v":
```

```
    aux2=""+Dv.get((int)(Dv.size()*random.nextFloat()))+"*";
    break;
case "w":
    aux2=""+Dw.get((int)(Dw.size()*random.nextFloat()))+"*";
    break;
case "x":
    aux2=""+Dx.get((int)(Dx.size()*random.nextFloat()))+"*";
    break;
case "y":
    aux2=""+Dy.get((int)(Dy.size()*random.nextFloat()))+"*";
    break;
case "z":
    aux2=""+Dz.get((int)(Dz.size()*random.nextFloat()))+"*";
    break;
case "$":
    aux2="";
```

```
                    break;

                case "*":

                    aux2="*";

                    break;

                default:

                    break;
```

De modo que en aux2 tendremos una palabra seleccionada al azar en el vector correspondiente Da,Db...Dz y la marca “*” para indicar que finaliza la palabra, si comienza con “*” , dejamos el “*” y en caso de la marca “$” que indica que comienza una palabra, borramos la marca.

```
}

            } while(aux2.length()>=l&&!s.equals("  "));
```

Haremos esto mientras la longitud de aux2 desborde las dimensiones de la matriz

```
            int ran=(int)(10*random.nextFloat());

            if (ran==0||ran==2||ran==4)   aux2+=" ";

            else if (ran==1||ran==3||ran==5)  aux2+="  ";
```

Tomamos en ran un numero al azar entre 0 y 10 si resulta par en aux2 concatenamos un espacio en blanco “ ” sino “ ” esto, al igual que los saltos se hace para facilitar las coincidencias verticales con horizontales en las combinaciones.

```
            String aux3="";
```

En este paso como ya hemos visto anteriormente encontramos una variable aux3 que al concatenarla con aux2 la longitud de una combinación de longitud igual a la dimensión de la matriz de solución

```
while (aux2.length()+aux3.length()<l){

ArrayList
arraux=((ArrayList)(DICCIONARIO.get((int)(DICCIONARIO.size()*random.nextFloat()))))
;
```

En el vector arraux guardamos un vector diccionario Da,Db,...Dz escogido al azar

```
int ran1=(int)(10*random.nextFloat());
```

En la variable ran1 guardaremos un valor al azar

```
String aux6=((String)(arraux.get((int)(arraux.size()*random.nextFloat()))));
```

Guardamos un valor al azar entre 0 y 10 en la variable ran1 y guardamos en aux6 una palabra al azar dentro del diccionario Da, Db ..Dz escogido al azar.

```
switch (ran1) {

   case 0:

   case 2:

   case 4:

      aux3+="$"+aux6+"*";

      break;

   case 1:

   case 3:

   case 5:

      aux3+="$"+aux6+"* ";
```

```
                    break;

                default:

                    aux3+="$"+aux6+"*  ";

                    break;
                }
```

En los casos pares concatenamos en aux3 la marca ”$” para indicar el espacio entre palabras , la palabra en aux6 y la marca “* ” y un espacio al final del asterisco y dos espacios para los casos impares “* “. esta discriminación se hace también con el objetivo de facilitar las combinaciones de palabras.

```
                }

            aux2+=aux3;
```

Ahora concatenamos las palabras en las variables aux2 y aux3 y guardamos en resultado en la variable aux2

```
            if (aux2.length()==l){
```

Si la longitud de la cadena contenida en la variable aux2 es igual a la dimensión de la matriz significa que hemos encontrado la combinación de palabras correctas.

```
                for(int x=0;x<l;x++)
                    {
                        String aux5="";
```

Inicializamos una variable aux5 con un valor vacío

```
                        for(int p=0;p<aux2.length();p++)
                          {
                           if (aux2.charAt(p)!=' ') aux5+=aux2.charAt(p)+"";
```

Si aux2 es vacía concatenmos en aux5 una cadena vacía “”

Sino el carácter que se encuentra en la fila “i” recorriendo las columnas “p” desde el 0 hasta la dimensión de la matriz de solución.

```
                           else aux5+=tabla[i][p];
```

Recorremos la cadena en cada uno de los caracteres de la cadena aux2 sino es un valor nulo y guardamos una copia en aux5

}

aux2=aux5;

Guardamos el valor de aux5 que es la combinación correcta en aux2 ,

Si el valor de la tabla de solución en la fila "i" recorriendo todas las columnas "x" desde 0 hasta la dimensión de la matriz de solución y no es un espacio vacío " " es porque la combinación de palabras no coincide con las columnas ya construidas en el crucigrama inicializamos la variable aux2="" para que se repita el proceso de encontrar otra combinación y rompemos la iteración de búsqueda con la combinación incorrecta.

```
if (!tabla[i][x].equals(aux2.charAt(x)+"")

    &&

    !tabla[i][x].equals("  ")) {aux2="";break;}
}
}
}//while
```

Esta operación se repetirá hasta encontrar una combinación correcta

Que se guardara en la variable aux2.

```
for(int x=0;x<l;x++) tabla[i][x]=aux2.charAt(x)+"";
```

Entonces recorremos la matriz de solución en la fila "i" en dirección horizontal a lo largo de las columnas "x" y colocamos los valores correspondientes de cada carácter de la cadena aux2, la combinación correcta en la matriz de solución y aquí finaliza el barrido horizontal en el proceso de búsqueda de combinaciones para el crucigrama.

```
}   //horizontal

////////////////////////////////////////////////////////
```

Al llegar hasta este punto tenemos la matriz solución en la variable "tabla[][]" de dimensión "l", hemos construido el crucigrama partiendo de completar la fila 0 de la tabla con una combinación, hacer un barrido en vertical y otro en horizontal.

```
String [][]tablaX=new String[l+1][l+1];
```

```
for(int f=0;f<l+1;f++) for(int c=0;c<l+1;c++) tablaX[f][c]=" ";

for(int f=0;f<l;f++) System.arraycopy(tabla[f], 0, tablaX[f+1], 1, l);
                //for(int c=0;c<l;c++) tablaX[f+1][c+1]=tabla[f][c];
```

Aquí copiamos a una tablaX la tabla de solución y agregamos una fila y una columna

Y asignamos nuevamente el valor a la variable tabla=tablaX, de esta manera hemos agregado una fila en blanco y una columna a la matriz solución

```
tabla=tablaX;

for(int i=1;i<l;i+=6)

   if (!tabla[1][i].equals("  ")&&!tabla[1][i].equals("*")

         &&

         !tabla[1][i].equals("$"))  tabla[0][i]="$";
```

Recorremos la tabla de solución en la fila "1" desde la columna "i" inicializada en "1" y con saltos de 6 y si en dicha posición no hay un espacio vacío o "*" o "$" entonces es porque es la primera letra de una palabra así que colocamos en la posición "i" de la columna la marca "$"

```
for(int i=1;i<l;i+=2)
   if (!tabla[i][1].equals("  ")

         &&

         !tabla[i][1].equals("*")

         &&

         !tabla[i][1].equals("$"))  tabla[i][0]="$";
```

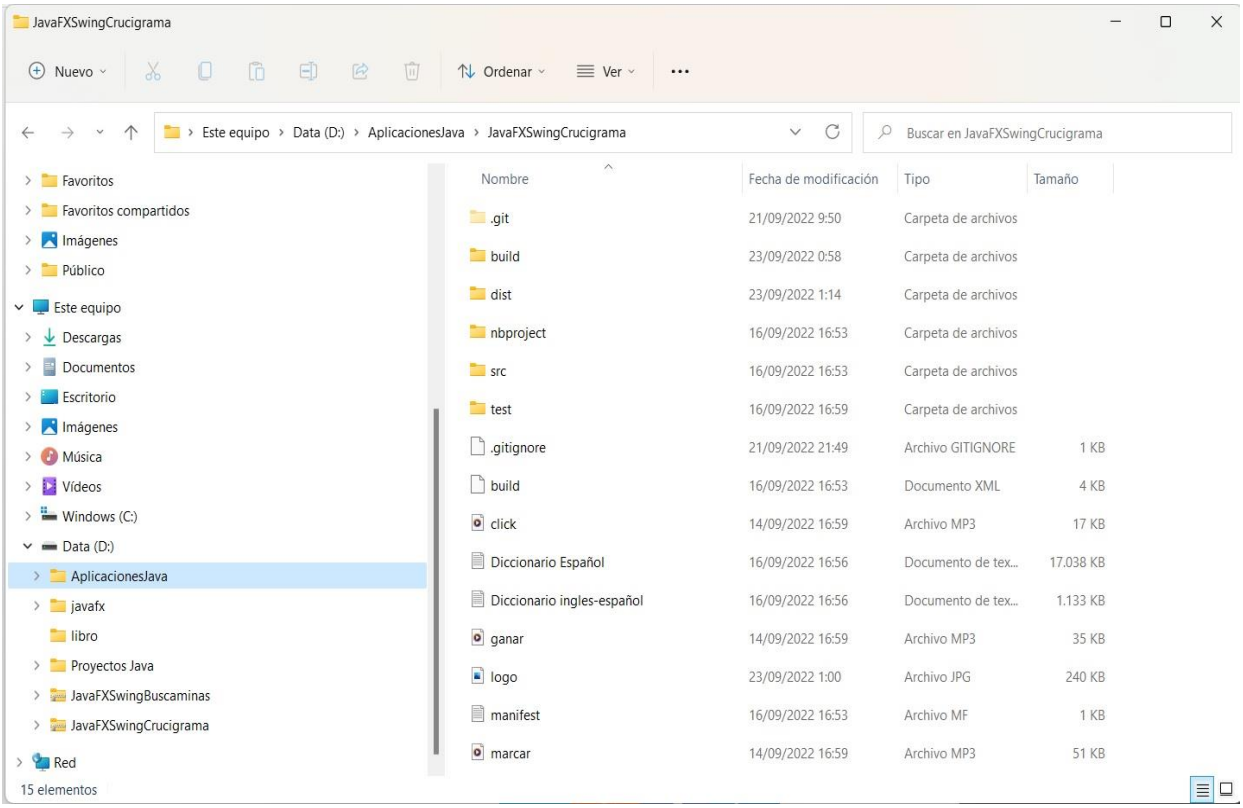
Ahora hacemos el mismo barrido en vertical y con saltos de a dos y colocamos las marcas "$" que indican la separación entre palabras.

Y aquí termina la construcción del crucigrama que guardamos en la variable tabla[][]

```
}
```

}

Hemos obtenido un crucigrama de dimensión “l” al azar a partir de un archivo diccionario de texto plano “.txt” con la estructura palabra-significado, recordemos que cambiando el archivo diccionario por otro en cualquier idioma obtendríamos el mismo resultado el algoritmo de construcción seria exactamente el mismo. Y también termina nuestra clase, a continuación se muestra el contenido de la carpeta del proyecto.



Contenido de la carpeta del proyecto .