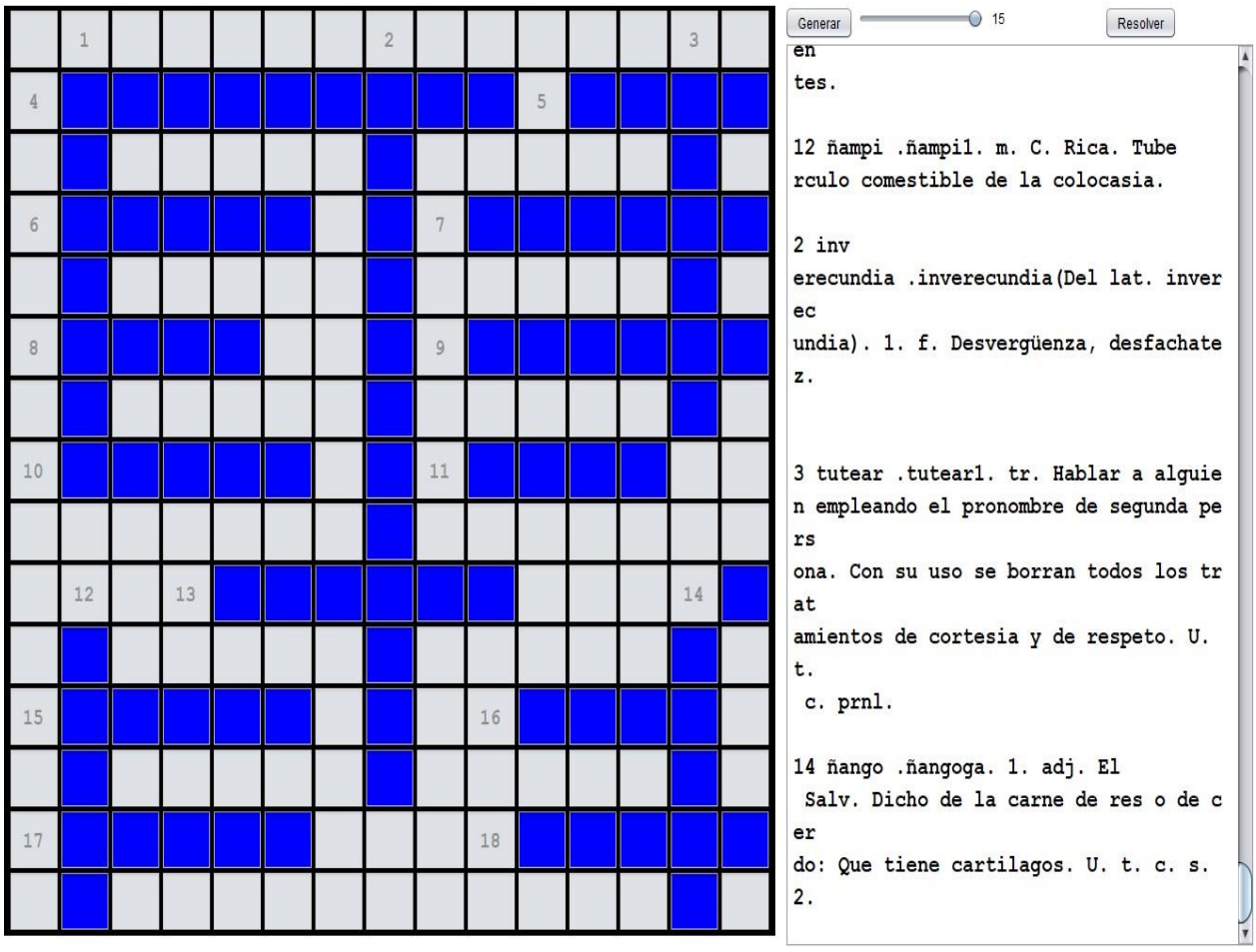
CROSSWORD

En JavaFX Swing.



A **crossword puzzle** is a game or **hobby** that consists of **filling in the gaps** of a drawing with letters. To find out which letter should be written in each space, the crossword puzzle indicates the **meaning of the words** to be read vertically and horizontally. The idea, therefore,

is that the already completed crossword template presents a series of words that can be read vertically and horizontally and that intersect with each other.

How to complete a crossword puzzle

To begin completing a crossword puzzle, the person must read the two lists of definitions that the hobby presents: one corresponding to the vertical sense and one for the horizontal sense. The template or drawing is divided into **white boxes** (where you have to write the individual letters) and **black boxes** (which serve to separate the words).

Thus, when reading a definition and knowing the word, the participant has to enter the crossword puzzle, writing a letter in each white box of the corresponding space. In this way, little by little the game will be completed.

History of crossword puzzles

The origin of crossword puzzles **dates back to the first century**. The first crossword puzzle was found in the ruins of Pompeii and is known as the **sator** square.

Sator's Square, the first crossword puzzle

Later, already in the nineteenth century a man known as Hyperion published the famous double diamond puzzles; they were games of a mental nature that closely resembled the current crossword puzzles (with squares where the user had to intertwine the words until he found the right combination). It is believed that these publications were made over several years.

Years later a game would emerge that was deeply inspired by the sator square and consisted of a four-by-four grid with some clues so that users could find the answers; however, this entertainment was not very successful.

Finally, Arthur Wynne, an English journalist published in the New York World the first crossword puzzle, as we know it today. It was called "word-cross" although a few years later the term by which we still know this entertainment was coined.

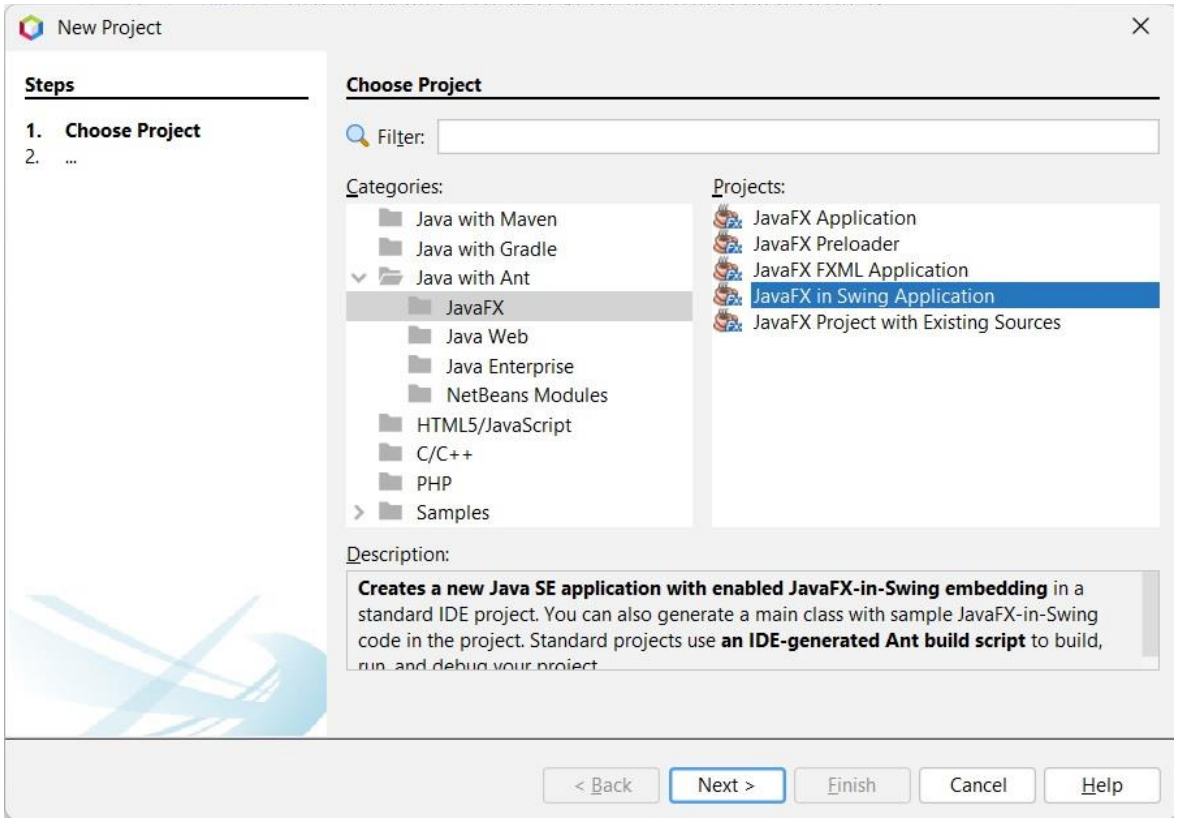
The fundamental elements of a crossword puzzle are: **the grid** (white cells with small numbers that are associated with the references) and **the references** (they are usually located at the bottom of the page and consist of a series of very concise definitions that allow the user to link a word with the space available to complete the grid).

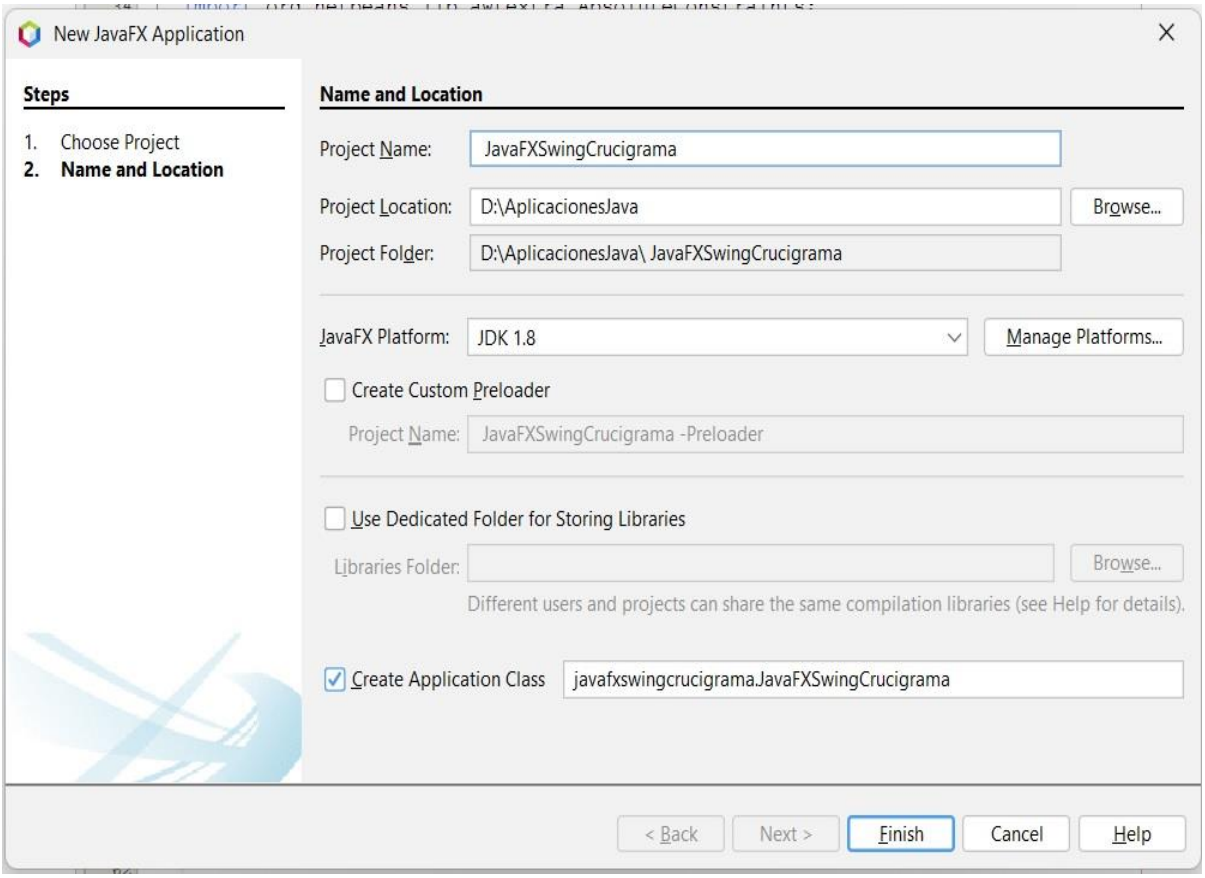
It is important to mention that the references are divided into two parts, those corresponding to the grid read **horizontally**, and those that correspond to the vertical distribution.

The words are chosen in such a way that **those boxes shared by two words have a letter that is equal** for the position of both terms, the horizontal and the vertical, in this way, as the grid is filled it is easier to find the missing words.

These hobbies are usually recommended for those people who have an easy time forgetting things since it keeps their mind awake and helps them associate concepts. They are usually published alone in magazines called crossword puzzles or crosswords and also in newspapers and magazines; in some cases, these **crossword puzzles are thematic**.

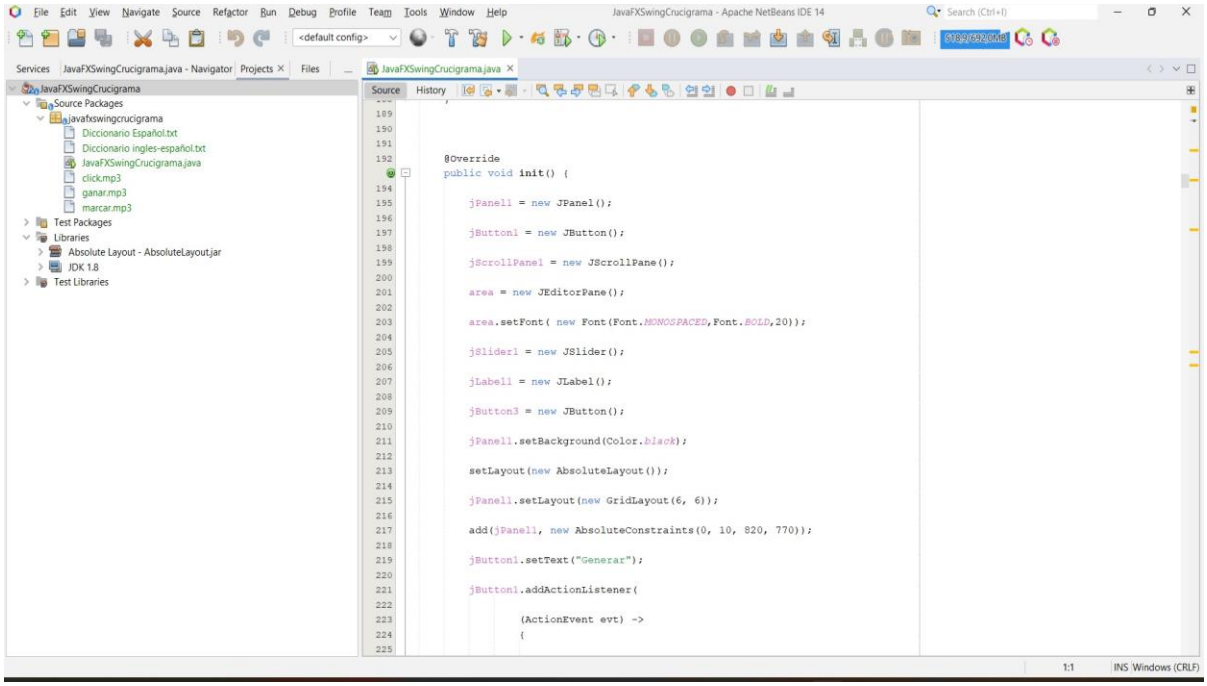
To get started we opened NetBeans and created our JavaFX Swing project.





And we add to our package the multimedia files that we will use as sound effects, specifically the sound files "win.mp3", "click.mp3", "mark.mp3", and the file "dictionary.txt", with which we will build the crossword puzzles with the words chosen at random from the file in this example we have a "Spanish Dictionary.txt" and an "English-Spanish Dictionary.txt", but any plain text file that contains the word structure can be used: meaning in any language and on any subject.

Source code:



/*

* 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;

We import the classes we will need

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 {
```

We declare the variables of the class

```
    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 call;

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();
```

In our main method we create the window and display an instance of our class.

```
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();
    });
  }
```

In the init() method, the variables and objects necessary for both graph and logic are initialized.

```
@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("Generate");
```

We create a "Generate" button, and the event of the click on the button is declared, so that when this happens it takes us to the jButtonActionPerformed() method, every time the user activates this "Generate" button the execution thread will go to the method and create a new crossword puzzle.

it also creates a text area that will be used later to write the descriptions of the horizontals and verticals and communications with the user, a Slider with which we can adjust the size of the words that we will use to generate our crossword puzzle, and a Panel that will contain the crossword.

```
jButton1.addActionListener(

    (ActionEvent possibly) >
    {

      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);
```

Here we delimit a minimum and a maximum of the size that the user can configure for the words of the crossword puzzle

```
jSlider1.setPaintLabels(true);

jSlider1.setPaintTicks(true);

jSlider1.setSnapToTicks(true);

jSlider1.setToolTipText("");

jSlider1.addChangeListener(

    (ChangeEvent possibly) >
    {

       jSlider1StateChanged();

    }
);
```

The method to follow is declared when an action is performed on the slider, the execution thread will be taken to the jSliderr1StateChanged() method

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

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

```
jButton3.setText("Resolve");

jButton3.addActionListener(

    (ActionEvent possibly) >
    {

      jButton3ActionPerformed();

    }
);

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

The event is declared on the "Resolve" button and the execution thread will be taken to the jButton3ActionPerformed() method, each time the user activates the button the program will execute this method to solve the crossword puzzle

```
Scanner scanner=null;

while(scanner==null){

  try {
    scanner =
        new Scanner(

            new BufferedReader(

                new FileReader(

                    new File("./Spanish Dictionary.txt")
```

```
                    )
                )
            );

        }

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

}
```

We create a scanner object that will read from the file "Diccionario Español.tx" that is contained in the package, a simple text without format with the word-meaning structure, in this section of the code we can modify the file from which scanner reads by another dictionary in any other language that the program uses to build the crossword puzzle. This program can create Crossword Puzzles in any language.

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

The scanner class works as an enumeration that iterates on the elements of a list, these elements are the lines of the document "Diccionario Español.txt". So we say that as long as scanner contains lines...

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

We save in the variable "aux" the content of the line of the document on which we are going to work.

```
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("-", " ");
```

We replace some special characters that java interprets differently to format the text.

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

{
```

if we are not working on an empty line or length 0 then...

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

In the vector arr we save the result of applying the Split method to our variable aux that contains the line of the document on which we are working, the Split method returns the line divided into 2 parts from the first ". " that I find, so we separate the word from the meaning.

```
String  aux1="";

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

   aux1+=scar[i];

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

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

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

We give a little more formatting to the text file and if the vector arr is not empty using a dictionary Hash Table we place the word-meaning pairs contained in the variable arr[0] and aux1 respectively.

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

```
        }
```

We will do this with each line of the document until the last one so that in the dictionary hash table we will have all the words of the document and their meanings, ordered.

```
    }


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

Now we iterate on our hash table specifically on the keys that are the words of the document that we have read.

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


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

We save in the variable "aux" the word in progress.

```
      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);
```

And in this step according to the letter that begins "aux" that we remember contains the word on which we work, we add it to the corresponding "ArrayList" vector, for example, if it begins with "a" we add it to the Da vector, if it begins with "b" we add it to the Db vector, and so with each of the words contained in the dictionary hash table

```
}


DICTIONARY.add(Da);

DICTIONARY.add(Db);

DICTIONARY.add(Dc);

DICTIONARY.add(Dd);

DICTIONARY.add(De);

DICTIONARY.add(Df);

DICTIONARY.add(Sun);

DICTIONARY.add(Dh);

DICCIONARIO.add(Di);
```

DICTIONARY.add(Dj);

DICTIONARY.add(Dk);

DICTIONARY.add(Dl);

DICTIONARY.add(Dm);

DICTIONARY.add(Dn);

DICTIONARY.add(Dñ);

Dictionary.add(Do);

DICTIONARY.add(Dp);

DICTIONARY.add(Dq);

DICTIONARY.add(Dr);

DICTIONARY.add(Ds);

DICTIONARY.add(Dt);

DICTIONARY.add(Du);

DICTIONARY.add(Fri);

DICTIONARY.add(Dw);

```
        DICTIONARY.add(Dx);

        DICTIONARY.add(Dy);

        DICTIONARY.add(Dz);
```

So so far we have a DICTIONARY vector that contains 27 vectors Da,Db,Dc... Dz, which contain the words of the document. And a Hash Table dictionary containing the pairs word, meaning. It is time to take a break, we already have the dictionary separated by words and meanings and grouped in alphabetical order. Every time I get to this part I have a coffee.

```
        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();

        });
```

We finished configuring the graphic features and audio files and will use them as sound effects. The audio files are located in the root folder of the project.

```
    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());

generate();
```

We call the "generate()" method that will create our crossword puzzle. It will be a grid of text areas, where we can enter letter by letter of the crossword, some textFields will be unused and others will contain the index of the horizontal or vertical word contained in the crossword.

We create an execution thread that in parallel to the execution thread of the program will check every 5000 millisecond or 5 seconds if we have completed the crossword puzzle satisfactorily.

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

        boolean correcto=true;
```

We take a correct Boolean variable and initialize it in True.

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

We save in the comp vector all the areas of text "textFields" that the crossword grid contains by applying the getComponents() method to jPanel1 which is where they are contained with a distribution of Table rows by Columns "GridLayout".

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

We go through the entire vector and if the component is enabled ...

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

    &&

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

Now, this part requires some mathematical clarification, the variable "l" contains the value of the slider that is the dimension of the grid of the crossword puzzle that is, it is the length and width, and "i" is the variable that marks the position of the button in the vector of graphic components, a list, later we will see that "table" is a matrix of characters with the solution to the crossword, if we divide the position "i" by "l" we will get the row to which comp[i] would belong in the table solution matrix, and if we take the rest of the division "%" we will obtain the column.

Clarified this we continue... If the text area "comp[i]" does not contain in the "i" position the same character as the solution table we put the correct variable in false.

```
    correct=false;
```

This is done with each of the text areas so that if when leaving this loop the correct variable continues with True value it is because all the text areas match the solution table and we have solved the crossword puzzle.

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

        efectoganar.setVolume(0.7);

        efectoganar.play();
```

We launch a sound effect that indicates to the user that he has solved the crossword satisfactorily and a dialogue window is also launched that contains a text message with the same objective.

```
        JOptionPane.showMessageDialog(frame,"You have completed the
crossword puzzle"

            ,"End game",javax.swing.JOptionPane.OK_OPTION);
```

Once the user confirms the window, a new crossword puzzle is generated to solve.

```
                    generate();
```

Remember that this check routine is performed every 5 seconds by a parallel execution thread

```
            }

    });


     timer.start();
```

We start the "time" thread of verification that we have declared above.

```
       }

   private void createScene() {

     StackPane root = new StackPane();

     fxContainer.setScene(new Scene(root));

   }
```

The window is launched and the graphics are displayed.

```
    private void generate(){
```

As already seen this method will be responsible for creating our grid of text areas that will graphically represent the crossword, contained in jPanel1, and the crossword in a table of characters table[l][l] whose dimension is "l" in length and width, solution table.

```
      l=jSlider1.getValue();
```

```
jLabel1.setText(l+"");

jPanel1.removeAll();

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

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

We initialize the work panel and give it size and table format l X l and call the method search and pass it as parameter l, this method will find random words within the dictionary and that form the crossword puzzle and place them in the solution table.

```
search(l);

int account=1;
```

We initialize a variable "cont", a counter, in value 1

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

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

we go through a square matrix of dimension "l", table solution table[f][c] in rows and columns, then f and c are its coordinates.

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

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

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

          jPanel1.add(tf);
```

The graphic text area is created, formatted, and added to the work pane

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

If in the solution table at position f,c there is a character "*", we replace it by putting a blank space " ".

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

and If in the solution table at position f,c there is a blank space " " or a character "$" we say that the text area "tf" will not be enabled. Or what is the same we disable all the text areas that are marked in the solution matrix as " " and "$"

```
tf.setEnabled(false);


else{
```

In another case we visualize the text area and format with blue background to the text areas that will be enabled so that the user can write the solutions, in short, the crossword puzzle is graphically configured from the solution table.

```
tf.setVisible(true);

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

We set the event over the text areas by pressing a key.

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

   @Override

   public void keyPressed(KeyEvent evt) {

      jTextFieldKeyPressed(tf);}});

tf.addKeyListener(new KeyAdapter() {

  @Override
```

We set the event over the text areas to release a key.

```
public void keyReleased(KeyEvent evt) {

   jTextFieldKeyReleased(tf);}});
```

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

We configure the event on the text areas, click on the graphic component.

```
@Override

public void mouseClicked(MouseEvent evt) {

   jTextFieldMouseClicked(tf);}});

}
```

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

                    account++;
                   }
```

In this step, if in the table of solutions in the position f,c there is a mark "$" means that it is the beginning of a word in the crossword puzzle both vertically and horizontally, then in that text area we place the value of the variable "cont" remember that it is initialized in the

value "1" and increase its value in each cycle, so that at the end each word of the crossword puzzle would be indexed.

```
                    }

    jPanel1.validate();

    this.validate();
```

And so far the graphic and logical configurations of our grid or crossword puzzle.

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

I save in the variable "text" the horizontal words of crossword, for which we go through the table of solutions skipping the even rows.

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

    {
       String aux="";

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

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

If in position f,c there is a mark "$", it means that a word begins, or is the first row of the table

```
          {
             to="";

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

If the row is not the first, we move through the row "f" to the next column to which it is marked with "$", until the beginning of the word .

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

We save in contH the horizontal index of the word that is in the position f*l+c in the previous coordinate that we obtain by subtracting 1.

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

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

While the " " or "$" marks are not in the solution table, we move through it and save in the accumulator variable "aux".

```
if (!aux.equals("")) if the word is not empty ""

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

We save in the accumulator variable "text", the string composed of the sum of the horizontal index, the word saved in aux, we put a "," and we concatenate the meaning of the word that we obtain from the dictionary hash table, we give it the key or key, it is the word saved in "aux", and it returns the meaning of the word that we have previously saved and a line break for each one.

```
        }

      }

    }
```

So far we have in the accumulator text the words and horizontal meanings

And now we will concatenate the words in Vertical.

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

We go through the table of solutions.

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

Initialize the aux accumulator in each column with the value "" and go through the column

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

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

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

If the table is marked with "$" we move vertically to the position before the beginning of the word and obtain the index that we will save in the variable "contV", basically we repeat the process followed for the horizontal ones, but moving through the solution matrix, but in a vertical direction.

```
          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++;}
```

We scroll through the solution table vertically and concatenate the characters in the testo areas and save the word in aux.

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

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

We check that we do not have an empty result, and we concatenate to the accumulator "text" the vertical index, the word and the meaning that we obtain from the dictionary hash table and a line break.

```
        }
   }

}

String temp="";
```

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

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

We divide the text into lines of 40 characters in length.

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

We write the result in the text area arranged in the window to communicate with the user.

At this point of the execution thread we have initialized and defined the graphic and logical components of the crossword puzzle, this private process of the program, "generate" is called by the execution threads at the beginning of the launch of the window and each time and is called every time the user activates the button "Generate".

```
 }

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

Here, the execution thread is directed to this method when the "generate" button is pressed and from here it is directed to the private method generate().

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

In this method, which is called when you activate the slider, the value of the text is updated in the label that shows the value of the dimension of the matrix or table that the crossword will have, or what is the same as the maximum length that the words of the crossword will have.

Remembering the above code is next defined method is called to execution when the user presses the "Resolve" button.

```
 private void jButton3ActionPerformed() {

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

We save in a comp vector all the text areas of the component matrix in the work panel, remember that we can obtain the position in the solution matrix by dividing the position "i" in the linear vector by the dimension of the matrix, we take the entire part as rows and the rest of the division as columns.

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

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

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

Traverses the component vector and compares the content in the text area to the content in the solution table

```
    }


   private void jTextFieldMouseClicked(JTextField tf) {
```

This method is called by the execution thread when the user performs a click action on any of the boxes or areas of text, the background of the boxes that form the word is painted yellow and the orientation of the cursor is updated horizontally or vertically alternately, if we click on the text area again the orientation of the writing cursor changes to vertical and are marked in yellow the corresponding boxes that form the word vertical

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

    efectomarcar.setVolume(0.7);

    efectomarcar.play();
```

We launch an audio effect that tells the user that the click is given and that the writing sense is going to be changed.

```
    int index=0;
```

An index counter is initialized in 0 and we update the horizontal Boolean variable with its logical opposite value, we alternate its value with each click in true or false, so when horizontal is false we will take it as vertical the sense of writing

```
    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);
```

Remember that in the variable "tf" that receives the method as an argument we have the component on which the click was made. Then we go through the linear vector of components we put the background of each one of White color and if it is the component on which it has been clicked we save the position in the index variable, and if the component is enabled for the user to write we put it in Blue color.

```
}

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

          &&

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

Let's analyze the following condition: if the next position to the index (which is the position in the linear vector on which the click was given) and the previous position are within the limits of the vector and the direction is the horizontal.

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

We travel in a horizontal direction from the index position to the end of the word and paint the box yellow

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

          index++;
```

```
        }

        index--;

        while(index>0&&comp[index].isEnabled())
```
We travel in a horizontal direction from the index position to the beginning of the word and paint the box yellow
```
        {
           comp[index].setBackground(Color.yellow);

           index--;
        }

        return;
```
We finish the execution of the event.
```
     }

     if (index+l<comp.length&&index-l>0&&!horizontal
```
In this case if the next and the previous component on which you clicked are within the limits of the linear vector of components and the sense is Vertical.
```
           &&

           (comp[index+l].isEnabled()|| comp[index-l].isEnabled()))
```
whether the previous or next component is enabled to type the user
```
     {
        while(index<comp.length&&comp[index].isEnabled())

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

```
            index+=l;
```

We go through the vector of components from the index position to the end of the word and paint the boxes yellow

```
 }

        index-=l;

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

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

            index-=l;
```

We go through the vector of components from the index position to the beginning of the word and paint yellow

```
}

    }
```

And so far we have the process to alternate on the boxes the direction of writing and signal the yellow boxes on which we will write this method is called by the execution thread every time there is a click event on one of the text areas of the work panel that graphically represents our crossword.

```
}
```

It is important to distinguish between runtime and compilation, because at compile time all the variables are initialized objects and graphic components and the application is launched, and the execution time is all that we program to occur once it is already launched from the sales application and throughout the life of the application, events are a good example of code running at run time. Below we see the runtime process that will be performed every time a key is pressed on any of the text areas of our component table.

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

```
MediaPlayer efectoclick=new MediaPlayer(mediaclick);

efectoclick.setVolume(0.7);

efectoclick.play();

tf.setText("");
```

We launch a sound effect that indicates that we have typed a character, and we clean the text area on which we have clicked.

```
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()))
```

We go through the solution table and compare with the corresponding text in the boxes if they match the solution we paint the background of the box white.

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

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

        &&

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

If it is not the correct answer and it is an empty or blank "" space then it is because it is an incorrect answer and we paint it red

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

If it is finally a blank or empty space we paint the box blue.

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


        }


}
```

This module is called by the execution thread when an event is made to release the key here the most important thing is to place the cursor or writing focus in the box that corresponds to the next letter of the word taking into account the writing direction selected in the click event.

```
    private void jTextFieldKeyReleased(JTextField tf) {


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

If the sense is horizontal we transfer the focus to the next component by calling the method inherited from the superclass that will pass to the next component enabled in the order of the natural sequence.

```


       else {
```

If the sense is vertical there is a little more work, we go through the linear vector of components and...

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


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

If we are in the "i" position of the component on which the key has been released, we clean the text area and transfer the writing focus to the box that is in the "i" position plus the dimension of the table "l" or what is the same, in the next box in the vertical direction and towards the end of the word on which we are writing.

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

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


                return;
```
We finish the execution of the method.
```
                }


            }


        }
    }
```

This private method of the class is the one that looks for the words that are going to be part of the crossword puzzle and are placed in the solution table, horizontally and vertically so that the crossword puzzle is formed and saved in the solution table, this method is called every time the generate() method is called as we have defined above.

```
public void busca(int l){


  tabla=new String[l][l];
```
We create a table, a square matrix of dimension l.
```
   String aux="";
```

We initialize the variable aux in "" an empty string
```
   while(aux.length()<l )
```
{ as long as the length of the auxiliary chain is less than l
```
      ArrayList
arraux=((ArrayList)(DICCIONARIO.get((int)(DICCIONARIO.size()*random.nextFloat()))));
```
We save in the variable arraux a vector chosen at random within the vector DICTIONARY that contains in turn the vectors Da, Db,... Dz, which contain the words grouped by the initial of the word. So in arraux we have randomly the Da or Db ... Dz.
```
      aux+=((String)(arraux.get((int)(arraux.size()*random.nextFloat()))))+"$";
```

And we will save a word also chosen at random within the vector Da, Db.. o Dz we will put the mark "$" indicating that one word ends and we will concatenate with the next one until the length of the variable aux is less than the dimension of the matrix, in that case aux is initialized in an empty field and the operation is repeated until the combination of words is found that its length added between is equal to the dimension of the matrix .

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


}



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

We initialize the square matrix by rows and columns and place a space " "

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

We move through row 0 and place the contents of the aux variable.

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


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

We jump in row 0 at a step of 6 and being less than the dimension and take the character that has been placed in the position "i" of row one and save it in the variable "s", thus we guarantee that this process will be done at most 2 times since the dimension l can not be greater than 15 , and we have to look for a word that begins with that character randomly within the vector of words Da, Db.. Dz as appropriate and we will place it vertically within the solution table to build the crossword puzzle from top to bottom. This jump of 6 in 6 when going through the columns limits the number of combinations of words to fill the vertical because if we try all the combinations the order of operations exceeds the limit of the virtual memory. As in the horizontal path we make a jump of 2, so we limit and space the combinations of words within the solution matrix so that there are a sufficient amount of white spaces in the crossword puzzle, and we save the number of operations to be performed, this can be modified but with this combination we get a good distribution of the crossword in the solution table

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

We initialize a variable aux2 with an empty value, in it we will save the word vertically that we will place in the crossword puzzle and as long as the length of aux2 is different from the dimension of the solution matrix we will repeat the search.

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

```
of {
    aux2="";
```

We randomly choose a word within the word vector of the dictionary that begins with the letter contained in the variable "s" that has been obtained in the previous steps.

```
    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;
```

```
houses "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;

houses "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;
```

```
Almost "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)(Two.size()-random.nextFloat());

               break;

            case "z":

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

               break;

            default:

               break;
         }

         aux2+="*";
```

We place at the end of the word the mark "*" to indicate that a word ends vertically and repeat the process by concatenating the words vertically to the string aux2 as long as the length of the string is greater than or equal to the dimension of the matrix or is an empty string.

```
         } 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+="  ";
```

In this part we make a random discrimination giving value to the variable "ran" with a value from 0 to 10 , if the value obtained at random is even in the variable aux2 we concatenate a blank space if it is odd we concatenate two blank spaces.

```
String aux3="";

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

{ we will do this operation as long as the length of the word contained in the variable aux2 + length of the word contained in aux3 is less than the dimension

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

We save in arraux in vector of random dictionaries Da, Db... or Dz.

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

We assign to the variable aux6 a word chosen at random from the vector dictionary Da, Db,.. o Dz saved in the variable arraux, also chosen at random.

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

Now we check that the length of the variable aux6 is equal to the dimension in which case we initialize it to an empty space

Otherwise we will have in aux6 a word of length less than or greater than the dimension of the matrix in the vertical direction.

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

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

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

We save a random value between 0 and 10 in the variable ran1 if it is even we concatenate to the variable aux3 that contains the other word vertically the mark "$" to indicate the separation between words and the mark "*" and if it is odd with the mark "* " at the end to indicate that the combination of words has finished. And this process will be repeated until we find the correct combination of words such that the sum of the string lengths are equal to the dimension of the solution matrix.

```
            }
```

And we repeat the operation by concatenating the word contained in aux2 with the new word obtained whose combination is correct.

```
            aux2+=aux3;


        }
```

We go through the column of the position "i" from row 0 to the last and place each of the characters of the string contained in aux2 in its respective place in the solution matrix.

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


   }     //vertircal
```

And here ends the search for combinations by the vertical.

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


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

{ Starting at position 2 with jumps of 2 and as long as the position "i" is less than the dimension of the solution matrix....

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

We obtain the character in column 0 by moving through the rows "i" as we have pointed out from 2 to 2, as in the previous cases the jumps of 2 in 2 in the horizontal are made to cut the number of possible combinations and facilitate the blanks when creating the crossword

puzzle do not be too dense, as you can surely imagine we will now look for horizontal combinations.

```
String aux2="";

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

        of {
          aux2="";
          We choose the dictionary that corresponds to the first letter of the row "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;

houses "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;

houses "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;

Almost "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)(Two.size()*random.nextFloat())";

    break;

case "z":

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

    break;

case "$":

    aux2="";

    break;

case "*":
```

```
                    aux2="*";

                    break;

                default:

                    break;
```

So in aux2 we will have a randomly selected word in the corresponding vector Da,Db... Dz and the mark "*" to indicate that the word ends, if it begins with "*", we leave the "*" and in case of the mark "$" that indicates that a word begins, we delete the mark.

```
}

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

We will do this as long as the length of aux2 overflows the dimensions of the matrix

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

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

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

We take in ran a random number between 0 and 10 if it turns out to be even in aux2 we concatenate a blank space " " but " this, just as jumps are done to facilitate vertical and horizontal matches in combinations.

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

In this step as we have seen above we find a variable aux3 that when concatenated with aux2 the length of a combination of length equal to the dimension of the solution matrix

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

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

In the vector arraux we save a dictionary vector Da,Db,... Dz chosen at random

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

In the variable ran1 we will save a random value

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

We save a random value between 0 and 10 in the variable ran1 and we save in aux6 a random word within the dictionary Da, Db .. Dz chosen at random.

```
switch (ran1) {

   case 0:

   case 2:

   case 4:

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

      break;

   case 1:

   case 3:

   case 5:

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

      break;
```

```
                default:

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

                    break;
                }
```

In even cases we concatenate in aux3 the mark "$" to indicate the space between words, the word in aux6 and the mark "* " and a space at the end of the asterisk and two spaces for odd cases "* ". this discrimination is also done with the aim of facilitating word combinations.

```
                }

            aux2+=aux3;
```

Now we concatenate the words in the variables aux2 and aux3 and save in result in the variable aux2

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

If the length of the string contained in the variable aux2 is equal to the dimension of the matrix it means that we have found the right combination of words.

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

We initialize an aux5 variable with an empty value

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

If aux2 is empty let's concatenate in aux5 an empty string ""

But the character found in row "i" running through the columns "p" from 0 to the dimension of the solution matrix.

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

We go through the string in each of the characters of the string aux2 but it is a null value and we save a copy in aux5

```
                    }


                    aux2=aux5;
```

We save the value of aux5 which is the correct combination in aux2 ,

If the value of the solution table in row "i" running through all the columns "x" from 0 to the dimension of the solution matrix and is not an empty space " " it is because the combination of words does not match the columns already constructed in the crossword puzzle we initialize the variable aux2="" so that the process of finding another combination is repeated and we break the search iteration with the wrong combination.

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


                        &&


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

this operation will be repeated until a correct combination is found

To be saved in the aux2 variable.

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

Then we go through the solution matrix in the row "i" in a horizontal direction along the columns "x" and place the corresponding values of each character of the string aux2, the correct combination in the solution matrix and here ends the horizontal sweep in the process of finding combinations for the crossword puzzle.

```
    }    //horizontal


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

When we get to this point we have the solution matrix in the variable "table[][]" of dimension "l", we have built the crossword puzzle starting from completing row 0 of the table with a combination, making a vertical sweep and another horizontally.

```
    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];
```

Here we copy the solution table to a tableX and add a row and a column

And we assign the value again to the variable table=tableX, in this way we have added a blank row and a column to the solution matrix

```
table=tableX;

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

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

         &&

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

We go through the solution table in row "1" from the column "i" initialized in "1" and with jumps of 6 and if in that position there is no empty space or "*" or "$" then it is because it is the first letter of a word so we place in the position "i" of the column the mark "$"

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

         &&

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

         &&

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

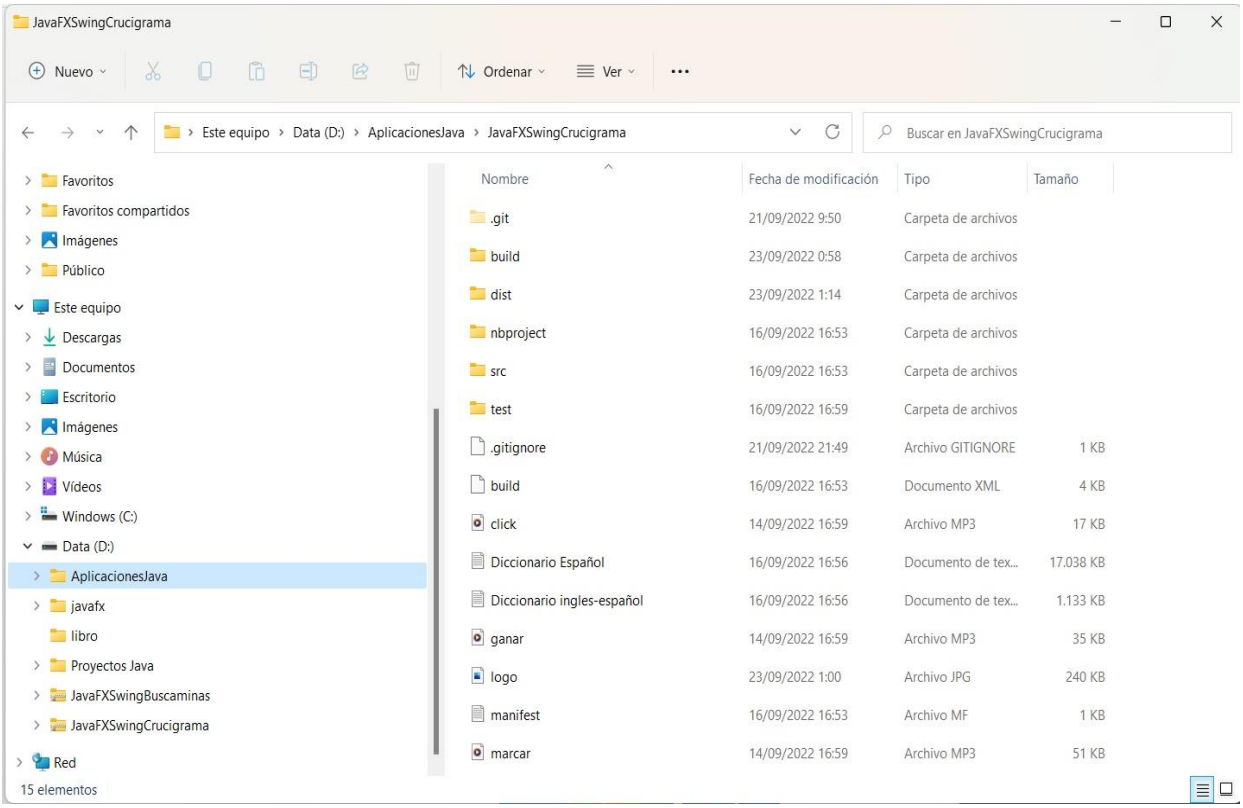
Now we do the same sweep vertically and with jumps of two and place the "$" marks that indicate the separation between words.

And here ends the construction of the crossword puzzle that we save in the variable table[][]

}

}

We have obtained a crossword puzzle of dimension "l" at random from a plain text dictionary file ".txt" with the word-meaning structure, remember that changing the dictionary file for another in any language we would obtain the same result the construction algorithm would be exactly the same. And also ends our class, below is the contents of the project folder.



Contents of the project folder.