Treball Final de Carrera

Teclat virtual amb predicció del llenguatge

Eloi Creus Casassas

Enginyeria Tècnica d'Informàtica de Gestió i Sistemes
Director: Joan Vancells i Flotats

Taradell, Febrer de 2012

Agraïments

Albert Baucells, per les idees aportades

Francesc Codina, per l'assessorament filològic

Joan Vancells, per la coordinació del projecte

Eumo Editorial, per facilitar el Cercamots

A totes les persones que ho han fet possible.

"Elimineu la instrucció i s'acabaran les preocupacions."
(LAO TSE, del Tao Te King)

ÍNDEX

Pàg.

Resum de Treball Final de Carrera
Enginyeria Tècnica d'Informàtica de Gestió i Sistemes

Títol: Teclat virtual amb predicció del llenguatge
Paraules clau: teclat virtual, predicció de paraules, gramàtica catalana, multi-plataforma
Autor: Eloi Creus Casassas
Direcció: Joan Vancells i Flotats
Data: Febrer de 2012

Resum

L'objectiu de la realització d'aquest treball és la creació d'un teclat virtual destinat a ajudar a persones amb mobilitat reduïda, que no poden utilitzar el teclat físic de l'ordinador, a escriure intentant aconseguir una velocitat d'escriptura raonable per a textos de qualsevol mida.

Per aconseguir aquesta velocitat d'escriptura raonable s'ha implementat un sistema de predicció del llenguatge que té dos aspectes. D'una banda es prediuen paraules segons la seva freqüència d'ús en un determinat diccionari i, d'altra banda, es prediuen paraules seguint les regles d'escriptura de la gramàtica catalana.

Un altre aspecte important era que el programa creat es pogués utilitzar en diferents sistemes operatius ja que només hi havia versions específiques per a cada un d'ells. El programa creat es pot executar en els sistemes operatius Windows XP, Mac OS i Ubuntu Linux.

El programa creat pretén ser una base per a posteriors millores i ampliacions en diferents parts del seu conjunt. No obstant això, com a resultat s'ha obtingut un programa que permet escriure raonablement ràpid i permet a l'usuari gestionar diccionaris i els dos tipus de predicció que s'han implementat.

Summary of Career Final Work
Technical Engineering of Computer Science

Title: Virtual keyboard with language prediction
Key words: virtual keyboard, word prediction, Catalan grammar, cross-platform
Author: Eloi Creus Casassas
Direction: Joan Vancells i Flotats
Date: February of 2012

Summary

The aim of this work is to create a virtual keyboard designed to help disabled people who can not use the physical keyboard of the computer trying to write in a reasonable typing speed for any size text.

In order to obtain this reasonable typing speed has been implemented a prediction language system that has two aspects. On one hand words are predicted by their frequency of use in a given dictionary, on the other hand, words are predicted according to the rules of the writing grammar in Catalan.

Another important aspect was that the created program could be used on different operating systems due that there were only specific versions for each one. The program created can be run on Windows XP, Mac OS and Ubuntu Linux operating systems.

The program created is the first step for future improvements and updates in the different parts of it. However, as a result we have got a program that allow to write reasonably fast and allows the user to manage dictionaries and the two types of prediction that have been implemented.

1. Introducció

El treball ha consistit en la creació d'un programa que simula la funció del teclat físic de l'ordinador permetent controlar i escriure a l'ordinador amb qualsevol dispositiu de control del cursor.

La finalitat del programa era ajudar a persones amb mobilitat reduïda, que no poden utilitzar el teclat físic de l'ordinador, a escriure textos de qualsevol format o mida intentant aconseguir una velocitat d'entrada de text raonable.

Per tant, s'ha creat un sistema de predicció de l'escriptura. Aquest sistema ha consistit en dues parts:
Una primera part ha estat la creació d'un sistema que ens permet, segons la freqüència de les paraules entrades, donar opcions d'entrada a l'usuari de tal manera que se'ns indiquen primer les paraules que han aparegut més vegades segons un diccionari.
La segona part d'aquest sistema ha estat la predicció segons les regles del llenguatge català. Aquesta segona part ens mostra opcions d'entrada seguint la gramàtica catalana. A causa de la complexitat d'aquesta part el sistema pretén ser una primera base per a posteriors ampliacions i millores, i es limita als aspectes més bàsics de la gramàtica catalana.

Un altre aspecte fonamental d'aquest treball ha estat l'adaptació del programa a diferents sistemes operatius, de tal manera que el programa és multi-plataforma. La motivació d'aquest aspecte l'ha provocat el fet que no existia cap teclat virtual que es pogués executar en diferents sistemes i cada un d'ells era específic a una plataforma concreta. Concretament s'ha implementat el programa de manera completa a les plataformes Windows i MacOS, i de forma parcial a Linux.

2. Objectius i metodologia

Com ja s'ha dit, els objectius de la realització del treball eren els següents:

- Creació d'un programa que permeti escriure, amb una velocitat raonable, sense utilitzar el teclat físic de l'ordinador per ajudar a persones amb mobilitat reduïda.
- Implementació d'un sistema de predicció del llenguatge que d'una banda predigui segons la freqüència d'ús de les paraules i, de l'altra, permeti escriure seguint la gramàtica catalana.
- Adaptació del programa perquè pugui executar-se en diferents sistemes operatius. La idea era assegurar un sistema operatiu i després mirar d'adaptar el programa a altres.

La metodologia seguida ha consistit en investigar com es podia fer cada part del programa i anar implementant cadascuna de les parts resolent els problemes que anaven sorgint durant la implementació. El procés d'implementació del programa ha estat un aprenentatge constant de com es podia fer el que es volia. Al principi no es sabia com realitzar diferents aspectes fonamentals del programa:

- Simular el teclat físic per poder escriure en qualsevol aplicació.
- En la part de predicció amb freqüència de paraules es va haver d'investigar les millors formes per guardar les paraules de manera que la seva cerca fos eficient en temps.
- En la part de la predicció segons la gramàtica catalana es va haver de consultar quins aspectes podrien ajudar a escriure amb rapidesa. Es van haver de definir quines coses podrien implementar-se i quines coses no eren factibles.

Per a la implementació del programa es va decidir utilitzar el llenguatge de programació **Java.** La propietat del llenguatge que es va tenir en compte per decidir-se per aquest llenguatge va ser la seva capacitat d'executar-se en diferents sistemes operatius. Per tant, la part principal del programa s'ha implementat amb aquest llenguatge.

Per a la part específica de cada sistema operatiu, que es comunica amb la part principal del programa i permet poder escriure en qualsevol programa, es va veure que s'havia d'utilitzar el llenguatge natiu de cada sistema operatiu. Concretament **C,** per a Windows i Linux, i **Objective-C** en el cas del MacOS.

Per unir la part específica del sistema operatiu i la part principal realitzada en Java, es va veure que seria necessari utilitzar les llibreries **JNI (Java Native Interface)** que permeten executar codi natiu des de Java.

Per a la realització de la predicció segons la gramàtica catalana s'ha utilitzat la base de dades **Cercamots,** gràcies a la col·laboració d'Eumo, que consta d'un gran grup de paraules catalanes i ens indica el tipus gramatical que tenen.

Per la part de base de dades del programa, que s'hauria d'encarregar de guardar els diccionaris de paraules que crearà l'usuari i que s'utilitzarien en la predicció per freqüència, es va decidir utilitzar el sistema de bases de dades **MySQL** ja que ens ofereix la característica que és multi-plataforma. Aquesta base de dades també inclouria la part utilitzada del Cercamots que estava feta amb aquest mateix sistema de base de dades.

Abans de la implementació del programa, i sobretot durant aquesta, es va haver d'aprendre el llenguatge de programació Objective-C que era desconegut abans de la implementació del programa. Un altre aspecte desconegut per a mi eren les llibreries JNI de les quals també se'n ha après la seva utilització i

funcionament. El llenguatge de programació Java ja era conegut però, tot i així, també s'han hagut d'aprendre nous conceptes sobretot per a la creació de la interfície gràfica i la gestió d'esdeveniments per crear el teclat virtual que permet escriure.

En els apartats següents s'explicaran tots els passos seguits per arribar al programa final, amb els problemes que es van presentar en cada un d'ells i com es van resoldre.

3. Creació del programa

En aquest apartat principal de la memòria del projecte s'aniran explicant tots els aspectes del programa realitzats per tal d'aconseguir els objectius explicats anteriorment. Primer de tot, s'explicarà la idea que es tenia de com hauria de ser el programa abans de la seva implementació per poder complir els objectius. En segon lloc, s'explicaran cada un dels passos seguits durant la realització del programa, mostrant els aspectes principals d'ell i com funcionen. Finalment es mostrarà l'estructura final del programa des d'un punt de vista més proper a l'usuari i la interacció amb el programa.

3.1. Idea general

La idea de programa que s'ha seguit durant tota la implementació i que es tenia abans de la implementació es pot resumir en els següents tres punts:

1. Una interfície gràfica que consisteix en dues finestres:
   Una primera finestra representa un teclat a la pantalla en el qual els usuaris seleccionen les tecles amb el cursor com si estiguessin escrivint amb el teclat físic. En aquesta finestra també hi ha una caixa de text que permet veure als usuaris el que estan escrivint. Aquest últim component es deu al fet que mentre estem seleccionant les tecles a la finestra del programa no veiem el programa on escrivim i així permetem veure-ho.
   Una segona finestra s'utilitza per mostrar les paraules que es van predient, permetent la selecció, creació i eliminació de diccionaris, i controlar el tipus de predicció que volem utilitzar.

2. Que el programa a partir de les tecles que va prement l'usuari envii un flux d'esdeveniments. Aquests esdeveniments són rebuts pel sistema de predicció que va generant i mostrant les paraules que podem escriure segons les lletres que hem entrat. Aquests esdeveniments, així mateix,

també produeixen que haguem de cridar el codi natiu del sistema i li enviem el text de la tecla que ha entrat l'usuari. Aquest codi s'encarrega de que el text el rebi l'aplicació que està activa en aquell moment.

3. Una base de dades que d'una banda té la base de dades Cercamots amb tota la col·lecció de paraules i les seves categories gramaticals per a la predicció amb la gramàtica catalana. D'altra banda, per a la predicció per freqüència, es guarden tots els diccionaris que pot decidir l'usuari juntament amb les paraules que contenen cadascun d'aquests diccionaris i les freqüències d'ús d'elles. En el primer cas, la base de dades serveix per guardar les paraules però també per a realitzar les consultes per indicar a l'usuari les paraules que pot escriure durant la predicció. En el segon cas, només l'utilitzem per guardar les paraules per a la predicció per freqüència i en el moment de la predicció es gestionen les paraules amb un arbre a la memòria RAM sense utilitzar la base de dades.

3.2. Procés de creació

En aquest apartat s'explicaran els passos seguits per a la realització del programa i s'explicarà des d'un punt de vista intern, és a dir, entrant en els punts de codi i aspectes més importants de cadascuna de les parts realitzades.

Abans d'entrar en cada un d'aquests aspectes l'evolució del programa ha seguit els següents passos:

1. Creació de la primera pantalla que conté el teclat per poder escriure i implementació de la lògica dels esdeveniments del teclat.

2. Implementació del codi natiu per a cada un dels sistemes operatius en el qual funciona el teclat.

3. Implementació de les estructures de dades utilitzades per a la predicció per freqüència de paraules.

4. Creació de la base de dades integrant el Cercamots i la gestió de diccionaris amb les seves paraules.

5. Creació de la segona pantalla que mostra a l'usuari les paraules de la predicció. Paral·lelament, implementació de les classes destinades als diccionaris i al sistema de predicció per freqüències.

6. Implementació del sistema de predicció seguint la gramàtica catalana estenent el sistema de predicció existent.

3.2.1. El teclat virtual

El primer pas per a la creació del programa va ser crear la classe **MainWindow** que conté el teclat virtual i una caixa de text perquè es pugui veure el que s'està escrivint en l'aplicació on es treballa.

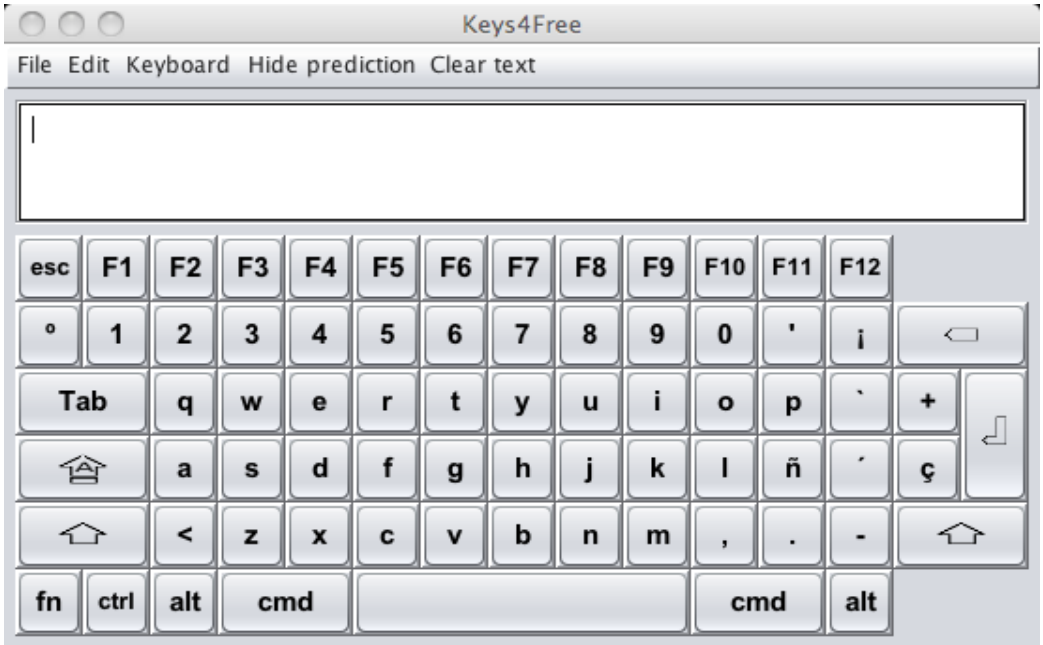

Figura 3.2.1.1 Imatge de la pantalla MainWindow

La classe anterior serà explicada més endavant ja que intervé poc en el funcionament del teclat virtual. L'única cosa que fa és contenir una instància de la classe **BasicKeyboard** que hereta de la classe **AbstractKeybord,** que és una classe abstracta que defineix propietats comunes dels teclats. En el moment de la creació del teclat la classe principal indica la caixa de text on ha d'escriure el teclat.

Centrant-nos ja en l'explicació d'aquest apartat dedicat al teclat virtual la primera classe que es va crear va ser la classe abstracta **AbstractKeybord.** En ella hi ha definides unes propietats que haurien de complir els diferents teclats que es creïn en el programa. Aquesta classe hereta de **JPanel** perquè es pugui mostrar gràficament a la interfície del programa.

La classe conté instàncies de la classe **KeyboardKey** que representen les tecles modificadores del teclat com Shift, Alt, etc i boleans que indiquen si aquestes tecles estan actives o no.
Aquesta classe també conté una instància de **JTextArea** que conté la caixa de text on escriuran les tecles que conté.
La classe té un boleà **dwellEnabled** que indica si les tecles del teclat es poden usar en lloc de fent clic amb el ratolí, mantenint el cursor sobre la tecla durant un període de temps, la qual cosa permet millorar la comoditat a l'usuari.

La classe **AbstractKeybord** s'encarrega de cridar el codi natiu del sistema operatiu on s'està executant el programa per enviar-li el text de la tecla a escriure o els caràcters Space, Enter, Backspace o Tab que com ja veurem funcionen de forma diferent. Això es realitza amb les funcions **public native void setKeyAtComponent (String keyTyped),** i **public native void setTextModifier (String modifier),** que estan implementades depenent de cada sistema operatiu.

Com que la classe anterior és abstracta i no la podem instanciar, la classe que usa les seves propietats és la classe **BasicKeyboard** que és la que conté totes les tecles del teclat que es mostra a **MainWindow.**

En aquesta classe hi ha contingudes totes les tecles en forma d'instàncies de la classe **KeyboardKey.** Durant la inicialització de la classe s'indiquen les tecles que faran de modificadores definides en la superclasse **AbstractKeyboard.** A més, s'indica a les tecles el teclat on estan i es registren els esdeveniments als quals han de respondre les tecles. A les tecles creades se les inicialitza segons el tipus de tecla que són.

Les tecles responen a tres tipus d'esdeveniments: dos que s'han creat i un d'estàndard que detecta quan interaccionem amb el cursor. Un primer esdeveniment creat que defineix la classe **ModifierEvent** permet a les tecles saber en quin estat estan, és a dir, quin caràcter han de mostrar i escriure depenent dels modificadors que estiguin activats en el teclat. Un segon esdeveniment creat es diu **TextAdjustEvent** i s'encarrega de canviar la mida del text del teclat depenent de la seva mida.

La classe **BasicKeyboard** s'encarrega de llançar aquest últim tipus d'esdeveniment quan detecta que ha canviat la seva mida. Com que està registrat a totes les tecles que conté, aquestes executen la funció que els canvia la mida de la lletra o de la icona en cas que en tinguin.

Per crear aquest esdeveniment intervé la classe **TextAdjustEvent** que hereta de **EventObject**, de manera que funciona com un esdeveniment estàndard. En aquest cas només necessitem el constructor per defecte. Les classes que vulguin utilitzar aquest esdeveniment han de implementar la interfície **TextAdjustEventListener** que defineix la funció que han de realitzar quan es llença un esdeveniment d'aquest tipus. En aquest cas la classe **KeyboardKey**

haurà d'implementar la funció **public void adjustText ();**. La interfície **TextAdjustEventListener** hereta de la classe estàndard **EventListener**, que utilitza el llenguatge per a definir les funcions que han d'implementar les classes per rebre els esdeveniments definits.

La funció de la classe **BasicKeyboard** com ja s'ha dit és llançar aquest esdeveniment. Per fer-ho es guarda les tecles que han de rebre l'esdeveniment, en aquest cas totes les que conté, en una llista de classe estàndard **EventListenerList**. S'ha de definir un mètode per afegir elements a aquesta llista que són els que rebran l'esdeveniment llençat. Quan es detecti un canvi de mida al teclat s'executarà la funció **protected synchronized void fireTextAdjustEvent (TextAdjustEvent i)**, que s'encarrega d'anar iterant per la llista i fer executar la funció corresponent a l'esdeveniment. En aquest cas concret farà executar la funció **adjustText ()**, que haurà d'implementar la classe **KeyboardKey**.

La funcionalitat més important la implementa la classe **KeyboardKey**. Aquesta classe s'encarrega de gestionar les interaccions amb el cursor, modificar l'estat de les tecles que formen el teclat depenent dels modificadors i, depenent del tipus de tecla, realitzar la funció corresponent tant a la caixa de text com en l'aplicació que té el cursor del sistema operatiu.

En aquest punt de l'explicació és important dir que s'ha limitat la funcionalitat de les tecles perquè només escriguin text. No s'han implementat les combinacions de tecles per a realitzar comandes ni les opcions F. Aquests aspectes queden per a posteriors millores ja que, per realitzar la part de predicció, amb les tecles per manejar text era suficient.

Per a la interacció amb el cursor, com ja s'ha dit anteriorment, tenim dues opcions per utilitzar una tecla. La primera és fer clic a la tecla en qüestió i la segona és mantenir el cursor sobre una tecla durant un període determinat de temps. En ambdós casos, quan es detecta que s'ha de fer l'acció d'escriure s'executa la funció **public void typeOperation (boolean isDwell)** que s'explicarà més tard.

Per implementar això s'utilitza una instància de la classe **MouseAdapter** que es registra en el moment de la creació de la classe **BasicKeyboard** perquè les tecles rebin els esdeveniments del cursor. L'únic que fa l'adaptador del cursor és implementar les funcions **keyExited**, **keyEntered** i **keyPressed** que permeten saber quan hem entrat dins la tecla, quan sortim d'aquesta i quan l'hem premut amb el ratolí. En aquest últim cas només hem de cridar la funció **typeOperation (false)** per indicar que hem premut fent clic.

Per poder implementar la pulsació de tecla mantenint el cursor sobre d'ella s'ha implementat la classe **DwellThread**. La idea és que quan entrem en una tecla amb el cursor es creï un fil d'execució que esperi un determinat període de temps, i quan acabi l'espera miri si el cursor ha sortit de la tecla o no. Si després de l'espera encara no hem sortit, escriurem la lletra corresponent i en cas contrari no escriurem res. Cada tecla té una única instància de **DwellThread** de manera que només es pot executar un fil per tecla en un moment determinat guanyant en eficiència.

La classe **DwellThread** només hereta de la classe estàndard **Thread** i defineix l'operació **run ()** que s'executa quan llancem el fil d'execució. La classe defineix els mil·lisegons que hauran d'esperar totes les instàncies creades abans de decidir si hem d'escriure. Quan creem cada fil indiquem a quina tecla està associat per poder executar la funció d'escriure on correspon. La funció **keyExited** canvia el boleà **buttonExited** a cert si sortim de la tecla amb un fil

corrent per indicar-li que no ha d'escriure. L'opció de mantenir el cursor es pot activar o desactivar, de manera que tot el que s'ha explicat d'aquesta opció no passa si no ho tenim activat.

Figura 3.2.1.2. Tipus de tecles del teclat virtual

Les instàncies de la classe **KeyboardKey** tenen associat un tipus definit a la classe **KeyType.** Depenent del tipus de tecla l'operació **typeOperation** s'executa d'una manera o altra. També l'utilitzem per saber què hem de fer quan es produeix un esdeveniment **ModifierEvent** en cadascuna de les tecles.

Com es pot veure a la Figura 5.2.2.3.1 de color taronja hi ha les tecles de tipus OPTION, de color verd les VARCHARACTER, de color ocre les TEXTMODIFIER, de color blau les MODIFIER i de color gris les CHARACTER.

Abans d'entrar a explicar com funciona la funció **typeOperation** que realitza l'acció de cadascuna de les tecles, cal dir que en el moment de creació d'una tecla de tipus CHARACTER o VARCHARACTER se li associen els caràcters que ha d'escriure depenent de l'estat dels modificadors del teclat, amb la funció **setCharacters.** Per exemple, la tecla ç té associats ç, Ç i }. Això ho guarda en l'atribut **chars**.

Per poder explicar com funciona l'operació **typeOperation** necessitem parlar de la classe **ModifierEvent**. Aquesta classe que s'ha anomenat anteriorment ens permet implementar els anomenats esdeveniments de modificador. Les tecles implementen les funcions de la interfície **ModifierEventListener** que defineixen què ha de fer cada tecla quan es prem un modificador. Bàsicament el que ha de fer la tecla és canviar el text que es veu al teclat, que és el que s'escriurà quan es premi la tecla.

Una altra classe utilitzada aquí és la classe **ModifierType** que indica els tipus de modificadors que generen esdeveniments **ModifierEvent**. Aquests poden ser de tipus CAPS_LOCK, SHIFT, OPENACCENT, etc. Quan llancem un esdeveniment **ModifierEvent** des de la classe **KeyboardKey** indiquem de quin tipus és i si s'està activant o desactivant el seu estat. El mètode que llença els esdeveniments és **protected synchronized void fireModifierEvent (ModifierEvent i)** que, com en el cas dels esdeveniments de canviar la mida del text de les tecles, va iterant per les tecles que tenen registrat l'esdeveniment i executa la funció corresponent de la interfície **ModifierEventListener** depenent del que ens indica l'esdeveniment creat.

La funció de llançar l'esdeveniment només l'executen les tecles modificadores juntament amb els accents i la dièresi. Durant la creació del **BasicKeyboard** aquestes tecles guarden en una llista les tecles que seran afectades pel seu estat. Per exemple, quan es produeixi un esdeveniment CAPS_LOCK les tecles de tipus CHARACTER executen la funció **onPressCapsLock** o **onUnpressCapsLock** depenent de si **CapsLock** estava actiu o no.

Per finalitzar aquest apartat on estem parlant del teclat virtual, s'explicarà el funcionament de la funció **typeOperation**. Aquesta funció, com s'ha dit, depenent del tipus de tecla que hem premut executa l'acció que li correspon.

En el cas que la tecla sigui de tipus **MODIFIER** la funció s'encarrega de veure quin tipus de modificador ha produït la crida a la funció. El programa només ha implementat els esdeveniments produïts per **CapsLock**, **Shift** i **Alt** que llancen el seu tipus d'esdeveniment associat quan executen aquesta funció amb el mètode updateModifier, que també actualitza l'estat lògic del teclat i també gràficament. En els modificadors que no tenen implementat cap esdeveniment (**Function**, **Control i Comand**) s'actualitza l'estat, però sense generar cap esdeveniment. Com s'ha explicat anteriorment, quan es produeix un esdeveniment d'aquest tipus l'estat gràfic del teclat mostra els caràcters que es poden escriure en cadascuna de les tecles segons l'estat dels modificadors.

El segon cas amb el qual ens podem trobar és que la tecla sigui del tipus **TEXTMODIFIER,** és a dir, les tecles **Tab, Backspace, Enter** o **Space**. Totes aquestes tecles envien el seu caràcter corresponent a la caixa de text de la pantalla principal menys en el cas de la tecla **Backspace** que elimina un caràcter d'aquesta caixa. Però el més important que fan és cridar a la funció nativa que està definida en el teclat **keyboard.setTextModifier (KeyName)**, que s'encarrega de gestionar aquest tipus de tecles.

En el cas que la tecla sigui de tipus **OPTION,** com ja s'ha dit anteriorment, no s'ha implementat la seva funcionalitat en el programa.

Finalment considerem el cas en què la tecla sigui de tipus **CHARACTER** o **VARCHARACTER.** Dins d'aquesta operació aquests dos tipus es comporten exactament igual. En aquest cas s'han de detectar els accents i la dièresi per saber si hem de llançar l'esdeveniment que els correspon de la mateixa manera que ho feien les tecles **MODIFIER**

En el cas més habitual doncs, depenent de l'estat del teclat, a part d'escriure el caràcter s'han de llançar els esdeveniments que desactiven els modificadors en el cas de Shift i Alt, o que desactiven els accents o la dièresi utilitzant la funció proccessAccentTyping en aquest últim cas. De qualsevol manera, s'envia a la caixa de text el caràcter que s'estava mostrant gràficament en el teclat virtual ja que els esdeveniments de modificador ja s'encarreguen que es mostri sempre el caràcter que s'escriurà. El caràcter també s'envia a l'aplicació activa utilitzant el mètode natiu **keyboard.setKeyAtComponent (key);**.

Per la realització de la implementació del teclat virtual es va aprendre com implementar esdeveniments propis amb Java que eren desconeguts abans de la realització del projecte. Un altre aspecte descobert va ser com organitzar els botons (tecles) dins de la interfície gràfica amb **GridBagLayout**.

3.2.2. Part nativa del sistema operatiu

En aquest apartat s'explicarà com s'ha implementat el codi natiu per cada sistema operatiu en què s'executa el programa. El programa pot funcionar en els sistemes operatius Windows XP, MacOS X 10.6 i Ubuntu Linux 10.8, en altres no s'ha provat el seu funcionament. En els dos primers el programa pot funcionar totalment, encara que en el segon no es pot utilitzar la pulsació mantenint el cursor sobre la tecla menys si s'escriu només a la caixa de text per una limitació de la implementació de Java del Mac. En el sistema operatiu Ubuntu només es poden utilitzar les tecles amb lletres i en minúscula així com els modificadors de text, ja que la implementació total no ha estat possible a causa de falta d'informació.

El codi natiu implementat s'encarrega de donar la funcionalitat a les funcions natives de la classe **AbstractKeyboard, setTextModifier** i **setKeyAtComponent,** que s'encarreguen d'enviar tant el modificador de text corresponent com el caràcter o paraula que s'ha d'escriure en l'aplicació activa.

Per poder executar funcions de codi natiu des de **Java** s'han d'utilitzar les llibreries **JNI** (Java Native Interface). Aquestes llibreries ens permeten que puguem executar funcions implementades amb el codi natiu del sistema en el qual estem executant el programa. En cada un dels sistemes operatius s'ha hagut de crear una llibreria dinàmica que el programa carrega al principi segons el sistema operatiu i que implementa aquestes funcions natives. Les llibreries JNI ens permeten intercanviar informació de les variables entre els dos llenguatges, Java i el natiu. La classe **MainWindow** s'encarrega de carregar la llibreria corresponent al principi, quan s'inicia el programa. La llibreria creada amb el codi natiu ha d'estar en el PATH de llibreries de Java perquè el programa pugui funcionar escrivint en l'aplicació desitjada.

En els tres sistemes operatius es va haver d'afrontar el problema que quan es polsava una tecla a la pantalla principal del programa, el focus del sistema operatiu se li assignava impedint que el caràcter s'escrivís també a l'aplicació activa amb el cursor del sistema operatiu, ja que aquesta perdia el focus. En els sistemes operatius Windows i Linux això s'ha pogut solucionar fàcilment desactivant la propietat de la **MainWindow,** heretada de **JFrame, focusableWindowState** la qual cosa ja ens impedeix que el focus es canvii de l'aplicació on volem escriure al teclat virtual. En el cas de Mac s'ha hagut de implementar una funció nativa que ens posa la finestra **MainWindow** en un estat en el qual no es canvii el focus en utilitzar algun component d'ella.

3.2.2.1. Implementació nativa per Windows

En aquest primer cas s'ha hagut d'implementar una llibreria anomenada libWindows.dll de tipus dinàmic. Aquesta llibreria conté un fitxer de capçalera **NativeIssue.h** i la implementació dels mètodes en el fitxer **NativeIssue.c**.

El llenguatge utilitzat per la implementació en Windows ha estat C i s'han utilitzat algunes funcions de la llibreria **windows.h** que és molt utilitzada per programar aplicacions en aquest sistema operatiu. Com ja s'ha dit anteriorment s'utilitza la llibreria JNI que en aquest sistema operatiu es troba en el fitxer de capçalera **jni.h**.

Per poder cridar la funció nativa corresponent aquesta ha de tenir un nom que identifiqui des de quin paquet de Java i des de quina classe serà cridada. Això s'ha de complir per tots els sistemes operatius i concretament han de tenir els següents noms:

JNIEXPORT void JNICALL
Java_keysforfree_AbstracrKeyboard_setKeyAtComponent
(JNIEnv *, jobject, jstring)

JNIEXPORT void JNICALL
Java_keysforfree_AbstracrKeyboard_setTextModifier
(JNIEnv *, jobject, jstring)

Els dos primers paràmetres són objectes de la llibreria JNI que en el cas del primer apunta a un grup de funcions d'aquestes llibreries i en el cas del segon representa l'objecte de Java que ha cridat la funció. El tercer paràmetre és el caràcter o paraula que volem enviar a l'aplicació activa del sistema o el tipus de modificador.

En les dues opcions anteriors, en aquest sistema operatiu, utilitzem la funció anomenada **getFocusedWindow()**, que s'ha implementat i té la funció d'obtenir el component gràfic del sistema que té el cursor d'escriptura.

Per fer-ho primer de tot obtenim la pantalla de l'aplicació activa de Windows amb la funció **GetForegroundWindow()** definida a les llibreries de Windows. Un cop tenim la pantalla de l'aplicació activa hem d'obtenir el procés de l'aplicació on volem escriure amb la funció **GetWindowThreadProcessId** que ens diu el fil d'execució que ha creat la pantalla que ens interessa. Un cop tenim això hem de connectar l'entrada del procés Java amb l'entrada del procés de la finestra on volem escriure. D'aquesta manera podrem obtenir el component que té el cursor d'escriptura.

Un cop hem obtingut el component amb el cursor d'escriptura amb la funció **getFocusedWindow(),** el codi transforma el caràcter o paraula o nom de modificador que li hem passat des de Java a un tipus de dades que pugui entendre el llenguatge natiu, en aquest cas un apuntador a **const TCHAR**.

En el cas de la funció **setTextModifier** un cop tenim el nom del modificador de text representat nativament hem de mirar quin tipus de modificador del text és i assignar-li el codi virtual de tecla que té definit a Windows per al modificador de text corresponent. Un cop sabem el codi virtual de modificador de text hem d'enviar al component actiu de l'aplicació activa els missatges **WM_KEYDOWN**, **WM_CHAR** i **WM_KEKUP** amb el codi virtual obtingut, per simular la pulsació de la tecla en el component.

En el cas de la funció **setKeyAtComponent** primer de tot hem de transformar el caràcter o paraula rebut des de Java a una representació, com ja s'ha dit, que entengui el llenguatge natiu. Un cop tenim aquesta representació el que fa aquesta funció és anar obtenint els codis virtuals dels caràcters enviats. Per passar cada caràcter al seu codi virtual utilitzem la funció **VkKeyScan.** Per cada un dels caràcters hem d'enviar al component actiu de l'aplicació activa els missatges **WM_KEYDOWN, WM_CHAR** i **WM_KEKUP** amb el codi virtual obtingut per simular la pulsació de cada un d'ells.

Vull fer notar que en aquest codi cada vegada que fem servir una tecla del teclat virtual hem de buscar el component actiu de l'aplicació activa, encara que aquest no hagi canviat. Una millora que es podria fer en aquest apartat seria realitzar un programa de Windows que controlés els canvis de finestres actives i fos aquesta aplicació la que informés al teclat virtual perquè canviés el component on escriure. En aquest cas s'utilitzaria una part més complexa de JNI que permet crear la màquina virtual de Java des del programa de Windows. En aquesta màquina virtual s'executaria la part de Java amb el teclat virtual, i quan l'aplicació Windows detectés un canvi de finestra ho comunicaria al teclat executant una funció Java des de C, de forma contrària al que es fa ara.

3.2.2.2. Implementació nativa per Linux

En el cas de Linux s'ha hagut de implementar una llibreria dinàmica anomenada libLinux.so. Aquesta llibreria conté un fitxer de capçalera **NativeIssue.h** i la implementació dels mètodes en el fitxer **NativeIssue.c**.

Les funcions que s'implementen, com el cas de Windows, tenen una part comuna i segueixen una estructura semblant ja que es tracta d'obtenir el component amb el cursor d'escriptura de la pantalla activa i simular la pulsació dels caràcters o el codi de modificador.

En aquest sistema operatiu les llibreries JNI es troben en el fitxer de capçalera jni.h. En aquest sistema operatiu el gestor de finestres la realitza el sistema X Window System o X11 i per tant s'han hagut d'utilitzar fitxers de capçalera d'aquest sistema, concretament els fitxers X11/Xlib.h i X11/keysym.

Amb aquest sistema de finestres hi ha un servidor anomenat **X Server** que pot rebre connexions de diferents clients o monitors i s'encarrega de gestionar-los. El que hem de fer des del codi és connectar-nos a aquest servidor per poder obtenir la informació de la nostra pantalla, és a dir, l'aplicació que té el cursor

en aquesta pantalla o monitor.

La funció utilitzada per connectar-nos s'anomena **XOpenDisplay (0)** i ens retorna una estructura de tipus **Display**. Un cop tenim aquesta estructura podem obtenir la pantalla arrel de l'aplicació activa amb la funció **XdefaultRootWindow(display).** Amb aquesta variable que indica la pantalla arrel podem obtenir la pantalla amb el focus d'escriptura utilitzant la funció **XGetInputFocus(display, & winFocus, & state)**. Un cop hem obtingut la finestra amb el cursor les dues funcions transformen el paràmetre que s'ha passat des de Java a la representació nativa.

En el cas de la funció **setTextModifier** també obtenim la representació en X11 depenent del tipus de modificador. El que obtenim és el tipus KeySym que representa el nom que té cada tecla en el sistema. Amb aquest KeySym podrem obtenir el codi virtual de la tecla.

En el cas de la funció **setKeyAtComponent** cada caràcter que volem escriure ja representa el KeySym.

En els dos casos, per cada KeySym obtingut hem de generar dos esdeveniments que simulin la pulsació d'una determinada tecla en el component que té el cursor d'escriptura. Hem de crear un esdeveniment pel **KeyPress** i un altre pel **KeyRelease**.

El primer que hem de fer és construir els esdeveniments que s'enviaran. Per això s'ha implementat la funció **createKeyEvent**. Aquesta funció permet construir l'esdeveniment que s'enviarà definint una sèrie d'informacions. Les més importants serien la pantalla al qual va dirigit, la pantalla amb el cursor, la pantalla arrel, si és press o release i sobretot el KeySym, que dins d'aquesta funció es transforma en el codi virtual corresponent utilitzant

XkeysymToKeycode.

Un cop hem creat l'esdeveniment hem d'enviar-lo al servidor utilitzant la funció **XSendEvent**. Hi indiquem la pantalla on s'ha generat, la pantalla amb el cursor on s'ha d'escriure, i hem d'indicar quin tipus d'esdeveniment ha d'esperar el servidor que en aquest cas serà **KeyPressMask** o **KeyReleaseMask**. També s'indica tota l'estructura de l'esdeveniment creat .

Com ja s'ha comentat, en aquest sistema operatiu la funcionalitat del teclat virtual és força limitada ja que amb el codi explicat només podem escriure lletres i en minúscula. No s'ha pogut trobar suficient informació per poder escriure qualsevol caràcter i el poc que s'ha pogut trobar indica que cada ordinador en aquest sistema pot tenir diferents representacions dels caràcters. En aquest punt es va optar per treballar en els altres sistemes en els quals el funcionament era millor i centrar-nos en la part de predicció, per tant, això queda com una millora posterior a realitzar.

3.2.2.3. Implementació nativa per Mac

En aquest últim sistema operatiu la manera d'implementar les funcions va variar una mica dels anteriors. S'ha hagut d'implementar una llibreria dinàmica anomenada libMacintosh.jnilib que implementa les mateixes funcions que els anteriors sistemes operatius. A més s'ha implementat l'aspecte propi ja comentat de posar les finestres del teclat virtual en un estat en el qual aquestes no agafin el cursor quan escrivim, impedint escriure en l'aplicació activa.

El llenguatge de programació d'aquest sistema operatiu és Objective-C, per la qual cosa els fitxers implementats en aquest cas es diuen, el de capçalera **NativeIssue.h** i el de la implementació **NativeIssue.m**.

Per desenvolupar aplicacions utilitzant les llibreries JNI en aquest sistema operatiu es necessita el fitxer de capçalera **JavaNativeFoundation / JavaNativeFoundation.h**. En aquest fitxer hi ha definides funcions per interactuar entre JNI i **Cocoa**, que conté gran part de les llibreries per programar en Mac.

Primer de tot explicaré les funcions comunes amb els altres sistemes operatius. Com ja és habitual obtenim el paràmetre de Java en una representació que pot entendre el sistema natiu. En el cas de la funció **setTextModifier** hem d'obtenir el codi del modificador de text corresponent. Per **setKeyAtComponent** no necessitem obtenir aquest codi ja que podem enviar el caràcter directament.

En ambdós casos s'ha de crear un esdeveniment per simular quan premem la tecla i quan la deixem. Per fer-ho fem servir la funció **CGEventCreateKeyboardEvent**.

Per **setTextModifier** hem de crear aquest esdeveniment amb el codi de modificador que hem obtingut i després enviem al sistema els dos esdeveniments utilitzant la funció **CGEventPost**. El que fa aquesta funció és comunicar al sistema que s'ha premut una determinada tecla i aquest envia l'esdeveniment a l'aplicació activa, de manera que no hem d'obtenir la finestra activa com fèiem en els altres sistemes operatius.

En la funció **setKeyAtComponent** per cada caràcter que hem d'enviar executem la funció **simulateKeyStroke** que s'ha implementat i que envia directament el caràcter sense obtenir el codi.

L'aspecte més diferent en aquest sistema operatiu va ser com posar a les pantalles del teclat virtual en un estat en el qual aquestes no agafin el focus pels

motius ja explicats. Les pantalles que s'havien de posar en aquest estat eren la **MainWindow** que conté el teclat i la **PredictionWindow** de la qual encara no s'ha parlat, però conté una llista amb les paraules que es poden escriure segons la predicció. S'han hagut d'implementar les funcions natives que declaren aquestes dues classes que són les següents:

JNIEXPORT void JNICALL
Java_keysforfree_MainWindow_setWindowInactivated
(JNIEnv * jnienv, jobject object)

JNIEXPORT void JNICALL
Java_keysforfree_PredictorWindow_setWindowInactivated
(JNIEnv * jnienv, jobject object)

Com que les dues funcions realitzen el mateix, la implementació es troba en la funció **setWindowInactivated (JNIEnv * jnienv, jobject object)** que és cridada per les dues funcions anteriors. El que fa aquesta funció és obtenir la finestra del teclat virtual que li ha assignat el sistema operatiu i un cop tenim aquesta finestra posar-la en l'estat que ens interessa.

Per obtenir la finestra que se li ha assignat en el sistema operatiu s'ha implementat la funció **NSView * GetViewFromComponent (jobject object, JNIEnv * jnienv)** que ens retorna una instància de la classe **NSView** que representa la part lògica de la finestra. Per a aquesta funció necessitem la capçalera **JavaVM / jawt_md.h** que ens permet obtenir l'anomenada **DrawingSurface** de la pantalla del programa del projecte. Amb aquesta informació podem obtenir la **NSView** de la pantalla.

A partir de la **NSView** de la pantalla podem obtenir la **NSWindow** i indicar quin estil dels que defineix el sistema operatiu volem que tingui. Concretament

l'estil que ens interessa és **NSNonactivatingPanelMask**.

Aquesta funció, que posa cada una de les pantalles del programa en aquest estat, s'executa al principi del programa en el cas que l'estiguem executant en el sistema operatiu Mac.

3.2.2.4. Cloenda

Per la realització de la part nativa de cada sistema operatiu s'han hagut d'investigar les maneres d'implementar les funcions per enviar el text o el modificador de text que es volia escriure a l'aplicació activa.

En el cas de Windows ja es tenien alguns coneixements de com programar en aquest sistema, apresos durant la carrera. Tot i així s'han hagut de buscar les funcions que feien el que volíem i s'ha hagut d'aprofundir en l'aspecte d'entrada del teclat.

En el cas de Linux els coneixements que es tenien no eren aplicables a la programació que s'ha fet ja que no tenia relació amb el sistema **X Window**. Precisament s'ha hagut d'aprendre el funcionament d'aquest sistema gestor de finestres encara que la informació obtinguda sobre ell no ha estat molt gran. Com que X11 és comú en diferents distribucions de Linux, pot ser que el programa funcioni en un altre sistema Linux que no sigui Ubuntu.

En l'últim sistema operatiu, Mac OS X, s'ha hagut d'aprendre tot des de zero ja que no es tenia cap idea de programar en aquest sistema operatiu amb les llibreries Cocoa. A més, tampoc es coneixia el llenguatge de programació Objective-C que s'ha hagut d'estudiar una mica. Com que s'han hagut de realitzar les funcions específiques del focus de les pantalles, en aquest sistema s'ha hagut de treballar una mica més investigant com impedir aquest problema.

En tots els sistemes operatius es va haver d'aprendre a utilitzar les llibreries JNI per poder realitzar la funció del teclat virtual. S'ha après a cridar funcions natives des del codi de Java i s'ha vist que aquestes llibreries es poden utilitzar en ambdós sentits, és a dir, cridar codi natiu des de Java o en el sentit contrari.

Aquesta part específica per cada sistema operatiu ha servit per veure com es poden integrar diferents tecnologies per la realització d'un programa i els problemes en els quals ens podem trobar quan integrem diferents tecnologies.

3.2.3. Estructura de Predicció

Una vegada que ja s'havia implementat la funcionalitat del teclat virtual, tant la manera com les tecles realitzarien la seva funció, com la implementació del codi natiu per a cada un dels sistemes operatius, es va començar a mirar quina estructura de dades s'utilitzaria per guardar les paraules a predir de la forma més eficient possible per obtenir-les durant l'ús de l'usuari.

L'estructura de dades a utilitzar havia de permetre anar guardant les paraules que anava introduint l'usuari amb el teclat virtual i actualitzar les freqüències de les paraules que ja s'havien escrit anteriorment. A més, aquestes paraules havien de guardar-se de manera que donada la paraula que estava escrivint l'usuari es poguessin obtenir totes les paraules que començaven pel prefix que estava escrivint.

Aquestes paraules obtingudes a partir del prefix que estava escrivint l'usuari també havien de mostrar-se ordenades de major a menor freqüència d'aparició per facilitar la velocitat d'escriptura.

Primer de tot es pensava en implementar-ho amb un **arbre òptim de cerca** en el qual en les fulles més properes a l'arrel es situarien les paraules amb més freqüència i en les més llunyanes les de menor freqüència. Aquesta primera

opció va ser descartada per complet ja que d'una banda, és una estructura que ha de ser fixa ja que cada vegada que s'introduïa una paraula nova s'havia de tornar a generar l'arbre per actualitzar les freqüències, però sobretot perquè era impossible obtenir paraules que comencessin per un determinat prefix ja que es guardaven les paraules senceres i no pels caràcters que les formaven.

La següent estructura que es va pensar utilitzar va ser un **Trie** i finalment va ser l'estructura utilitzada però en la seva implementació amb un **arbre ternari de cerca**.

La idea d'un **Trie** consisteix en un arbre en el qual cada node representa un determinat prefix d'una paraula. En un nivell de l'arbre hi ha els caràcters que ocupen la posició nivell dins d'alguna paraula. Seguint les connexions d'aquest arbre des de l'arrel fins a les fulles es van formant les paraules que conté aquesta estructura. Les paraules que estan dins no sempre s'acaben a les fulles sinó que poden acabar-se en algun node intermedi. A la figura 5.2.3.1 hi ha representat un exemple que conté les paraules aire, aigua, pera, pere, poma i pe.

Amb aquesta estructura podem aconseguir el que buscàvem. Donat un prefix format per les lletres que escriu un usuari podem trobar fàcilment totes les paraules que comencen per aquell prefix ja que només hem d'agafar les paraules que queden en el subarbre sota del node de l'última lletra del prefix. Per exemple, si l'usuari ha escrit "pe" podem trobar les altres paraules que comencen així, pere i pera.

Aquesta estructura també ens permet guardar fàcilment les paraules que va escrivint l'usuari. A mesura que aquest vagi escrivint lletres d'una paraula es poden anar creant els nodes fins que acabi la paraula. D'aquesta manera, les noves paraules es van incorporant a l'estructura de dades. En cas que la paraula escrita ja existís, quan l'usuari acaba d'escriure la paraula només s'ha

d'incrementar en un el comptador que es pot posar als nodes que representen el final d'una paraula i que ens indica les vegades que s'ha utilitzat .

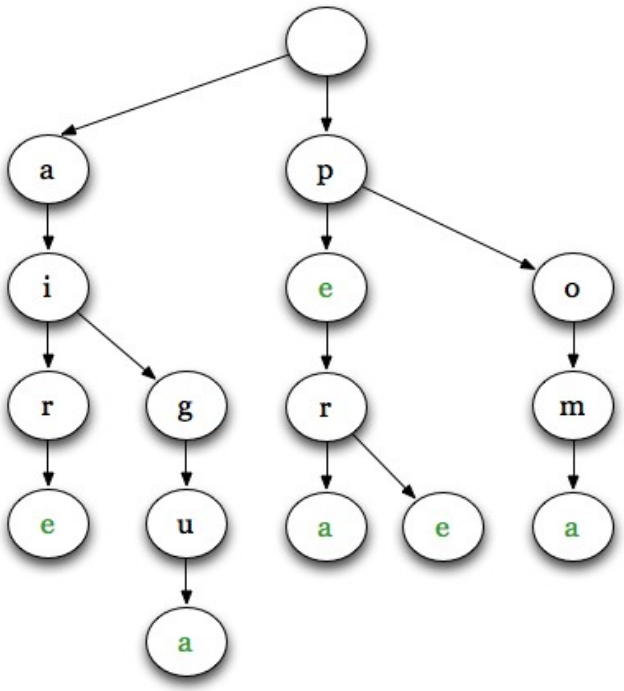

Figura 5.2.3.1. Representació d'un Trie

Per la implementació d'aquesta estructura de dades el primer que se'ns podria ocórrer és que cada node tingués una taula indexada per lletres en què cada posició fos un apuntador al següent node que contindria les paraules que comencessin per l'índex de la taula de la posició corresponent. De seguida es va veure que implementant d'aquesta forma s'utilitza molt espai.

En les referències d'aquest apartat es va trobar una implementació molt més adequada i eficient. Aquesta implementació, que és la que s'ha utilitzat, es diu **arbre ternari de cerca**. Aquesta estructura de dades combina el millor dels arbres binaris de cerca i els tries.

Com el seu nom indica aquest arbre és ternari de manera que cada node té tres fills. Des d'ara considerem el fill esquerre, el fill dret i el fill del mig.
Al fill del mig hi ha el node que conté la següent lletra de la paraula, de forma semblant a tots els apuntadors en el cas d'un Trie ja que tots ells apunten a la següent lletra possible. Aquesta part ens manté la funcionalitat del Trie.

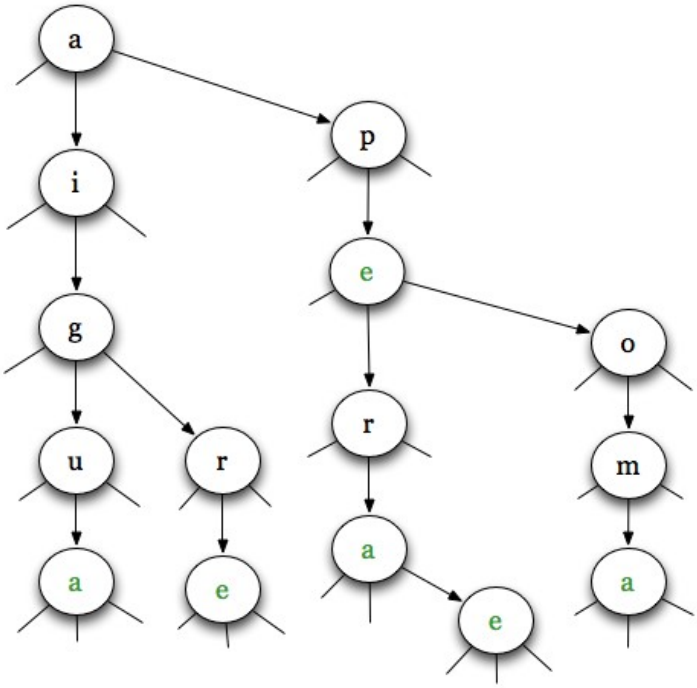

Figura 5.2.3.2. Representació d'un arbre ternari de cerca

Els fills esquerre i dret representen les altres lletres que poden tenir les paraules en el nivell en què estem. És a dir, quan viatgem per un d'aquests dos fills no canviem de posició dins de la paraula sinó que canviem la lletra de la posició actual. A més, per obtenir la funcionalitat de l'arbre binari de cerca, el fill esquerre conté les lletres que van abans segons l'orde lexicogràfic del caràcter del node pare i al fill dret les paraules que van després.

Aquesta implementació permet que l'ús d'apuntadors ens ocupi menys espai ja que hi haurà molts menys apuntadors que si cada node tingués tants apuntadors com lletres de l'abecedari. D'altra banda, l'eficiència en temps de la cerca de paraules dins de l'estructura es veu molt millorada ja que en cada node descartem la meitat de possibilitats, ja que podem comparar el caràcter actual amb el node on ens trobem i anar cap al fill lateral en el qual es troba el caràcter buscat, com en una recerca dicotòmica en un diccionari.

Un cop explicada l'estructura de dades que es va decidir implementar passem ara a explicar com es va implementar.

Per representar l'estructura de l'arbre s'ha creat la classe **TernarySearchTree.** Cada node de l'arbre està implementat amb una instància de la classe **Node** que està definida dins de la classe anterior. En cada un dels nodes de l'arbre hi ha la següent informació: un enter que ens indica la freqüència de la paraula que acaba en aquell node si és que representa alguna paraula, el caràcter que representa el node, un apuntador al node dret, esquerre i del mig respectivament i un boleà que indica si aquell node representa el final de la paraula. La classe **Node** implementa alguns mètodes d'utilitat per saber si un determinat node té fills i per construir nous **Node**.

```
int frequency = 0;
char character;
Node leftChild, middleChild, rightChild;
boolean isWord = false;
```

La classe **TernarySearchTree** té d'atributs un apuntador al **Node** arrel **(root)**, un apuntador al node actual de l'arbre **(currentNode)** que ens permet anar pujant o baixant per l'arbre mentre l'usuari va escrivint o esborrant caràcters, i una altra variable anomenada **cent** que ens apunta al node arrel. També tenim una llista dels nodes previs pels quals hem passat per arribar al **currentNode** a la variable **previousLevels**. Això ens permet pujar cap amunt

de l'arbre ja que en els nodes no tenim apuntador al nostre pare i no podríem pujar. S'ha preferit implementar-ho amb una pila.

```
Node root;
Node currentNode;
Node cent;
int numberOfWords;
LinkedList<Node> previousLevels;
```

El constructor de la classe s'encarrega de inicialitzar el **root**, el **currentNode** i la pila que guardarà el nostre recorregut per l'arbre. Aquest primer constructor només s'usa internament.

```
private TernarySearchTree()
{
    root = null;
    currentNode = null;
    numberOfWords = 0;
    previousLevels = new LinkedList<Node>();
}
```

L'altre constructor que s'ha creat, i que s'usarà posteriorment en el programa, ens construeix l'arbre a partir d'una llista de parelles de paraula i freqüència. Per aquesta estructura de parella s'ha creat la classe **WordInfo** que no és més que una paraula amb la seva corresponent freqüència.

```
public TernarySearchTree(ArrayList<WordInfo> wordList)
{
    this();
    balancedInsertion(wordList,0,wordList.size()-1);
    cent = new Node('0');
    cent.middleChild = root;
    currentNode = cent;
}
```

Per construir l'arbre es van inserint cadascuna de les paraules i la seva freqüència dins de l'arbre. Per fer-ho s'utilitza la funció **balancedInsertion** que insereix les paraules de manera que primer s'insereix la paraula del mig segons el seu ordre lexicogràfic, després les paraules més petites i finalment les

més grans, de forma recursiva. La llista de paraules ha d'estar ordenada alfabèticament.

```
private void balancedInsertion(ArrayList<WordInfo> wordList, int first, int last)
{
    if(last >= first)
    {
        int middle = ((last - first)/2)+first;

         WordInfo wordInfo = wordList.get(middle);
         insert(wordInfo.word,wordInfo.frequency);
         System.out.println(wordInfo.word);

         balancedInsertion(wordList,first,middle-1);
         balancedInsertion(wordList,middle+1,last);
    }
}
```

El motiu de la inserció d'aquesta forma està explicat a la referència BENTLEY, JL; SEDGEWICK, R. "Fast Algorithms for Sorting and Searching Strings "Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, January 1997 concretament a la pàgina 6 columna de la dreta.

Inserint les paraules d'aquesta manera el nombre de comparacions realitzades disminueix significativament que en altres formes d'inserció. A més, inserint d'aquesta manera ens assegurem que l'arbre quedi totalment equilibrat de manera que no ens degeneri en una llista, la qual cosa faria que les operacions implementades baixessin molt la seva eficiència.

Atès que, com ja s'explicarà més endavant, l'operació d'inserció d'una paraula té complexitat computacional de **O (log n)** la funció d'inserció balancejada té un cost de **O (n * log n)**. En un arbre no balancejat la inserció tindria un cost de **O (n)**.

Faltava dir que el constructor després de l'execució de la inserció balancejada ens construeix el node **cent** que apunta a l'arrel de l'arbre i és el primer node en

el qual es troba **currentNode** cada vegada que l'usuari comença a escriure una paraula.

Passem ara a explicar l'operació **insert (String word, int frequency)** que ja s'ha comentat. Aquesta operació té una complexitat de **O (log n)** ja que es tracta d'anar baixant per les branques d'un arbre. Aquesta funció va inserint els caràcters que formen part de la paraula a inserir.

En aquesta funció primer de tot ens situem al node arrel de l'arbre. Hi ha un cas especial en el qual hem de crear aquest node **root** que es produeix quan enviem la primera paraula dins de l'arbre. En tots els altres casos el que fem és anar comparant el primer caràcter de la paraula amb els caràcters dels nodes ja inserits. Com ja s'ha explicat, quan comparem sabem a quin fill anar segons l'abecedari.

- En el cas que el nostre caràcter sigui més petit que el caràcter del node actual seguim buscant pel node esquerre. En el cas que no existeixi node esquerre haurem trobat la posició de la primera lletra de la paraula i haurem de crear un nou Node amb aquesta lletra i acabarem el procés de comparació. En cas que ja existeixi fill esquerre haurem de continuar comparant fins a trobar la nostra posició.

- En el cas que el nostre caràcter sigui més gran que el caràcter del node actual ens trobem en el cas contrari de l'anterior i tindrem la mateixa casuística però amb el fill dret.

- En el cas que els caràcters coincideixin voldrà dir que hem trobat la posició del primer caràcter de la paraula. En aquest cas hem de repetir tot això però amb el segon caràcter de la paraula fins que s'acabi la paraula o trobem un node fill inexistent. Si anem per aquesta última opció també inserim el caràcter en cas que ens trobem amb un fill buit,

però en aquest cas inserim el caràcter següent.

En el cas que trobem un fill buit en el qual s'ha de posar el nostre caràcter, el bucle anterior s'acaba i només hem d'anar inserint els caràcters que falten per acabar la paraula en el fill del mig, creant la seqüència que conforma la paraula. En cadascun dels casos, quan detectem que estem al final de la paraula, posem a cert la variable **isWord** de l'últim node i posem la freqüència corresponent en la seva variable **frequency**.

```
    public void insert(String word,int frequency)
{
     numberOfWords++;
     int level = 0;
     int wordLength = word.length();
     if(root == null)
     {
          root = new Node(word.charAt(level),level+1 == wordLength);
          if(level == wordLength-1)
          {
               root.frequency = frequency;
               root.isWord = true;
               return;
          }
     }

     char character;

     Node travesalNode;
     travesalNode = root;

     boolean continueDown = true;
     character = word.charAt(level);

     while(continueDown)
     {
          if(character < travesalNode.character)
          {
               if(!travesalNode.existLeft())
               {
                    travesalNode.leftChild = new Node(character,level+1 == wordLength);
                    continueDown = false;
               }
               travesalNode = travesalNode.leftChild;
          }
          else if(character == travesalNode.character)
          {
```

```
                if(level == wordLength-1)
                {
                    travesalNode.frequency = frequency;
                    travesalNode.isWord = true;
                    return;
                }
                if(!travesalNode.existMiddle())
                {
                    travesalNode.middleChild = new Node(word.charAt(level+1),level+1 ==
                                                        wordLength-1);
                    continueDown = false;
                }
                travesalNode = travesalNode.middleChild;
                level++;
                character = word.charAt(level);
            }
            else
            {
                if(!travesalNode.existRight())
                {
                    travesalNode.rightChild = new Node(character,level+1 == wordLength);
                    continueDown = false;
                }
                travesalNode = travesalNode.rightChild;
            }

        }


        while(level < wordLength-1)
        {
            travesalNode.middleChild = new Node(word.charAt(level+1),level+1 ==
                                                wordLength-1);
            level++;
            travesalNode = travesalNode.middleChild;
        }
        travesalNode.frequency = frequency;

    }
```

Fins ara les funcions anteriors són utilitzades en el moment de la creació de l'estructura de dades que contindrà les paraules d'un determinat diccionari. Vegem ara les funcions utilitzades per anar pujant i baixant per l'estructura durant l'escriptura d'una paraula per part de l'usuari.

La funció **nextLevel (char character)** s'executa cada vegada que l'usuari escriu un nou caràcter de la paraula (els **TEXTMODIFIER** no). Aquesta funció

realitza pràcticament el mateix que la **insert (String word, int frequency)** però només per trobar la posició del caràcter entrat. La funció executa pràcticament el mateix bucle que en la inserció. Quan executem aquesta funció es crea el node corresponent si no existia. Dins d'aquesta funció guardem el Node en què estàvem abans a la variable **previousLevels** que ens permetrà pujar si l'usuari esborra el caràcter.

```
    public void nextLevel(char character)
{
     if(root == null)
     {
          root = new Node(character);
          cent.middleChild = root;
          currentNode = root;
          return;
     }

     if(currentNode.existMiddle())
     {
          Node travesalNode;
          previousLevels.push(currentNode);
          travesalNode = currentNode.middleChild;
          boolean continueDown = true;

          while(continueDown)
          {
               if(character < travesalNode.character)
               {
                    if(!travesalNode.existLeft())
                    {
                         travesalNode.leftChild = new Node(character);
                         continueDown = false;
                    }
                    travesalNode = travesalNode.leftChild;
               }
               else if(character == travesalNode.character)
               {
                    continueDown = false;
               }
               else
               {
                   if(!travesalNode.existRight())
                   {
                         travesalNode.rightChild = new Node(character);
                         continueDown = false;
                   }
                   travesalNode = travesalNode.rightChild;
               }
```

```
            }
            currentNode = travesalNode;
        }
        else
        {
            currentNode.middleChild = new Node(character);
            previousLevels.push(currentNode);
            currentNode = currentNode.middleChild;
        }
    }
```

Aquesta funció ens permet anar creant l'arbre a mesura que l'usuari va escrivint lletres de paraules. En cas que s'activi l'opció d'aprendre paraules, quan s'acabi la paraula aquesta quedarà guardada en l'arbre. En cas que no es vulguin aprendre paraules l'última paraula escrita s'esborrarà de l'arbre sense afectar les altres paraules existents. Aquí hi havia l'opció de no anar creant els nodes de l'arbre a mesura que s'escrivia sinó inserir la paraula quan ja se sabia que es volia aprendre-la. S'ha optat per anar creant l'arbre ja que d'aquesta manera podem anar obtenint les paraules que comencen pel prefix que hem escrit i, a més, podem canviar d'opinió d'aprendre la paraula al final i així assegurem que no es perdi la paraula escrita. Per cert, tenim la complexitat **O (log n)** en aquesta última funció.

Com que en un moment donat podem escriure de cop una paraula de les que se'ns prediuen, s'ha creat una funció anomenada **positioning (String character)** que s'encarregarà d'anar baixant per l'arbre, construint els nodes si cal, com si la paraula escrita fos un prefix, de manera que podrem predir les paraules que comencen per la paraula entrada.

```
public void positioning(String character)
{
    int length = character.length();
    for(int i = 0; i < length; i++)
    {
        nextLevel(character.charAt(i));
    }
}
```

Una altra funció que s'utilitza bastant és **previousLevel ()** que permet pujar per l'arbre quan l'usuari esborra un caràcter, de manera que podem predir les paraules que comencen pels caràcters previs. Com ja s'ha explicat, es va guardant el camí de baixada en una pila **(previousLevels)**. Quan volem pujar cap amunt només hem d'agafar el primer element de la pila. Una altra cosa que hem de fer és esborrar el node de l'últim caràcter escrit de l'arbre. Per fer-ho hem d'anar baixant des del node anterior fins a trobar el node que volem esborrar. Aquesta funció, per tant, té una complexitat **O (log n)**.

S'ha de tenir en compte que el node previ no és el node que té algun dels seus fills apuntant-nos, sinó que és el primer node en el qual comencen les lletres del nivell anterior de la paraula. Per exemple, si havíem escrit "pom" i volem anar al nivell anterior hauríem d'anar al node "e", veure Figura 5.2.3.2 pàg. 30.

```
    public boolean previousLevel()
{
     if(!previousLevels.isEmpty())
     {
          Node previousNode = previousLevels.pop();

          if(!currentNode.hasChilds() && !currentNode.isWord)
          {
               char character = currentNode.character;

               Node travesalNode;
               travesalNode = previousNode.middleChild;
               boolean continueDown = true;

               while(continueDown)
               {
                    if(character < travesalNode.character)
                    {
                         if(travesalNode.leftChild.character == character)
                         {
                              travesalNode.leftChild = null;
                              break;
                         }else
                         {
                              travesalNode = travesalNode.leftChild;
                         }
                    }
                    else if(character == travesalNode.character)
                    {
                         previousNode.middleChild = null;
```

```
                    break;
                }
                else
                {
                    if(travesalNode.rightChild.character == character)
                    {
                        travesalNode.rightChild = null;
                        break;
                    }
                    else
                    {
                        travesalNode = travesalNode.rightChild;
                    }
                }

            }
        }
        currentNode = previousNode;

        return true;
    }
    return false;
}
```

Cada vegada que l'usuari acaba una paraula i escriu un separador (espai, enter, tabulador ...) s'executa la funció **confirmWord (boolean learning)**. Aquesta funció canvia el seu comportament depenent de si volem guardar la paraula que hem escrit i de si l'últim node de la paraula indica **isWord**.

• En cas que l'últim node indiqui **isWord** només haurem d'incrementar en 1 la freqüència d'aparició.

•En cas que **isWord** no sigui cert, depenent de si aprenem o no, fem una cosa o l'altra. Si volem aprendre, només hem de posar **isWord** a cert. Si no volem aprendre haurem d'esborrar els nodes d'aquest mot sense afectar als altres amb la funció **deleteNodes**. L'única cosa que fa és anar executat la funció **previousLevel** mentre no afectem a altres nodes.

```
public void confirmWord(boolean learning)
{
    if(currentNode.isWord)
    {
        currentNode.frequency++;
    }
    else
    {
        if(learning)
        {
            currentNode.isWord = true;
        }
        else
        {
            deleteNodes();
        }
    }
    currentNode = cent;
}

public void deleteNodes()
{
    while(!currentNode.hasChilds() && !currentNode.isWord &&
    previousLevel())
    {
        ;
    }
    if(!root.hasChilds())
    {
        root = null;
    }
}
```

L'última funció que conté aquesta classe s'encarrega d'obtenir totes les paraules que es prediran que comencen pel prefix que ha escrit l'usuari. La funció es diu **getSubtree (Node subtreeRoot, String word, ArrayList <WordInfo> map)** i va recorrent l'arbre de sota d'un determinat node. La funció retorna una llista de les paraules que contenen els nodes d'aquest arbre ordenades alfabèticament. Això ens ajuda a que la posterior inserció tingui una llista ordenada alfabèticament tal com ja s'ha comentat que calia. La complexitat d'aquesta funció és **O(n*log n)**.

```
public void getSubtree(Node subtreeRoot, String word, ArrayList<WordInfo> map)
```

```
{
    if( subtreeRoot != null)
    {
        Node p = subtreeRoot;

        if(p.existLeft()) getSubtree(p.leftChild,word,map);

        if(p.isWord)
        {
            map.add(new WordInfo(word+p.character, p.frequency));
        }

        if(p.existMiddle()) getSubtree(p.middleChild,word+p.character,map);

        if(p.existRight()) getSubtree(p.rightChild,word,map);
    }
}
```

En aquest apartat s'ha explicat extensament l'estructura de dades utilitzada per a la predicció per freqüències. S'ha tractat més profundament que altres apartats a causa de la importància de l'estructura en el rendiment del programa.

En aquesta estructura d'arbre s'ha comentat anteriorment que s'ha d'inserir de manera que quedi balancejat l'arbre. Durant la creació dels nodes que representen les paraules no s'ha implementat l'operació de re-balanceig ja que s'ha considerat que és bastant difícil, si no es fa a propòsit, que l'arbre quedi molt mal equilibrat, per exemple, si escrivim una paraula que comenci per a, després per b, etcètera. Aquesta és una millora que es podria implementar de manera que cada vegada que afegim un nou node, en un nivell de posició dins d'una paraula, balancegem aquest nivell.

3.2.4. Base de dades

Un cop es tenia implementada l'estructura de dades que suportaria la predicció amb freqüència, es va crear la base de dades per guardar els diccionaris que creés l'usuari i les seves paraules. A més, es va integrar dins la base de dades Cercamots per utilitzar-la en la predicció seguint la gramàtica catalana.

La base de dades, creada en MySQL, conté dues taules utilitzades en la predicció

per freqüència i una altra taula amb dades incloses al Cercamots.

Per a la predicció per freqüència s'han creat les taules **Dictionary** i **Word**. En la primera d'elles, es guarden els noms dels diccionaris que crea l'usuari juntament amb el seu identificador que funciona com a clau primària.
En la segona, es guarden les paraules que conté un determinat diccionari i la freqüència d'aparició en aquest. En aquest últim cas, la clau primària és la paraula i el diccionari del qual forma part.
Entre les dues taules hi ha una relació d'1 a N que ens associa el nom del diccionari amb les seves paraules.

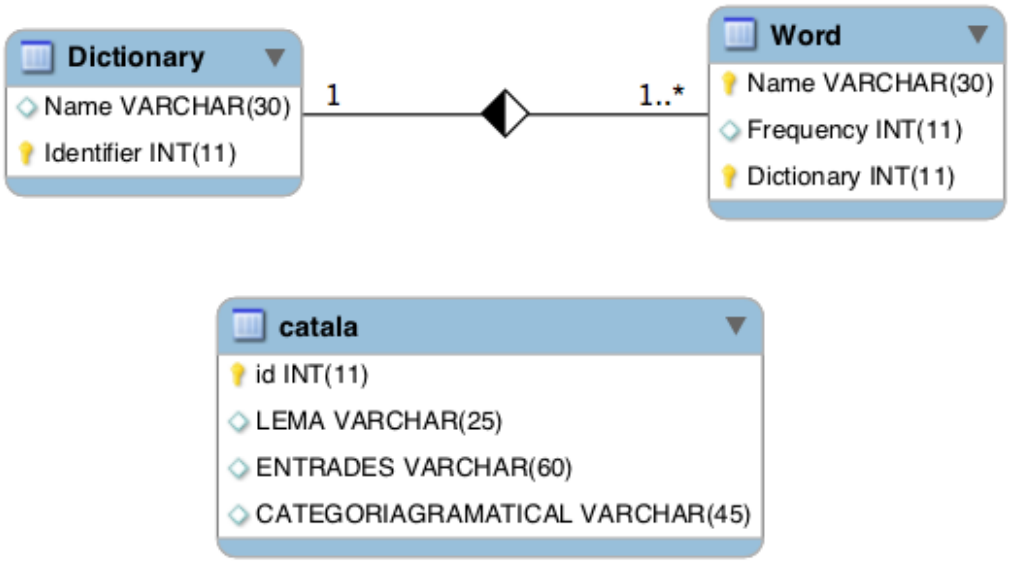

Figura 5.2.4.1 Diagrama de la base de dades

La definició en SQL d'aquestes dues primeres taules és la següent:

```
CREATE  TABLE IF NOT EXISTS `Keys4Free`.`Dictionary` (
`Name` VARCHAR(30) CHARACTER SET 'latin1' COLLATE 'latin1_general_ci'
NULL DEFAULT NULL ,
`Identifier` INT(11) NOT NULL AUTO_INCREMENT ,
```

```
PRIMARY KEY (`Identifier`) )

CREATE  TABLE IF NOT EXISTS `Keys4Free`.`Word` (
`Name` VARCHAR(30) CHARACTER SET 'latin1' COLLATE 'latin1_general_ci'
NOT NULL ,
`Frequency` INT(11) NULL DEFAULT NULL ,
`Dictionary` INT(11) NOT NULL ,
PRIMARY KEY (`Name`, `Dictionary`) ,
INDEX `fk_Word_Dictionary` (`Dictionary` ASC) ,
CONSTRAINT `fk_Word_Dictionary`
FOREIGN KEY (`Dictionary`)
REFERENCES `Keys4Free`.`Dictionary` (`Identifier` )
ON DELETE NO ACTION
ON UPDATE NO ACTION)
```

L'altra taula creada, conté les dades de Cercamots utilitzades per a la predicció seguint la gramàtica catalana. Aquesta taula no conté tota la informació que es tenia a la base de dades Cercamots sinó només els camps que s'han utilitzat en aquest tipus de predicció.

En aquesta taula, anomenada **catala,** es guarden unes 17.000 entrades de paraules de la llengua catalana, així com la seva categoria gramatical. En cadascuna d'aquestes entrades, si és necessari, hi ha les diferents formes en que pot trobar-se la paraula segons el gènere o el nombre.

```
CREATE  TABLE IF NOT EXISTS `Keys4Free`.`catala` (
 `id` INT(11) NOT NULL AUTO_INCREMENT ,
`LEMA` VARCHAR(25) CHARACTER SET 'latin1' COLLATE
'latin1_general_ci' NULL DEFAULT NULL ,
`ENTRADES` VARCHAR(60) CHARACTER SET 'latin1' COLLATE
```

```
'latin1_general_ci' NULL DEFAULT NULL ,
`CATEGORIAGRAMATICAL` VARCHAR(45) CHARACTER SET 'latin1'
COLLATE 'latin1_general_ci' NULL DEFAULT NULL ,
PRIMARY KEY (`id`) )
```

En l'apartat dedicat a la predicció en català s'explicarà més detingudament l'estructura d'aquesta taula de la base de dades i la seva utilització dins del programa.

Per gestionar la connexió amb la base de dades, s'ha creat la classe **DatabaseConnection** que conté la informació necessària per connectar-se a la base de dades i donar connexions a les classes que necessitin utilitzar-la. Aquesta classe ens guarda el nom d'usuari, la contrasenya i un enllaç de connexió amb la base de dades. A aquesta classe se li pot demanar que ens obri una connexió perquè puguem usar-la en un altre lloc sense conèixer els paràmetres de connexió.

Cal dir que la base de dades és externa al programa i s'ha d'iniciar el servidor MySQL des de fora del programa. Per a un posterior ús real del programa, aquesta base de dades hauria d'integrar-se dins el programa de manera que es gestionés interiorment la iniciació o finalització de l'execució del servidor. Per fer-ho es podria utilitzar la versió encastada de MySQL que s'ha descobert que hi ha durant la realització d'aquest apartat.

El driver que s'ha utilitzat per connectar-se a la base de dades, que està integrat dins de les llibreries utilitzades en el programa, és el mysql-connector-java-5.1.17.

3.2.5. Sistema de predicció

Una vegada que teníem les estructures que suportarien el sistema de predicció,

es va començar a implementar aquest sistema.

La idea era que quan es produís la inserció d'un nou caràcter a la caixa de text de la pantalla principal **MainWindow** es detectés aquesta inserció, i a partir d'aquest flux de dades anar predient les paraules que es podien escriure i mostrar-les a l'usuari. Aquestes paraules formarien part d'un diccionari, i aquest diccionari contindria l'estructura de dades que s'ha explicat.

La primera classe que es va crear va ser la classe **Dictionary** que permet gestionar els diferents diccionaris. Aquesta classe conté mètodes que realitzen accions contra la base de dades explicada anteriorment. Concretament, en aquesta classe s'han implementat les consultes a la base de dades per crear nous diccionaris, per carregar els diccionaris que estan creats, per eliminar diccionaris i les seves paraules, per carregar les paraules dels diccionaris o per guardar les paraules dels diccionaris. En aquesta classe s'utilitza la classe **DatabaseConnection**.

Cada vegada que s'executa una d'aquestes accions es fa en una nova connexió amb base de dades. En aquesta classe no mantenim una connexió constant amb la base de dades ja que es considera que aquestes accions seran puntuals, perquè la major part del temps l'usuari estarà escrivint utilitzant un determinat diccionari.

Com atribut d'aquesta classe hi ha una instància de la classe **Predictor** que és la que realment implementa la funcionalitat de predicció d'un determinat diccionari. Quan es carreguen les paraules d'un diccionari aquestes passen directament a l'estructura **TernarySearchTree,** que es gestionada per la classe **Predictor.** Quan es canvia de diccionari o es tanca el programa, les paraules que formen l'arbre es tornen a la base de dades actualitzant el conjunt

de paraules i freqüències anteriors.

La classe **Predictor** s'encarregarà d'anar executant una sèrie de funcions que interactuen amb el **TernarySearchTree** que gestiona. Aquestes funcions són cridades des de la classe **PredictorWindow** que és la que rep els esdeveniments generats a la caixa de text de **MainWindow**. Aquests esdeveniments es produeixen quan s'introdueix un nou caràcter o quan s'elimina un caràcter. Depenent d'aquests dos esdeveniments i de si s'ha introduït un separador de paraula, s'executa una o altra acció en el predictor del diccionari actiu.

L'acció que es produeix quan s'insereix un nou caràcter a la caixa de text, que no és un separador de paraula, és **newCharacter (String character)**. Els separadors de paraula no són més que caràcters que fan el que el seu nom indica. Aquests estan definits a la classe **Predictor** i s'ha considerat que són la nova línia, l'espai, el tabulador, el punt, la coma i el punt i coma.

```
public void newCharacter(String character)
{
    if(!writingWord)
    {
        writingWord = true;
    }
    if(character.length() < 2)
    {
        predictionWords.nextLevel(character.charAt(0));
    }
    else
    {
        predictionWords.positioning(character);
    }
    currentWord = currentWord+character;
}
```

El que fa aquesta funció és baixar un nivell per l'arbre, com s'ha explicat anteriorment. A més també contempla rebre com a paràmetre una paraula seleccionada de la finestra de predicció. En aquest cas baixem els nivells de

l'arbre que convinguin. Amb aquest caràcter o grup de caràcters que es reben com a paràmetre, actualitzem la variable **currentWord** que ens indicarà en cada moment el prefix que està actualment escrit d'una paraula.

En cas que el caràcter entrat a la caixa de text sigui un separador de paraula, s'executa la funció **addSeparator (String separator)**. Aquesta funció comunica a l'arbre que ens acabi el camí d'entrada d'una paraula. Se li indicarà a l'arbre si estàvem en mode d'aprendre paraules o no i s'actualitza el **currentWord** a la paraula buida.

```
void addSeparator(String separator) {
    if(writingWord)
    {
        predictionWords.confirmWord(isLearningOn());
        currentWord = "";
        writingWord = false;
        if(predictedEntered)
        {
            predictedEntered = false;
        }
    }
}
```

En el cas que s'esborri un caràcter de la caixa de text s'executa la funció **processRemove()**. Aquesta funció comunica a l'arbre que vagi un nivell cap amunt i actualitza **currentWord** eliminant l'últim caràcter de la paraula.

```
void processRemove() {

    if(writingWord && predictionWords.previousLevel())
    {
        currentWord = currentWord.substring(0, currentWord.length()-1);
        if(currentWord.equals(""))
        {
            writingWord = false;
        }
    }
}
```

En tots els casos anteriors es controla si estem escrivint o no amb l'atribut de la classe **writingWord**. Aquest serveix perquè no considerem els separadors de paraula si no estàvem escrivint cap paraula, per indicar que estem escrivint quan inserim el primer caràcter d'una paraula, i perquè no fem l'acció de pujar per l'arbre si esborrem i no estem escrivint cap paraula.

A la classe **Predictor** també hi ha definides dues funcions que obtenen les paraules de l'arbre. La funció **ArrayList <String> getCandidateWords()** envia a la finestra de predicció les paraules que comencen per **currentWord**. Com que ja estem posicionats dins de l'arbre, només hem d'utilitzar la funció de l'arbre que ens retorna totes les paraules a partir d'un determinat node. Recordem que aquesta funció ens retorna les paraules ordenades alfabèticament juntament amb les freqüències. Com ja s'ha dit, ens interessa mostrar aquestes paraules ordenades segons la seva freqüència i, per això, s'executa un heapsort implementat a la classe **Heap.** Amb les parelles de paraules i freqüències com a entrada, i amb les paraules ordenades per freqüència com a sortida.

```
public ArrayList<String> getCandidateWords()
{
    Heap heap;

    ArrayList<WordInfo> words = new ArrayList<WordInfo>();
    predictionWords.getSubtree(predictionWords.currentNode.
            middleChild, currentWord, words);

    heap = new Heap(words);
    return heap.heapsort();
}
```

La classe **Heap** implementa aquesta estructura juntament amb les operacions per inserir i per eliminar elements. Les paraules i freqüències obtingudes de l'arbre es van entrant a aquesta estructura complint amb les restriccions que imposa, és a dir, mentre es va inserint ens assegurem que es compleixin les condicions bàsiques d'un heap. Un cop tenim totes les parelles inserides, utilitzant la funció **ArrayList <String> heapsort()**, anem eliminant les

paraules una a una del heap, mantenint les seves condicions, de manera que obtenim les paraules ordenades de major a menor freqüència.

S'ha utilitzat aquest mètode d'ordenació ja que és l'algoritme més estable que es coneix per a l'ordenació i ens assegura que sempre tindrem una complexitat computacional de **O (n * log n)**. Això permet que el rendiment del programa en el moment d'indicar les paraules que es poden escriure sigui bastant bo. L'operació que obté les paraules de l'arbre té la mateixa complexitat.

L'altra funció que retorna paraules de l'arbre és **ArrayList <WordInfo> getAllWords()**. En aquest cas, s'utilitza per agafar totes les parelles de paraula i freqüència de l'arbre. Aquesta funció s'utilitza per guardar les paraules i freqüències del diccionari, de forma alfabètica, a la base de dades. Recordem que en el moment de creació de l'arbre necessitem les paraules ordenades d'aquesta manera per millorar l'eficiència de la inserció. Així doncs, quan tornem a carregar les paraules d'aquest diccionari ja ho tindrem bé per a la creació de l'arbre.

```
public ArrayList<WordInfo> getAllWords()
{
    ArrayList<WordInfo> words = new ArrayList<WordInfo>();
    predictionWords.getSubtree(predictionWords.root, "", words);
    return words;
}
```

La classe **Predictor** té l'atribut **learningOn** que indica si es volen aprendre paraules quan s'utilitza aquest diccionari. Aquest atribut pot ser canviat per l'usuari depenent de les seves necessitats.

Com ja s'ha dit anteriorment, totes les funcions de la classe **Predictor** són cridades per la classe gràfica **PredictionWindow** que és la que s'encarrega de

connectar el flux de text, que va entrant l'usuari pel teclat virtual, amb el sistema de predicció.

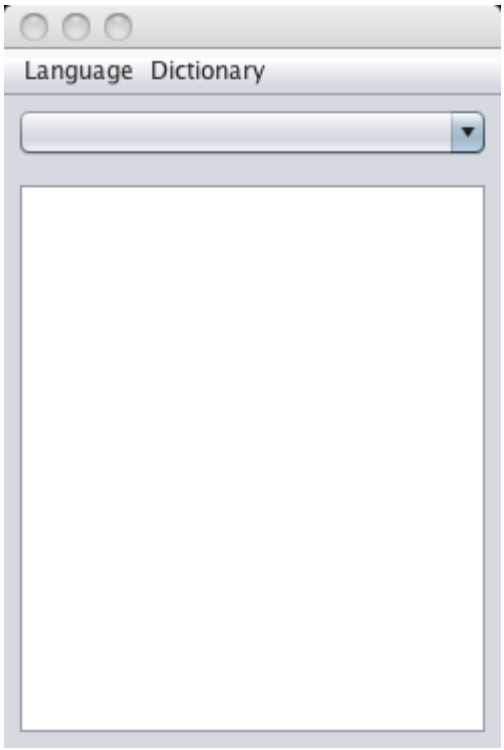

Figura 3.2.6.1 Imatge de la pantalla de predicció

A més de la funció anterior, s'encarrega de mostrar les paraules que es poden escriure en un determinat moment perquè l'usuari pugui seleccionar-les. Una altra funció que té, és la de triar el diccionari que es vol utilitzar i gestionar els diccionaris. També permet que l'usuari triï el tipus de predicció que es vol utilitzar en el diccionari actiu.

Aquesta finestra és creada per la pantalla principal **MainWindow** al principi del programa. Per a la seva creació, se li ha de passar el teclat del qual obtindrà la informació d'on ha d'escriure quan l'usuari seleccioni una paraula, i amb la caixa de text que indica el teclat podrem registrar que volem escoltar el que es va escrivint en aquesta caixa, per realitzar les accions apropiades.

Aquesta classe implementa la interfície **DocumentListener** que permet escoltar els canvis que es produeixen a la caixa de text on escrivim. En aquesta interfície hem de implementar els mètodes **insertUpdate (DocumentEvent i)** i **removeUpdate (DocumentEvent i)** que s'executen quan s'insereix un nou caràcter a la caixa de text o quan s'elimina el caràcter respectivament.

La primera funció, com ja s'ha comentat, depenent del caràcter que escrivim cridarà als mètodes de la classe **Predictor addSeparator** o **newCharacter** que s'han explicat anteriorment. Després que el predictor actualitzi l'arbre amb una d'aquestes dues funcions, es cridarà a la funció **getCandidateWords** del predictor perquè ens digui les paraules que pot escriure l'usuari, i enviar-les a la llista perquè l'usuari pugui veure-les i seleccionar-les.

En la segona funció es crida el mètode **processRemove** del predictor i també es mostren les paraules possibles a la llista gràfica.

La classe **PredictorWindow** té com atribut una instància de la classe **Dictionary** anomenada **currentDictionary** que ens indicarà en tot moment el diccionari que tenim activat, i des del qual podem accedir a les funcions del predictor que té associat.

Aquesta classe, quan es construeix, carrega de la base de dades tots els noms dels diccionaris creats per l'usuari que es mostren en el combo box de la pantalla.
Un cop aquests estan carregats l'usuari ha de seleccionar el diccionari que vol utilitzar, si és que en vol usar algun. Quan l'usuari selecciona el diccionari, es construeix aquest diccionari en l'atribut **currentDictionary** executant el mètode **comboBoxDictionariesActionPerformed**. Això vol dir que es carreguen les paraules associades al diccionari, i depenent de les opcions del llenguatge es crearà dins del diccionari un predictor per freqüència o un

predictor seguint la gramàtica catalana. Això últim es fa utilitzant el mètode del diccionari **setPredictor**.

Els menús d'aquesta pantalla són d'una banda el menú Language i d'altra banda el menú Dictionary. El primer d'ells permet activar el mode de predicció catalana, i activar l'opció de si volem que aquesta predicció inclogui la predicció amb freqüència o només segons la gramàtica. El segon menú permet crear diccionaris, eliminar diccionaris, carregar els diccionaris i indicar si volem que el predictor aprengui les paraules noves. Totes aquestes operacions utilitzen les funcions de la classe **Dictionary** o del seu predictor.

La funció que s'executa quan es selecciona una paraula de la llista de paraules és **wordsListValueChanged**.

```
 private void wordsListValueChanged(javax.swing.event.ListSelectionEvent evt)
{
    String word = (String)
    ((JList)evt.getSource()).getSelectedValue();

     if(word!=null)
     {
          String partialWord =
         (currentDictionary.predictor.currentWord.equals("") ?
               word :
               word.substring(word.indexOf(currentDictionary.predictor.currentWord)
               +currentDictionary.predictor.currentWord.length(),word.length()));

          currentDictionary.predictor.predictedEntered = true;

          keyboard.setKeyAtComponent(partialWord);
          try {
               keyboard.output.getDocument().insertString(keyboard.output.getText().length(
                    ) ,partialWord, null);
          } catch (BadLocationException ex) {

          }
     }

}
```

En ella el que es fa és obtenir la cadena de caràcters que ha d'escriure el

programa, tant en la caixa de text com en l'aplicació activa del sistema operatiu. Per fer-ho hem de comparar el prefix de paraula escrit amb la paraula seleccionada. Hem d'agafar la cadena de caràcters que encara no hem escrit de la paraula per saber què hem d'escriure. Un cop hem trobat la cadena a escriure, s'utilitza la funció **setKeyAtComponent** del teclat que l'envia a l'aplicació activa, i també l'enviem a la caixa de text.

Quan es tanca el programa o es canvia de diccionari, es guarden les paraules d'aquest a la base de dades. Per fer-ho es crida la funció **saveDictionary** del **currentDictionary**.

3.2.6. Predicció del català

L'últim pas de la implementació del programa va ser la creació d'un predictor seguint algunes de les regles de la gramàtica catalana.

Durant la realització del projecte es van concretar, amb l'assessorament d'un filòleg, les regles més bàsiques que podien ajudar en major mesura a aconseguir una velocitat d'escriptura raonable amb aquest tipus de predicció. Concretament les regles que es van definir i que s'implementaren foren les següents:

1. Dues regles sintàctiques productives:
    - Després d'un article (el, la, etc., un, una, etc.) no hi poden anar verbs ni adverbis ni conjuncions ni preposicions ni pronoms, sinó noms i, de vegades, adjectius.
    - Després d'un adverbi no seguit de punt o coma, només hi poden anar adjectius o, en alguns casos, un altre adverbi.

2. Aspectes morfològics

- -Verbs regulars de la primera conjugació (model cantar), que són la majoria: un cop escrit el radical i la vocal temàtica (cant/a, menj/a) comença la part variable del verb segons el temps (canta/ré, canta/va, canta/ria); el problema és que en els temps de subjuntiu la vocal temàtica no hi apareix (cant/és, cant/i).
- -Verbs regulars incoatius de la tercera conjugació (model partir o dividir): tots segueixen el mateix patró (radical+(increment "eix" en els alguns temps)+ terminació. Exemples: part/eix/o, part/eix/i, però part/ís... .

3. Altres
   - Després d'un apostrof la paraula comença amb vocal o amb “h”.
   - Després d'un guionet la paraula comença amb consonant i pronom.

Per la implementació de la predicció segons la gramàtica catalana s'ha creat la classe **CatalanPredictor** que hereta de la classe **Predictor**. En aquesta classe s'han d'implementar els mètodes que s'han d'utilitzar quan s'insereix un nou caràcter **(newCharacter)**, quan s'insereix un separador de paraula **(addSeparator)**, quan s'elimina un caràcter **(processRemove)**, o per obtenir les paraules per mostrar a l'usuari les opcions de predicció **(getCandidateWords)**. En definitiva, els mateixos mètodes de la classe **Predictor** però adaptant-los a la nova funcionalitat. A banda d'implementar aquestes funcions, s'han implementat els mètodes auxiliars necessaris per realitzar la predicció del català.
El sistema predictiu s'ha dissenyat de tal forma que es pugui combinar la predicció per freqüències i la predicció seguint la gramàtica catalana. Les funcions esmentades anteriorment, en el cas del predictor en català, combinen les funcions del predictor per freqüència amb la nova funcionalitat. L'usuari pot decidir mitjançant l'atribut **strictLanguagePrediction** si vol combinar les

dues funcionalitats quan utilitza la predicció en català o si només vol que se l'ajudi segons la gramàtica catalana.

Abans d'explicar com s'ha implementat aquest predictor català s'explicarà l'estructura de la base de dades Cercamots, ja que la implementació d'aquesta classe és molt dependent d'aquesta per a l'obtenció de les diferents dades que es necessiten per a la predicció.

La taula que conté la informació utilitzada durant la predicció és la que es diu **catala.** Com ja s'ha mostrat anteriorment, aquesta conté una columna anomenada **LEMA**, una altra anomenada **ENTRADES** i una altra anomenada **CATEGORIAGRAMATICAL**.

• La columna **LEMA** conté un representant d'una determinada paraula, és a dir, en el cas d'un verb ens indica la seva forma d'infinitiu i en el cas d'una categoria que varii en gènere i/o nombre ens indica només una forma de la paraula. En altres categories aquesta columna conté la paraula en si.

• La columna **ENTRADES** conté les diferents formes de la paraula, en cas que la paraula pugui presentar diferents formes segons el gènere o el nombre. En aquesta columna hi ha diferents patrons depenent d'aquestes variacions de les paraules. Aquests són detectats pel programa de manera que ens permet obtenir la concordança desitjada de la paraula. Per exemple, podem detectar el gènere i nombre d'un article, i després, consultar a la base de dades noms dels quals seleccionarem els que siguin concordants amb l'article.

• La columna **CATEGORIAGRAMATICAL**, com el seu nom indica, ens diu els tipus de categoria gramatical que pot tenir la paraula o paraules

en qüestió. En el cas dels verbs, aquesta columna ens indica el model de conjugació[1] al qual pertany la paraula. Aquest s'usarà per generar els possibles verbs de la primera i la tercera conjugació, que són els que s'han implementat com s'ha dit al principi d'aquest apartat. En les altres categories gramaticals s'indica una llista de les possibles categories de la paraula, referides sempre a la paraula representativa de la columna LEMA. En el cas dels noms se'ns indica el gènere quan aquest és únic, i també se'ns indica el nombre quan aquest també és únic.

id	LEMA	ENTRADES	CATEGORIAGRAMATICAL
64	abrigall	abrigalls	nom m
65	abrigar		v (cantar)
66	abril	abrils	nom m
67	abrillantar		v (cantar)
68	abrivar		v (cantar)
69	abrogar		v (cantar)
70	abrupte	abrupta abruptes	adj
71	abrusar		v (cantar)
72	absència	absències	nom f
73	absent	absents	adj
74	àbsida	àbsides	nom f
75	absis	absis	nom m
76	absoldre		v (valer)
77	absolució	absolucions	nom f
78	absolut	absoluta absoluts absolutes	adj
79	absolutament		adv
80	absorbent	absorbents	adj

Figura 3.2.6.1 Algunes entrades de la taula catala

Començarem explicant la implementació amb la manera com estarem connectats a la base de dades durant la utilització del predictor català. Quan l'usuari seleccioni el mode de predicció en català, obtenim una connexió amb la base de dades que es mantindrà oberta durant tota l'estona en què estiguem en

1 Un model de conjugació agrupa els verbs que tenen el mateix patró de conjugació en la creació dels diferents temps verbals.

aquest mode.

Com ja s'ha comentat, en aquesta manera de predicció l'estructura de dades utilitzada és la mateixa base de dades que conté la informació sobre les paraules catalanes. Per tant, durant tota la predicció anem realitzant consultes contra la base de dades i, per això, no ens convé anar obrint i tancant connexions. Quan sortim d'aquesta manera de predicció o s'acaba el programa, s'alliberen els recursos utilitzats per la connexió.

Per realitzar les consultes que convé s'ha creat un mètode perquè es pugui consultar la consulta que volem a la base de dades. Aquest mètode es diu **executeQuery (String string)** i ens crea l'estament, amb la consulta SQL que li passem a l'atribut **databaseStatement**. La connexió activa la tenim guardada en l'atribut **databaseConnection**.

Com ja hem dit, la classe **CatalanPredictor** ha d'implementar les funcions bàsiques d'un predictor adaptant-lo a la nova funcionalitat, i mantenint la funcionalitat del predictor per freqüències. La funció **processRemove** té el mateix funcionament que en el predictor per freqüències. La funció **newCharacter** també ens fa el mateix que en l'altre predictor però, a més, s'encarregarà de detectar quan escrivim un apòstrof o un guionet, cosa que produeix l'execució de la funció **addSeparator** ja que el considerem com un separador.

En aquest sistema de predicció en català, prediem depenent de la categoria de la paraula anterior, de manera que cada vegada que escrivim una paraula hem de buscar la categoria gramatical a la qual pertany, aquesta funció la realitza **addSeparator**.

En aquest cas, també s'han definit separadors de frase que es detecten en

aquesta funció, i que produeixen que les variables utilitzades per guardar la categoria gramatical, el nombre i el gènere de l'última paraula entrada, s'anul·lin permetent poder començar la nova frase sense tenir influències de la frase anterior. Aquests atributs són **categoriaActual, genereActual** i **nombreActual**.

La funció **addSeparator** el primer que fa és descobrir la categoria gramatical de la paraula acabada d'entrar, que està guardada en l'atribut **currentWord**. Això es realitza mitjançant el mètode **EnumMap <CategoriaGramatical, String> consultaCategoriesPossibles (String paraula)**. A la base de dades pot ser que hi hagi dues entrades per la mateixa paraula, per la qual cosa aquesta funció retorna per cada entrada la categoria més prioritària i la concatenació dels camps LEMA i ENTRADES. El tipus **CategoriaGramatical** defineix una enumeració de les possibles categories en català.

Dins de la funció realitzem una consulta a la base de dades buscant les entrades que continguin o al LEMA o a ENTRADES la paraula que s'ha escrit. A causa del format de la segona columna hem d'utilitzar les expressions regulars de MySQL per detectar la presència de la paraula desitjada. Per cadascuna de les entrades obtingudes de la paraula executem el mètode **CategoriaGramatical determinaCategoriaString (String stringCategory)**. A aquesta funció li passem com a paràmetre el camp **CATEGORIAGRAMATICAL**.

Aquesta última funció defineix unes prioritats per assignar a una paraula una categoria de totes les possibilitats que ens indiquen les dades obtingudes. Com ja s'ha comentat, en el camp de la categoria gramatical obtenim una llista amb les categories possibles de la paraula, i hem de seleccionar-ne alguna ja que no s'ha implementat un tractament tenint en compte totes les possibilitats.

```
public CategoriaGramatical determinaCategoriaString(String
```

```
    stringCategory)
{
    Matcher encaixa;

    if((encaixa =
    Pattern.compile("det").matcher(stringCategory)).find())
    {
        return CategoriaGramatical.DETERMINANT;
    }
    if((encaixa = Pattern.compile("nom m i f")).matcher(stringCategory)).find())
    {
        genereActual = Genere.INDIFERENT;
        return CategoriaGramatical.NOM;
    }
    if((encaixa = Pattern.compile("nom m").matcher(stringCategory)).find())
    {
        genereActual = Genere.MASCULÍ;
        return CategoriaGramatical.NOM;
    }
    if((encaixa = Pattern.compile("nom f").matcher(stringCategory)).find())
    {
        genereActual = Genere.FEMENÍ;
        return CategoriaGramatical.NOM;
    }
    if((encaixa = Pattern.compile("adj").matcher(stringCategory)).find())
    {
        return CategoriaGramatical.ADJECTIU;
    }
    if((encaixa = Pattern.compile("\\Aadv").matcher(stringCategory)).find())
    {
        return CategoriaGramatical.ADVERBI;
    }
    if((encaixa = Pattern.compile("prep").matcher(stringCategory)).find())
    {
        return CategoriaGramatical.PREPOSICIO;
    }
    if((encaixa = Pattern.compile("conj").matcher(stringCategory)).find())
    {
        return CategoriaGramatical.CONJUNCIO;
    }
    if((encaixa = Pattern.compile("pron").matcher(stringCategory)).find())
    {
        return CategoriaGramatical.PRONOM;
    }
    return CategoriaGramatical.ALTRE;
}
```

Amb expressions regulars anem determinant si la paraula conté alguna de les categories gramaticals. La primera categoria gramatical que contingui serà la que se li assignarà, com a categoria gramatical a la paraula. En el codi anterior

es pot veure que la prioritat és: determinant, nom, adjectiu, adverbi, preposició, conjunció i pronom. En el cas dels noms ja determinem el seu gènere ja que es troba en el camp **CATEGORIAGRAMATICAL**.

Després d'assignar una categoria gramatical per cada entrada trobada acabem el mètode **consultaCategoriesPossibles** retornant a **addSeparator**, com ja s'ha dit, una llista on s'indica per cada categoria gramatical assignada els camps LEMA i ENTRADES concatenats.

Amb la llista de possibilitats, una altra vegada aplicant la mateixa prioritat que anteriorment en cas que hi hagi diverses opcions, realitzem una cosa o una altra depenent de la categoria gramatical de la paraula. En el cas que la categoria sigui un determinant obtindrem el gènere i el nombre d'aquest, en el cas d'un adjectiu o d'un nom només determinarem el gènere i nombre si abans no hem escrit un determinant, en el cas d'un pronom mirarem la persona i el nombre per poder concordar-lo amb els verbs. En altres categories gramaticals i en les anteriorment esmentades guardem la categoria en l'atribut **categoriaActual**. Ara veurem una mica més detalladament el que es fa en cada categoria:

- Els tres casos en els quals pot ser necessari obtenir el gènere i/o el nombre de la paraula són quan aquesta és un determinant, un nom o un adjectiu. En els tres actualitzarem l'atribut **categoriaActual** i, a més, indiquem que estem en tercera persona amb l'atribut **personaActual** ja que sempre que estem en aquests casos estarem en aquesta persona. Per obtenir el gènere i el nombre utilitzem la funció **buscarGenereINombre (String paraula, String possibilitats)** en el cas dels determinants i adjectius, i la funció **buscarNombre(String paraula, String possibilitats)** en el cas dels noms ja què el gènere s'ha determinant anteriorment quan buscàvem la categoria gramatical de la paraula escrita.

• En el cas dels adverbis només caldrà que actualitzem l'atribut **categoriaActual**.

• En el cas de les preposicions, a part d'actualitzar la categoria actual, hem d'esborrar el gènere i nombre actual dels atributs **genereActual** i **nombreActual**, i posar-los a indiferent**.** Això es fa perquè no mantinguem la concordança amb el que ve abans de la preposició. Considerem que pot venir una paraula de qualsevol gènere i nombre després de la preposició.

• En el cas de les conjuncions, haurem d'executar la funció **newFrase** que ens posa la variable de la categoria actual, del gènere actual, i del nombre actual a null. Per tant, les tractem com si ens fessin canviar de frase.

• En el cas dels pronoms, actualitzarem el tipus de categoria gramatical i executarem la funció **obtenirPersonaPronom (String pronom)** que, en el cas dels pronoms forts, obté la persona i el nombre d'aquest pronom per la posterior concordança amb el verb.

```
EnumMap<CategoriaGramatical,String> categoriesPossibles =
        consultaCategoriesPossibles(currentWord);

if(categoriesPossibles.containsKey(CategoriaGramatical.DETERMINANT))
{

        buscarGenereINombre(currentWord,categoriesPossibles.
                get(CategoriaGramatical.DETERMINANT).trim());

      categoriaActual = CategoriaGramatical.DETERMINANT;
      personaActual = Persona.TERCERA;
}
else if(categoriesPossibles.containsKey(CategoriaGramatical.NOM))
{
        if(categoriaActual != CategoriaGramatical.DETERMINANT)
```

```
        {
                buscarNombre(currentWord,categoriesPossibles.
                        get(CategoriaGramatical.NOM).trim());
            }

            personaActual = Persona.TERCERA;
            categoriaActual = CategoriaGramatical.NOM;
    }
    else if(categoriesPossibles.containsKey(CategoriaGramatical.ADJECTIU))
    {
            if(categoriaActual != CategoriaGramatical.DETERMINANT)
            {
                buscarGenereINombre(currentWord,categoriesPossibles.
                        get(CategoriaGramatical.ADJECTIU).trim());
            }

            categoriaActual = CategoriaGramatical.ADJECTIU;
            personaActual = Persona.TERCERA;
    }
    else if(categoriesPossibles.containsKey(CategoriaGramatical.ADVERBI))
    {
            categoriaActual = CategoriaGramatical.ADVERBI;
    }
    else if(categoriesPossibles.containsKey(CategoriaGramatical.PREPOSICIO))
    {
            genereActual = Genere.INDIFERENT;
            nombreActual = Nombre.INDIFERENT;
            categoriaActual = CategoriaGramatical.PREPOSICIO;
    }
    else if(categoriesPossibles.containsKey(CategoriaGramatical.CONJUNCIO))
    {
            newFrase();
    }
    else if(categoriesPossibles.containsKey(CategoriaGramatical.PRONOM))
    {
            obtenirPersonaPronom(currentWord);
            categoriaActual = CategoriaGramatical.PRONOM;
    }
```

La funció **buscarGenereINombre (String paraula, String possibilitats)** rep com a paràmetres la paraula escrita i la concatenació dels camps LEMA i ENTRADES. En el segon paràmetre hi haurà una cadena de caràcters amb les diferents possibilitats de gènere i nombre de l'entrada de la base de dades. Del que es tracta en aquesta funció és de veure quina possibilitat es correspon amb la paraula escrita, obtenint el seu gènere i el nombre. Per veure-ho millor, mostro els patrons:

```
Pattern mfsp = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*(\\p{L}*)");
Pattern mfp = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*)");
Pattern sp = Pattern.compile("(\\p{L}*) (\\p{L}*s)");
Pattern mf = Pattern.compile("(\\p{L}*) (\\p{L}*a|\\p{L}*)");
Pattern la = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*') (\\p{L}*) (\\p{L}*)");
Pattern en = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*')");
```

El primer d'ells correspon a quan hi ha variació de gènere i de nombre, el segon a quan només hi ha variació de gènere en el singular i el plural és comú, el tercer a quan hi ha només variació de nombre, el quart a quan hi ha només variació de gènere, el cinquè és específic pel determinant "la" per detectar quan s'escriu abreujat, i l'últim cas és específic per l'article "en" quan s'escriu abreujat.

Dins de la funció busquem el patró en el qual ens trobem. Un cop dins del patró, depenent del lloc que ocupi la paraula escrita dins d'ell, podrem saber el seu gènere i el nombre.

Per exemple, si tenim l'entrada "un una uns unes" i hem escrit "uns" encaixarem amb el primer patró i com que la paraula escrita coincideix amb la tercera opció sabem que és un article de gènere masculí i de nombre plural.

La funció **obtenirPersonaPronom (String pronom)** que executen els pronoms només mira si el pronom escrit pertany a algun pronom fort. En el cas que ho sigui, actualitza la persona actual i el nombre als que té el pronom, de manera que podrem fer concordar correctament el verb.

Fins ara detectem la categoria gramatical quan hem entrat una paraula utilitzant la funció **addSeparator**. A més, obtenim el gènere i el nombre i la persona el que ens permetrà predir paraules que concordin amb això.

L'altra funció important a la classe **CatalanPredictor** és el mètode **ArrayList <String> getCandidateWords()** que, depenent de la categoria actual, ens indicarà paraules de categories gramaticals perquè es mostrin a la pantalla de

predicció. En cas que s'hagi de concordar amb la paraula anterior només es mostren les opcions que són correctes gramaticalment. En aquesta funció, depenent del que vol l'usuari, també s'agafen les paraules possibles per freqüència. A la base de dades es consulten sempre les paraules que comencen per **currentWord** per reduir el nombre d'entrades obtingudes.

```
@Override
public ArrayList<String> getCandidateWords()
{
    ArrayList<String> paraules = new ArrayList<String>();

    if(!isStrictLanguagePrediction())
    {
        paraules.addAll(super.getCandidateWords());
    }

    if(writingWord||postApostrof||postGuionet)
    {
        if(categoriaActual == CategoriaGramatical.DETERMINANT)
        {
            paraules.addAll(buscarNomsConcordants(paraules));
            paraules.addAll(buscarAdjectiusConcordants(paraules));
        }
        else
        {
            paraules.addAll(buscarVerbsPossibles(paraules));
        }

        if(categoriaActual == CategoriaGramatical.ADVERBI)
        {
            paraules.addAll(buscarAdverbis(paraules));
            paraules.addAll(buscarAdjectiusConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.NOM)
        {
            paraules.addAll(buscarAdjectiusConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.ADJECTIU)
        {
            paraules.addAll(buscarNomsConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.PREPOSICIO)
        {
            paraules.addAll(buscarNomsConcordants(paraules));
        }
```

```
        }
        return paraules;
    }
```

En aquesta funció apareixen els mètodes **buscarNomsConcordants**, **buscarAdjectiusConcordants**, **buscarAdverbis** i **buscar VerbsPossibles**.

Les dues primeres funcions ens retornen els noms i els adjectius que concorden amb el gènere i el nombre actual. Depenent de si és un nom o un adjectiu es realitza una consulta o una altra a la base de dades per obtenir la categoria gramatical desitjada. En els dos casos utilitzem la funció **buscarConcordancia** per a cadascuna de les entrades que ens dóna la base de dades, per obtenir les paraules concordants. Aquesta funció de nou utilitza patrons per trobar la forma desitjada de la paraula. En aquest cas només necessitem dos patrons.

La funció **buscarAdverbis** consulta a la base de dades totes les paraules d'aquesta categoria gramatical i només els ha de retornar ja que no hi ha variació de gènere o nombre en aquesta categoria gramatical.

En el cas dels verbs s'utilitza la funció **buscarVerbsPossibles** que s'encarrega de generar els verbs que comencen per **currentWord** a partir dels infinitius de la base de dades. Com ja s'ha comentat, només es construiran els verbs que segueixen els models cantar, en el cas de la primera conjugació, i servir, en el cas de la tercera conjugació.

Per construir les terminacions en els diferents temps verbals s'ha creat una matriu per cada un dels models en la qual les columnes ens indiquen les sis combinacions de persona i nombre, i les files ens indiquen cada un dels temps verbals que es prediran. Concretament només es prediuen els temps d'indicatiu

present, imperfet, passat simple, futur i condicional.

```
public static String[][] modelCantar =
        {{"o","es","a","em","eu","en"}, //present
       {"ava","aves","ava","àvem","àveu","aven"}, //imperfet
       {"í","ares","à","àrem","àreu","aren"}, //passat simple
       {"aré","aràs","arà","arem","areu","aran"}, //futur
       {"aria","aries","aria","aríem","aríeu","arien"}}; //condicional

public static String[][] modelServir =
        {{"eixo","eixes","eix","im","iu","eixen"}, //present
       {"ia","ies","ia","íem","íeu","ien"}, //imperfet
       {"í","ires","í","írem","íreu","iren"}, //passat simple
       {"iré","iràs","irà","irem","ireu","iran"}, //futur
       {"iria","iries","iria","iriem","irieu","irien"}}; //condicional
```

La funció d'obtenir els verbs realitza una consulta per trobar els infinitius que segueixen el model cantar, i una altra, pels infinitius que segueixen el model servir. Per cada un dels verbs trobats s'obté l'arrel del verb amb la funció **obtenirArrel**. Un cop obtinguda l'arrel de la paraula, s'executarà la funció **conjugaVerb** que, depenent de la columna de la taula on s'indica la terminació obtinguda amb la funció **obtenirPersona**, obtindrà totes les possibilitats combinant l'arrel amb la terminació.

Com es pot veure en el codi de la funció **ArrayList <String> getCandidateWords()** mostrat una mica enrere, depenent de la categoria gramatical de la paraula anterior s'executaran algunes de les funcions explicades anteriorment.

Quan la categoria gramatical anterior és un determinant es prediuen els noms i adjectius que concorden amb el gènere i el nombre. D'aquesta manera es compleix la norma de que després d'un determinant ve un nom o un adjectiu.
Quan la categoria gramatical anterior és un adverbi, fem que es compleixi la regla que diu que després d'un adverbi normalment ve un altre adverbi o un adjectiu.

En cas que la categoria gramatical anterior sigui un nom prediem tots els adjectius que concorden en gènere i nombre. En el cas sigui un adjectiu prediem tots els noms que concorden en gènere i nombre amb ell.

En el cas de les preposicions prediem tots els noms, en aquest cas sense tenir en compte cap concordança ja que no existeix.

En tots els casos anteriors, menys en el cas en què la categoria anterior sigui un determinant, es prediuen els verbs que es puguin escriure ja que pràcticament ens podem trobar un verb en qualsevol part d'una frase. En el cas que hi hagi una preposició davant només es prediran els verbs en infinitiu.

En totes les funcions explicades que busquen a la base de dades paraules d'una determinada categoria, se'ls passa un paràmetre amb la llista de paraules ja seleccionades per mostrar a la predicció, aquesta llista serveix perquè no repetim la paraula en les opcions que mostrem. Aquesta llista és usada només quan utilitzem la forma de predicció en el qual es combinen la predicció per freqüència i la predicció en català. En aquest cas, cada vegada que obtinguem una paraula hem de mirar que no estigui ja a la llista. Per fer-ho, utilitzem la funció **cercaDicotomica**, que ens permet una cerca amb un temps d'execució bastant raonable.

Com s'ha pogut veure durant l'explicació d'aquest apartat el predictor seguint les regles del català es limita a les regles que es van considerar que podrien ajudar més a escriure de forma ràpida. Aquest predictor pretén ser una petita base per a futures ampliacions i es limita a algunes coses molt concretes del llenguatge.

La base de dades utilitzada per aquesta predicció ha causat que la

implementació d'aquesta part sigui una mica més complexa, ja que la seva estructura de guardar les variacions de les paraules ha fet que haguem de fer servir el sistema de patrons que s'ha explicat. Per exemple, si tinguéssim cadascuna de les variacions de les paraules en una sola entrada indicant el seu gènere i el nombre, la recerca de paraules a la base de dades seria molt més fàcil.

Durant la realització d'aquesta part es va plantejar la possibilitat de reorganitzar la base de dades, de forma que cada variació de la paraula fóra en una entrada diferent, amb una operació que només s'executés una vegada. Això pot ser una de les primeres millores que es podrien implementar en aquest sistema de predicció.

En algunes categories gramaticals, com en el cas dels adverbis, no se'ns mostra a la base de dades els diferents tipus d'aquestes categoria. Per exemple, se'ns podria indicar si són adverbis de temps, de lloc, etc. D'aquesta manera es podrien introduir al predictor aspectes semàntics del llenguatge.

Durant la realització d'aquest apartat s'ha constatat la dificultat de treballar en el camp de tractament del llenguatge natural. En català, a més, les dificultats de la llengua són més grans que en altres idiomes a causa de la complexitat del llenguatge.

3.3. Estructura final

Un cop explicada tota la implementació que s'ha realitzat del programa, en aquest apartat s'explica les possibilitats que té l'usuari de configuració del programa i d'ús de les interfícies gràfiques.

Durant la realització del projecte s'ha descobert el funcionament de les APIS de les preferències en Java. Elles s'encarreguen de que en cada sistema operatiu es guardin les preferències de l'usuari del programa. En els diferents sistemes

operatius on funciona el programa les preferències es guarden a un lloc específic que té el sistema per guardar-les.

Al programa es guarden com a preferències la posició de teclat virtual en pantalla, el nom d'usuari i la contrasenya de la base de dades, i el temps d'espera per a la funció d'escriure mantenint el cursor sobre la tecla. Aquestes preferències són carregades en el moment d'inicialització del programa, i són guardades quan es tanca el programa. Coneixent el funcionament d'aquestes APIS es podrien estendre a altres paràmetres del teclat virtual.

Començam primer per l'explicació de la interfície gràfica que conté el teclat virtual. Aquesta està definida a la classe **MainWindow**. Aquesta finestra conté els menús File, Edit, Keyboard, Hide prediction i Clear text, com es pot veure a la Figura 3.3.1.

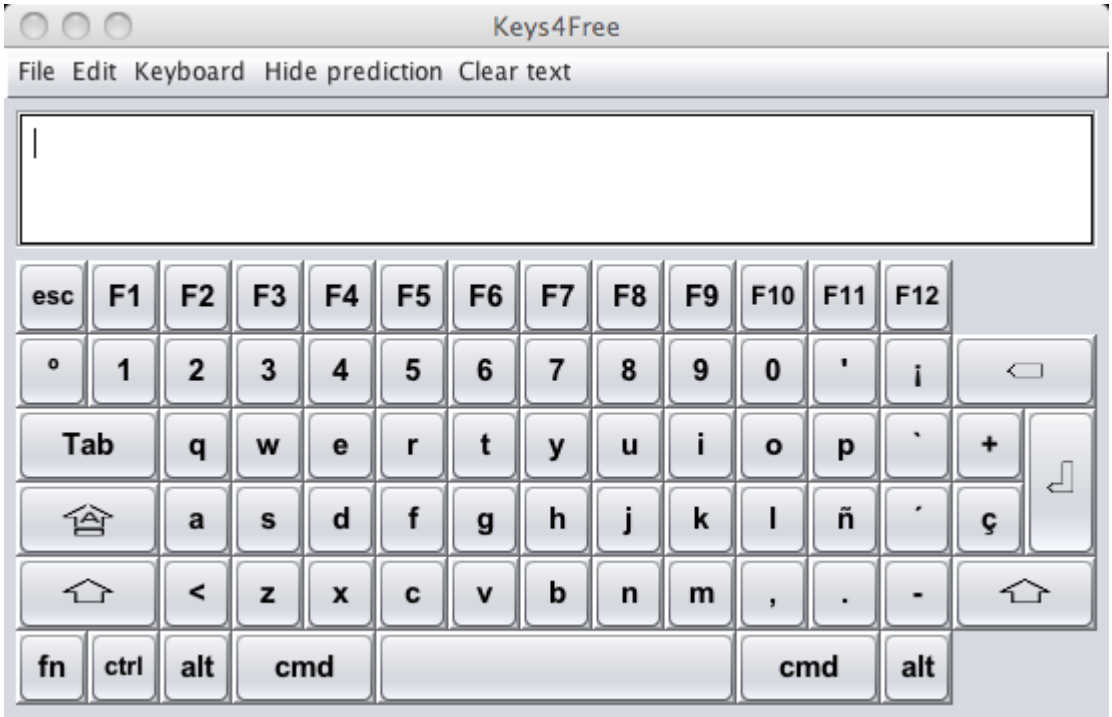

Figura 3.3.1. Imatge de la pantalla principal

Al menú **File,** tenim l'opció **Database options** que ens permet canviar els paràmetres de connexió amb la base de dades del programa. Concretament permet indicar el nom d'usuari i la contrasenya a utilitzar. La pantalla per canviar la configuració es mostra a la Figura 3.3.2.

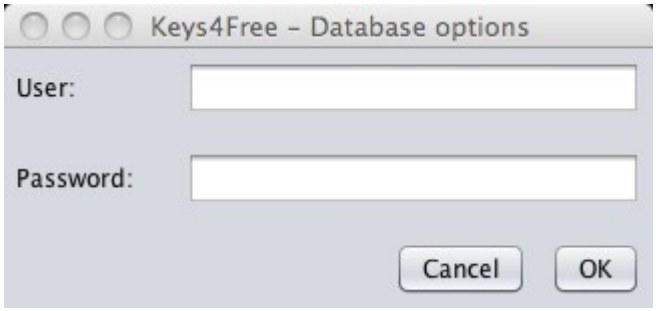

Figura 3.3.2. Pantalla de configuració de la base de dades

En el cas que, durant l'execució del programa no ens puguem connectar a la base de dades, ja sigui perquè no tenim els paràmetres ben configurats o perquè la base de dades no està corrent , es mostrarà un missatge d'error semblant al de la Figura 3.3.3 que és per quan no es poden carregar els noms dels diccionaris.

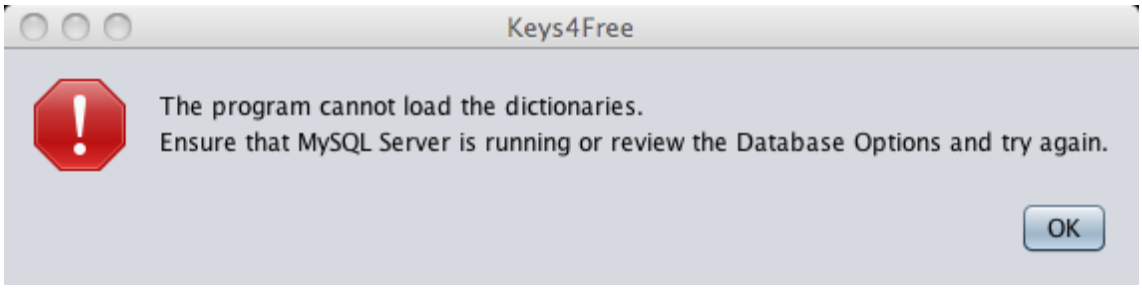

Figura 3.3.3. Missatge d'error de l'aplicació

Al menú **Keyboard** tenim l'opció **Dwell active** que permet seleccionar o des-seleccionar un check box indicant si volem utilitzar la funció d'escriure mantenint el cursor sobre la tecla. També tenim l'opció **Dwell time** que ens farà aparèixer una finestra en la qual podrem escollir el temps d'espera de la

funció anterior. Aquesta finestra es mostra a la Figura 3.3.4.

Figura 3.3.4. Pantalla de selecció del temps de manteniment del cursor

Els altres dos menús ens permeten, d'una banda amagar la finestra de predicció al menú **Hide prediction**, i d'altra banda esborrar el text en la caixa de text, al menú **Clear text**.

Pel que fa a la finestra de predicció definida a la classe **PredictorWindow** els menús que es poden utilitzar són **Language** i **Dictionary.** La Figura 3.3.5 mostra la finestra de predicció.

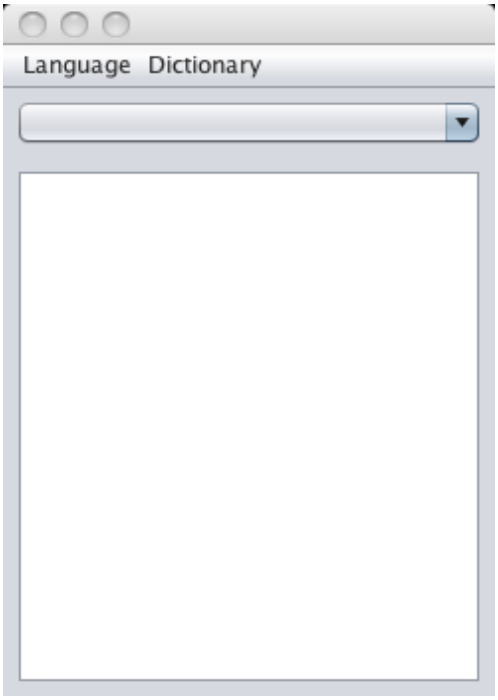

Figura 3.3.5. Pantalla de predicció

En el primer d'ells tenim les opcions de configuració del predictor en català. Concretament, tenim un check box anomenat **Predict Català** que permet activar o desactivar aquesta manera de predicció, i un altre anomenat **Strict grammar** que permet indicar si volem mantenir la funcionalitat de predictor per freqüència o si només volem predicció en català.

Al menú anomenat **Dictionary hi ha** les opcions de gestió dels diccionaris i de la forma d'aprenentatge del predictor usat.

La primera opció que tenim es diu **Create** i permet crear un nou diccionari introduint el nom a la pantalla que es mostra a la Figura 3.3.6.

Figura 3.3.6. Pantalla de creació de diccionari

La segona opció permet eliminar diccionaris de la base de dades juntament amb les seves paraules i es diu **Delete**. La finestra que ens apareix permet seleccionar entre els diccionaris creats per escollir quin es vol eliminar. Veure Figura 3.3.7.

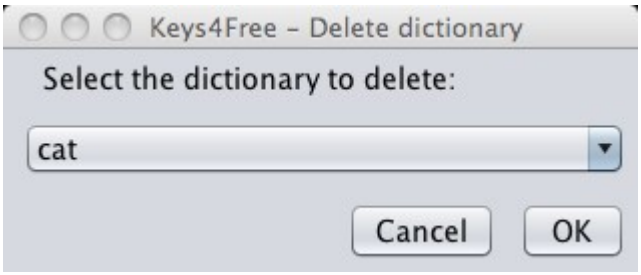

Figura 3.3.7. Pantalla per eliminar diccionaris

La tercera opció permet carregar els noms dels diccionaris que estan creats. Aquesta opció s'anomena **Load dictionaries** i només es pot utilitzar si en el moment de la inicialització del programa aquests diccionaris no s'han pogut carregar a causa d'un error en la configuració de la base de dades. Això se'ns indicarà tal com s'ha dit anteriorment amb un missatge.

L'última opció que tenim en aquest menú, permet indicar si volem que el predictor aprengui paraules noves o no. Aquesta opció s'anomena **Learning model** i ja s'ha explicat on influeix anteriorment.

En aquest apartat només es volien mostrar les opcions que té l'usuari de

configuració del programa. Òbviament aquestes es poden ampliar per oferir millors funcionalitats a l'usuari, però per al propòsit del projecte aquestes mínimes opcions ja eren representatives.

4. Conclusions

Com s'ha anat dient durant tota l'explicació de la memòria, el projecte pretenia ser una base d'un teclat virtual perquè posteriorment es poguessin realitzar ampliacions i millores a partir d'una base ja consolidada. Durant tota l'explicació ja s'han anat apuntant algunes de les millores que es podrien fer en el programa en una primera fase.

Pel que fa als objectius que es volien realitzar s'han obtingut els següents resultats:

El primer objectiu era que es pogués escriure en diferents aplicacions sense utilitzar el teclat físic de l'ordinador. A més, es pretenia aconseguir una velocitat raonable d'escriptura.

El programa realitzat ha aconseguit que es pogués escriure en les diferents aplicacions del sistema operatiu on s'executarà el programa utilitzant la interfície del teclat virtual creat per substituir el teclat físic. Amb el sistema de predicció implementat la velocitat d'escriptura és bastant més raonable que si aquest sistema no existís.

El segon objectiu era que es pogués realitzar predicció per freqüència o utilitzant algunes regles del català. Aquests dos elements permetrien obtenir una freqüència raonable d'escriptura. Com s'ha explicat, aquests dos tipus de predicció s'han implementat dins del programa.

El sistema predictiu creat permet la combinació dels dos tipus de predicció, fent que en el cas d'escriure en català aquesta velocitat d'escriptura sigui més ràpida ja que tindrem a la nostra disposició totes les paraules de Cercamots.

En el cas d'altres idiomes hem d'utilitzar el predictor per freqüència, que també ens permetrà obtenir una bona velocitat quan hi hagi un bon nombre de

paraules en el diccionari que utilitzem.

El tercer objectiu era aconseguir l'execució del programa en diferents sistemes operatius assegurant la funcionalitat en un d'ells i posteriorment ampliar-ho a altres. El programa creat ha implementat totalment el teclat virtual en els sistemes operatius Windows i Mac i, com ja s'ha comentat, en Ubuntu la implementació ha estat parcial. Per tant, respecte a l'objectiu marcat, s'ha aconseguit poder obtenir la funcionalitat desitjada en algun sistema operatiu i després fer-la extensiva a altres.

Amb tot això, es pot dir que s'han complert els objectius que es volien aconseguir abans de la realització del projecte final de carrera.

En el projecte es podria haver dedicat més temps en algun dels seus diferents aspectes però s'havien de complir les limitacions de temps dels termes de la realització del projecte. A més, com que es volien tractar els diferents aspectes que són necessaris per a la creació d'un teclat virtual, tampoc s'ha pogut aprofundir molt en cada un d'aquests.

A partir d'aquest projecte es pot anar aprofundint en cada un d'aquests aspectes, de manera que s'obtingui un teclat virtual molt més potent i amb més opcions per als usuaris que pugui tenir.

5. Bibliografia

Las referències utilitzades para la realització del projecte es presenten agrupades segons la part del programa a la qual es refereixen.

Teclat virtual

Organització de les tecles dintre el teclat

- http://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html
- http://netbeans.org/kb/docs/java/gbcustomizer-basic.html
- http://netbeans.org/kb/docs/java/gbcustomizer-advanced.html

Esdeveniments en Java

- http://www.scribd.com/doc/3118986/Events-in-JAVA
- http://www.javaworld.com/javaworld/javaqa/2000-08/01-qa-0804-events.html?page=1
- http://www.exampledepot.com/egs/java.util/CustEvent.html

Part nativa del sistema operatiu

Llibreries JNI

- http://www.atwistedweb.com/java/jni.html
- http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jni.html
- LIANG, Sheng**.** *The JavaTM Native Interface Programmer's Guide and Specification*. 1999 Sun Microsystems, Inc.http://java.sun.com/docs/books/jni/download/jni.pdf

Windows

- Win 32 SDK Online Help
- http://msdn.microsoft.com/en-us/library/ms645530(v=VS.85).aspx

X11/Linux

- http://tronche.com/gui/x/xlib/
- http://cnd.netbeans.org/docs/jni/nb6-linux/beginning-jni-linux.html
- http://ubuntuforums.org/archive/index.php/t-864566.html

Mac OS X

- http://developer.apple.com/library/mac/#technotes/tn2147/_index.html
- http://developer.apple.com/library/mac/#documentation/Java/Conceptual/Java14Development/05-CoreJavaAPIs/CoreJavaAPIs.html
- *The Objective-C Programming Language*2 2002 Apple Computer, Inc.http://pj.freefaculty.org/ps905/ObjC.pdf
- http://developer.apple.com/library/mac/#documentation/CrossPlatform/Reference/JavaNativeFoundation_Functions/Reference/reference.html

Estructura de predicció

- https://blog.itu.dk/SGDM-F2011/files/2011/03/52tries-jeslkey.pdf
- BENTLEY, J.L. ; SEDGEWICK, R. "Fast Algorithms for Sorting and Searching Strings" Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, January 1997 http://www.cs.princeton.edu/~rs/strings/paper.pdf
- http://drdobbs.com/windows/184410528
- www.ki.informatik.hu-berlin.de/wbi/.../ss11/.../16_optimal_trees.pdf

Base de dades

- http://dev.mysql.com/doc/refman/5.0/en/index.html
- http://dev.mysql.com/usingmysql/java/
- http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

Predicció en català

- BAPTISTA XURIGUERA, Joan. *Els verbs catalans conjugats*. Editorial Claret 10à edició, Barcelona, 1981
- FABRA, Pompeu. *Gramàtica catalana*. Institut d'Estudis Catalans, 7à edició, Barcelona, 1995
- http://en.wikipedia.org/wiki/Conjugation_of_regular_Catalan_verbs

Preferències d'usuari en Java

- http://java.sun.com/developer/technicalArticles/releases/preferences/
- http://java.sun.com/developer/Books/javaprogramming/jdk14/javapch10.PDF

6. Annex: Codi del programa

6.1. Codi de Java

6.1.1. AbstractKeyboard

```
package keysforfree;

import javax.swing.JPanel;
import javax.swing.JTextArea;

/**
 * Classe abstracta que defineix les propietats bàsiques dels teclats. Hereta
 * de la classe gràfica JPanel, que permet que es pugui mostrar gràficament.
 * @author eloicreus
 */
abstract public class AbstractKeyboard extends JPanel
{
    /*
     * Tecles modificadores que controlaran el teclat
     */
    protected KeyboardKey capsLock;
    protected KeyboardKey shiftLeft;
    protected KeyboardKey altLeft;
    protected KeyboardKey comandLeft;
    protected KeyboardKey comandRight;
    protected KeyboardKey altRight;
    protected KeyboardKey shiftRight;
    protected KeyboardKey openAccent;
    protected KeyboardKey closeAccent;
    protected KeyboardKey control;

    /**
     * Funció que permet associar a les tecles creades gràficament la funció de
     * modificador que volem per elles. Associem també el teclat a la caixa de
     * text on han d'escriure les tecles.
     * @param jTextArea1
     * @param keyboardKeyCapsLock
     * @param keyboardKeyShiftLeft
     * @param keyboardKeyAltLeft
     * @param keyboardKeyComandLeft
     * @param keyboardKeyComandRight
     * @param keyboardKeyAltRight
     * @param keyboardKeyShiftRight
     * @param keyboardKeyControl
     */
    protected void setKeyboardProperties(JTextArea jTextArea1,
            KeyboardKey keyboardKeyCapsLock, KeyboardKey keyboardKeyShiftLeft,
            KeyboardKey keyboardKeyAltLeft, KeyboardKey keyboardKeyComandLeft,
            KeyboardKey keyboardKeyComandRight, KeyboardKey keyboardKeyAltRight,
            KeyboardKey keyboardKeyShiftRight, KeyboardKey keyboardKeyControl)

    {
```

```
        capsLock = keyboardKeyCapsLock;
        shiftLeft = keyboardKeyShiftLeft;
        altLeft = keyboardKeyAltLeft;
        comandLeft = keyboardKeyComandLeft;
        comandRight = keyboardKeyComandRight;
        altRight = keyboardKeyAltRight;
        shiftRight = keyboardKeyShiftRight;
        control = keyboardKeyControl;

        output = jTextArea1;
    }

    /**
     * Caixa de text on han d'escriure les tecles
     */
    JTextArea output;

    /*
     * Indiquen l'estat dels modificadors.
     */
    boolean isShift = false;
    boolean isAlt = false;
    boolean isCapsLock = false;
    boolean isFunction = false;
    boolean isControl = false;
    boolean isComand = false;
    boolean isOpenAccent = false;
    boolean isCloseAccent = false;
    boolean isDiaeresis = false;

    /**
     * Funció nativa que envia un caràcter o una paraula a l'aplicació activa
     * del sistema.
     * @param keyTyped Caràcter o paraula a escriure
     */
    public native void setKeyAtComponent(String keyTyped);
    /**
     * Funció nativa que envia el nom del modificador que s'ha premut per que
     * l'apliqui a l'aplicació activa del sistema
     * @param modifier Nom del modificador premut
     */
    public native void setTextModifier(String modifier);

    /**
     * Indica si esta activada la pulsació de tecles mantenint el cursor a sobre
     */
    protected boolean dwellEnabled = false;

    /**
     * Canvia l'estat gràfic dels modificadors que li diuen
     * @param state Indica si s'ha d'activar o desactivar
     * @param keys  Indica les tecles a modificar
     */
```

```
public void changeModifierState(boolean state, KeyboardKey ... keys)
{
    for(KeyboardKey key : keys)
        key.getModel().setPressed(state);
}

/**
 * @return the capsLock
 */
public KeyboardKey getCapsLock()
{
    return capsLock;
}

/**
 * @return the shiftLeft
 */
public KeyboardKey getShiftLeft()
{
    return shiftLeft;
}

/**
 * @return the altLeft
 */
public KeyboardKey getAltLeft()
{
    return altLeft;
}

/**
 * @return the comandLeft
 */
public KeyboardKey getComandLeft()
{
    return comandLeft;
}

/**
 * @return the comandRight
 */
public KeyboardKey getComandRight()
{
    return comandRight;
}

/**
 * @return the altRight
 */
public KeyboardKey getAltRight()
{
    return altRight;
}
```

```
/**
 * @return the shiftRight
 */
public KeyboardKey getShiftRight()
{
    return shiftRight;
}

/**
 * @param capsLock the capsLock to set
 */
public void setCapsLock(KeyboardKey capsLock)
{
    this.capsLock = capsLock;
}

/**
 * @param shiftLeft the shiftLeft to set
 */
public void setShiftLeft(KeyboardKey shiftLeft)
{
    this.shiftLeft = shiftLeft;
}

/**
 * @param altLeft the altLeft to set
 */
public void setAltLeft(KeyboardKey altLeft)
{
    this.altLeft = altLeft;
}

/**
 * @param comandLeft the comandLeft to set
 */
public void setComandLeft(KeyboardKey comandLeft)
{
    this.comandLeft = comandLeft;
}

/**
 * @param comandRight the comandRight to set
 */
public void setComandRight(KeyboardKey comandRight)
{
    this.comandRight = comandRight;
}

/**
 * @param altRight the altRight to set
 */
public void setAltRight(KeyboardKey altRight)
```

```
{
    this.altRight = altRight;
}

/**
 * @param shiftRight the shiftRight to set
 */
public void setShiftRight(KeyboardKey shiftRight)
{
    this.shiftRight = shiftRight;
}

/**
 * @return the openAccent
 */
public KeyboardKey getOpenAccent()
{
    return openAccent;
}

/**
 * @param openAccent the openAccent to set
 */
public void setOpenAccent(KeyboardKey openAccent)
{
    this.openAccent = openAccent;
}

/**
 * @return the closeAccent
 */
public KeyboardKey getCloseAccent()
{
    return closeAccent;
}

/**
 * @param closeAccent the closeAccent to set
 */
public void setCloseAccent(KeyboardKey closeAccent)
{
    this.closeAccent = closeAccent;
}

/**
 * @return the control
 */
public KeyboardKey getControl()
{
    return control;
}

/**
```

```
     * @param control the control to set
     */
    public void setControl(KeyboardKey control)
    {
        this.control = control;
    }

    /**
     * @return the dwellEnabled
     */
    public boolean isDwellEnabled()
    {
        return dwellEnabled;
    }

    /**
     * @param dwellEnabled the dwellEnabled to set
     */
    public void setDwellEnabled(boolean dwellEnabled)
    {
        this.dwellEnabled = dwellEnabled;
    }
}
```

6.1.2. BasicKeyboard

```
package keysforfree;

import javax.swing.JTextArea;
import javax.swing.event.EventListenerList;

/**
 * Classe que implementa la representació gràfica del teclat virtual
 * @author eloicreus
 */
public class BasicKeyboard extends AbstractKeyboard
{
    /**
     * Creem les tecles que formen part del teclat i les empaquetem.
     * Indiquem a les tecles que s'associïn amb aquest teclat i indiquem al
     * teclat les tecles que faran de modificadors.
     * També indiquem la caixa de text on escriurà el teclat.
     * També registrem els esdeveniments que ha d'escoltar cada tecla.
     * @param textOutput Caixa de text on s'ha d'escriure.
     */
    public BasicKeyboard(JTextArea textOutput)
    {
        initComponents();
        KeyboardKey.setKeyboard(this);
        setKeyboardProperties(textOutput,keyboardKeyCapsLock,
                keyboardKeyShiftLeft,keyboardKeyAltLeft,keyboardKeyComandLeft,
                keyboardKeyComandRight,keyboardKeyAltRight,keyboardKeyShiftRight,
                keyboardKeyControl);
        setOpenAccent(keyboardKeyOpenAccent);
        setCloseAccent(keyboardKeyCloseAccent);

        initLettersListeners();
    }

    public BasicKeyboard()
    {
        initComponents();
    }

    /**
     * Funció que s'encarrega de registrar els esdeveniments a cada tecla.
     * A totes les tecles les registrem perquè rebin l'esdeveniment d'ajustament
     * de text i els esdeveniments per interactuar amb el cursor.
     * També registrem les tecles perquè rebin els esdeveniments de modificador,
     * depenent del tipus de tecla.
     */
    private void initLettersListeners()
    {
        int numComponents = getComponentCount();

        for(int i=0;i<=numComponents-1;i++)
        {
```

```
                KeyboardKey   key = (KeyboardKey) getComponent(i);

                addTextAdjustEventListener(key);
                key.addMouseListener(key.mouseAdapter);

                 if(key.keyType == KeyType.CHARACTER || key.keyType ==
KeyType.VARCHARACTER)
                {
                     shiftLeft.addModifierEventListener(key);
                     shiftRight.addModifierEventListener(key);

                    if(key.keyType == KeyType.CHARACTER)
capsLock.addModifierEventListener(key);
                }
            }
            capsLock.addModifierEventListener(keyboardKeyÇ);
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        keyboardKeySpace = new keysforfree.KeyboardKey(KeyType.TEXTMODIFIER);
        keyboardKeyEsc = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF1 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF2 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF3 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF4 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF5 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF6 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF7 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF8 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF9 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF10 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF11 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyF12 = new keysforfree.KeyboardKey(KeyType.OPTION);
        keyboardKeyGrade = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey1 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyTab = new keysforfree.KeyboardKey(KeyType.TEXTMODIFIER);
        keyboardKeyCapsLock = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyShiftLeft = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyFunction = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKey2 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey3 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey4 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey5 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey6 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey7 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey8 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKey9 = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keysforfree.KeyboardKey keyboardKey0 = new
keysforfree.KeyboardKey(KeyType.VARCHARACTER);
```

```
        keysforfree.KeyboardKey keyboardKeyApostrophe = new
keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyCloseAdmiration = new
keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyBackspace = new keysforfree.KeyboardKey(KeyType.TEXTMODIFIER);
        keyboardKeyQ = new keysforfree.KeyboardKey();
        keyboardKeyW = new keysforfree.KeyboardKey();
        keyboardKeyE = new keysforfree.KeyboardKey();
        keyboardKeyR = new keysforfree.KeyboardKey();
        keyboardKeyT = new keysforfree.KeyboardKey();
        keyboardKeyY = new keysforfree.KeyboardKey();
        keyboardKeyU = new keysforfree.KeyboardKey();
        keyboardKeyI = new keysforfree.KeyboardKey();
        keyboardKeyO = new keysforfree.KeyboardKey();
        keyboardKeyP = new keysforfree.KeyboardKey();
        keyboardKeyOpenAccent = new
keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyPlus = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyEnter = new keysforfree.KeyboardKey(KeyType.TEXTMODIFIER);
        keyboardKeyA = new keysforfree.KeyboardKey();
        keyboardKeyS = new keysforfree.KeyboardKey();
        keyboardKeyD = new keysforfree.KeyboardKey();
        keyboardKeyF = new keysforfree.KeyboardKey();
        keyboardKeyG = new keysforfree.KeyboardKey();
        keyboardKeyH = new keysforfree.KeyboardKey();
        keyboardKeyJ = new keysforfree.KeyboardKey();
        keyboardKeyK = new keysforfree.KeyboardKey();
        keyboardKeyL = new keysforfree.KeyboardKey();
        keyboardKeyÑ = new keysforfree.KeyboardKey();
        keyboardKeyCloseAccent = new
keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyÇ = new keysforfree.KeyboardKey(KeyType.CHARACTER);
        keyboardKeyLessThen = new
keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyZ = new keysforfree.KeyboardKey();
        keyboardKeyX = new keysforfree.KeyboardKey();
        keyboardKeyC = new keysforfree.KeyboardKey();
        keyboardKeyV = new keysforfree.KeyboardKey();
        keyboardKeyB = new keysforfree.KeyboardKey();
        keyboardKeyN = new keysforfree.KeyboardKey();
        keyboardKeyM = new keysforfree.KeyboardKey();
        keyboardKeyComma = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyDot = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyLess = new keysforfree.KeyboardKey(KeyType.VARCHARACTER);
        keyboardKeyShiftRight = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyControl = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyAltLeft = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyComandLeft = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyComandRight = new keysforfree.KeyboardKey(KeyType.MODIFIER);
        keyboardKeyAltRight = new keysforfree.KeyboardKey(KeyType.MODIFIER);

        addComponentListener(new java.awt.event.ComponentAdapter() {
            public void componentResized(java.awt.event.ComponentEvent evt) {
```

```
                keyboardResized(evt);
            }
        });
        setLayout(new java.awt.GridBagLayout());

        keyboardKeySpace.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeySpace.setAlignmentY(0.0F);
        keyboardKeySpace.setFocusable(false);
        keyboardKeySpace.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeySpace.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeySpace.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeySpace.setMinimumSize(new java.awt.Dimension(25, 5));
        keyboardKeySpace.setName("Space"); // NOI18N
        keyboardKeySpace.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 5;
        gridBagConstraints.gridwidth = 5;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeySpace, gridBagConstraints);

        keyboardKeyEsc.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.
border.BevelBorder.RAISED));
        keyboardKeyEsc.setText("esc");
        keyboardKeyEsc.setAlignmentY(0.0F);
        keyboardKeyEsc.setFocusable(false);
        keyboardKeyEsc.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyEsc.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyEsc.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyEsc.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyEsc.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyEsc, gridBagConstraints);

        keyboardKeyF1.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF1.setText("F1");
        keyboardKeyF1.setAlignmentY(0.0F);
        keyboardKeyF1.setFocusable(false);
        keyboardKeyF1.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
```

```
        keyboardKeyF1.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF1.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF1.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF1, gridBagConstraints);

        keyboardKeyF2.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF2.setText("F2");
        keyboardKeyF2.setAlignmentY(0.0F);
        keyboardKeyF2.setFocusable(false);
        keyboardKeyF2.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF2.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF2.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF2.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF2.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 2;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF2, gridBagConstraints);

        keyboardKeyF3.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF3.setText("F3");
        keyboardKeyF3.setAlignmentY(0.0F);
        keyboardKeyF3.setFocusable(false);
        keyboardKeyF3.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF3.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF3.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF3.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF3.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 3;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF3, gridBagConstraints);
```

```
        keyboardKeyF4.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF4.setText("F4");
        keyboardKeyF4.setAlignmentY(0.0F);
        keyboardKeyF4.setFocusable(false);
        keyboardKeyF4.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF4.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF4.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF4.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF4.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 4;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF4, gridBagConstraints);

        keyboardKeyF5.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF5.setText("F5");
        keyboardKeyF5.setAlignmentY(0.0F);
        keyboardKeyF5.setFocusable(false);
        keyboardKeyF5.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF5.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF5.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF5.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF5.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF5, gridBagConstraints);

        keyboardKeyF6.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF6.setText("F6");
        keyboardKeyF6.setAlignmentY(0.0F);
        keyboardKeyF6.setFocusable(false);
        keyboardKeyF6.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF6.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF6.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF6.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF6.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 6;
        gridBagConstraints.gridy = 0;
```

```
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF6, gridBagConstraints);

        keyboardKeyF7.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF7.setText("F7");
        keyboardKeyF7.setAlignmentY(0.0F);
        keyboardKeyF7.setFocusable(false);
        keyboardKeyF7.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF7.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF7.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF7.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF7.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 7;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF7, gridBagConstraints);

        keyboardKeyF8.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF8.setText("F8");
        keyboardKeyF8.setAlignmentY(0.0F);
        keyboardKeyF8.setFocusable(false);
        keyboardKeyF8.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF8.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF8.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF8.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF8.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 8;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF8, gridBagConstraints);

        keyboardKeyF9.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.b
order.BevelBorder.RAISED));
        keyboardKeyF9.setText("F9");
        keyboardKeyF9.setAlignmentY(0.0F);
        keyboardKeyF9.setFocusable(false);
        keyboardKeyF9.setFont(new java.awt.Font("Arial", 1, 14));
```

```
        keyboardKeyF9.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF9.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF9.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF9.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 9;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF9, gridBagConstraints);

        keyboardKeyF10.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.
border.BevelBorder.RAISED));
        keyboardKeyF10.setText("F10");
        keyboardKeyF10.setAlignmentY(0.0F);
        keyboardKeyF10.setFocusable(false);
        keyboardKeyF10.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF10.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF10.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF10.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF10.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 10;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF10, gridBagConstraints);

        keyboardKeyF11.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.
border.BevelBorder.RAISED));
        keyboardKeyF11.setText("F11");
        keyboardKeyF11.setAlignmentY(0.0F);
        keyboardKeyF11.setFocusable(false);
        keyboardKeyF11.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF11.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF11.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF11.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF11.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 11;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF11, gridBagConstraints);
```

```
        keyboardKeyF12.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.
border.BevelBorder.RAISED));
        keyboardKeyF12.setText("F12");
        keyboardKeyF12.setAlignmentY(0.0F);
        keyboardKeyF12.setFocusable(false);
        keyboardKeyF12.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyF12.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyF12.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyF12.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyF12.setPreferredSize(new java.awt.Dimension(20, 20));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 12;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyF12, gridBagConstraints);

        keyboardKeyGrade.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyGrade.setText("º");
        keyboardKeyGrade.setAlignmentY(0.0F);
        keyboardKeyGrade.setFocusable(false);
        keyboardKeyGrade.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyGrade.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyGrade.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyGrade.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyGrade.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyGrade.setCharacters("º","ª","\\");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 0;
            gridBagConstraints.gridy = 1;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyGrade, gridBagConstraints);

            keyboardKey1.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
            keyboardKey1.setText("1");
            keyboardKey1.setAlignmentY(0.0F);
            keyboardKey1.setFocusable(false);
            keyboardKey1.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKey1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKey1.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKey1.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKey1.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKey1.setCharacters("1","!","|");
```

```
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 1;
                gridBagConstraints.gridy = 1;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKey1, gridBagConstraints);

                keyboardKeyTab.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.s
wing.border.BevelBorder.RAISED));
                keyboardKeyTab.setText("Tab");
                keyboardKeyTab.setAlignmentY(0.0F);
                keyboardKeyTab.setFocusable(false);
                keyboardKeyTab.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyTab.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER
);
                keyboardKeyTab.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyTab.setMinimumSize(new java.awt.Dimension(10, 5));
                keyboardKeyTab.setName("Tab"); // NOI18N
                keyboardKeyTab.setPreferredSize(new java.awt.Dimension(20, 20));
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 0;
                gridBagConstraints.gridy = 2;
                gridBagConstraints.gridwidth = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyTab, gridBagConstraints);

                keyboardKeyCapsLock.setBorder(javax.swing.BorderFactory.createBevelBorder(ja
vax.swing.border.BevelBorder.RAISED));
                keyboardKeyCapsLock.setAlignmentY(0.0F);
                keyboardKeyCapsLock.setFocusable(false);
                keyboardKeyCapsLock.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyCapsLock.setHorizontalTextPosition(javax.swing.SwingConstants.CE
NTER);
                keyboardKeyCapsLock.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyCapsLock.setMinimumSize(new java.awt.Dimension(10, 5));
                keyboardKeyCapsLock.setName("CapsLock"); // NOI18N
                keyboardKeyCapsLock.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyCapsLock.setImageIcon("/keysforfree/images/CapsLock.png");
                keyboardKeyCapsLock.setHasIcon(true);
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 0;
                gridBagConstraints.gridy = 3;
                gridBagConstraints.gridwidth = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
```

```
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyCapsLock, gridBagConstraints);

                keyboardKeyShiftLeft.setBorder(javax.swing.BorderFactory.createBevelBorder(jav
ax.swing.border.BevelBorder.RAISED));
                keyboardKeyShiftLeft.setAlignmentY(0.0F);
                keyboardKeyShiftLeft.setFocusable(false);
                keyboardKeyShiftLeft.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyShiftLeft.setHorizontalTextPosition(javax.swing.SwingConstants.CEN
TER);
                keyboardKeyShiftLeft.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyShiftLeft.setMinimumSize(new java.awt.Dimension(10, 5));
                keyboardKeyShiftLeft.setName("ShiftLeft"); // NOI18N
                keyboardKeyShiftLeft.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyShiftLeft.setImageIcon("/keysforfree/images/Shift.png");
                keyboardKeyShiftLeft.setHasIcon(true);
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 0;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.gridwidth = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyShiftLeft, gridBagConstraints);

                keyboardKeyFunction.setBorder(javax.swing.BorderFactory.createBevelBorder(jav
ax.swing.border.BevelBorder.RAISED));
                keyboardKeyFunction.setText("fn");
                keyboardKeyFunction.setAlignmentY(0.0F);
                keyboardKeyFunction.setFocusable(false);
                keyboardKeyFunction.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyFunction.setHorizontalTextPosition(javax.swing.SwingConstants.CEN
TER);
                keyboardKeyFunction.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyFunction.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyFunction.setName("Function"); // NOI18N
                keyboardKeyFunction.setPreferredSize(new java.awt.Dimension(20, 20));
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 0;
                gridBagConstraints.gridy = 5;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyFunction, gridBagConstraints);

                keyboardKey2.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
                keyboardKey2.setText("2");
```

```
        keyboardKey2.setAlignmentY(0.0F);
        keyboardKey2.setFocusable(false);
        keyboardKey2.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey2.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey2.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey2.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey2.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey2.setCharacters("2","\"","@");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 2;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey2, gridBagConstraints);

        keyboardKey3.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey3.setText("3");
        keyboardKey3.setAlignmentY(0.0F);
        keyboardKey3.setFocusable(false);
        keyboardKey3.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey3.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey3.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey3.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey3.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey3.setCharacters("3","·","#");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 3;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey3, gridBagConstraints);

        keyboardKey4.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey4.setText("4");
        keyboardKey4.setAlignmentY(0.0F);
        keyboardKey4.setFocusable(false);
        keyboardKey4.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey4.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey4.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey4.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey4.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey4.setCharacters("4","$");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 4;
        gridBagConstraints.gridy = 1;
```

```
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey4, gridBagConstraints);

        keyboardKey5.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey5.setText("5");
        keyboardKey5.setAlignmentY(0.0F);
        keyboardKey5.setFocusable(false);
        keyboardKey5.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey5.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey5.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey5.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey5.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey5.setCharacters("5","%");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey5, gridBagConstraints);

        keyboardKey6.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey6.setText("6");
        keyboardKey6.setAlignmentY(0.0F);
        keyboardKey6.setFocusable(false);
        keyboardKey6.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey6.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey6.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey6.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey6.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey6.setCharacters("6","&","¬");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 6;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey6, gridBagConstraints);

        keyboardKey7.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey7.setText("7");
        keyboardKey7.setAlignmentY(0.0F);
```

```
        keyboardKey7.setFocusable(false);
        keyboardKey7.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey7.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey7.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey7.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey7.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey7.setCharacters("7","/");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 7;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey7, gridBagConstraints);

        keyboardKey8.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey8.setText("8");
        keyboardKey8.setAlignmentY(0.0F);
        keyboardKey8.setFocusable(false);
        keyboardKey8.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey8.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey8.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey8.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey8.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey8.setCharacters("8","(");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 8;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKey8, gridBagConstraints);

        keyboardKey9.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKey9.setText("9");
        keyboardKey9.setAlignmentY(0.0F);
        keyboardKey9.setFocusable(false);
        keyboardKey9.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKey9.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKey9.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKey9.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKey9.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKey9.setCharacters("9",")");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 9;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
```

```
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKey9, gridBagConstraints);

                keyboardKey0.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
                keyboardKey0.setText("0");
                keyboardKey0.setAlignmentY(0.0F);
                keyboardKey0.setFocusable(false);
                keyboardKey0.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKey0.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
                keyboardKey0.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKey0.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKey0.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKey0.setCharacters("0","=");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 10;
                gridBagConstraints.gridy = 1;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKey0, gridBagConstraints);

                keyboardKeyApostrophe.setBorder(javax.swing.BorderFactory.createBevelBorder(
javax.swing.border.BevelBorder.RAISED));
                keyboardKeyApostrophe.setText("'");
                keyboardKeyApostrophe.setAlignmentY(0.0F);
                keyboardKeyApostrophe.setFocusable(false);
                keyboardKeyApostrophe.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyApostrophe.setHorizontalTextPosition(javax.swing.SwingConstants.C
ENTER);
                keyboardKeyApostrophe.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyApostrophe.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyApostrophe.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyApostrophe.setCharacters("'","?");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 11;
                gridBagConstraints.gridy = 1;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyApostrophe, gridBagConstraints);

                keyboardKeyCloseAdmiration.setBorder(javax.swing.BorderFactory.createBevelBo
rder(javax.swing.border.BevelBorder.RAISED));
                keyboardKeyCloseAdmiration.setText("¡");
                keyboardKeyCloseAdmiration.setAlignmentY(0.0F);
```

```
            keyboardKeyCloseAdmiration.setFocusable(false);
            keyboardKeyCloseAdmiration.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyCloseAdmiration.setHorizontalTextPosition(javax.swing.SwingConsta
nts.CENTER);
            keyboardKeyCloseAdmiration.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyCloseAdmiration.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyCloseAdmiration.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyCloseAdmiration.setCharacters("¡","¿");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 12;
            gridBagConstraints.gridy = 1;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyCloseAdmiration, gridBagConstraints);

            keyboardKeyBackspace.setBorder(javax.swing.BorderFactory.createBevelBorder(j
avax.swing.border.BevelBorder.RAISED));
            keyboardKeyBackspace.setAlignmentY(0.0F);
            keyboardKeyBackspace.setFocusable(false);
            keyboardKeyBackspace.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyBackspace.setHorizontalTextPosition(javax.swing.SwingConstants.C
ENTER);
            keyboardKeyBackspace.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyBackspace.setMinimumSize(new java.awt.Dimension(10, 5));
            keyboardKeyBackspace.setName("Backspace"); // NOI18N
            keyboardKeyBackspace.setOpaque(true);
            keyboardKeyBackspace.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyBackspace.setImageIcon("/keysforfree/images/Backspace.png");
            keyboardKeyBackspace.setHasIcon(true);
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 13;
            gridBagConstraints.gridy = 1;
            gridBagConstraints.gridwidth = 2;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyBackspace, gridBagConstraints);

            keyboardKeyQ.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyQ.setText("q");
            keyboardKeyQ.setAlignmentY(0.0F);
            keyboardKeyQ.setFocusable(false);
            keyboardKeyQ.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyQ.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyQ.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyQ.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyQ.setPreferredSize(new java.awt.Dimension(20, 20));
```

```
                keyboardKeyQ.setCharacters("q","Q");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 2;
                gridBagConstraints.gridy = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyQ, gridBagConstraints);

                keyboardKeyW.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
                keyboardKeyW.setText("w");
                keyboardKeyW.setAlignmentY(0.0F);
                keyboardKeyW.setFocusable(false);
                keyboardKeyW.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyW.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
                keyboardKeyW.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyW.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyW.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyW.setCharacters("w","W");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 3;
                gridBagConstraints.gridy = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyW, gridBagConstraints);

                keyboardKeyE.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
                keyboardKeyE.setText("e");
                keyboardKeyE.setAlignmentY(0.0F);
                keyboardKeyE.setFocusable(false);
                keyboardKeyE.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyE.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
                keyboardKeyE.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyE.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyE.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyE.setCharacters("e","E","€","è","é","ë","È","É","Ë");
                keyboardKeyE.setIsVocal(true);
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 4;
                gridBagConstraints.gridy = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyE, gridBagConstraints);
```

```
            keyboardKeyR.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyR.setText("r");
            keyboardKeyR.setAlignmentY(0.0F);
            keyboardKeyR.setFocusable(false);
            keyboardKeyR.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyR.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyR.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyR.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyR.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyR.setCharacters("r","R");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 5;
            gridBagConstraints.gridy = 2;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyR, gridBagConstraints);

            keyboardKeyT.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
            keyboardKeyT.setText("t");
            keyboardKeyT.setAlignmentY(0.0F);
            keyboardKeyT.setFocusable(false);
            keyboardKeyT.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyT.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyT.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyT.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyT.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyT.setCharacters("t","T");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 6;
            gridBagConstraints.gridy = 2;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyT, gridBagConstraints);

            keyboardKeyY.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
            keyboardKeyY.setText("y");
            keyboardKeyY.setAlignmentY(0.0F);
            keyboardKeyY.setFocusable(false);
            keyboardKeyY.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyY.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyY.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyY.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyY.setPreferredSize(new java.awt.Dimension(20, 20));
```

```
        keyboardKeyY.setCharacters("y","Y");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 7;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyY, gridBagConstraints);

        keyboardKeyU.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyU.setText("u");
        keyboardKeyU.setAlignmentY(0.0F);
        keyboardKeyU.setFocusable(false);
        keyboardKeyU.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyU.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyU.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyU.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyU.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyU.setCharacters("u","U","","ù","ú","ü","Ù","Ú","Ü");
        keyboardKeyU.setIsVocal(true);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 8;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyU, gridBagConstraints);

        keyboardKeyI.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKeyI.setText("i");
        keyboardKeyI.setAlignmentY(0.0F);
        keyboardKeyI.setFocusable(false);
        keyboardKeyI.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyI.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyI.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyI.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyI.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyI.setCharacters("ï","I","","ì","í","ï","Ì","Í","Ï");
        keyboardKeyI.setIsVocal(true);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 9;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
```

```
        add(keyboardKeyI, gridBagConstraints);

        keyboardKeyO.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyO.setText("o");
        keyboardKeyO.setAlignmentY(0.0F);
        keyboardKeyO.setFocusable(false);
        keyboardKeyO.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyO.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyO.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyO.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyO.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyO.setCharacters("o","O","","ò","ó","ö","Ò","Ó","Ö");
        keyboardKeyO.setIsVocal(true);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 10;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyO, gridBagConstraints);

        keyboardKeyP.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKeyP.setText("p");
        keyboardKeyP.setAlignmentY(0.0F);
        keyboardKeyP.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        keyboardKeyP.setFocusable(false);
        keyboardKeyP.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyP.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyP.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyP.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyP.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyP.setCharacters("p","P");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 11;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyP, gridBagConstraints);

        keyboardKeyOpenAccent.setBorder(javax.swing.BorderFactory.createBevelBorder
(javax.swing.border.BevelBorder.RAISED));
        keyboardKeyOpenAccent.setText("`");
        keyboardKeyOpenAccent.setAlignmentY(0.0F);
        keyboardKeyOpenAccent.setFocusable(false);
        keyboardKeyOpenAccent.setFont(new java.awt.Font("Arial", 1, 14));
```

```
            keyboardKeyOpenAccent.setHorizontalTextPosition(javax.swing.SwingConstants.
CENTER);
            keyboardKeyOpenAccent.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyOpenAccent.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyOpenAccent.setName("OpenAccent"); // NOI18N
            keyboardKeyOpenAccent.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyOpenAccent.setCharacters("`","^","[");
            keyboardKeyOpenAccent.addModifierEventListener(keyboardKeyE);
            keyboardKeyOpenAccent.addModifierEventListener(keyboardKeyU);
            keyboardKeyOpenAccent.addModifierEventListener(keyboardKeyI);
            keyboardKeyOpenAccent.addModifierEventListener(keyboardKeyO);
            keyboardKeyOpenAccent.addModifierEventListener(keyboardKeyA);
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 12;
            gridBagConstraints.gridy = 2;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyOpenAccent, gridBagConstraints);

            keyboardKeyPlus.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.s
wing.border.BevelBorder.RAISED));
            keyboardKeyPlus.setText("+");
            keyboardKeyPlus.setAlignmentY(0.0F);
            keyboardKeyPlus.setFocusable(false);
            keyboardKeyPlus.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyPlus.setHorizontalTextPosition(javax.swing.SwingConstants.CENTE
R);
            keyboardKeyPlus.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyPlus.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyPlus.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyPlus.setCharacters("+","*","]");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 13;
            gridBagConstraints.gridy = 2;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyPlus, gridBagConstraints);

            keyboardKeyEnter.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.
swing.border.BevelBorder.RAISED));
            keyboardKeyEnter.setAlignmentY(0.0F);
            keyboardKeyEnter.setFocusable(false);
            keyboardKeyEnter.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyEnter.setHorizontalTextPosition(javax.swing.SwingConstants.CENTE
R);
            keyboardKeyEnter.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyEnter.setMinimumSize(new java.awt.Dimension(5, 10));
```

```
        keyboardKeyEnter.setName("Enter"); // NOI18N
        keyboardKeyEnter.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyEnter.setImageIcon("/keysforfree/images/Enter.png");
        keyboardKeyEnter.setHasIcon(true);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 14;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyEnter, gridBagConstraints);

        keyboardKeyA.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyA.setText("a");
        keyboardKeyA.setAlignmentY(0.0F);
        keyboardKeyA.setFocusable(false);
        keyboardKeyA.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyA.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyA.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyA.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyA.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyA.setCharacters("a","A","","à","á","ä","À","Á","Ä");
        keyboardKeyA.setIsVocal(true);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 2;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyA, gridBagConstraints);

        keyboardKeyS.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyS.setText("s");
        keyboardKeyS.setAlignmentY(0.0F);
        keyboardKeyS.setFocusable(false);
        keyboardKeyS.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyS.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyS.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyS.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyS.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyS.setCharacters("s","S");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 3;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
```

```
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyS, gridBagConstraints);

            keyboardKeyD.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyD.setText("d");
            keyboardKeyD.setAlignmentY(0.0F);
            keyboardKeyD.setFocusable(false);
            keyboardKeyD.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyD.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyD.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyD.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyD.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyD.setCharacters("d","D");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 4;
            gridBagConstraints.gridy = 3;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyD, gridBagConstraints);

            keyboardKeyF.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
            keyboardKeyF.setText("f");
            keyboardKeyF.setAlignmentY(0.0F);
            keyboardKeyF.setFocusable(false);
            keyboardKeyF.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyF.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyF.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyF.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyF.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyF.setCharacters("f","F");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 5;
            gridBagConstraints.gridy = 3;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyF, gridBagConstraints);

            keyboardKeyG.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyG.setText("g");
            keyboardKeyG.setAlignmentY(0.0F);
            keyboardKeyG.setFocusable(false);
            keyboardKeyG.setFont(new java.awt.Font("Arial", 1, 14));
```

```
        keyboardKeyG.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyG.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyG.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyG.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyG.setCharacters("g","G");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 6;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyG, gridBagConstraints);

        keyboardKeyH.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyH.setText("h");
        keyboardKeyH.setAlignmentY(0.0F);
        keyboardKeyH.setFocusable(false);
        keyboardKeyH.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyH.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyH.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyH.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyH.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyH.setCharacters("h","H");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 7;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyH, gridBagConstraints);

        keyboardKeyJ.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
        keyboardKeyJ.setText("j");
        keyboardKeyJ.setAlignmentY(0.0F);
        keyboardKeyJ.setFocusable(false);
        keyboardKeyJ.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyJ.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyJ.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyJ.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyJ.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyJ.setCharacters("j","J");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 8;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
```

```
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyJ, gridBagConstraints);

            keyboardKeyK.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyK.setText("k");
            keyboardKeyK.setAlignmentY(0.0F);
            keyboardKeyK.setFocusable(false);
            keyboardKeyK.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyK.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyK.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyK.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyK.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyK.setCharacters("k","K");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 9;
            gridBagConstraints.gridy = 3;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyK, gridBagConstraints);

            keyboardKeyL.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
            keyboardKeyL.setText("l");
            keyboardKeyL.setAlignmentY(0.0F);
            keyboardKeyL.setFocusable(false);
            keyboardKeyL.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyL.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyL.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyL.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyL.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyL.setCharacters("l","L");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 10;
            gridBagConstraints.gridy = 3;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyL, gridBagConstraints);

            keyboardKeyÑ.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyÑ.setText("ñ");
            keyboardKeyÑ.setAlignmentY(0.0F);
            keyboardKeyÑ.setFocusable(false);
            keyboardKeyÑ.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyÑ.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
```

```
        keyboardKeyÑ.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyÑ.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyÑ.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyÑ.setCharacters("ñ","Ñ");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 11;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyÑ, gridBagConstraints);

        keyboardKeyCloseAccent.setBorder(javax.swing.BorderFactory.createBevelBorde
r(javax.swing.border.BevelBorder.RAISED));
        keyboardKeyCloseAccent.setText("´");
        keyboardKeyCloseAccent.setAlignmentY(0.0F);
        keyboardKeyCloseAccent.setFocusable(false);
        keyboardKeyCloseAccent.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyCloseAccent.setHorizontalTextPosition(javax.swing.SwingConstants.
CENTER);
        keyboardKeyCloseAccent.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyCloseAccent.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyCloseAccent.setName("CloseAccent"); // NOI18N
        keyboardKeyCloseAccent.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyCloseAccent.setCharacters("´","¨","{");
        keyboardKeyCloseAccent.addModifierEventListener(keyboardKeyE);
        keyboardKeyCloseAccent.addModifierEventListener(keyboardKeyU);
        keyboardKeyCloseAccent.addModifierEventListener(keyboardKeyI);
        keyboardKeyCloseAccent.addModifierEventListener(keyboardKeyO);
        keyboardKeyCloseAccent.addModifierEventListener(keyboardKeyA);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 12;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 20;
        gridBagConstraints.ipady = 20;
        gridBagConstraints.weightx = 0.5;
        gridBagConstraints.weighty = 0.5;
        add(keyboardKeyCloseAccent, gridBagConstraints);

        keyboardKeyÇ.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
        keyboardKeyÇ.setText("ç");
        keyboardKeyÇ.setAlignmentY(0.0F);
        keyboardKeyÇ.setFocusable(false);
        keyboardKeyÇ.setFont(new java.awt.Font("Arial", 1, 14));
        keyboardKeyÇ.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        keyboardKeyÇ.setMaximumSize(new java.awt.Dimension(80, 80));
        keyboardKeyÇ.setMinimumSize(new java.awt.Dimension(5, 5));
        keyboardKeyÇ.setPreferredSize(new java.awt.Dimension(20, 20));
        keyboardKeyÇ.setCharacters("ç","Ç","}");
```

```
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 13;
                gridBagConstraints.gridy = 3;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyÇ, gridBagConstraints);

                keyboardKeyLessThen.setBorder(javax.swing.BorderFactory.createBevelBorder(ja
vax.swing.border.BevelBorder.RAISED));
                keyboardKeyLessThen.setText("<");
                keyboardKeyLessThen.setAlignmentY(0.0F);
                keyboardKeyLessThen.setFocusable(false);
                keyboardKeyLessThen.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyLessThen.setHorizontalTextPosition(javax.swing.SwingConstants.CE
NTER);
                keyboardKeyLessThen.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyLessThen.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyLessThen.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyLessThen.setCharacters("<",">");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 2;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyLessThen, gridBagConstraints);

                keyboardKeyZ.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
                keyboardKeyZ.setText("z");
                keyboardKeyZ.setAlignmentY(0.0F);
                keyboardKeyZ.setFocusable(false);
                keyboardKeyZ.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyZ.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
                keyboardKeyZ.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyZ.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyZ.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyZ.setCharacters("z","Z");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 3;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyZ, gridBagConstraints);
```

```
            keyboardKeyX.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyX.setText("x");
            keyboardKeyX.setAlignmentY(0.0F);
            keyboardKeyX.setFocusable(false);
            keyboardKeyX.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyX.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyX.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyX.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyX.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyX.setCharacters("x","X");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 4;
            gridBagConstraints.gridy = 4;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyX, gridBagConstraints);

            keyboardKeyC.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyC.setText("c");
            keyboardKeyC.setAlignmentY(0.0F);
            keyboardKeyC.setFocusable(false);
            keyboardKeyC.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyC.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyC.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyC.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyC.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyC.setCharacters("c","C");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 5;
            gridBagConstraints.gridy = 4;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyC, gridBagConstraints);

            keyboardKeyV.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swin
g.border.BevelBorder.RAISED));
            keyboardKeyV.setText("v");
            keyboardKeyV.setAlignmentY(0.0F);
            keyboardKeyV.setFocusable(false);
            keyboardKeyV.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyV.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyV.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyV.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyV.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyV.setCharacters("v","V");
```

```
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 6;
            gridBagConstraints.gridy = 4;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyV, gridBagConstraints);

            keyboardKeyB.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyB.setText("b");
            keyboardKeyB.setAlignmentY(0.0F);
            keyboardKeyB.setFocusable(false);
            keyboardKeyB.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyB.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyB.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyB.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyB.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyB.setCharacters("b","B");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 7;
            gridBagConstraints.gridy = 4;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyB, gridBagConstraints);

            keyboardKeyN.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
ng.border.BevelBorder.RAISED));
            keyboardKeyN.setText("n");
            keyboardKeyN.setAlignmentY(0.0F);
            keyboardKeyN.setFocusable(false);
            keyboardKeyN.setFont(new java.awt.Font("Arial", 1, 14));
            keyboardKeyN.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
            keyboardKeyN.setMaximumSize(new java.awt.Dimension(80, 80));
            keyboardKeyN.setMinimumSize(new java.awt.Dimension(5, 5));
            keyboardKeyN.setPreferredSize(new java.awt.Dimension(20, 20));
            keyboardKeyN.setCharacters("n","N");
            gridBagConstraints = new java.awt.GridBagConstraints();
            gridBagConstraints.gridx = 8;
            gridBagConstraints.gridy = 4;
            gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
            gridBagConstraints.ipadx = 20;
            gridBagConstraints.ipady = 20;
            gridBagConstraints.weightx = 0.5;
            gridBagConstraints.weighty = 0.5;
            add(keyboardKeyN, gridBagConstraints);

            keyboardKeyM.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swi
```

```
ng.border.BevelBorder.RAISED));
                keyboardKeyM.setText("m");
                keyboardKeyM.setAlignmentY(0.0F);
                keyboardKeyM.setFocusable(false);
                keyboardKeyM.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyM.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
                keyboardKeyM.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyM.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyM.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyM.setCharacters("m","M");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 9;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyM, gridBagConstraints);

                keyboardKeyComma.setBorder(javax.swing.BorderFactory.createBevelBorder(jav
ax.swing.border.BevelBorder.RAISED));
                keyboardKeyComma.setText(",");
                keyboardKeyComma.setAlignmentY(0.0F);
                keyboardKeyComma.setFocusable(false);
                keyboardKeyComma.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyComma.setHorizontalTextPosition(javax.swing.SwingConstants.CEN
TER);
                keyboardKeyComma.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyComma.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyComma.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyComma.setCharacters(",",";");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 10;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyComma, gridBagConstraints);

                keyboardKeyDot.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.s
wing.border.BevelBorder.RAISED));
                keyboardKeyDot.setText(".");
                keyboardKeyDot.setAlignmentY(0.0F);
                keyboardKeyDot.setFocusable(false);
                keyboardKeyDot.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyDot.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER)
;
                keyboardKeyDot.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyDot.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyDot.setPreferredSize(new java.awt.Dimension(20, 20));
```

```
                keyboardKeyDot.setCharacters(".",":");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 11;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyDot, gridBagConstraints);

                keyboardKeyLess.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.s
wing.border.BevelBorder.RAISED));
                keyboardKeyLess.setText("-");
                keyboardKeyLess.setAlignmentY(0.0F);
                keyboardKeyLess.setFocusable(false);
                keyboardKeyLess.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyLess.setHorizontalTextPosition(javax.swing.SwingConstants.CENTE
R);
                keyboardKeyLess.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyLess.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyLess.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyLess.setCharacters("-","_");
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 12;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyLess, gridBagConstraints);

                keyboardKeyShiftRight.setBorder(javax.swing.BorderFactory.createBevelBorder(ja
vax.swing.border.BevelBorder.RAISED));
                keyboardKeyShiftRight.setAlignmentY(0.0F);
                keyboardKeyShiftRight.setFocusable(false);
                keyboardKeyShiftRight.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyShiftRight.setHorizontalTextPosition(javax.swing.SwingConstants.CE
NTER);
                keyboardKeyShiftRight.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyShiftRight.setMinimumSize(new java.awt.Dimension(10, 5));
                keyboardKeyShiftRight.setName("ShiftRight"); // NOI18N
                keyboardKeyShiftRight.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyShiftRight.setImageIcon("/keysforfree/images/Shift.png");
                keyboardKeyShiftRight.setHasIcon(true);
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 13;
                gridBagConstraints.gridy = 4;
                gridBagConstraints.gridwidth = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
```

```
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyShiftRight, gridBagConstraints);

                keyboardKeyControl.setBorder(javax.swing.BorderFactory.createBevelBorder(java
x.swing.border.BevelBorder.RAISED));
                keyboardKeyControl.setText("ctrl");
                keyboardKeyControl.setAlignmentY(0.0F);
                keyboardKeyControl.setFocusable(false);
                keyboardKeyControl.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyControl.setHorizontalTextPosition(javax.swing.SwingConstants.CENT
ER);
                keyboardKeyControl.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyControl.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyControl.setName("Control"); // NOI18N
                keyboardKeyControl.setPreferredSize(new java.awt.Dimension(20, 20));
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 1;
                gridBagConstraints.gridy = 5;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyControl, gridBagConstraints);

                keyboardKeyAltLeft.setBorder(javax.swing.BorderFactory.createBevelBorder(javax
.swing.border.BevelBorder.RAISED));
                keyboardKeyAltLeft.setText("alt");
                keyboardKeyAltLeft.setActionCommand("Alt");
                keyboardKeyAltLeft.setAlignmentY(0.0F);
                keyboardKeyAltLeft.setFocusable(false);
                keyboardKeyAltLeft.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyAltLeft.setHorizontalTextPosition(javax.swing.SwingConstants.CENT
ER);
                keyboardKeyAltLeft.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyAltLeft.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyAltLeft.setName("AltLeft"); // NOI18N
                keyboardKeyAltLeft.setOpaque(true);
                keyboardKeyAltLeft.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyAltLeft.addModifierEventListener(keyboardKeyGrade);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKey1);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKey2);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKey3);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKey6);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKeyCloseAccent);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKeyPlus);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKeyOpenAccent);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKeyÇ);
                keyboardKeyAltLeft.addModifierEventListener(keyboardKeyÉ);
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 2;
                gridBagConstraints.gridy = 5;
```

```
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyAltLeft, gridBagConstraints);

                keyboardKeyComandLeft.setBorder(javax.swing.BorderFactory.createBevelBorder
(javax.swing.border.BevelBorder.RAISED));
                keyboardKeyComandLeft.setText("cmd");
                keyboardKeyComandLeft.setAlignmentY(0.0F);
                keyboardKeyComandLeft.setFocusable(false);
                keyboardKeyComandLeft.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyComandLeft.setHorizontalTextPosition(javax.swing.SwingConstants.
CENTER);
                keyboardKeyComandLeft.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyComandLeft.setMinimumSize(new java.awt.Dimension(10, 5));
                keyboardKeyComandLeft.setName("ComandLeft"); // NOI18N
                keyboardKeyComandLeft.setPreferredSize(new java.awt.Dimension(20, 20));
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 3;
                gridBagConstraints.gridy = 5;
                gridBagConstraints.gridwidth = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyComandLeft, gridBagConstraints);

                keyboardKeyComandRight.setBorder(javax.swing.BorderFactory.createBevelBord
er(javax.swing.border.BevelBorder.RAISED));
                keyboardKeyComandRight.setText("cmd");
                keyboardKeyComandRight.setAlignmentY(0.0F);
                keyboardKeyComandRight.setFocusable(false);
                keyboardKeyComandRight.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyComandRight.setHorizontalTextPosition(javax.swing.SwingConstants
.CENTER);
                keyboardKeyComandRight.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyComandRight.setMinimumSize(new java.awt.Dimension(10, 5));
                keyboardKeyComandRight.setName("ComandRight"); // NOI18N
                keyboardKeyComandRight.setPreferredSize(new java.awt.Dimension(20, 20));
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 10;
                gridBagConstraints.gridy = 5;
                gridBagConstraints.gridwidth = 2;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyComandRight, gridBagConstraints);
```

```
                keyboardKeyAltRight.setBorder(javax.swing.BorderFactory.createBevelBorder(jav
ax.swing.border.BevelBorder.RAISED));
                keyboardKeyAltRight.setText("alt");
                keyboardKeyAltRight.setActionCommand("Alt");
                keyboardKeyAltRight.setAlignmentY(0.0F);
                keyboardKeyAltRight.setFocusable(false);
                keyboardKeyAltRight.setFont(new java.awt.Font("Arial", 1, 14));
                keyboardKeyAltRight.setHorizontalTextPosition(javax.swing.SwingConstants.CEN
TER);
                keyboardKeyAltRight.setMaximumSize(new java.awt.Dimension(80, 80));
                keyboardKeyAltRight.setMinimumSize(new java.awt.Dimension(5, 5));
                keyboardKeyAltRight.setName("AltRight"); // NOI18N
                keyboardKeyAltRight.setPreferredSize(new java.awt.Dimension(20, 20));
                keyboardKeyAltRight.addModifierEventListener(keyboardKeyGrade);
                keyboardKeyAltRight.addModifierEventListener(keyboardKey1);
                keyboardKeyAltRight.addModifierEventListener(keyboardKey2);
                keyboardKeyAltRight.addModifierEventListener(keyboardKey3);
                keyboardKeyAltRight.addModifierEventListener(keyboardKey6);
                keyboardKeyAltRight.addModifierEventListener(keyboardKeyCloseAccent);
                keyboardKeyAltRight.addModifierEventListener(keyboardKeyPlus);
                keyboardKeyAltRight.addModifierEventListener(keyboardKeyOpenAccent);
                keyboardKeyAltRight.addModifierEventListener(keyboardKeyÇ);
                keyboardKeyAltRight.addModifierEventListener(keyboardKeyE);
                gridBagConstraints = new java.awt.GridBagConstraints();
                gridBagConstraints.gridx = 12;
                gridBagConstraints.gridy = 5;
                gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
                gridBagConstraints.ipadx = 20;
                gridBagConstraints.ipady = 20;
                gridBagConstraints.weightx = 0.5;
                gridBagConstraints.weighty = 0.5;
                add(keyboardKeyAltRight, gridBagConstraints);
            }// </editor-fold>

    /**
     * S'executa quan detectem un canvi en la mida del teclat.
     * Llancem un esdeveniment de canvi de text perquè les tecles canviïn la
     * mida del text.
     * @param evt
     */
    private void keyboardResized(java.awt.event.ComponentEvent evt) {
        fireTextAdjustEvent(new TextAdjustEvent(this));
    }

    /**
     * Guardem els escoltadors de les tecles que rebran esdeveniments de canvi
     * de text.
     */
    private EventListenerList textAdjustListeners = new EventListenerList();

    /**
     * Serveix per afegir un escoltador d'una tecla a la llista anterior.
     * @param listener Escoltador de la tecla.
```

```
     */
    public synchronized void addTextAdjustEventListener(TextAdjustEventListener listener)
    {
        textAdjustListeners.add(TextAdjustEventListener.class, listener);
    }

    /**
     * Serveix per treure un escoltador de la llista.
     * @param listener Escoltador que es vol treure
     */
    public synchronized void removeTextAdjustEventListener(TextAdjustEventListener
listener)
    {
        textAdjustListeners.remove(TextAdjustEventListener.class, listener);
    }

    /**
     * Llença l'esdeveniment d'ajustament de text.
     * Anem iterant per la llista d'escoltadors i fem executar l'operació
     * d'ajustament de text definida a la interfície TextModifierEventListener
     * a cada tecla.
     * @param e
     */
    protected synchronized void fireTextAdjustEvent(TextAdjustEvent e)
    {
        Object[] listeners = textAdjustListeners.getListenerList();
        for (int i = 0; i < listeners.length; i+=2)
        {
            if(listeners[i] == TextAdjustEventListener.class)
            {
                    ((TextAdjustEventListener) listeners[i+1]).adjustText();
            }
        }
    }

    /**
     * Serveix per obtenir la llista d'escoltadors que esperen l'esdeveniment
     * d'ajustament de text.
     * @return the textAdjustListeners
     */
    public EventListenerList getTextAdjustListeners()
    {
        return textAdjustListeners;
    }

    /**
     * Serveix per indicar la llista d'escoltadors d'esdeveniments de text
     * @param textAdjustListeners the textAdjustListeners to set
     */
    public void setTextAdjustListeners(EventListenerList textAdjustListeners)
    {
        this.textAdjustListeners = textAdjustListeners;
    }
```

```
// Variables declaration - do not modify
private keysforfree.KeyboardKey keyboardKey1;
private keysforfree.KeyboardKey keyboardKey2;
private keysforfree.KeyboardKey keyboardKey3;
private keysforfree.KeyboardKey keyboardKey4;
private keysforfree.KeyboardKey keyboardKey5;
private keysforfree.KeyboardKey keyboardKey6;
private keysforfree.KeyboardKey keyboardKey7;
private keysforfree.KeyboardKey keyboardKey8;
private keysforfree.KeyboardKey keyboardKey9;
private keysforfree.KeyboardKey keyboardKeyA;
private keysforfree.KeyboardKey keyboardKeyAltLeft;
private keysforfree.KeyboardKey keyboardKeyAltRight;
private keysforfree.KeyboardKey keyboardKeyB;
private keysforfree.KeyboardKey keyboardKeyBackspace;
private keysforfree.KeyboardKey keyboardKeyC;
private keysforfree.KeyboardKey keyboardKeyCapsLock;
private keysforfree.KeyboardKey keyboardKeyCloseAccent;
private keysforfree.KeyboardKey keyboardKeyCloseAdmiration;
private keysforfree.KeyboardKey keyboardKeyComandLeft;
private keysforfree.KeyboardKey keyboardKeyComandRight;
private keysforfree.KeyboardKey keyboardKeyComma;
private keysforfree.KeyboardKey keyboardKeyControl;
private keysforfree.KeyboardKey keyboardKeyD;
private keysforfree.KeyboardKey keyboardKeyDot;
private keysforfree.KeyboardKey keyboardKeyE;
private keysforfree.KeyboardKey keyboardKeyEnter;
private keysforfree.KeyboardKey keyboardKeyEsc;
private keysforfree.KeyboardKey keyboardKeyF;
private keysforfree.KeyboardKey keyboardKeyF1;
private keysforfree.KeyboardKey keyboardKeyF10;
private keysforfree.KeyboardKey keyboardKeyF11;
private keysforfree.KeyboardKey keyboardKeyF12;
private keysforfree.KeyboardKey keyboardKeyF2;
private keysforfree.KeyboardKey keyboardKeyF3;
private keysforfree.KeyboardKey keyboardKeyF4;
private keysforfree.KeyboardKey keyboardKeyF5;
private keysforfree.KeyboardKey keyboardKeyF6;
private keysforfree.KeyboardKey keyboardKeyF7;
private keysforfree.KeyboardKey keyboardKeyF8;
private keysforfree.KeyboardKey keyboardKeyF9;
private keysforfree.KeyboardKey keyboardKeyFunction;
private keysforfree.KeyboardKey keyboardKeyG;
private keysforfree.KeyboardKey keyboardKeyGrade;
private keysforfree.KeyboardKey keyboardKeyH;
private keysforfree.KeyboardKey keyboardKeyI;
private keysforfree.KeyboardKey keyboardKeyJ;
private keysforfree.KeyboardKey keyboardKeyK;
private keysforfree.KeyboardKey keyboardKeyL;
private keysforfree.KeyboardKey keyboardKeyLess;
private keysforfree.KeyboardKey keyboardKeyLessThen;
private keysforfree.KeyboardKey keyboardKeyM;
```

```
    private keysforfree.KeyboardKey keyboardKeyN;
    private keysforfree.KeyboardKey keyboardKeyO;
    private keysforfree.KeyboardKey keyboardKeyOpenAccent;
    private keysforfree.KeyboardKey keyboardKeyP;
    private keysforfree.KeyboardKey keyboardKeyPlus;
    private keysforfree.KeyboardKey keyboardKeyQ;
    private keysforfree.KeyboardKey keyboardKeyR;
    private keysforfree.KeyboardKey keyboardKeyS;
    private keysforfree.KeyboardKey keyboardKeyShiftLeft;
    private keysforfree.KeyboardKey keyboardKeyShiftRight;
    private keysforfree.KeyboardKey keyboardKeySpace;
    private keysforfree.KeyboardKey keyboardKeyT;
    private keysforfree.KeyboardKey keyboardKeyTab;
    private keysforfree.KeyboardKey keyboardKeyU;
    private keysforfree.KeyboardKey keyboardKeyV;
    private keysforfree.KeyboardKey keyboardKeyW;
    private keysforfree.KeyboardKey keyboardKeyX;
    private keysforfree.KeyboardKey keyboardKeyY;
    private keysforfree.KeyboardKey keyboardKeyZ;
    private keysforfree.KeyboardKey keyboardKeyÇ;
    private keysforfree.KeyboardKey keyboardKeyÑ;
    // End of variables declaration
}
```

6.1.3. CatalanPredictor

```
package keysforfree;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.EnumMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.List;

/**
 * Classe que implementa la predicció seguint les regles del català.
 * Hereta del predictor per freqüència per poder utilitzar a la plegada
 * les dues funcionalitats.
 * @author eloicreus
 */
public class CatalanPredictor extends Predictor
{

    /**
     * @return the strictLanguagePrediction
     */
    public boolean isStrictLanguagePrediction() {
        return strictLanguagePrediction;
    }

    /**
     * @param strictLanguagePrediction the strictLanguagePrediction to set
     */
    public void setStrictLanguagePrediction(boolean strictLanguagePrediction) {
        this.strictLanguagePrediction = strictLanguagePrediction;
    }

    /**
     * Enumeració per indicar els diferents tipus de categoria gramatical del
     * català.
     */
    public enum CategoriaGramatical {NOM, VERB, ADVERBI, ADJECTIU,
DETERMINANT, PRONOM, PREPOSICIO, CONJUNCIO, ALTRE}

    /**
     * Enumeració per indicar els diferents gèneres del català.
     */
    enum Genere {MASCULÍ, FEMENÍ, INDIFERENT}

    /**
```

```
    * Enumeració per indicar els diferents nombres del català.
    */
    enum Nombre {SINGULAR, PLURAL, INDIFERENT}

    /**
    * Enumeració per indicar les diferents persones del català.
    */
    enum Persona {PRIMERA, SEGONA, TERCERA}

   /**
    * Indica el gènere de l'última paraula.
    */
    Genere genereActual;
    /**
    * Indica el nombre de l'última paraula.
    */
    Nombre nombreActual;
    /**
    * Indica la persona de l'última paraula.
    */
    Persona personaActual;
    /**
    * Indica la categoria gramatical de l'última paraula.
    */
    CategoriaGramatical categoriaActual;


    /**
    * Conté el caràcters que faran acabar una frase.
    */
    static String[] separadorsFrase = {".",",","!",";",":"};
    List<String> separaFrase;

    /**
    * Enllaç a la base de dades per consultar paraules.
    */
    Connection databaseConnection;
    /**
    * Estructura per realitzar les consultes.
    */
    Statement databaseStatement;

    String ultimaParaula;
    String fraseActual;

    /**
    * Indica que hem escrit un apòstrof
    */
    boolean postApostrof;
    /**
    * Indica que hem escrit un guionet.
    */
    boolean postGuionet;
```

```
    /**
     * Indica si volem utilitzar només la funcionalitat de predicció en català o
     * volem combinar amb la predicció per freqüència.
     */
    protected boolean strictLanguagePrediction = false;

    /*
     * Patrons per detectar la conjugació dels verbs en infinitiu.
     */
    Pattern patroPrimeraConjugacio = Pattern.compile("([\\p{L}|·]*)ar[-se]?");
    Pattern patroSegonaConjugacio = Pattern.compile("(\\p{L}*)[er|re]");
    Pattern patroTerceraConjugacio = Pattern.compile("([\\p{L}|·]*)ir[-se]?");

    /**
     * Indica amb quines lletres pot començar una paraula després d'un apòstrof.
     */
    String començamentApostrof = "^[h|a|e|i|o|u]";
    /**
     * Indica amb quines lletres pot començar una paraula després d'un guionet.
     */
    String començamentGuionet = "^[b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|y|z]";


    /**
     * Patró de conjugació cantar de la primera conjugació.
     */
    public static String[][] modelCantar = {{"o","es","a","em","eu","en"}, //present
                                            {"ava","aves","ava","àvem","àveu","aven"}, //
imperfet
                                            {"í","ares","à","àrem","àreu","aren"}, //passat
simple
                                            {"aré","aràs","arà","arem","areu","aran"},
//futur
                                            {"aria","aries","aria","aríem","aríeu","arien"}};
//condicional


    /**
     * Model de conjugació servir de la tercera conjugació.
     */
    public static String[][] modelServir = {{"eixo","eixes","eix","im","iu","eixen"}, //present
                                            {"ia","ies","ia","íem","íeu","ien"}, //imperfet
                                            {"í","ires","í","írem","íreu","iren"}, //passat
simple
                                            {"iré","iràs","irà","irem","ireu","iran"}, //futur
                                            {"iria","iries","iria","iriem","irieu","irien"}};
//condicional

    /**
     * Constructor de la classe que fa les inicialitzacions necessàries.
     * @param words Llista de paraules del diccionari que utilitza el predictor.
     */
```

```
public CatalanPredictor(ArrayList<WordInfo> words)
{
    super(words, true);
    separaFrase = Arrays.asList(separadorsFrase);
}

/**
 * Obté un enllaç per realitzar consultes contra la base de dades.
 * S'executa al principi del funcionament del predictor en català.
 * @return Si l'operació s'ha realitzat correctament.
 */
public boolean connectDatabase()
{
    boolean exception = false;
    try
    {
        databaseConnection = DatabaseConnection.connectToDatabase();
    }
    catch (SQLException ex)
    {
        exception = true;
    }
    finally
    {
        return exception;
    }
}

/**
 * Serveix per realitzar consultes a la base de dades.
 * @param string Consulta SQL a executar.
 * @return L'estructura que gestiona la consulta que permet obtenir els
 *         resultats d'aquesta.
 * @throws SQLException Si hi ha hagut algun problema.
 */
ResultSet executeQuery(String string) throws SQLException
{
    databaseStatement = databaseConnection.createStatement();
    return databaseStatement.executeQuery(string);
}

/**
 * Neteja els recursos utilitzats de la base de dades.
 */
public void cleanStuff()
{
    try
    {
        if(databaseStatement != null)
        {
            databaseStatement.close();
        }
    }
```

```
        catch (SQLException ex)
        {}
    }

    /**
     * Serveix per acabar la connexió amb la base de dades.
     * S'executa quan deixem d'utilitzar la predicció en català.
     * @return Si l'operació s'ha realitzat correctament.
     */
    public void disconnectDatabase()
    {
        try
        {
            if(databaseConnection != null)
            {
                databaseConnection.close();
            }
        }
        catch (SQLException ex)
        {}
    }

    /**
     * Funció utilitzada per obtenir els verbs dels models cantar i servir que
     * comencen pel prefix que ha escrit l'usuari.
     * Obtenim de la base de dades tots els infinitius possibles i els
     * conjuguem.
     * @return La llista de verbs possibles.
     */
    public ArrayList<String> buscarVerbsPossibles(ArrayList<String> paraules)
    {
        String consultaVerbs = "SELECT LEMA FROM Catala WHERE LEMA REGEXP "
                + "'^"+ currentWord+ "'" + " AND CATEGORIAGRAMATICAL REGEXP ";
        String Cantar = "'^v [(]cantar[)]'";
        String Partir = "'^v [(]servir[)]'";
        String lema;

        ArrayList<String> verbsPossibles = new ArrayList<String>();

        try {

            ResultSet categories = executeQuery(consultaVerbs+Cantar);
            int persona = obtenirPersona();

            while(categories.next())
            {
                lema = categories.getString("LEMA");

                if(categoriaActual != CategoriaGramatical.PREPOSICIO)
                {
                    verbsPossibles.addAll(conjugaVerb(obtenirArrel(lema,patroPrimeraConju
gacio),modelCantar,persona,
                                paraules));
```

```
                }
                else
                {
                    if(isStrictLanguagePrediction() || !cercaDicotomica(paraules, lema))
                    {
                        verbsPossibles.add(lema);
                    }
                }
            }

            categories = executeQuery(consultaVerbs+Partir);

            while(categories.next())
            {
                lema = categories.getString("LEMA");

                if(categoriaActual != CategoriaGramatical.PREPOSICIO)
                {
                    verbsPossibles.addAll(conjugaVerb(obtenirArrel(lema,patroTerceraConju
gacio),modelServir,persona
                            ,paraules));
                }
                else
                {
                    if(isStrictLanguagePrediction() || !cercaDicotomica(paraules, lema))
                    {
                        verbsPossibles.add(lema);
                    }
                }
            }

            categories.close();

        } catch (SQLException ex) {

        } finally {
            cleanStuff();
        }
        return verbsPossibles;
    }

    /**
     * Funció que donada  l'arrel d'un verb ens fa la conjugació, segons el model
     * que ens passen i la persona i gènera.
     * @param arrel Arrel del verb a conjugar.
     * @param model Model de conjugació a utilitzar.
     * @param persona Columna de la matriu que representa el model de conjugació
     *                que conté el nombre i persona desitjat.
     * @return Les conjugacions possibles.
     */
    public ArrayList<String> conjugaVerb(String arrel, String[][] model, int persona,
ArrayList<String> paraules )
    {
```

```
        ArrayList<String> verbConjugat = new ArrayList<String>();

        for(int i = 0; i < 5; i++)
        {
            String verb = arrel+model[i][persona];
            if(isStrictLanguagePrediction() || !cercaDicotomica(paraules, verb))
            {
                verbConjugat.add(verb);
            }
        }

        return verbConjugat;
    }

    /**
     * Funció que ens obté donat el gènera i la persona de l'última paraula ens
     * retrona la columna de la matriu de conjugació que correspon a aquestes
     * variables.
     * @return La columna corresponent.
     */
    public int obtenirPersona()
    {
        if(personaActual == Persona.PRIMERA)
        {
            if(nombreActual == Nombre.SINGULAR)
            {
                return 0;
            }
            else
            {
                return 3;
            }
        }
        else if(personaActual == Persona.SEGONA)
        {
            if(nombreActual == Nombre.SINGULAR)
            {
                return 1;
            }
            else
            {
                return 4;
            }
        }
        else if(personaActual == Persona.TERCERA)
        {
            if(nombreActual == Nombre.SINGULAR)
            {
                return 2;

            }
            else
            {
```

```
                return 5;
            }
        }
        return 2;
    }

    /**
     * Donat un pronom fort ens retorna la seva persona i gènere per fer-lo
     * concordar amb el verb.
     * @param pronom Pronom escrit.
     */
    public void obtenirPersonaPronom(String pronom)
    {
        genereActual = Genere.INDIFERENT;
        if(pronom.equals("jo"))
        {
            personaActual = Persona.PRIMERA;
            nombreActual = Nombre.SINGULAR;
        }
        else if(pronom.equals("nosaltres"))
        {
            personaActual = Persona.PRIMERA;
            nombreActual = Nombre.PLURAL;
        }
        else if(pronom.equals("tu"))
        {
            personaActual = Persona.SEGONA;
            nombreActual = Nombre.SINGULAR;
        }
        else if(pronom.equals("vosaltres"))
        {
            personaActual = Persona.SEGONA;
            nombreActual = Nombre.PLURAL;
        }
        else if(pronom.equals("ell") || pronom.equals("ella"))
        {
            personaActual = Persona.TERCERA;
            nombreActual = Nombre.SINGULAR;
        }
        else if(pronom.equals("ells") || pronom.equals("elles"))
        {
            personaActual = Persona.TERCERA;
            nombreActual = Nombre.PLURAL;
        }
        else
        {
            personaActual = Persona.TERCERA;
        }
    }

    /**
     * Donat un verb en infinitiu obtenim la seva arrel depenent de la seva
     * conjugació.
```

```
     * @param verbInfinitiu Verb del que s'ha d'obtenir l'arrel.
     * @param conjugacio Conjugació de l'infinitiu-
     * @return L'arrel obtinguda.
     */
    public String obtenirArrel(String verbInfinitiu, Pattern conjugacio)
    {
        Matcher encaixa = conjugacio.matcher(verbInfinitiu);
        encaixa.find();
        return encaixa.group(1);
    }

    /**
     * Serveix per construir la cadena de caràcters que hem de buscar a la base
     * de dades en el camp LEMA.
     * @param paraula Prefix de paraula escrit per l'usuari.
     * @return La cadena que s'ha d'utilitzar.
     */
    public String lemaString(String paraula)
    {
        if(postApostrof)
        {
            postApostrof = false;
            return començamentApostrof;
        }

        if(postGuionet)
        {
            postGuionet = false;
            return començamentGuionet;
        }

        return "'^"+paraula+"'";
    }

    /**
     * Serveix per construir la cadena que hem de buscar en el camp ENTRADES de
     * la base de dades.
     * @param paraula Prefix de paraula escrit per l'usuari.
     * @return La cadena que s'ha d'utilitzar.
     */
    public String entradesString(String paraula)
    {
        return "'^"+paraula+" | "+paraula+" | "+paraula+"$|^"+paraula+"$'";
    }

    /**
     * Ens retorna per cada entrada de la paraula escrita que conté la base de
     * dades, la categoria gramatical més prioritària i els camps de la base de
     * dades que ens permeten determinar el gènere i nombre de la paraula.
     * @param paraula Paraula que ha escrit l'usuari.
     * @return El que s'ha explicat abans.
     */
    public EnumMap<CategoriaGramatical,String> consultaCategoriesPossibles(String
```

```
paraula)
    {
        String consultCategoriesGramaticals = "SELECT
LEMA,ENTRADES,CATEGORIAGRAMATICAL FROM Catala WHERE LEMA REGEXP
'^"+paraula+"$' OR ENTRADES REGEXP "
            + entradesString(paraula) +" AND NOT CATEGORIAGRAMATICAL REGEXP
'locució'";
        String lema = null;
        String entrades = null;
        EnumMap<CategoriaGramatical,String> categoriesPossibles = null;

        try {

            ResultSet categories = executeQuery(consultCategoriesGramaticals);
            categoriesPossibles = new
EnumMap<CategoriaGramatical,String>(CategoriaGramatical.class);

            while(categories.next())
            {
                System.out.println(categories.getString("CATEGORIAGRAMATICAL"));
                lema = categories.getString("LEMA");
                entrades = categories.getString("ENTRADES");
                categoriesPossibles.put(determinaCategoriaString(categories.getString("CAT
EGORIAGRAMATICAL")),lema+" "+entrades);
            }

            categories.close();

        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            cleanStuff();
        }
        return categoriesPossibles;
    }

    /**
     * Funció que ens retorna la categoria a assignar a la paraula entrada.
     * Definim una prioritat donada una cadena que ens indica les diferents
     * categories que pot tenir la paraula.
     * En el cas del noms, ja determinem el seu gènere.
     * @param stringCategory Cadena amb les categories possibles.
     * @return La categoria que assignem a la paraula.
     */
    public CategoriaGramatical determinaCategoriaString(String stringCategory)
    {
        Matcher encaixa;

        if((encaixa = Pattern.compile("det").matcher(stringCategory)).find())
        {
            return CategoriaGramatical.DETERMINANT;
        }
        if((encaixa = Pattern.compile("nom m i f").matcher(stringCategory)).find())
```

```
        {
            genereActual = Genere.INDIFERENT;
            return CategoriaGramatical.NOM;
        }
        if((encaixa = Pattern.compile("nom m").matcher(stringCategory)).find())
        {
            genereActual = Genere.MASCULÍ;
            return CategoriaGramatical.NOM;
        }
        if((encaixa = Pattern.compile("nom f").matcher(stringCategory)).find())
        {
            genereActual = Genere.FEMENÍ;
            return CategoriaGramatical.NOM;
        }
        if((encaixa = Pattern.compile("adj").matcher(stringCategory)).find())
        {
            return CategoriaGramatical.ADJECTIU;
        }
            if((encaixa = Pattern.compile("\\Aadv").matcher(stringCategory)).find())
        {
            return CategoriaGramatical.ADVERBI;
        }
        if((encaixa = Pattern.compile("prep").matcher(stringCategory)).find())
        {
            return CategoriaGramatical.PREPOSICIO;
        }
        if((encaixa = Pattern.compile("conj").matcher(stringCategory)).find())
        {
            return CategoriaGramatical.CONJUNCIO;
        }
        if((encaixa = Pattern.compile("pron").matcher(stringCategory)).find())
        {
            return CategoriaGramatical.PRONOM;
        }
        return CategoriaGramatical.ALTRE;
    }

    /**
     * S'executa quan l'usuari acaba d'entrar una paraula per determinar la
     * categoria gramatical d'aquesta i realitzar les accions necessàries en
     * cada cas.
     * Obtenim el gènere i el nombre de la paraula en cas de que sigui necessari,
     * i actualitzem l'atribut que ens indica la categoria de l'última paraula,
     * per saber que podem predir posteriorment.
     * @param separator Separador amb el que hem acabat d'escriure la paraula.
     */
    @Override
    public void addSeparator(String separator)
    {
        if(separaFrase.contains(separator))
        {
            newFrase();
        }
```

```
        if(writingWord)
        {
            EnumMap<CategoriaGramatical,String> categoriesPossibles =
consultaCategoriesPossibles(currentWord);
            if(categoriesPossibles.containsKey(CategoriaGramatical.DETERMINANT))
            {
                buscarGenereINombre(currentWord,categoriesPossibles.get(CategoriaGrama
tical.DETERMINANT).trim());

                categoriaActual = CategoriaGramatical.DETERMINANT;
                personaActual = Persona.TERCERA;
            }
            else if(categoriesPossibles.containsKey(CategoriaGramatical.NOM))
            {
                if(categoriaActual != CategoriaGramatical.DETERMINANT)
                {
                    buscarNombre(currentWord,categoriesPossibles.get(CategoriaGramatica
l.NOM).trim());
                }

                personaActual = Persona.TERCERA;
                categoriaActual = CategoriaGramatical.NOM;
            }
            else if(categoriesPossibles.containsKey(CategoriaGramatical.ADJECTIU))
            {
                if(categoriaActual != CategoriaGramatical.DETERMINANT)
                {
                    buscarGenereINombre(currentWord,categoriesPossibles.get(CategoriaG
ramatical.ADJECTIU).trim());
                }

                categoriaActual = CategoriaGramatical.ADJECTIU;
                personaActual = Persona.TERCERA;
            }
            else if(categoriesPossibles.containsKey(CategoriaGramatical.ADVERBI))
            {
                categoriaActual = CategoriaGramatical.ADVERBI;
            }
            else if(categoriesPossibles.containsKey(CategoriaGramatical.PREPOSICIO))
            {
                genereActual = Genere.INDIFERENT;
                nombreActual = Nombre.INDIFERENT;
                categoriaActual = CategoriaGramatical.PREPOSICIO;
            }
            else if(categoriesPossibles.containsKey(CategoriaGramatical.CONJUNCIO))
            {
                newFrase();
            }
            else if(categoriesPossibles.containsKey(CategoriaGramatical.PRONOM))
            {
                obtenirPersonaPronom(currentWord);
                categoriaActual = CategoriaGramatical.PRONOM;
```

```
            }
        }

        super.addSeparator(separator);
    }

    /**
     * S'executa quan realitzem un canvi de frase.
     * Eliminem les interferències amb la frase anterior
     */
    public void newFrase()
    {
        categoriaActual = null;
        genereActual = null;
        nombreActual = null;
    }

    /**
     * S'utilitza per obtenir el nombre dels noms escrits sense determinant al
     * davant. El gènere es determina al moment de saber que la paraula és un
     * nom.
     * @param paraula Paraula escrita.
     * @param possibilitats Variacions possibles de la paraula.
     */
    public void buscarNombre(String paraula, String possibilitats)
    {
        Pattern mfsp = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*) (\\p{L}*)");
        Pattern sp = Pattern.compile("(\\p{L}*) (\\p{L}*s)");

        Matcher encaixa;

        if((encaixa = mfsp.matcher(possibilitats)).matches())
        {
            if(encaixa.group(1).matches(paraula) || encaixa.group(2).matches(paraula))
            {
                nombreActual = Nombre.SINGULAR;
            }
            else
            {
                nombreActual = Nombre.PLURAL;
            }
        }
        else
        {
            (encaixa = sp.matcher(possibilitats)).matches();

            if(encaixa.group(1).matches(paraula))
            {
                nombreActual = Nombre.SINGULAR;
            }
            else
            {
                nombreActual = Nombre.PLURAL;
```

```
            }
        }
    }

    /**
     * S'utilitza per obtenir el gènere i el nombre dels determinants i dels
     * adjectius que escriu l'usuari. Utilitzem patrons per determinar aquesta
     * informació depenent de les variacions que pugui tenir la paraula escrita.
     * @param paraula Paraula escrita per l'usuari.
     * @param possibilitats Variacions que pot tenir la paraula.
     */
    public void buscarGenereINombre(String paraula, String possibilitats)
    {
        Pattern mfsp = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*) (\\p{L}*)");
        Pattern mfp = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*)");
        Pattern sp = Pattern.compile("(\\p{L}*) (\\p{L}*s)");
        Pattern mf = Pattern.compile("(\\p{L}*) (\\p{L}*a|\\p{L}*)");
        Pattern la = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*') (\\p{L}*) (\\p{L}*)");
        Pattern en = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*')");

        Matcher encaixa;

        genereActual = Genere.MASCULÍ;
        nombreActual = Nombre.SINGULAR;

        if((encaixa = mfsp.matcher(possibilitats)).matches())
        {
            if(encaixa.group(2).matches(paraula))
            {
                genereActual = Genere.FEMENÍ;
            }
            if(encaixa.group(3).matches(paraula))
            {
                nombreActual = Nombre.PLURAL;
            }
            if(encaixa.group(4).matches(paraula))
            {
                genereActual = Genere.FEMENÍ;
                nombreActual = Nombre.PLURAL;
            }
        }
        else if((encaixa = mfp.matcher(possibilitats)).matches())
        {
            if(encaixa.group(2).matches(paraula))
            {
                genereActual = Genere.FEMENÍ;
            }
            if(encaixa.group(3).matches(paraula))
            {
                genereActual = Genere.INDIFERENT;
                nombreActual = Nombre.PLURAL;
            }
        }
```

```
else if((encaixa = sp.matcher(possibilitats)).matches())
{
    genereActual = Genere.INDIFERENT;
    if(encaixa.group(2).matches(paraula))
    {
        nombreActual = Nombre.PLURAL;
    }
}
else if(((encaixa = mf.matcher(possibilitats)).matches()))
{
    nombreActual = Nombre.INDIFERENT;
    if(encaixa.group(2).matches(paraula))
    {
        genereActual = Genere.FEMENÍ;
    }
}
else if(((encaixa = la.matcher(possibilitats)).matches()))
{
    if(encaixa.group(2).matches(paraula))
    {
        genereActual = Genere.FEMENÍ;
    }
    if(encaixa.group(3).matches(paraula))
    {
        nombreActual = Nombre.INDIFERENT;
    }
    if(encaixa.group(4).matches(paraula))
    {
        nombreActual = Nombre.PLURAL;
    }
    if(encaixa.group(5).matches(paraula))
    {
        genereActual = Genere.FEMENÍ;
        nombreActual = Nombre.PLURAL;
    }
}
else if(((encaixa = en.matcher(possibilitats)).matches()))
{
    nombreActual = Nombre.INDIFERENT;
    if(encaixa.group(2).matches(paraula))
    {
        genereActual = Genere.FEMENÍ;
    }
}
else
{
    genereActual = Genere.INDIFERENT;
    if(paraula.equals("cada") || paraula.equals("cap"))
    {
        nombreActual = Nombre.SINGULAR;
    }
    else
    {
```

```
                    nombreActual = Nombre.PLURAL;
                }
            }
        }

        /**
         * Funció que obté els adverbis que comencen pel prefix de paraula que ha
         * escrit l'usuari.
         * @param paraules Llista de les paraules que ja s'han seleccionat amb la
         *                 predicció per freqüència.
         * @return La llista d'adverbis possibles.
         */
        public ArrayList<String> buscarAdverbis(ArrayList<String> paraules)
        {
            String consultaAdverbis = "SELECT LEMA FROM Catala WHERE LEMA
REGEXP "
                    + "'^"+ currentWord+ "'" + " AND CATEGORIAGRAMATICAL REGEXP
"+"'^adv'";

            ArrayList<String> adverbisPossibles = new ArrayList<String>();

            try {

                ResultSet categories = executeQuery(consultaAdverbis);

                while(categories.next())
                {
                    String paraula = categories.getString("LEMA");
                    if(isStrictLanguagePrediction() || !cercaDicotomica(paraules,paraula.trim()))
                    {
                        adverbisPossibles.add(paraula);
                    }
                }

                categories.close();

            } catch (SQLException ex) {
                Logger.getLogger(CatalanPredictor.class.getName()).log(Level.SEVERE, null,
ex);
            }
            return adverbisPossibles;
        }

        /**
         * Funció que obté els noms que comencen amb un prefix de paraula escrit
         * fent-los concordar amb el gènere i el nombre actuals.
         * @param paraules Conté la llista de les paraules que ja s'han seleccionat
         *                 a la predicció per freqüència.
         */
        public ArrayList<String> buscarNomsConcordants(ArrayList<String> paraules)
        {
            String buscar;
            if(genereActual == Genere.INDIFERENT)
```

```
        {
            buscar = "'nom'";
        }
        else if (genereActual == Genere.FEMENÍ)
        {
            buscar = "'nom f|nom m i f'";
        }
        else
        {
            buscar = "'nom m|nom m i f'";
        }

        String consultaNoms = "SELECT LEMA,ENTRADES FROM Catala WHERE
CATEGORIAGRAMATICAL REGEXP "+buscar+
                " AND LEMA REGEXP "+lemaString(currentWord);
        ArrayList<String> nomsPossibles = new ArrayList<String>();

        try {

            ResultSet categories = executeQuery(consultaNoms);

            while(categories.next())
            {
                String lema = categories.getString("LEMA");
                String entrades = categories.getString("ENTRADES");
                for(String paraula: buscarConcordancia(lema,lema+" "+entrades))
                {
                    System.out.println(paraula);
                    if(isStrictLanguagePrediction() || !cercaDicotomica(paraules,
paraula.trim()))
                    {
                        nomsPossibles.add(paraula);
                    }
                }
            }

            categories.close();

        } catch (SQLException ex) {

        } finally {
            cleanStuff();
        }
        return nomsPossibles;
    }

    /**
     * Funció que ens obté els adjectius que comencen pel prefix de paraula que
     * ha escrit l'usuari, fent-los concordar amb el gènere i nombre actuals.
     * @param paraules Conté les paraules que ja s'ha seleccionat en la
     *                 predicció per freqüència.
     * @return La llista d'adjectius possibles.
     */
```

```
    public ArrayList<String> buscarAdjectiusConcordants(ArrayList<String> paraules)
    {
        String consultaAdjectius = "SELECT LEMA,ENTRADES FROM Catala WHERE
CATEGORIAGRAMATICAL REGEXP 'adj'"+
                " AND LEMA REGEXP "+lemaString(currentWord);
        ArrayList<String> adjectiusPossibles = new ArrayList<String>();

         try {

            ResultSet categories = executeQuery(consultaAdjectius);

            while(categories.next())
            {
                String lema = categories.getString("LEMA");
                String entrades = categories.getString("ENTRADES");
                for(String paraula: buscarConcordancia(lema,lema+" "+entrades))
                {
                    System.out.println(paraula);
                    if(isStrictLanguagePrediction() || !cercaDicotomica(paraules,
paraula.trim()))
                    {
                        adjectiusPossibles.add(paraula);
                    }
                }
            }

            categories.close();

        } catch (SQLException ex) {

        } finally {
            cleanStuff();
        }
        return adjectiusPossibles;
    }

    /**
     * Serveix per fer concordar els noms i adjectius obtinguts de la base de
     * dades amb el gènere i nombre actuals.
     * @param paraula Paraula del camp LEMA obtinguda de la base de dades.
     * @param possibilitats Variacions que pot tenir la paraula.
     * @return La llista de les possibilitats que concorden.
     */
    public ArrayList<String> buscarConcordancia(String paraula, String possibilitats)
    {
        Pattern mfsp = Pattern.compile("(\\p{L}*) (\\p{L}*) (\\p{L}*) (\\p{L}*)");
        Pattern sp = Pattern.compile("(\\p{L}*) (\\p{L}*s)");

        Matcher encaixa;

        ArrayList<String> paraulesConcordants = new ArrayList<String>();

        if(((encaixa = mfsp.matcher(possibilitats)).matches()))
```

```
{
    if(genereActual == Genere.INDIFERENT)
    {
        if(nombreActual == Nombre.SINGULAR)
        {
            paraulesConcordants.add(encaixa.group(1));
            paraulesConcordants.add(encaixa.group(2));
        }
        else if(nombreActual == Nombre.PLURAL)
        {
            paraulesConcordants.add(encaixa.group(3));
            paraulesConcordants.add(encaixa.group(4));
        }
        else
        {
            paraulesConcordants.add(encaixa.group(1));
            paraulesConcordants.add(encaixa.group(2));
            paraulesConcordants.add(encaixa.group(3));
            paraulesConcordants.add(encaixa.group(4));
        }
    }
    else if(genereActual == Genere.FEMENÍ)
    {
        if(((encaixa = mfsp.matcher(possibilitats)).matches()))
        {
            if(nombreActual == Nombre.SINGULAR)
            {
                paraulesConcordants.add(encaixa.group(2));
            }
            else if(nombreActual == Nombre.PLURAL)
            {
                paraulesConcordants.add(encaixa.group(4));
            }
            else
            {
                paraulesConcordants.add(encaixa.group(2));
                paraulesConcordants.add(encaixa.group(4));
            }
        }
    }
    else
    {
        if(((encaixa = mfsp.matcher(possibilitats)).matches()))
        {
            if(nombreActual == Nombre.SINGULAR)
            {
                paraulesConcordants.add(encaixa.group(1));
            }
            else if(nombreActual == Nombre.PLURAL)
            {
                paraulesConcordants.add(encaixa.group(3));
            }
            else
```

```
                    {
                        paraulesConcordants.add(encaixa.group(1));
                        paraulesConcordants.add(encaixa.group(3));
                    }
                }
            }
        }
        else if(((encaixa = sp.matcher(possibilitats)).matches()))
        {
            if(nombreActual == Nombre.SINGULAR)
            {
                paraulesConcordants.add(encaixa.group(1));
            }
            else if(nombreActual == Nombre.PLURAL)
            {
                paraulesConcordants.add(encaixa.group(2));
            }
            else
            {
                paraulesConcordants.add(encaixa.group(1));
                paraulesConcordants.add(encaixa.group(2));
            }
        }
        else
        {
            paraulesConcordants.add(paraula);
        }

        return paraulesConcordants;
    }

    /**
     * Serveix per obtenir les paraules que conté el predictor per freqüència.
     * @return La llista de paraules que conté.
     */
    @Override
    public ArrayList<WordInfo> getAllWords()
    {
        return super.getAllWords();
    }

    /**
     * Serveix per obtenir les paraules que es poden escriure en tot moment, per
     * mostrar a la pantalla de predicció.
     * Depenent de la categoria gramatical de l'última paraula escrita obtenim
     * paraules de les categories gramaticals que poden venir després de
     * l'actual. Si cal hem de fer concordar les paraules obtingudes.
     * Depenent de si volem tenir la funcionalitat de predicció per freqüència o
     * no, també obtenim les paraules possibles amb aquest tipus de predicció.
     * @return La llista de paraules que pot escriure l'usuari.
     */
    @Override
    public ArrayList<String> getCandidateWords()
```

```
{
    ArrayList<String> paraules = new ArrayList<String>();

    if(!isStrictLanguagePrediction())
    {
        paraules.addAll(super.getCandidateWords());
    }

    if(writingWord||postApostrof||postGuionet)
    {
        if(categoriaActual == CategoriaGramatical.DETERMINANT)
        {
            paraules.addAll(buscarNomsConcordants(paraules));
            paraules.addAll(buscarAdjectiusConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.ADVERBI)
        {
            paraules.addAll(buscarAdverbis(paraules));
            paraules.addAll(buscarAdjectiusConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.NOM)
        {
            paraules.addAll(buscarAdjectiusConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.ADJECTIU)
        {
            paraules.addAll(buscarNomsConcordants(paraules));
        }

        if(categoriaActual == CategoriaGramatical.PREPOSICIO)
        {
            paraules.addAll(buscarNomsConcordants(paraules));
        }

        if(categoriaActual != CategoriaGramatical.DETERMINANT)
        {
            paraules.addAll(buscarVerbsPossibles(paraules));
        }
    }

    return paraules;
}

/**
 * Serveix per saber si una determinada paraula pertany a una llista de
 * paraules o no. L'utilitzem per no mostrar a l'usuari la mateixa paraula
 * més d'una vegada.
 * @param paraules Llista de paraules on volem mirar si h ha una paraula.
 * @param paraula Paraula que volem buscar a la llista.
 * @return Si la paraula buscada es troba dintre de la llista o no.
```

```
     */
    public boolean cercaDicotomica(ArrayList<String> paraules, String paraula)
    {
        int middle;
        int first = 0;
        int last = paraules.size()-1;

        while(first <= last)
        {
            middle = ((last - first)/2)+first;

            if(paraula.compareTo(paraules.get(middle)) < 0)
            {
                last = middle-1;
            }
            else if(paraula.compareTo(paraules.get(middle)) > 0)
            {
                first = middle+1;
            }
            else
            {
                return true;
            }
        }
        return false;
    }

    /**
     * S'executa quan l'usuari escriu un caràcter. Realitzem el mateix que en el
     * predictor per freqüència, i a més, detectem els apòstrofs i els guionets.
     * @param character Caràcter que ha escrit l'usuari.
     */
    @Override
    public void newCharacter(String character)
    {
        super.newCharacter(character);

        if(writingWord)
        {
            if(character.equals("'"))
            {
                postApostrof = true;
                currentWord = currentWord+"'";
                addSeparator("'");

            }
            else if(character.equals("-"))
            {
                postGuionet = true;
                addSeparator("-");
            }
        }
    }
```

```
    /**
     * S'executa quan l'usuari esborra un caràcter i fa el mateix que el
     * predictor per freqüència.
     */
    @Override
    void processRemove() {
        super.processRemove();
    }

}
```

6.1.4. DatabaseConnection

```
package keysforfree;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.SQLException;
import java.util.prefs.BackingStoreException;
import java.util.prefs.Preferences;

/**
 * Classe que gestiona el paràmetres de la base de dades. Retorna connexions a
 * les classes que necessiten utilitzar la base de dades.
 * @author eloicreus
 */
public class DatabaseConnection {

    /**
     * Indica la base de dades on ens hem de connectar.
     */
    static String DATABASE_URL = "jdbc:mysql://localhost/Keys4Free";
    /**
     * Indica el nom d'usuari amb el que ens hem de connectar.
     */
    public static String USER;
    /**
     * Indica la contrasenya amb la que ens hem de connectar.
     */
    public static String PASSWORD;

    /**
     * Conté les preferències de la base de dades.
     */
    static final Preferences preferences =
Preferences.userNodeForPackage(DatabaseConnection.class).node("database");

    /*
     * Carreguem de les preferències el nom d'usuari i la contrasenya.
     */
    static
    {
        setUSER(preferences.get("database.user", ""));
        setPASSWORD(preferences.get("database.password", ""));
    }

    public static Statement statement;

    /**
     * Retorna un enllaç obert amb la base de dades. Ens connectem a la base de
     * dades dintre d'aquesta funció.
     * @return L'enllaç obert amb la base de dades.
     * @throws SQLException Si no ens hem pogut connectar.
```

```
     */
    static public Connection connectToDatabase() throws SQLException
    {
        Connection connection = DriverManager.getConnection(DATABASE_URL,
getUSER(), getPASSWORD());
        return connection;
    }


    /**
     * Guarda el nom d'usuari i la contrasenya a les preferències del programa.
     * @throws BackingStoreException Si no hem pogut guardar les preferències.
     */static void savePreferences() throws BackingStoreException
    {
        preferences.put("database.user", USER);
        preferences.put("database.password", PASSWORD);
        preferences.flush();
    }

    /**
     * @return the USER
     */
    public static String getUSER() {
        return USER;
    }

    /**
     * @param aUSER the USER to set
     */
    public static void setUSER(String aUSER) {
        USER = aUSER;
    }

    /**
     * @return the PASSWORD
     */
    public static String getPASSWORD() {
        return PASSWORD;
    }

    /**
     * @param aPASSWORD the PASSWORD to set
     */
    public static void setPASSWORD(String aPASSWORD) {
        PASSWORD = aPASSWORD;
    }
}
```

6.1.5. Dictionary

```
package keysforfree;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

/**
 * Classe que implementa la gestió dels diccionaris. Conté el predictor que
 * s'utilitzarà en el diccionari
 * @author eloicreus
 */
public class Dictionary
{
    /**
     * Predictor que utilitzarà el diccionari
     */
    public Predictor predictor;

    /**
     * Nom del diccionari
     */
    private String name;
    /**
     * Identificador del diccionari a la base de dades.
     */
    public int identifier;
    /**
     * Conté temporalment les paraules del diccionari, mentre no es generi
     * l'arbre ternari de cerca del predictor.
     */
    public ArrayList<WordInfo> words;

    /**
     * Conté els noms dels diccionaris creats per l'usuari.
     */
    static ArrayList<String> dictionariesName;

    /**
     * Conté un enllaç amb la base de dades. S'utilitza per realitzar les
     * accions necessàries contra la base de dades.
     */
    static Connection databaseConnection;
    /**
     * Permet fer les accions a la base de dades.
     */
    static Statement databaseStatement;

    /**
```

```
     * Constructor de la classe indicant el nom del diccionari.
     * @param name Nom del diccionari.
     */
    public Dictionary(String name) {
        this.name = name;
    }

    /**
     * Allibera els recursos de la base de dades utilitzats per la connexió que
     * hi tenim.
     */
    public static void cleanStuff()
    {
        try
        {
            if(databaseStatement != null)
            {
                databaseStatement.close();
            }
            if(databaseConnection != null)
            {
                databaseConnection.close();
            }
        }
        catch (SQLException ex)
        {}
    }

    /**
     * Construeix el predictor que utilitzarà el diccionari i carrega les
     * paraules d'aquesta.
     * Depenen del tipus de predictor que vulgui l'usuari, creem un predictor
     * per freqüència o pel català.
     * Construïm l'arbre ternari de cerca del predictor carregant les paraules
     * del diccionari de la base de dades que es troben a l'atribut words.
     * Aquest últim s'allibera després de la construcció.
     * @param catalanPredictor Indica el tipus de predictor
     * @return Indica si l'operació s'ha pogut realitzar
     */
    public boolean setPredictor(boolean catalanPredictor)
    {
        boolean exception;
        exception = loadDictionary();
        if(catalanPredictor)
        {
            predictor = new CatalanPredictor(words);
            exception = ((CatalanPredictor)predictor).connectDatabase() || exception;
        }
        else
        {
            predictor = new Predictor(words);
        }
        words = null;
```

```
        return exception;
    }

    /**
     * Obté de la base de dades l'identificador del diccionari per a usos
     * posteriors.
     * @return L'identificador o un número d'error.
     */
    private int getDictionaryIdentifier()
    {
        String GET_DICTIONARY_IDENTIFIER = "SELECT Identifier "+
                                           "FROM Dictionary "+
                                           "WHERE Dictionary.Name = '"+name+"'";
        int ident = -1;

        try
        {
            databaseConnection = DatabaseConnection.connectToDatabase();
            databaseStatement = databaseConnection.createStatement();

            ResultSet result =
databaseStatement.executeQuery(GET_DICTIONARY_IDENTIFIER);
            result.first();
            ident = result.getInt("Identifier");

        }
        catch (SQLException ex)
        {

        }
        finally
        {
            cleanStuff();
            return ident;
        }
    }

    /**
     * Carrega els noms dels diccionaris de la base de dades i els guarda en
     * l'atribut dictionariesName.
     * @return Si l'operació s'ha realitzat correctament.
     */
    public static boolean loadDictionaries()
    {
        boolean exception = false;
        String GET_DICTIONARIES_CONSULT = "SELECT Name FROM Dictionary";
        dictionariesName = new ArrayList<String>();

        try
        {
            databaseConnection = DatabaseConnection.connectToDatabase();
            databaseStatement = databaseConnection.createStatement();
```

```
                ResultSet result =
databaseStatement.executeQuery(GET_DICTIONARIES_CONSULT);

                while(result.next())
                {
                    dictionariesName.add(result.getString("Name"));
                }

            }
            catch (SQLException ex)
            {
                exception = true;
            }
            finally
            {
                cleanStuff();
                return !exception;
            }
}

    /**
     * Crea una nova entrada de diccionari a la base de dades.
     * @param dictionaryName Nom del nou diccionari.
     * @return Si l'operació s'ha realitzat correctament.
     */
    public static boolean createDictionary(String dictionaryName)
    {
        boolean exception = false;
        PreparedStatement result = null;
        try {
            databaseConnection = DatabaseConnection.connectToDatabase();

            result = databaseConnection.prepareStatement("INSERT INTO Dictionary
(Name) VALUES ('"+dictionaryName+"')");
            result.executeUpdate();

            dictionariesName.add(dictionaryName);

        } catch (SQLException ex) {
            exception = true;
        }
        finally
        {
            try
            {
                if(result != null)
                {
                    result.close();
                }
                if(databaseConnection != null)
                {
                    databaseConnection.close();
                }
```

```
            }
            catch (SQLException ex)
            {}
            return !exception;
        }
    }

    /**
     * Carrega les paraules que té el diccionari de la base de dades i els
     * guarda temporalment a la variable words.
     * @return Si l'operació ha anat bé.
     */
    public boolean loadDictionary()
    {
        boolean exception = false;
        String GET_WORDS_CONSULT = "SELECT Word.Name,Frequency "
            + "FROM Word,Dictionary "
            + "WHERE Word.Dictionary = Dictionary.Identifier AND "
            + "Dictionary.Name = '"+name+"'";

        try {
            databaseConnection = DatabaseConnection.connectToDatabase();
            databaseStatement = databaseConnection.createStatement();

            ResultSet result =
databaseStatement.executeQuery(GET_WORDS_CONSULT);
            words = new ArrayList<WordInfo>();

            while(result.next())
            {
                words.add(new
WordInfo(result.getString("Name").toLowerCase(),result.getInt("Frequency")));
            }

        } catch (SQLException ex) {
            exception = true;
        }
        finally
        {
            cleanStuff();
            return !exception;
        }

    }

    /**
     * Guarda les paraules de l'arbre ternari de cerca que gestiona el predictor
     * a la base de dades. En cas que una paraula ja existeixi només s'actualitza
     * la seva freqüència.
     * @return Si l'operació s'ha executat correctament.
     */
    public boolean saveDictionary()
    {
```

```
        boolean exception = false;
        String SAVE_WORD_CONSULT = "INSERT INTO Word
(Name,Frequency,Dictionary) "+
                                   "VALUES (?,?,?) "+
                                   "ON DUPLICATE KEY UPDATE Frequency=?";
        PreparedStatement saveWord = null;
        try
        {
            databaseConnection = DatabaseConnection.connectToDatabase();

            saveWord = databaseConnection.prepareStatement(SAVE_WORD_CONSULT);

            for(WordInfo wordInfo : predictor.getAllWords())
            {

                saveWord.setString(1, wordInfo.word);
                saveWord.setInt(2, wordInfo.frequency);
                saveWord.setInt(3, getIdentifier());
                saveWord.setInt(4, wordInfo.frequency);

                saveWord.executeUpdate();
            }

        } catch (SQLException ex) {
            exception = true;
        }
        finally
        {
            try
            {
                if(saveWord != null)
                {
                    saveWord.close();
                }
                if(databaseConnection != null)
                {
                    databaseConnection.close();
                }
            }
            catch (SQLException ex)
            {}
            return !exception;
        }
    }

    /**
     * Elimina totes les paraules del diccionari de la base de dades i el nom
     * del diccionari.
     * @return Si l'operació s'ha executat bé.
     */
    public boolean removeDictionary()
    {
        boolean exception = false;
```

```
        String DELETE_DICTIONARY_WORDS = "DELETE FROM Word WHERE
Dictionary = "+getIdentifier();
        String DELETE_DICTIONARY = "DELETE FROM Dictionary WHERE Identifier =
"+getIdentifier();

        PreparedStatement deleteStatement = null;

        try{
            databaseConnection = DatabaseConnection.connectToDatabase();

            deleteStatement =
databaseConnection.prepareStatement(DELETE_DICTIONARY_WORDS);
            deleteStatement.executeUpdate();

            deleteStatement =
databaseConnection.prepareStatement(DELETE_DICTIONARY);
            deleteStatement.executeUpdate();

            Dictionary.dictionariesName.remove(name);

        }
        catch (SQLException ex)
        {
            exception = true;
        }
        finally
        {
            try
            {
                if(deleteStatement != null)
                {
                    deleteStatement.close();
                }
                if(databaseConnection != null)
                {
                    databaseConnection.close();
                }
            }
            catch (SQLException ex)
            {}
            return !exception;
        }
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
```

```
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the identifier
     */
    public int getIdentifier() {
        return identifier;
    }

    /**
     * @param identifier the identifier to set
     */
    public int setIdentifier() {
        return (identifier = getDictionaryIdentifier());
    }

}
```

6.1.6. DwellThread

```
package keysforfree;

import java.util.prefs.BackingStoreException;
import java.util.prefs.Preferences;

/**
 * Classe que s'utilitza pel mode de pulsació mantenint el cursor a sobre les
 * tecles.
 * Hereta de la classe Thread, ja que s'executa en un nou fil d'execució cada
 * vegada que entrem en una tecla.
 * @author eloicreus
 */
    public class DwellThread extends Thread
    {

     /**
      * Carrega les preferències associades a la funció de pulsació
      */
     static final Preferences preferences =
Preferences.userNodeForPackage(DwellThread.class).node("dwelling");
     /**
      * Temps d'espera que indica l'estona que s'ha de mantenir el cursor sobre
      * la tecla per escriure-la
      */
     static public int milisecondsDelay;

     /**
      * Tecla que ha produït l'execució del fil.
      */
     KeyboardKey key;

     /*
      * Carreguem de les preferències el temps d'espera que vol l'usuari.
      * S'executa en iniciar la classe.
      */
     static
     {
          setMilisecondsDelay(preferences.getInt("dwelling.delay", 1000));
     }

     /**
      * @return the milisecondsDelay
      */
     public static int getMilisecondsDelay()
     {
          return milisecondsDelay;
     }

     /**
      * @param aMilisecondsDelay the milisecondsDelay to set
      */
```

```
public static void setMilisecondsDelay(int aMilisecondsDelay)
{
    milisecondsDelay = aMilisecondsDelay;
}

/**
 * Guarda el temps d'espera que vol l'usuari a les preferències del programa
 * @throws BackingStoreException Si no s'han pogut guardar les preferències.
 */
static public void savePreferences() throws BackingStoreException
{
    preferences.putInt("time", milisecondsDelay);
    preferences.flush();
}

/**
 * Constructor per defecte de la classe.
 */
public DwellThread()
{

}

/**
 * Constructor de la classe indicant la tecla que crearà el fil d'execució
 * @param keyAssociated Tecla associada
 */
public DwellThread(KeyboardKey keyAssociated)
{
    key = keyAssociated;
    key.setButtonExited(false);
}

/**
 * Constructor de la classe indicant la tecla i el temps d'espera que volem
 * @param keyAssociated Tecla associada
 * @param delay Temps d'espera
 */
public DwellThread(KeyboardKey keyAssociated,int delay)
{
    key = keyAssociated;
    key.setButtonExited(false);
    milisecondsDelay = delay;
}

/**
 * Funció que s'executa en el fil d'execució creat.
 * Esperem el temps d'espera que tenim definit i si el cursor ha sortit de
 * la tecla mentre esperàvem no fem res, en cas contrari, escrivim el
 * caràcter de la tecla associada.
 */
@Override
public void run()
```

```
    {
        try
        {
                sleep(milisecondsDelay);
        }
        catch (InterruptedException ex)
        {

        }

        if(!key.isButtonExited())
        {
            key.typeOperation(true);
        }
    }
}
```

6.1.7. Heap

```
package keysforfree;

import java.util.ArrayList;

/**
 * Classe que implementa l'estructura de dades Heap.
 * L'utilitzem per ordenar la llista de paraules de la predicció de major a
 * menor freqüència.
 * Per tant, el Heap contindrà els nodes ordenats de major a menor.
 * @author eloicreus
 */
public class Heap
{

    /**
     * Taula que guarda les parelles de freqüència i paraula que conté
     * l'estructura
     */
    WordInfo elementTable[];

    /**
     * Indica el nombre d'elements que conté l'estructura.
     */
    int numberOfElements;

    /**
     * Constructor de la classe que donada la llista de paraules i freqüències
     * les va introduint a l'estructura del Heap.
     * @param words Llista de paraules i freqüències.
     */
    public Heap(ArrayList<WordInfo> words)
    {
        elementTable = new WordInfo[words.size()];

        for( WordInfo wordInfo: words)
        {
            add(wordInfo);
        }
    }

    /**
     * Inserim la parella a la posició que li pertoca segons la seva freqüència.
     * Primer situem l'element al final del Heap i anem pujant per l'arbre fins
     * a la posició que li pertoca.
     * @param word Parella de paraula i freqüència a introduir.
     */
    final void add(WordInfo word)
    {
        if(numberOfElements == 0)
        {
            elementTable[numberOfElements++] = word ;
```

```
        }
        else
        {

            int i = ++numberOfElements;
            while((i/2)-1 >= 0 && word.frequency > elementTable[(i/2)-1].frequency)
            {
                elementTable[i-1] = elementTable[(i/2)-1];
                i = i/2;
            }
            elementTable[i-1] = word;
        }
    }

    /**
     * Funció que agafa l'element de més freqüència i l'elimina del Heap.
     * Es reestructura tot l'arbre perquè mantingui les propietats del Heap.
     * @return Retorna la parella de l'arrel del Heap que conté la major
     * freqüència.
     */
    WordInfo remove(){

        if(numberOfElements == 0)
        {
            return null;
        }
        else if(numberOfElements == 1)
        {
            WordInfo head = elementTable[0];
            elementTable[0] = null;
            numberOfElements--;
            return head;
        }
        else
        {

            WordInfo head = elementTable[0];
            WordInfo tail = elementTable[numberOfElements-1];
            int value = tail.frequency;

            elementTable[0] = tail;
            int i = 1;
            int pot = 2;
            while((i*2)<numberOfElements && (elementTable[(i*2)-1].frequency>value ||
                    elementTable[i*2].frequency>value))
            {
                if(elementTable[(i*2)-1].frequency > value)
                {
                    if(elementTable[(i*2)-1].frequency > elementTable[i*2].frequency)
                    {
                        swap(i-1,(i*2)-1);
                        i = i*2;
                    }
```

```
                    else
                    {
                        swap(i-1,i*2);
                        i = (i*2)+1;
                    }
                }
                else
                {
                    swap(i-1,i*2);
                    i = (i*2)+1;
                }
            }

            elementTable[numberOfElements-1] = null;
            numberOfElements--;
            return head;
        }
    }

    /**
     * Intercanvia els elements de dues posicions.
     * @param x Posició origen.
     * @param y Posició destí.
     */
    private void swap(int x,int y)
    {

        if(x != y)
        {
            WordInfo aux = elementTable[x];
            elementTable[x] = elementTable[y];
            elementTable[y] = aux;
        }
    }

    /**
     * Indica si el Heap està buit.
     * @return
     */
    boolean isEmpty()
    {
        return numberOfElements == 0;
    }

    /**
     * Funció que obté la llista de parelles de major a menor freqüència.
     * Per fer-ho, anem eliminant els elements del Heap i anem posant la paraula
     * de l'element eliminat en una llista de manera que quedarà ordenada tal
     * com volem.
     * @return La llista de paraules ordenada.
     */
    public ArrayList<String> heapsort()
    {
```

```
        ArrayList<String> words = new ArrayList<String>(numberOfElements);
        while(!isEmpty())
        {
            words.add(remove().word);
        }
        return words;
    }
}
```

6.1.8. KeyboardKey

```
package keysforfree;

import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.RenderingHints;
import java.awt.Toolkit;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.event.EventListenerList;
import javax.swing.text.BadLocationException;


/**
 * Classe que implementa la funcionalitat de les tecles del teclat virtual.
 * @author eloicreus
 */
public class KeyboardKey extends JButton implements ModifierEventListener,
TextAdjustEventListener
{
    /**
     * Indica el teclat on es troba la tecla.
     */
    private static AbstractKeyboard keyboard;

    /*
     * Indiquen les tres mides que pot tenir el text de les tecles.
     */
    static float SMALL_FONT_SIZE = 12F;
    static float MEDIUM_FONT_SIZE = 16F;
    static float BIG_FONT_SIZE = 20F;

    /**
     * @return the keyboard
     */
    public static AbstractKeyboard getKeyboard()
    {
        return keyboard;
    }

    /**
     * @param aKeyboard the keyboard to set
     */
```

```
public static void setKeyboard(AbstractKeyboard aKeyboard)
{
    keyboard = aKeyboard;
}

/**
 * Indica el tipus de tecla
 */
protected KeyType keyType;

/**
 * Indica si la tecla és una vocal.
 */
private boolean isVocal = false;

/**
 * Indica si la tecla conté una icona.
 */
protected boolean hasIcon = false;

/**
 * Fil d'execució associat a la tecla per la funció d'escriure mantenint el
 * cursor sobre la tecla.
 */
DwellThread dwellThread = new DwellThread();

/**
 * Indica si hem sortit de la tecla durant l'execució del fil anterior.
 */
protected boolean buttonExited = false;

/**
 * Indica per cada estat dels modificadors el caràcter que ha d'escriure la
 * tecla.
 */
private HashMap chars = null;

/**
 * Constructor de la classe indicant el tipus de tecla.
 * @param type Tipus de tecla a crear.
 */
public KeyboardKey(KeyType type)
{
    super();
    keyType = type;
}

/**
 * Constructor de la classe per defecte.
 */
public KeyboardKey()
{
    super();
```

```
        keyType = KeyType.CHARACTER;
    }

    /**
     * Indica per cada estat de modificador del teclat el caràcter que ha
     * d'escriure la tecla. Inicialitza l'atribut chars.
     * @param characters Llista dels caràcter que ha d'escriure la tecla.
     */
    public void setCharacters(String... characters)
    {
        chars = new HashMap(characters.length);

        chars.put("default", characters[0]);

        if(characters.length > 1)
        {
            chars.put("shift", characters[1]);

            if (characters.length > 2)
            {

                if(!"".equals(characters[2]))
                {
                    chars.put("alt", characters[2]);
                }

                if (characters.length > 3)
                {
                    chars.put("openaccent", characters[3]);
                    chars.put("closeaccent", characters[4]);
                    chars.put("diaeresis", characters[5]);
                    chars.put("mayusopenaccent", characters[6]);
                    chars.put("mayuscloseaccent", characters[7]);
                    chars.put("mayusdiaeresis", characters[8]);
                }

            }
        }
    }

    /**
     * @return the hasIcon
     */
    public boolean isHasIcon()
    {
        return hasIcon;
    }

    /**
     * @param hasIcon the hasIcon to set
     */
    public void setHasIcon(boolean hasIcon)
    {
```

```
        this.hasIcon = hasIcon;
    }

    /**
     * @return the imageIcon
     */
    public BufferedImage getImageIcon()
    {
        return imageIcon;
    }

    /**
     * @param imageIcon the imageIcon to set
     */
    public void setImageIcon(BufferedImage imageIcon)
    {
        this.imageIcon = imageIcon;
    }

    /**
     * Carrega dels recursos del programa la icona d'una tecla
     * @param resourcePath Ruta de la icona.
     */
    public void setImageIcon(String resourcePath)
    {
        try {
            setImageIcon(ImageIO.read(getClass().getResourceAsStream(resourcePath)));
        } catch (IOException ex) {

        }
    }

    /**
     * @return the buttonExited
     */
    public boolean isButtonExited() {
        return buttonExited;
    }

    /**
     * @param buttonExited the buttonExited to set
     */
    public void setButtonExited(boolean buttonExited) {
        this.buttonExited = buttonExited;
    }

    /**
     * Classe que implementa els mètodes d'interacció amb el cursor de la tecla.
     */
    protected class MouseAdapter extends java.awt.event.MouseAdapter
    {
        /**
         * S'executa quan el cursor surt de la tecla.
```

```
         * @param evt
         */
        @Override
        public void mouseExited(java.awt.event.MouseEvent evt)
        {
            keyExited();
        }

        /**
         * S'executa quan el cursor entra a la tecla.
         * @param evt
         */
        @Override
        public void mouseEntered(java.awt.event.MouseEvent evt)
        {
            keyEntered();
        }

        /**
         * S'executa quan fem clic a la tecla.
         * @param evt
         */
        @Override
        public void mousePressed(java.awt.event.MouseEvent evt)
        {
            keyPressed();
        }
    }

    /**
     * Quan el cursor entra a la tecla i utilitzem la funció d'escriure mantenint
     * el cursor sobre la tecla, engeguem el fil d'execució associat per
     * posteriorment saber si hem d'escriure el text de la tecla.
     */
    public void keyEntered()
    {
        if(keyboard.isDwellEnabled())
        {
            if(!dwellThread.isAlive()){
                dwellThread = new DwellThread(this);
                dwellThread.start();
            }
        }
    }

    /**
     * Quan el cursor surt de la tecla ho indiquem per tal de que el fil
     * d'execució de la pulsació per manteniment sàpiga si ha d'escriure la
     * tecla o no.
     */
    public void keyExited()
    {
        if(keyboard.isDwellEnabled())
```

```
        {
            if(dwellThread.isAlive())
                buttonExited = true;
        }
    }

    /**
    * Quan fem clic a la tecla executem l'acció d'escriptura.
    */
    public void keyPressed()
    {
        typeOperation(false);
    }

    /**
     * Instància de l'adaptador de ratolí utilitzada per la classe.
     */
    MouseAdapter mouseAdapter = new MouseAdapter();

   /**
    * Realitza la escriptura de la tecla en la caixa de text de la pantalla
     * principal i en l'aplicació activa del sistema operatiu. En cas d'alguna
     * tecla modificadora llancem els esdeveniments de modificador corresponents.
    * @param isDwell Indica si hem polsat fent clic o mantenint el cursor.
    */
    public void typeOperation(boolean isDwell)
    {
        try {
            String key = "";
            String keyName = getName();

            if(isDwell)
            {
                model.setPressed(true);
                model.setArmed(true);
            }
            Toolkit.getDefaultToolkit().beep();

            switch (getKeyType())
            {
                /*
                 * En aquest cas llancem els esdeveniments corresponents per
                 * activar o desactivar l'estat dels modificadors del teclat.
                 */
                case MODIFIER:

                    if (keyName.equals("CapsLock"))
                    {
                        keyboard.isCapsLock = !keyboard.isCapsLock;
                        if(!keyboard.isCapsLock)
                        {
                            keyboard.changeModifierState(false,keyboard.getCapsLock());
                        }
```

```
            fireModifierEvent(new
          ModifierEvent(this,ModifierType.CAPS_LOCK,keyboard.isCapsLock));
        }
        else if (keyName.equals("ShiftRight") || keyName.equals("ShiftLeft"))
        {
            updateModifier("Shift", false, new ModifierEvent(this,
                ModifierType.SHIFT),
                    this, keyboard.getShiftLeft(), keyboard.getShiftRight());
        }
        else if (keyName.equals("Function"))
        {
            keyboard.isFunction = !keyboard.isFunction;
        }
        else if (keyName.equals("AltRight") || keyName.equals("AltLeft"))
        {
            updateModifier("Alt", false, new ModifierEvent(this,
                ModifierType.ALT),
                    this, keyboard.getAltLeft(), keyboard.getAltRight());
        }
        else if (keyName.equals("Control"))
        {
            keyboard.isControl = !keyboard.isControl;
            keyboard.changeModifierState(keyboard.isControl,
                keyboard.getControl());
        }
        else if (keyName.equals("ComandRight") ||
                keyName.equals("ComandLeft"))
        {
            keyboard.isComand = !keyboard.isComand;
            keyboard.changeModifierState(keyboard.isComand,
                keyboard.getComandLeft(), keyboard.getComandRight());
        }

        return;


    /*
     * En el cas dels modificadors de text, executem la funció
     * nativa que tracta aquests tipus de tecla.
     */
    case TEXTMODIFIER:

        if("Tab".equals(keyName))
        {
            if(keyboard.isShift)
            {

            }
            else
            {
                key = "\t";
            }
        }
```

```
        else if ("Backspace".equals(keyName))
        {
            String text = keyboard.output.getText();
            if(text.length() > 0)
                keyboard.output.getDocument().remove(text.length()-1, 1);
            if(isDwell) typeSignal();
            keyboard.setTextModifier(keyName);
            return;
        }
        else if("Enter".equals(keyName))
        {
            key = "\n";
        }
        else
        {
            key = " ";
        }

        if(isDwell)
        {
            typeSignal();
        }

        keyboard.output.getDocument().insertString(keyboard.output.getText().len
                gth() ,key, null);
         keyboard.setTextModifier(keyName);
        return;

    case OPTION:

        if(isDwell)
        {
            typeSignal();
        }
        break;

    /*
     * En el cas dels accents i la dièresi gestionem els seus
     * esdeveniments, en cas d'una tecla normal escrivim el text que
     * es mostra al teclat virtual. Utilitzem la funció nativa que
     * escriu text a la aplicació activa del sistema.
     */
    default:

        if(keyName != null)
        {
            if (keyName.equals("OpenAccent") && !keyboard.isCloseAccent
             && !keyboard.isDiaeresis && !keyboard.isAlt && !keyboard.isShift)
            {
                updateModifier("OpenAccent", false, new ModifierEvent(this,
                        ModifierType.OPEN_ACCENT),
                         this, this);
                return;
```

```
        }
        if (keyName.equals("CloseAccent"))
        {
            if (keyboard.isShift && !keyboard.isCloseAccent && !
                    keyboard.isOpenAccent)
            {
                keyboard.isDiaeresis = !keyboard.isDiaeresis;
                keyboard.changeModifierState(false,
                    keyboard.getCloseAccent());

                updateModifier("Shift", true, new ModifierEvent(this,
                    ModifierType.SHIFT, false),
                     keyboard.getShiftLeft(), keyboard.getShiftLeft(),
                    keyboard.getShiftRight());

                if(keyboard.isAlt)
                {
                     updateModifier("Alt", true, new ModifierEvent(this,
                            ModifierType.ALT, false),
                             keyboard.getAltLeft(), keyboard.getAltLeft(),
                            keyboard.getAltRight());
                }
                fireModifierEvent(new
  ModifierEvent(this,ModifierType.DIAERESIS,keyboard.isDiaeresis));
                return;
            }
            else if(!keyboard.isAlt && !keyboard.isOpenAccent && !
                    keyboard.isDiaeresis)
            {
                updateModifier("CloseAccent", false, new
                    ModifierEvent(this,ModifierType.CLOSE_ACCENT),
                     this, this);
                return;
            }
        }
}

key = getText();

if (keyboard.isCapsLock)
{
    proccessAccentTyping();

    if(keyboard.isAlt)
    {
        updateModifier("Alt", true, new
                ModifierEvent(this,ModifierType.ALT,false),
               keyboard.getAltLeft(), keyboard.getAltLeft(),
                keyboard.getAltRight());
    }

    if(keyboard.isShift)
    {
```

```
                        updateModifier("Shift", true, new
                                ModifierEvent(this,ModifierType.ALT,false),
                                keyboard.getShiftLeft(), keyboard.getShiftLeft(),
                                keyboard.getShiftRight());
                    }

                }
                else if (keyboard.isShift) //case for "shift" and "shift and alt"
                {
                    proccessAccentTyping();

                    updateModifier("Shift", true, new
                        ModifierEvent(this,ModifierType.ALT,false),
                         keyboard.getShiftLeft(), keyboard.getShiftLeft(),
                        keyboard.getShiftRight());

                    if(keyboard.isAlt)
                    {
                        updateModifier("Alt", true, new
                                ModifierEvent(this,ModifierType.ALT,false),
                                 keyboard.getAltLeft(), keyboard.getAltLeft(),
                                keyboard.getAltRight());
                    }
                }

                else if (keyboard.isAlt) //case for only "alt"
                {
                    proccessAccentTyping();

                    updateModifier("Alt", true, new
                        ModifierEvent(this,ModifierType.ALT,false),
                         keyboard.getAltLeft(), keyboard.getAltLeft(),
                        keyboard.getAltRight());

                }
                else //case without general modifiers
                {
                    proccessAccentTyping();
                }

                if(isDwell) typeSignal();
                break;
            }

            keyboard.setKeyAtComponent(key);
            keyboard.output.getDocument().insertString(keyboard.output.getText().length()
               ,key, null);

        } catch (BadLocationException ex) {
            Logger.getLogger(KeyboardKey.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

```
    /**
     * Actualitza l'estat dels modificadors del teclat tant gràficament com
     * lògicament i llancem l'esdeveniment corresponent.
     * @param keyString Nom del modificador a tractar.
     * @param isUnpress Indica si hem d'activar o desactivar el modificador.
     * @param evt Esdeveniment de modificador que hem de llançar.
     * @param modifier Tecla que ha generat l'esdeveniment.
     * @param keysToDeselect Indica els modificadors que s'ha d'actualitzar
     *                       gràficament.
     */
    private void updateModifier(String keyString, boolean isUnpress, ModifierEvent evt,
KeyboardKey modifier, KeyboardKey ... keysToDeselect)
    {
        boolean state;

        if(keyString.equals("Shift"))
        {
            state = keyboard.isShift = !keyboard.isShift;
        }
        else if(keyString.equals("Alt"))
        {
            state = keyboard.isAlt = !keyboard.isAlt;
        }
        else if(keyString.equals("OpenAccent"))
        {
            state = keyboard.isOpenAccent = !keyboard.isOpenAccent;
        }
        else if(keyString.equals("CloseAccent"))
        {
            state = keyboard.isCloseAccent = !keyboard.isCloseAccent;
        }
        else
        {
            state = keyboard.isDiaeresis = !keyboard.isDiaeresis;
        }

        keyboard.changeModifierState(state, keysToDeselect);


        if (!isUnpress)
        {
            evt.setToPress(state);
        }

        modifier.fireModifierEvent(evt);
    }

    /**
     * Serveix per gestionar els accents i la dièresi.
     */
    private void proccessAccentTyping()
    {
        if(isVocal && (keyboard.isOpenAccent || keyboard.isCloseAccent ||
```

```
keyboard.isDiaeresis))
            {
                if(keyboard.isOpenAccent)
                {
                    updateModifier("OpenAccent", true, new
ModifierEvent(this,ModifierType.OPEN_ACCENT,false),
                            keyboard.getOpenAccent(),keyboard.getOpenAccent());
                }
                else if(keyboard.isCloseAccent)
                {
                    updateModifier("CloseAccent", true, new
ModifierEvent(this,ModifierType.CLOSE_ACCENT,false),
                            keyboard.getCloseAccent(),keyboard.getCloseAccent());
                }
                else
                {
                    keyboard.isDiaeresis = false;
                    keyboard.getCloseAccent().fireModifierEvent(new
ModifierEvent(this,ModifierType.DIAERESIS,keyboard.isDiaeresis));
                }
            }
        }

        /**
         * Simula la pulsació gràfica de la tecla.
         */
        private void typeSignal()
        {
            try
            {
                Thread.sleep(5);
            } catch (InterruptedException ex)
            {
                Logger.getLogger(KeyboardKey.class.getName()).log(Level.SEVERE, null, ex);
            }

            model.setPressed(false);
        }

        /**
         * Serveix per modificar les dimensions de la icona d'una tecla quan canviem
         * la mida del teclat.
         * @param srcImg Imatge original de la tecla.
         * @param w Indica l'amplada de la tecla en les noves dimensions.
         * @param h Indica l'alçada de la tecla en les noves dimensions.
         * @return La imatge en les noves dimensions.
         */
        private Image getScaledImage(Image srcImg, int w, int h){
            BufferedImage resizedImg = new BufferedImage(w, h, BufferedImage.BITMASK);
            Graphics2D g2 = resizedImg.createGraphics();
            g2.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);
            g2.drawImage(srcImg, 0, 0, w, h, null);
```

```
        g2.dispose();
        return resizedImg;
    }

    /**
     * Conté la icona de la tecla.
     */
    protected BufferedImage imageIcon;

    /**
     * Funció que s'executa quan es produeix un canvi en les dimensions del
     * teclat. L'executa cada tecla quan rep l'esdeveniment d'ajust de text.
     */
    @Override
    public void adjustText()
    {
        if(hasIcon)
        {
            ImageIcon icon = new
ImageIcon(getScaledImage(imageIcon,getWidth()/2,getHeight()/2));
            setIcon(icon);
        }
        else
        {
            Font actualFont = getFont();
            FontMetrics fontMetrics = getFontMetrics(actualFont);

            Rectangle2D textRectangle = fontMetrics.getStringBounds(getText(),
getGraphics());

            int textWidth = (int) textRectangle.getWidth();
            int textHeight = (int) textRectangle.getHeight();

            Rectangle2D keyboardKeyBounds = getBounds().getBounds2D();

            int keyboardKeyWidth = (int) keyboardKeyBounds.getWidth();
            int keyboardKeyHeight = (int) keyboardKeyBounds.getHeight();

            float actualSize = actualFont.getSize();

            if(textWidth > keyboardKeyWidth/2 || textHeight > keyboardKeyHeight/2)
            {
                float newFontSize = --actualSize;
                if(!(newFontSize < KeyboardKey.SMALL_FONT_SIZE))
                {
                    setFont(actualFont.deriveFont(newFontSize));
                }
            }
            else
            {
                float newFontSize = ++actualSize;
                if(!(newFontSize > KeyboardKey.BIG_FONT_SIZE))
                {
```

```
                    setFont(actualFont.deriveFont(newFontSize));
                }
            }
        }
    }

    /**
     * @return the chars
     */
    public HashMap getChars()
    {
        return chars;
    }

    /**
     * @param chars the chars to set
     */
    public void setChars(HashMap chars)
    {
        this.chars = chars;
    }

    /**
     * Llista que conté totes les tecles que esperen esdeveniments de la tecla.
     */
    protected EventListenerList modifierListeners = new EventListenerList();

    /**
     * Serveix per omplir la llista de tecles que esperen el nostre esdeveniment
     * @param listener Escoltador de la tecla que rep els nostres esdeveniments.
     */
    public synchronized void addModifierEventListener(ModifierEventListener listener)
    {
        modifierListeners.add(ModifierEventListener.class, listener);
    }

    /**
     * Serveix per eliminar de la llista algunes de les tecles que esperen els
     * nostres esdeveniments.
     * @param listener Escoltador de la tecla a eliminar.
     */
    public synchronized void removeModifierEventListener(ModifierEventListener listener)
    {
        modifierListeners.remove(ModifierEventListener.class, listener);
    }

    /**
     * Llança els esdeveniments de modificador en cas que la tecla en tingui.
     * Depenent del que l'hi indiqui l'esdeveniment a llançar farà executar a
     * les tecles que rebin l'esdeveniment l'acció de resposta corresponent.
     * @param e Esdeveniment de modificador que s'ha de llançar.
     */
    protected synchronized void fireModifierEvent(ModifierEvent e)
```

```
{
    ModifierType modifierType = e.getModifierType();
    ModifierEventListener listener;
    Object[] listeners = modifierListeners.getListenerList();
    for (int i = 0; i < listeners.length; i+=2)
    {
        if(listeners[i]==ModifierEventListener.class)
        {
            listener = ((ModifierEventListener)listeners[i+1]);
            if(e.isToPress())
            {
                switch(modifierType)
                {
                    case CAPS_LOCK:
                        listener.onPressCapsLock(e);
                        break;
                    case SHIFT:
                        listener.onPressShift(e);
                        break;
                    case ALT:
                        listener.onPressAlt(e);
                        break;
                    case OPEN_ACCENT:
                        listener.onPressOpenAccent(e);
                        break;
                    case CLOSE_ACCENT:
                        listener.onPressCloseAccent(e);
                        break;
                    default:
                        listener.onPressDiaeresis(e);
                }
            }
            else
            {
                switch(modifierType)
                {
                    case CAPS_LOCK:
                        listener.onUnpressCapsLock(e);
                        break;
                    case SHIFT:
                        listener.onUnpressShift(e);
                        break;
                    case ALT:
                        listener.onUnpressAlt(e);
                        break;
                    case OPEN_ACCENT:
                        listener.onUnpressOpenAccent(e);
                        break;
                    case CLOSE_ACCENT:
                        listener.onUnpressCloseAccent(e);
                        break;
                    default:
                        listener.onUnpressDiaeresis(e);
```

```
                    }
                }
            }
        }
    }

    /**
     * @return the modifierListeners
     */
    public EventListenerList geModifierListeners()
    {
        return modifierListeners;
    }

    /**
     * @param modifierListeners the modifierListeners to set
     */
    public void setModifierListeners(EventListenerList modifierListeners)
    {
        this.modifierListeners = modifierListeners;
    }

    /**
     * S'executa quan activem el modificador Shift.
     * @param e
     */
    @Override
    public void onPressShift(ModifierEvent e)
    {
        if(!keyboard.isCapsLock)
        {
            if(!
        processAccentView("mayusopenaccent","mayuscloseaccent","mayusdiaeresis"))
            {
                setText((String)chars.get("shift"));
            }
        }
        else
        {
            if(keyType == KeyType.VARCHARACTER)
            {
                setText((String)chars.get("shift"));
            }
        }
    }

    /**
     * Funció que s'executa quan es desactiva la tecla Shift.
     * @param e
     */
    @Override
    public void onUnpressShift(ModifierEvent e)
    {
```

```
            if(keyboard.isCapsLock)
            {
                if(!
processAccentView("mayusopenaccent","mayuscloseaccent","mayusdiaeresis"))
                {
                    if(keyType == KeyType.VARCHARACTER)
                    {
                        setText((String)chars.get("default"));
                    }
                }
            }
            else
            {
                if(!processAccentView("openaccent","closeaccent","diaeresis"))
                {
                    if(keyboard.isAlt && chars.containsKey("alt"))
                    {
                            setText((String)chars.get("alt"));
                    }
                    else
                    {
                        setText((String)chars.get("default"));
                    }
                }
            }
        }

        /**
         * Funció que s'executa quan s'activa la tecla CapsLock.
         * @param e
         */
        @Override
        public void onPressCapsLock(ModifierEvent e)
        {
            if(!keyboard.isShift)
            {
                if(!
processAccentView("mayusopenaccent","mayuscloseaccent","mayusdiaeresis"))
                {
                        setText((String)chars.get("shift"));
                }
            }
        }

        /**
         * Funció que s'executa quan es desactiva la tecla CapsLock.
         * @param e
         */
        @Override
        public void onUnpressCapsLock(ModifierEvent e)
        {
            if(!keyboard.isShift)
            {
```

```
                if(keyboard.isAlt)
                {
                    if(chars.containsKey("alt"))
                    {
                        setText((String)chars.get("alt"));
                    }
                    else
                    {
                        setText((String)chars.get("default"));
                    }
                }
                else
                {
                    if(!processAccentView("openaccent","closeaccent","diaeresis"))
                    {
                        setText((String)chars.get("default"));
                    }
                }
            }
        }

        /**
         * Funció que s'executa quan s'activa la tecla Alt.
         * @param e
         */
        @Override
        public void onPressAlt(ModifierEvent e)
        {
            setText((String)chars.get("alt"));
        }

        /**
         * Funció que s'executa quan desactiva la tecla Alt.
         * @param e
         */
        @Override
        public void onUnpressAlt(ModifierEvent e)
        {
            if(keyboard.isShift)
            {
                setText((String)chars.get("shift"));
            }
            else if (keyboard.isCapsLock && !
processAccentView("mayusopenaccent","mayuscloseaccent","mayusdiaeresis"))
            {
                if(keyType == KeyType.VARCHARACTER)
                {
                    setText((String)chars.get("default"));
                }
                else
                {
                    setText((String)chars.get("shift"));
                }
```

```
        }
        else
        {
            if(!processAccentView("openaccent","closeaccent","diaeresis"))
            {
                setText((String)chars.get("default"));
            }
        }
}

/**
 * Funció que s'executa quan s'activa l'accent obert.
 * @param e
 */
@Override
public void onPressOpenAccent(ModifierEvent e)
{
    executeAccentEvent("mayusopenaccent","openaccent");
}

/**
 *  Funció que s'executa quan es desactiva l'accent obert.
 * @param e
 */
@Override
public void onUnpressOpenAccent(ModifierEvent e)
{
    executeAccentEvent("shift","default");
}

/**
 *  Funció que s'executa quan s'activa l'accent tancat.
 * @param e
 */
@Override
public void onPressCloseAccent(ModifierEvent e)
{
    executeAccentEvent("mayuscloseaccent","closeaccent");
}

/**
 * Funció que s'executa quan es desactiva l'accent tancat.
 * @param e
 */
@Override
public void onUnpressCloseAccent(ModifierEvent e)
{
    executeAccentEvent("shift","default");
}

/**
 * Funció que s'executa quan s'activa la dièresi.
 * @param e
```

```
     */
    @Override
    public void onPressDiaeresis(ModifierEvent e)
    {
        executeAccentEvent("mayusdiaeresis","diaeresis");
    }

    /**
     * Funció que s'executa quan es desactiva la dièresi
     * @param e
     */
    @Override
    public void onUnpressDiaeresis(ModifierEvent e)
    {
        executeAccentEvent("shift","default");
    }

    /**
     * Serveix per modificar el text de les tecles que són afectades pels
     * accents i la dièresi.
     * @param closeaccentText Text per l'accent obert.
     * @param openaccentTest Text per l'accent tancat.
     * @param diaeresisText Text per la dièresi.
     * @return Indica si la tecla estava afectada pels accents o la dièresi.
     */
    private boolean processAccentView(String openaccentText, String closeaccentText,
String diaeresisText)
    {
        if(isVocal && (keyboard.isCloseAccent || keyboard.isOpenAccent ||
keyboard.isDiaeresis))
        {
                if(keyboard.isCloseAccent)
                {
                    setText((String) chars.get(closeaccentText));
                }
                else if(keyboard.isOpenAccent)
                {
                    setText((String) chars.get(openaccentText));
                }
                else
                {
                    setText((String) chars.get(diaeresisText));
                }

                return true;
         }

        return false;
    }

    /**
     * Depenent de l'estat de CapsLock obtenim el text que han de tenir les
     * tecles afectades pels accents o la dièresi.
```

```
     * @param afirmative Text que s'ha d'escriure en cas que CapsLock estigui
     *                   actiu.
     * @param negative Text que s'ha d'escriure en cas que CapsLock estigui
     *                 inactiu.
     */
    private void executeAccentEvent(String afirmative, String negative)
    {
        if(keyboard.isCapsLock)
        {
            setText((String) chars.get(afirmative));
        }
        else
        {
            setText((String) chars.get(negative));
        }
    }

    /**
     * @return the isVocal
     */
    public boolean isIsVocal()
    {
        return isVocal;
    }
    /**
     * @param isVocal the isVocal to set
     */
    public void setIsVocal(boolean isVocal)
    {
        this.isVocal = isVocal;
    }

    /**
     * Get the value of keyType.
     * @return The value of keyType.
     */
    public KeyType getKeyType()
    {
        return keyType;
    }

    /**
     * Set the value of keyType.
     * @param keyType new value of keyType.
     */
    public void setKeyType(KeyType keyType)
    {
        this.keyType = keyType;
    } }
```

6.1.9. KeyType

```
package keysforfree;

/**
 * Classe enumerativa que defineix els tipus de tecla
 * @author eloicreus
 */
public enum KeyType {
    CHARACTER,VARCHARACTER,MODIFIER,OPTION,TEXTMODIFIER
}
```

6.1.10. MainWindow

```
package keysforfree;

import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.prefs.BackingStoreException;
import javax.swing.JFrame;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import java.util.prefs.Preferences;
import javax.swing.JOptionPane;
import javax.swing.text.BadLocationException;

/**
 * Classe que conté la pantalla amb el teclat virtual.
 * @author eloicreus
 */
public class MainWindow extends JFrame
{
    /**
     * Indica el sistema operatiu on estem executant el programa.
     */
    public static String OSName = System.getProperty("os.name");

    public static int DEFAULT_WIDTH = 430;
    public static int DEFAULT_HEIGHT = 350;

    public static int DEFAULT_X = 40;
    public static int DEFAULT_Y = 22;

    /**
     * Indica l'alçada que té la pantalla.
     */
    public int height;
    /**
     * Indica l'amplada que té la pantalla.
     */
    public int width;

    /**
     * Indica la posició x de la pantalla.
     */
    public int x;
    /**
     * Indica la posició y de la pantalla.
     */
    public int y;

    /**
     * Indica si hem pogut carregar la llibreria nativa del sistema operatiu.
     */
    private static boolean nativeLibraryLoaded = false;
```

```
/**
 * Conté preferències de l'usuari.
 */
public Preferences preferences;

/**
 * Conté la finestra de predicció.
 */
public PredictorWindow predictorWindow;

/**
 * Constructor de la classe que inicialitza els seus components i carrega
 * les preferències de la pantalla principal del programa.
 */
public MainWindow()
{
    loadPreferences();
    initComponents();
    predictorWindow = new PredictorWindow(keyboard);
}

/**
 * Carrega les preferències d'usuari de la pantalla principal.
 */
private void loadPreferences()
{
    preferences = Preferences.userNodeForPackage(getClass()).node("mainwindow");

    height = preferences.getInt("height", DEFAULT_HEIGHT);
    width = preferences.getInt("width", DEFAULT_WIDTH);

    x = preferences.getInt("x", DEFAULT_X);
    y = preferences.getInt("y", DEFAULT_Y);

}

/**
 * Guarda les preferències d'usuari del programa.
 */
private void savePreferences()
{
    try {
        preferences.putInt("width", getWidth());
        preferences.putInt("height", getHeight());
        preferences.putInt("x", getX());
        preferences.putInt("y", getY());
        DwellThread.savePreferences();
        DatabaseConnection.savePreferences();
        preferences.flush();
    } catch (BackingStoreException ex) {
        Logger.getLogger(MainWindow.class.getName()).log(Level.SEVERE, null, ex);
    }
```

```
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        dialogDwellTime = new javax.swing.JDialog();
        buttonOKDwellTime = new javax.swing.JButton();
        comboBoxDwellTimes = new javax.swing.JComboBox();
        buttonCancelDwellTime = new javax.swing.JButton();
        labelMessageDwellTime = new javax.swing.JLabel();
        dialogDatabaseOptions = new javax.swing.JDialog();
        labelUser = new javax.swing.JLabel();
        labelPassword = new javax.swing.JLabel();
        textFieldUser = new javax.swing.JTextField();
        buttonCancelDatabaseOptions = new javax.swing.JButton();
        buttonOKDatabaseOptions = new javax.swing.JButton();
        textFieldPassword = new javax.swing.JPasswordField();
        scrollPanelAreaTyping = new javax.swing.JScrollPane();
        textAreaTyping = new javax.swing.JTextArea();
        keyboard = new keysforfree.BasicKeyboard(textAreaTyping);
        menuBarMain = new javax.swing.JMenuBar();
        menuBarFile = new javax.swing.JMenu();
        menuItemDatabaseOptions = new javax.swing.JMenuItem();
        menuBarEdit = new javax.swing.JMenu();
        menuKeyboard = new javax.swing.JMenu();
        checkBoxMenuItemDwellActive = new javax.swing.JCheckBoxMenuItem();
        menuItemDwellTime = new javax.swing.JMenuItem();
        menuShowPrediction = new javax.swing.JMenu();
        menuClearText = new javax.swing.JMenu();

        dialogDwellTime.setTitle("Keys4Free - Dwell time");
        dialogDwellTime.setAlwaysOnTop(true);

        buttonOKDwellTime.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        buttonOKDwellTime.setText("OK");
        buttonOKDwellTime.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonOKDwellTimeActionPerformed(evt);
            }
        });

        comboBoxDwellTimes.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        comboBoxDwellTimes.setModel(new javax.swing.DefaultComboBoxModel(new
String[] { "500 miliseconds", "1 second", "2 seconds" }));

        buttonCancelDwellTime.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        buttonCancelDwellTime.setText("Cancel");
        buttonCancelDwellTime.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonCancelDwellTimeActionPerformed(evt);
            }
        });
```

```
        labelMessageDwellTime.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        labelMessageDwellTime.setText("Select the time to dwell:");

        org.jdesktop.layout.GroupLayout dialogDwellTimeLayout = new
org.jdesktop.layout.GroupLayout(dialogDwellTime.getContentPane());
        dialogDwellTime.getContentPane().setLayout(dialogDwellTimeLayout);
        dialogDwellTimeLayout.setHorizontalGroup(
            dialogDwellTimeLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
            .add(dialogDwellTimeLayout.createSequentialGroup()
                .add(dialogDwellTimeLayout.createParallelGroup(org.jdesktop.layout.GroupLa
yout.LEADING)
                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
dialogDwellTimeLayout.createSequentialGroup()
                        .addContainerGap()
                        .add(buttonCancelDwellTime)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                        .add(buttonOKDwellTime,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 53,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(dialogDwellTimeLayout.createSequentialGroup()
                        .add(10, 10, 10)
                        .add(comboBoxDwellTimes, 0, 330, Short.MAX_VALUE))
                    .add(dialogDwellTimeLayout.createSequentialGroup()
                        .addContainerGap()
                        .add(labelMessageDwellTime)))
                .addContainerGap())
        );
        dialogDwellTimeLayout.setVerticalGroup(
            dialogDwellTimeLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
            .add(dialogDwellTimeLayout.createSequentialGroup()
                .add(25, 25, 25)
                .add(labelMessageDwellTime,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 17,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(comboBoxDwellTimes,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .add(13, 13, 13)
                .add(dialogDwellTimeLayout.createParallelGroup(org.jdesktop.layout.GroupLa
yout.BASELINE)
                    .add(buttonOKDwellTime,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 29,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(buttonCancelDwellTime,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );
```

```
        dialogDwellTime.pack();

        dialogDatabaseOptions.setTitle("Keys4Free - Database options");
        dialogDatabaseOptions.setAlwaysOnTop(true);

        labelUser.setText("User:");

        labelPassword.setText("Password:");

        buttonCancelDatabaseOptions.setText("Cancel");
        buttonCancelDatabaseOptions.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonCancelDatabaseOptionsActionPerformed(evt);
            }
        });

        buttonOKDatabaseOptions.setText("OK");
        buttonOKDatabaseOptions.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonOKDatabaseOptionsActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout dialogDatabaseOptionsLayout = new
org.jdesktop.layout.GroupLayout(dialogDatabaseOptions.getContentPane());
        dialogDatabaseOptions.getContentPane().setLayout(dialogDatabaseOptionsLayout);
        dialogDatabaseOptionsLayout.setHorizontalGroup(
            dialogDatabaseOptionsLayout.createParallelGroup(org.jdesktop.layout.GroupLayo
ut.LEADING)
            .add(dialogDatabaseOptionsLayout.createSequentialGroup()
                .addContainerGap()
                .add(dialogDatabaseOptionsLayout.createParallelGroup(org.jdesktop.layout.G
roupLayout.LEADING)
                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
dialogDatabaseOptionsLayout.createSequentialGroup()
                        .add(dialogDatabaseOptionsLayout.createParallelGroup(org.jdeskto
p.layout.GroupLayout.LEADING)
                            .add(labelUser)
                            .add(labelPassword))
                        .add(26, 26, 26)
                        .add(dialogDatabaseOptionsLayout.createParallelGroup(org.jdeskto
p.layout.GroupLayout.LEADING)
                            .add(textFieldPassword,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 227, Short.MAX_VALUE)
                            .add(textFieldUser,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 227, Short.MAX_VALUE)))
                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
dialogDatabaseOptionsLayout.createSequentialGroup()
                        .add(buttonCancelDatabaseOptions)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                        .add(buttonOKDatabaseOptions)))
```

```
                .addContainerGap())
        );
        dialogDatabaseOptionsLayout.setVerticalGroup(
            dialogDatabaseOptionsLayout.createParallelGroup(org.jdesktop.layout.GroupLayo
ut.LEADING)
            .add(dialogDatabaseOptionsLayout.createSequentialGroup()
                .addContainerGap()
                .add(dialogDatabaseOptionsLayout.createParallelGroup(org.jdesktop.layout.G
roupLayout.BASELINE)
                    .add(labelUser)
                    .add(textFieldUser,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .add(18, 18, 18)
                .add(dialogDatabaseOptionsLayout.createParallelGroup(org.jdesktop.layout.G
roupLayout.BASELINE)
                    .add(labelPassword)
                    .add(textFieldPassword,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .add(18, 18, 18)
                .add(dialogDatabaseOptionsLayout.createParallelGroup(org.jdesktop.layout.G
roupLayout.BASELINE)
                    .add(buttonCancelDatabaseOptions)
                    .add(buttonOKDatabaseOptions))
                .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        dialogDatabaseOptions.pack();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Keys4Free");
        setAlwaysOnTop(true);
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        setFocusableWindowState(false);
        setLocation(new java.awt.Point(x,y));
        setMinimumSize(new java.awt.Dimension(420, 350));
        setName("MainWindow"); // NOI18N
        setPreferredSize(new java.awt.Dimension(width,height));
        setSize(new java.awt.Dimension(width, height));
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosed(java.awt.event.WindowEvent evt) {
                formWindowClosed(evt);
            }
        });
        addWindowStateListener(new java.awt.event.WindowStateListener() {
            public void windowStateChanged(java.awt.event.WindowEvent evt) {
                formWindowStateChanged(evt);
            }
        });
```

```
        scrollPanelAreaTyping.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        scrollPanelAreaTyping.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.
HORIZONTAL_SCROLLBAR_NEVER);
        scrollPanelAreaTyping.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VE
RTICAL_SCROLLBAR_NEVER);
        scrollPanelAreaTyping.setViewportBorder(javax.swing.BorderFactory.createLineBorder
(new java.awt.Color(0, 0, 0)));

        textAreaTyping.setColumns(20);
        textAreaTyping.setEditable(false);
        textAreaTyping.setRows(5);
        textAreaTyping.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        textAreaTyping.setMinimumSize(new java.awt.Dimension(480, 10));
        textAreaTyping.getCaret().setVisible(true);
        textAreaTyping.getCaret().setSelectionVisible(true);
        scrollPanelAreaTyping.setViewportView(textAreaTyping);

        keyboard.setBorder(new javax.swing.border.MatteBorder(null));

        menuBarMain.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));

        menuBarFile.setText("File");

        menuItemDatabaseOptions.setText("Database options");
        menuItemDatabaseOptions.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemDatabaseOptionsActionPerformed(evt);
            }
        });
        menuBarFile.add(menuItemDatabaseOptions);

        menuBarMain.add(menuBarFile);

        menuBarEdit.setText("Edit");
        menuBarMain.add(menuBarEdit);

        menuKeyboard.setText("Keyboard");

        checkBoxMenuItemDwellActive.setText("Dwell active");
        checkBoxMenuItemDwellActive.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                checkBoxMenuItemDwellActiveActionPerformed(evt);
            }
        });
        menuKeyboard.add(checkBoxMenuItemDwellActive);

        menuItemDwellTime.setText("Dwell time");
        menuItemDwellTime.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemDwellTimeActionPerformed(evt);
```

```
            }
        });
        menuKeyboard.add(menuItemDwellTime);

        menuBarMain.add(menuKeyboard);

        menuShowPrediction.setText("Hide prediction");
        menuShowPrediction.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                menuShowPredictionMouseClicked(evt);
            }
        });
        menuBarMain.add(menuShowPrediction);

        menuClearText.setText("Clear text");
        menuClearText.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                menuClearTextMouseClicked(evt);
            }
        });
        menuBarMain.add(menuClearText);

        setJMenuBar(menuBarMain);

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(scrollPanelAreaTyping,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 500, Short.MAX_VALUE)
                    .add(keyboard, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 500,
Short.MAX_VALUE))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(scrollPanelAreaTyping,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 69,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(keyboard, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 161,
Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>
```

```
    /**
     * Avisa al teclat que canvii la mida del text de les tecles ja que hem
     * maximitzat la pantalla principal.
     * @param evt
     */
    private void formWindowStateChanged(java.awt.event.WindowEvent evt) {
        if(evt.getNewState() == JFrame.MAXIMIZED_BOTH)
            keyboard.fireTextAdjustEvent(new TextAdjustEvent(this));
    }

    /**
     * S'executa quan tanquem la pantalla principal.
     * @param evt
     */
    private void formWindowClosed(java.awt.event.WindowEvent evt) {
        savePreferences();
        predictorWindow.closeAction();
        System.exit(0);
    }

    /**
     * S'executa quan activem o desactivem l'opció d'escriure mantenint el
     * cursor sobre les tecles.
     * @param evt
     */
    private void checkBoxMenuItemDwellActiveActionPerformed(java.awt.event.ActionEvent
evt) {
        keyboard.setDwellEnabled(checkBoxMenuItemDwellActive.isSelected());
    }

    /**
     * S'executa quan amaguem o mostrem la pantalla de predicció.
     * @param evt
     */
    private void menuShowPredictionMouseClicked(java.awt.event.MouseEvent evt) {
        if(predictorWindow.isVisible())
        {
            predictorWindow.setVisible(false);
            menuShowPrediction.setText("Show prediction");
        }
        else
        {
            predictorWindow.setVisible(true);
            menuShowPrediction.setText("Hide prediction");
        }
    }

    /**
     * S'executa quan es vol canviar el temps que s'ha de mantindre el cursor
     * sobre la tecla que es vol escriure.
     * @param evt
     */
    private void menuItemDwellTimeActionPerformed(java.awt.event.ActionEvent evt) {
```

```
        dialogDwellTime.setLocationRelativeTo(this);
        dialogDwellTime.setVisible(true);
    }

    /**
     * S'executa quan hem triat el temps d'espera que hem de tenir el cursor
     * sobre la tecla.
     * @param evt
     */
    private void buttonOKDwellTimeActionPerformed(java.awt.event.ActionEvent evt) {
        String time = (String) comboBoxDwellTimes.getSelectedItem();
        if(time.equals("500 miliseconds"))
        {
            DwellThread.setMilisecondsDelay(500);
        }
        else if(time.equals("1 second"))
        {
            DwellThread.setMilisecondsDelay(1000);
        }
        else
        {
            DwellThread.setMilisecondsDelay(2000);
        }
        dialogDwellTime.setVisible(false);
    }

    /**
     * S'executa quan amaguem la pantalla per triar el temps d'espera a
     * mantindre el cursor sobre la tecla.
     * @param evt
     */
    private void buttonCancelDwellTimeActionPerformed(java.awt.event.ActionEvent evt) {
        dialogDwellTime.setVisible(false);
    }

    /**
     * S'executa quan volem mostrar la pantalla de configuració dels paràmetres
     * de la base de dades.
     * @param evt
     */
    private void menuItemDatabaseOptionsActionPerformed(java.awt.event.ActionEvent evt)
{
        dialogDatabaseOptions.setLocationRelativeTo(this);
        dialogDatabaseOptions.setVisible(true);
    }
    /**
     * S'executa quan amaguem la pantalla de configuració de la base de dades.
     * @param evt
     */
    private void buttonCancelDatabaseOptionsActionPerformed(java.awt.event.ActionEvent
evt) {
        textFieldUser.setText("");
        textFieldPassword.setText("");
```

```
            dialogDatabaseOptions.setVisible(false);
        }

        /**
         * S'executa quan canviem els paràmetres de la base de dades.
         * @param evt
         */
        private void buttonOKDatabaseOptionsActionPerformed(java.awt.event.ActionEvent evt)
{
            String user = textFieldUser.getText();
            char[] password = textFieldPassword.getPassword();
            if(user.length() > 0)
            {
                DatabaseConnection.setUSER(user);
                if(password.length > 0)
                {
                    DatabaseConnection.setPASSWORD(password.toString());
                }
                else
                {
                    DatabaseConnection.setPASSWORD("");
                }
                JOptionPane.showMessageDialog(this,
                                              "The database preferences has been
correctly saved",
                                              "Keys4Free",
                                              JOptionPane.INFORMATION_MESSAGE);
                textFieldUser.setText("");
                textFieldPassword.setText("");
            }
            dialogDatabaseOptions.setVisible(false);
        }

        /**
         * Esborra el text de la caixa de text de la pantalla principal.
         * @param evt
         */
        private void menuClearTextMouseClicked(java.awt.event.MouseEvent evt) {
            try {
                textAreaTyping.getDocument().insertString(keyboard.output.getText().length() ," ",
null);
                textAreaTyping.setText("");
            } catch (BadLocationException ex) {
                Logger.getLogger(MainWindow.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

        /**
         * Funció per Mac que posa la finestra en un estat en que no agafi el focus.
         */
        public native void setWindowInactivated();

        /**
```

```
     * Funció que inicia l'execució del programa
     * @param args the command line arguments
     */
    public static void main(String args[]) throws ClassNotFoundException,
UnsupportedLookAndFeelException, IllegalAccessException, InstantiationException
    {
        /*
         * Carreguem l'estil de la interfície gràfica a l'estil que és
         * multi-plataforma.
         */
        try
        {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFee
l");
        }
        catch(Exception e)
        {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassNam
e());
        }
        /*
         * Carreguem el driver per connectar-nos a la base de dades.
         */
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("MySQL Driver installed successfuly");
        }
        catch(Exception e)
        {
            System.out.println("Cannot found MySQL Driver");
        }

        System.out.println("Attempting to load library from "
                + System.getProperty("java.library.path"));

        /*
         * Carreguem la llibreria nativa del sistema operatiu que estem
         * utilitzant.
         */
        try
        {
            if(OSName.equals("Mac OS X"))
                System.loadLibrary("Macintosh");
            else if (OSName.contains("Windows"))
                System.loadLibrary("libWindows");
            else
                System.loadLibrary("Linux");
            nativeLibraryLoaded = true;
        }
        catch(UnsatisfiedLinkError ex)
        {
            System.out.println("The native library cannot be loaded");
```

```
        }

        /*
         * Construïm d'interfície gràfica del programa i la mostrem a la
         * pantalla.
         * En cas de Mac, executem la funció que fa que les finestres del
         * programa no agafin el cursor d'escriptura.
         */
        java.awt.EventQueue.invokeLater(new Runnable() {
            @Override
            public void run()
            {
                MainWindow window = new MainWindow();

                if(nativeLibraryLoaded && OSName.equals("Mac OS X"))
                {
                    window.setWindowInactivated();
                    window.predictorWindow.setWindowInactivated();
                }
                window.setVisible(true);
                window.predictorWindow.setVisible(true);
            }
        });

    }

    // Variables declaration - do not modify
    private javax.swing.JButton buttonCancelDatabaseOptions;
    private javax.swing.JButton buttonCancelDwellTime;
    private javax.swing.JButton buttonOKDatabaseOptions;
    private javax.swing.JButton buttonOKDwellTime;
    private javax.swing.JCheckBoxMenuItem checkBoxMenuItemDwellActive;
    private javax.swing.JComboBox comboBoxDwellTimes;
    private javax.swing.JDialog dialogDatabaseOptions;
    private javax.swing.JDialog dialogDwellTime;
    private keysforfree.BasicKeyboard keyboard;
    private javax.swing.JLabel labelMessageDwellTime;
    private javax.swing.JLabel labelPassword;
    private javax.swing.JLabel labelUser;
    private javax.swing.JMenu menuBarEdit;
    private javax.swing.JMenu menuBarFile;
    private javax.swing.JMenuBar menuBarMain;
    private javax.swing.JMenu menuClearText;
    private javax.swing.JMenuItem menuItemDatabaseOptions;
    private javax.swing.JMenuItem menuItemDwellTime;
    private javax.swing.JMenu menuKeyboard;
    private javax.swing.JMenu menuShowPrediction;
    private javax.swing.JScrollPane scrollPanelAreaTyping;
    private javax.swing.JTextArea textAreaTyping;
    private javax.swing.JPasswordField textFieldPassword;
    private javax.swing.JTextField textFieldUser;
    // End of variables declaration
}
```

6.1.11. ModifierEvent

```
package keysforfree;

import java.util.EventObject;

/**
 * Classe que serveix per crear els esdeveniments de les tecles modificadores
 * @author eloicreus
 */
public class ModifierEvent extends EventObject
{
    /**
     * Indica el tipus d'esdeveniment de modificador
     */
    private ModifierType modifierType;
    /**
     * Indica si l'esdeveniment és d'activació o de desactivació
     */
    private boolean toPress;

    /**
     * Crea un esdeveniment del tipus indicat i indica si és d'activació o de
     * desactivació
     * @param source Paràmetre utilitzar pel constructor de la superclasse
     * @param type Tipus d'esdeveniment de modificador
     * @param press Indica si és d'activació o de desactivació
     */
    public ModifierEvent(Object source, ModifierType modifierType, boolean toPress)
    {
        super(source);
        this.modifierType = modifierType;
        this.toPress = toPress;
    }

    /**
     * Crea un esdeveniment del tipus indicat
     * @param source Paràmetre utilitzat pel constructor de la superclasse
     * @param modifierType Tipus d'esdeveniment de modificador
     */
    public ModifierEvent(Object source, ModifierType modifierType)
    {
        super(source);
        this.modifierType = modifierType;
    }

    /**
     * @return the toPress
     */
    public boolean isToPress()
    {
        return toPress;
    }
```

```
    /**
     * @return the modifierType
     */
    public ModifierType getModifierType()
    {
        return modifierType;
    }

    /**
     * @param toPress the toPress to set
     */
    public void setToPress(boolean toPress)
    {
        this.toPress = toPress;
    }
}
```

6.1.12. ModifierEventListener

```
package keysforfree;

import java.util.EventListener;

/**
 * Interfície que defineix les funcions que han de implementar les classes que
 * vulguin rebre esdeveniments ModifierEvent
 * @author eloicreus
 */
public interface ModifierEventListener extends EventListener
{

    /**
     * Funció que s'executa quan s'activa la tecla Shift.
     * @param e
     */
    public void onPressShift(ModifierEvent e);

    /**
     * Funció que s'executa quan es desactiva la tecla Shift.
     * @param e
     */
    public void onUnpressShift(ModifierEvent e);

    /**
     * Funció que s'executa quan s'activa la tecla CapsLock.
     * @param e
     */
    public void onPressCapsLock(ModifierEvent e);

    /**
     * Funció que s'executa quan es desactiva  la tecla CapsLock.
     * @param e
     */
    public void onUnpressCapsLock(ModifierEvent e);

    /**
     * Funció que s'executa quan s'activa la tecla Alt.
     * @param e
     */
    public void onPressAlt(ModifierEvent e);

    /**
     * Funció que s'executa quan desactiva la tecla Alt.
     * @param e
     */
    public void onUnpressAlt(ModifierEvent e);

    /**
     * Funció que s'executa quan s'activa l'accent obert.
     * @param e
```

```
     */
    public void onPressOpenAccent(ModifierEvent e);

    /**
     *  Funció que s'executa quan es desactiva l'accent obert.
     * @param e
     */
    public void onUnpressOpenAccent(ModifierEvent e);

    /**
     *  Funció que s'executa quan s'activa l'accent tancat.
     * @param e
     */
    public void onPressCloseAccent(ModifierEvent e);

    /**
     * Funció que s'executa quan es desactiva l'accent tancat.
     * @param e
     */
    public void onUnpressCloseAccent(ModifierEvent e);

    /**
     * Funció que s'executa quan s'activa la dièresi.
     * @param e
     */
    public void onPressDiaeresis(ModifierEvent e);

    /**
     * Funció que s'executa quan es desactiva la dièresi
     * @param e
     */
    public void onUnpressDiaeresis(ModifierEvent e);
}
```

6.1.13. ModifierType

```
package keysforfree;

/**
 * Classe enumerativa que defineix els tipus de ModifierEvent
 * @author eloicreus
 */
public enum ModifierType {
    CAPS_LOCK,SHIFT,ALT,OPEN_ACCENT,CLOSE_ACCENT,DIAERESIS
}
```

6.1.14. Predictor

```
package keysforfree;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Classe que implementa les funcions de predicció.
 * Gestiona l'estructura de l'arbre ternari de cerca.
 * @author eloicreus
 */
public class Predictor
{
    /**
     * Indica si el predictor prediu segons el català o la freqüència.
     */
    public boolean isLanguagePredictor;

    /**
     * Conté el prefix de paraula que ha escrit l'usuari.
     */
    String currentWord;

    /**
     * Conté l'arbre ternari de cerca que gestiona el predictor.
     */
    TernarySearchTree predictionWords;

    /**
     * Indica si s'està escrivint una paraula o no.
     */
    boolean writingWord;

    /**
     * Indica si es volen aprendre paraules noves quan s'utilitza el predictor.
     */
    protected boolean learningOn = false;

    /**
     * Indica que l'usuari ha seleccionat una paraula de la llista de predicció.
     */
    public boolean predictedEntered;

    /**
     * Conté els caràcters que faran de separadors de paraula.
     */
    static String defaultSeparators[] = {"\n"," ","\t",".",";",","};
   /**
     * Conté els separadors anteriors però en una llista més manipulable.
     */
    List<String> separators;
```

```
/**
 * Constructor bàsic de la classe
 */
public Predictor()
{
    currentWord = "";
    writingWord = false;
    predictedEntered = false;
    separators  = Arrays.asList(defaultSeparators);
}

/**
 * Constructor que ja ens crea l'arbre ternari de cerca donada una llista
 * de parelles de paraula i freqüència.
 * @param words Llista de parelles per entrar a l'arbre.
 */
public Predictor(ArrayList<WordInfo> words)
{
    this();
    predictionWords = new TernarySearchTree(words);
    isLanguagePredictor = false;
}
/**
 * Constructor com l'anterior que a més permet indicar si volem un
 * predictor per freqüències o segons el català.
 * @param words Llista de parelles per entrar a l'arbre.
 * @param isLanguagePredictor Indica el tipus de predicció.
 */
public Predictor(ArrayList<WordInfo> words, boolean isLanguagePredictor)
{
    this();
    predictionWords = new TernarySearchTree(words);
    this.isLanguagePredictor = isLanguagePredictor;
}

/**
 * Acció realitzada quan l'usuari entra un nou caràcter o una paraula
 * seleccionada de la finestra de predicció.
 * Posiciona l'arbre en el node corresponent.
 * @param character Caràcter o paraula que ha entrat l'usuari.
 */
public void newCharacter(String character)
{
    if(!writingWord)
    {
        writingWord = true;
    }
    if(character.length() < 2)
    {
        predictionWords.nextLevel(character.charAt(0));
    }
    else
```

```
            {
                predictionWords.positioning(character);
            }
            currentWord = currentWord+character;
        }

        /**
         * Funció que ens retorna la llista de paraules donat el seu prefix que ha
         * escrit l'usuari totes les paraules de l'arbre que comencen per aquest
         * prefix.
         * Aquesta llista s'ordena de major a menor freqüència utilitzant un
         * heapsort.
         * @return La llista de paraules possibles ordenades per freqüència
         * descendent
         */
        public ArrayList<String> getCandidateWords()
        {
            Heap heap;

            ArrayList<WordInfo> words = new ArrayList<WordInfo>();
            predictionWords.getSubtree(predictionWords.currentNode.middleChild, currentWord,
words);

            heap = new Heap(words);
            return heap.heapsort();
        }

        /**
         * Funció que ens retorna totes les paraules de l'arbre ordenades
         * alfabeticament
         * @return La llista ordenada.
         */
        public ArrayList<WordInfo> getAllWords()
        {
            ArrayList<WordInfo> words = new ArrayList<WordInfo>();
            predictionWords.getSubtree(predictionWords.root, "", words);
            return words;
        }

        /**
         * S'executa quan l'usuari acaba d'escriure una paraula. Això passa quan
         * s'escriu un separador de paraula.
         * Ens situem al cap de munt de l'arbre.
         * @param separator Separador entrat.
         */
        void addSeparator(String separator) {
            if(writingWord)
            {
                predictionWords.confirmWord(isLearningOn());
                currentWord = "";
                writingWord = false;
                if(predictedEntered)
                {
```

```
                predictedEntered = false;
            }
        }
    }
    /**
     * S'executa quan l'usuari esborra un caràcter. Retrocedim un nivell de
     * l'arbre.
     */
    void processRemove() {

        if(writingWord && predictionWords.previousLevel())
        {
            currentWord = currentWord.substring(0, currentWord.length()-1);
            if(currentWord.equals(""))
            {
                writingWord = false;
            }
        }
    }

    /**
     * @return the learningOn
     */
    public boolean isLearningOn() {
        return learningOn;
    }

    /**
     * @param learningOn the learningOn to set
     */
    public void setLearningOn(boolean learningOn) {
        this.learningOn = learningOn;
    }


}
```

6.1.15. PredictorWindow

```
package keysforfree;

import java.util.ArrayList;
import javax.swing.DefaultListModel;
import javax.swing.JComboBox;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;

/**
 * Classe que mostra les opcions de predicció a l'usuari. Rep els caràcters que va
 * escrivint l'usuari amb el teclat.
 * @author eloicreus
 */
public class PredictorWindow extends javax.swing.JFrame implements DocumentListener
{
    /**
     * Conté en cada moment el diccionari que s'està utilitzant
     */
    Dictionary currentDictionary;

    /**
     * Coné el teclat que ha d'escriure les paraules predites
     */
    AbstractKeyboard keyboard;


    /**
     * Serveix per modificar els elements que conté el combo box per seleccionar
     * el diccionari a utilitzar.
     */
    DefaultListModel listModel = new DefaultListModel();

    /**
     * Indica si fem servir predicció per freqüència o pel català.
     */
    public boolean catalanPrediction = false;

    /**
     * Constructor de la finestra. Registrem que volem escoltar els esdeveniments
     * de canvis de la caixa de text, per detectar els caracters que escriu
     * l'usuari.
     * @param keyboard Indica el teclat que ens escriurà les paraules que
     *                 seleccioni l'usuari per escriure.
     */
    public PredictorWindow(AbstractKeyboard keyboard) {
        initComponents();
```

```
        this.keyboard = keyboard;
        keyboard.output.getDocument().addDocumentListener(this);
        loadDictionaries();
    }

    /**
     * Funció per Mac que ens posa la pantalla en un estat en que no agafi el
     * focus quan seleccionem una paraula.
     */
    public native void setWindowInactivated();

    /**
     * S'executa quan sortim del programa.
     */
    public void closeAction()
    {
        if(dictionaryActive())
        {
            currentDictionary.saveDictionary();
            if(checkBoxMenuItemPredictCatalan.isEnabled())
            {
                ((CatalanPredictor)currentDictionary.predictor).disconnectDatabase();
            }
        }
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        dialogCreateDictionary = new javax.swing.JDialog();
        labelMessage = new javax.swing.JLabel();
        buttonOKCreateAction = new javax.swing.JButton();
        buttonCancelCreateAction = new javax.swing.JButton();
        textFieldName = new javax.swing.JTextField();
        dialogDeleteDictionary = new javax.swing.JDialog();
        labelMessageDelete = new javax.swing.JLabel();
        comboBoxDictionaryDelete = new javax.swing.JComboBox();
        buttonCancelDelete = new javax.swing.JButton();
        buttonOKDelete = new javax.swing.JButton();
        comboBoxDictionaries = new javax.swing.JComboBox();
        scrollPaneWordsList = new javax.swing.JScrollPane();
        wordsList = new javax.swing.JList(listModel);
        menuBarPrediction = new javax.swing.JMenuBar();
        menuLanguage = new javax.swing.JMenu();
        checkBoxMenuItemPredictCatalan = new javax.swing.JCheckBoxMenuItem();
        checkBoxMenuItemStrictGrammar = new javax.swing.JCheckBoxMenuItem();
        menuDictionary = new javax.swing.JMenu();
        menuItemCreateDictionary = new javax.swing.JMenuItem();
        menuItemDeleteDictionary = new javax.swing.JMenuItem();
        menuItemLoadDictionaries = new javax.swing.JMenuItem();
        separatorMenuDictionary = new javax.swing.JPopupMenu.Separator();
        checkBoxMenuItemLearningMode = new javax.swing.JCheckBoxMenuItem();
```

```
        dialogCreateDictionary.setTitle("Keys4Free - Create dictionary");
        dialogCreateDictionary.setAlwaysOnTop(true);
        dialogCreateDictionary.setResizable(false);

        labelMessage.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        labelMessage.setText("Enter the new dictionary name:");

        buttonOKCreateAction.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
        buttonOKCreateAction.setText("OK");
        buttonOKCreateAction.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonOKCreateActionActionPerformed(evt);
            }
        });

        buttonCancelCreateAction.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
        buttonCancelCreateAction.setText("Cancel");
        buttonCancelCreateAction.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonCancelCreateActionActionPerformed(evt);
            }
        });

        textFieldName.setFont(new java.awt.Font("Lucida Grande", 0, 14));

        org.jdesktop.layout.GroupLayout dialogCreateDictionaryLayout = new
org.jdesktop.layout.GroupLayout(dialogCreateDictionary.getContentPane());
        dialogCreateDictionary.getContentPane().setLayout(dialogCreateDictionaryLayout);
        dialogCreateDictionaryLayout.setHorizontalGroup(
            dialogCreateDictionaryLayout.createParallelGroup(org.jdesktop.layout.GroupLayo
ut.LEADING)
            .add(dialogCreateDictionaryLayout.createSequentialGroup()
                .add(dialogCreateDictionaryLayout.createParallelGroup(org.jdesktop.layout.G
roupLayout.LEADING)
                    .add(dialogCreateDictionaryLayout.createSequentialGroup()
                        .add(160, 160, 160)
                        .add(buttonCancelCreateAction)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                        .add(buttonOKCreateAction,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 55,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(dialogCreateDictionaryLayout.createSequentialGroup()
                        .addContainerGap()
                        .add(textFieldName,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 290,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(dialogCreateDictionaryLayout.createSequentialGroup()
                        .add(29, 29, 29)
                        .add(labelMessage,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 220,
```

```
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        dialogCreateDictionaryLayout.setVerticalGroup(
            dialogCreateDictionaryLayout.createParallelGroup(org.jdesktop.layout.GroupLayo
ut.LEADING)
            .add(dialogCreateDictionaryLayout.createSequentialGroup()
                .addContainerGap()
                .add(labelMessage, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
29, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(textFieldName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                .add(dialogCreateDictionaryLayout.createParallelGroup(org.jdesktop.layout.G
roupLayout.LEADING)
                    .add(buttonOKCreateAction)
                    .add(buttonCancelCreateAction))
                .addContainerGap())
        );

        dialogCreateDictionary.pack();

        dialogDeleteDictionary.setTitle("Keys4Free - Delete dictionary");
        dialogDeleteDictionary.setAlwaysOnTop(true);
        dialogDeleteDictionary.setResizable(false);
        dialogDeleteDictionary.setSize(new java.awt.Dimension(300, 160));

        labelMessageDelete.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        labelMessageDelete.setText("Select the dictionary to delete:");

        comboBoxDictionaryDelete.setFont(new java.awt.Font("Lucida Grande", 0, 14));

        buttonCancelDelete.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        buttonCancelDelete.setText("Cancel");
        buttonCancelDelete.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonCancelDeleteActionPerformed(evt);
            }
        });

        buttonOKDelete.setFont(new java.awt.Font("Lucida Grande", 0, 14));
        buttonOKDelete.setText("OK");
        buttonOKDelete.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                buttonOKDeleteActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout dialogDeleteDictionaryLayout = new
org.jdesktop.layout.GroupLayout(dialogDeleteDictionary.getContentPane());
```

```
        dialogDeleteDictionary.getContentPane().setLayout(dialogDeleteDictionaryLayout);
        dialogDeleteDictionaryLayout.setHorizontalGroup(
            dialogDeleteDictionaryLayout.createParallelGroup(org.jdesktop.layout.GroupLayo
ut.LEADING)
            .add(dialogDeleteDictionaryLayout.createSequentialGroup()
                .add(dialogDeleteDictionaryLayout.createParallelGroup(org.jdesktop.layout.Gr
oupLayout.LEADING)
                    .add(dialogDeleteDictionaryLayout.createSequentialGroup()
                        .add(20, 20, 20)
                        .add(labelMessageDelete))
                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
dialogDeleteDictionaryLayout.createSequentialGroup()
                        .addContainerGap()
                        .add(buttonCancelDelete)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                        .add(buttonOKDelete,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 53,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(dialogDeleteDictionaryLayout.createSequentialGroup()
                        .add(10, 10, 10)
                        .add(comboBoxDictionaryDelete, 0, 301, Short.MAX_VALUE)))
                .addContainerGap())
        );
        dialogDeleteDictionaryLayout.setVerticalGroup(
            dialogDeleteDictionaryLayout.createParallelGroup(org.jdesktop.layout.GroupLayo
ut.LEADING)
            .add(dialogDeleteDictionaryLayout.createSequentialGroup()
                .addContainerGap()
                .add(labelMessageDelete)
                .add(13, 13, 13)
                .add(comboBoxDictionaryDelete,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .add(13, 13, 13)
                .add(dialogDeleteDictionaryLayout.createParallelGroup(org.jdesktop.layout.Gr
oupLayout.BASELINE)
                    .add(buttonOKDelete,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 29,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(buttonCancelDelete,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );

        dialogDeleteDictionary.pack();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setAlwaysOnTop(true);
        setModalExclusionType(java.awt.Dialog.ModalExclusionType.TOOLKIT_EXCLUDE);
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter() {
```

```
            public void windowClosed(java.awt.event.WindowEvent evt) {
                formWindowClosed(evt);
            }
        });

        comboBoxDictionaries.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        comboBoxDictionaries.setFocusable(false);
        comboBoxDictionaries.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                comboBoxDictionariesActionPerformed(evt);
            }
        });

        wordsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        wordsList.setFocusable(false);
        wordsList.setSelectedIndex(0);
        wordsList.setSelectionBackground(new java.awt.Color(255, 255, 255));
        wordsList.setSelectionForeground(new java.awt.Color(0, 0, 0));
        wordsList.addListSelectionListener(new javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                wordsListValueChanged(evt);
            }
        });
        scrollPaneWordsList.setViewportView(wordsList);

        menuLanguage.setText("Language");

        checkBoxMenuItemPredictCatalan.setText("Predict Català");
        checkBoxMenuItemPredictCatalan.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                checkBoxMenuItemPredictCatalanActionPerformed(evt);
            }
        });
        menuLanguage.add(checkBoxMenuItemPredictCatalan);

        checkBoxMenuItemStrictGrammar.setText("Strict grammar");
        checkBoxMenuItemStrictGrammar.setEnabled(false);
        checkBoxMenuItemStrictGrammar.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                checkBoxMenuItemStrictGrammarActionPerformed(evt);
            }
        });
        menuLanguage.add(checkBoxMenuItemStrictGrammar);

        menuBarPrediction.add(menuLanguage);

        menuDictionary.setText("Dictionary");

        menuItemCreateDictionary.setText("Create");
        menuItemCreateDictionary.addActionListener(new java.awt.event.ActionListener() {
```

```
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemCreateDictionaryActionPerformed(evt);
            }
        });
        menuDictionary.add(menuItemCreateDictionary);

        menuItemDeleteDictionary.setText("Delete");
        menuItemDeleteDictionary.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemDeleteDictionaryActionPerformed(evt);
            }
        });
        menuDictionary.add(menuItemDeleteDictionary);

        menuItemLoadDictionaries.setText("Load dictionaries");
        menuItemLoadDictionaries.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemLoadDictionariesActionPerformed(evt);
            }
        });
        menuDictionary.add(menuItemLoadDictionaries);
        menuDictionary.add(separatorMenuDictionary);

        checkBoxMenuItemLearningMode.setText("Learning mode");
        checkBoxMenuItemLearningMode.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                checkBoxMenuItemLearningModeActionPerformed(evt);
            }
        });
        menuDictionary.add(checkBoxMenuItemLearningMode);

        menuBarPrediction.add(menuDictionary);

        setJMenuBar(menuBarPrediction);

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING,
false)
                    .add(org.jdesktop.layout.GroupLayout.LEADING, scrollPaneWordsList,
0, 0, Short.MAX_VALUE)
                    .add(org.jdesktop.layout.GroupLayout.LEADING, comboBoxDictionaries,
0, 236, Short.MAX_VALUE))
                .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
```

```
                layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()
                    .addContainerGap()
                    .add(comboBoxDictionaries,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                    .add(scrollPaneWordsList,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 277,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
            );

            pack();
        }// </editor-fold>

        /**
         * S'executa quan tanquem la finestra.
         * @param evt
         */
        private void formWindowClosed(java.awt.event.WindowEvent evt) {
            setVisible(false);
        }

        /**
         * S'executa quan seleccionem el diccionari a utilitzar.
         * Guardem les paraules del diccionari anterior i creem el nou diccionari.
         * També indiquem al diccionari el tipus de predictor que volem. Omplim la
         * llista d'opcions de predicció amb les paraules del nou diccionari.
         * @param evt
         */
        private void comboBoxDictionariesActionPerformed(java.awt.event.ActionEvent evt) {
            String dictionaryName = (String) comboBoxDictionaries.getSelectedItem();
            if(currentDictionary != null)
            {
                currentDictionary.saveDictionary();
            }
            currentDictionary = new Dictionary(dictionaryName);
            if(currentDictionary.setIdentifier() == -1)
            {
                JOptionPane.showMessageDialog(this, "The program cannot connect to the
database.\n"+
                        "Ensure that MySQL Server is running or review the Database
Options and try again.",
                        "Keys4Free", JOptionPane.ERROR_MESSAGE);
                return;
            }
            if(!currentDictionary.setPredictor(catalanPrediction))
            {
                JOptionPane.showMessageDialog(this, "The program cannot load the
dictionary words.\n"+
```

```
                    "Ensure that MySQL Server is running or review the Database
Options and try again.",
                    "Keys4Free", JOptionPane.ERROR_MESSAGE);
                return;
            }
            if(checkBoxMenuItemStrictGrammar.isSelected())
            {
                ((CatalanPredictor)currentDictionary.predictor).setStrictLanguagePrediction(true);
            }

            if(checkBoxMenuItemLearningMode.isSelected())
            {
                currentDictionary.predictor.setLearningOn(true);
            }

            sendWords(currentDictionary.predictor.getCandidateWords());
        }

        /**
         * S'executa quan l'usuari selecciona una paraula per escriure de les opcions
         * que se li mostren.
         * Escrivim la part de paraula que falta al prefix que tingués entrat l'usuari
         * a la caixa de text i a l'aplicació activa del sistema.
         * @param evt
         */
        private void wordsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
            String word = (String) ((JList)evt.getSource()).getSelectedValue();

            if(word!=null)
            {
                String partialWord = (currentDictionary.predictor.currentWord.equals("") ?
                    word :
                    word.substring(word.indexOf(currentDictionary.predictor.currentWord)+
                    currentDictionary.predictor.currentWord.length(),word.length()));

                currentDictionary.predictor.predictedEntered = true;

                keyboard.setKeyAtComponent(partialWord);
                try {
                    keyboard.output.getDocument().insertString(keyboard.output.getText().length(
) ,partialWord, null);
                } catch (BadLocationException ex) {

                }
            }
        }

        /**
         * S'executa quan seleccionem l'opció de menú d'eliminar diccionaris.
         * @param evt
         */
        private void menuItemDeleteDictionaryActionPerformed(java.awt.event.ActionEvent evt) {
            if(Dictionary.dictionariesName.size() > 0)
```

```
            {
                comboBoxDictionaryDelete.setSelectedIndex(0);
                dialogDeleteDictionary.setLocationRelativeTo(this);
                dialogDeleteDictionary.setVisible(true);
            }
            else
            {
                JOptionPane.showMessageDialog(this,
                                              "The are no dictionaries to delete.",
                                              "Keys4Free",
                                              JOptionPane.WARNING_MESSAGE);
            }
        }

        /**
         * S'executa quan activem o desactivem el mode de predicció en català.
         * Creem o alliberem la connexió amb la base de dades que utilitza el
         * predictor.
         * @param evt
         */
        private void
checkBoxMenuItemPredictCatalanActionPerformed(java.awt.event.ActionEvent evt) {
            if(dictionaryActive())
            {
                if(catalanPrediction = checkBoxMenuItemPredictCatalan.isSelected())
                {
                    currentDictionary.predictor = new
CatalanPredictor(currentDictionary.predictor.getAllWords());
                    ((CatalanPredictor)currentDictionary.predictor).connectDatabase();
                    checkBoxMenuItemStrictGrammar.setEnabled(true);
                }
                else
                {
                    ((CatalanPredictor)currentDictionary.predictor).disconnectDatabase();
                    currentDictionary.predictor = new
Predictor(currentDictionary.predictor.getAllWords());
                    checkBoxMenuItemStrictGrammar.setEnabled(false);
                    checkBoxMenuItemStrictGrammar.setSelected(false);
                }
            }
            else
            {
                checkBoxMenuItemPredictCatalan.setSelected(false);
            }
        }

        /**
         * S'executa quan polsem l'opció de menú d'activar el mode de predicció
         * estricte del català.
         * @param evt
         */
        private void
checkBoxMenuItemStrictGrammarActionPerformed(java.awt.event.ActionEvent evt) {
```

```
            if(checkBoxMenuItemPredictCatalan.isSelected() && dictionaryActive())
            {
                ((CatalanPredictor)currentDictionary.predictor).setStrictLanguagePrediction(check
BoxMenuItemStrictGrammar.isSelected());
            }
        }

        /**
         * S'executa quan seleccionem l'opció de menú de crear diccionari.
         * @param evt
         */
        private void menuItemCreateDictionaryActionPerformed(java.awt.event.ActionEvent evt)
{
            dialogCreateDictionary.setLocationRelativeTo(this);
            dialogCreateDictionary.setVisible(true);
        }

        /**
         * S'executa quan activem o desactivem l'aprenentatge de paraules del
         * predictor.
         * @param evt
         */
        private void
checkBoxMenuItemLearningModeActionPerformed(java.awt.event.ActionEvent evt) {
            if(dictionaryActive())
            {
                currentDictionary.predictor.setLearningOn(checkBoxMenuItemLearningMode.isSel
ected());
            }
            else
            {
                checkBoxMenuItemLearningMode.setSelected(false);
            }
        }

       /**
         * S'executa quan tanquem la pantalla d'eliminar diccionaris.
         * @param evt
         */
        private void buttonCancelDeleteActionPerformed(java.awt.event.ActionEvent evt) {
            dialogDeleteDictionary.setVisible(false);
        }

        /**
         * S'executa quan confirmen el diccionari que volem esborrar.
         * @param evt
         */
        private void buttonOKDeleteActionPerformed(java.awt.event.ActionEvent evt) {
            String dictionaryName = (String) comboBoxDictionaryDelete.getSelectedItem();

            Dictionary deletedDictionary = new Dictionary(dictionaryName);
            if(deletedDictionary.setIdentifier() == -1)
            {
```

```
                JOptionPane.showMessageDialog(this, "The program cannot remove the
dictionary.\n"+
                        "Ensure that MySQL Server is running or review the Database
Options and try again.",
                        "Keys4Free", JOptionPane.ERROR_MESSAGE);
            }
            if(!deletedDictionary.removeDictionary())
            {
                JOptionPane.showMessageDialog(this, "The program cannot remove the
dictionary.\n"+
                        "Ensure that MySQL Server is running or review the Database
Options and try again.",
                        "Keys4Free", JOptionPane.ERROR_MESSAGE);
                return;
            }
            if(comboBoxDictionaries.getSelectedItem() != null &&
                    ((String)comboBoxDictionaries.getSelectedItem()).equals(dictionaryName))
            {
                listModel.clear();
            }
            comboBoxDictionaries.removeItem(dictionaryName);
            comboBoxDictionaryDelete.removeItem(dictionaryName);
            JOptionPane.showMessageDialog(this,
                                          "The dictionary "+dictionaryName+" has been
correctly removed.",
                                          "Keys4Free",
                                          JOptionPane.INFORMATION_MESSAGE);
        }

        /**
         * S'executa quan tanquem la pantalla de creació de diccionari.
         * @param evt
         */
        private void buttonCancelCreateActionActionPerformed(java.awt.event.ActionEvent evt) {
            dialogCreateDictionary.setVisible(false);
            textFieldName.setText("");
        }

        /**
         * S'executa quan confirmem la creació d'un nou diccionari.
         * @param evt
         */
        private void buttonOKCreateActionActionPerformed(java.awt.event.ActionEvent evt) {
            String dictionaryName = textFieldName.getText();
            if (dictionaryName.length() > 0)
            {
                if(!Dictionary.dictionariesName.contains(dictionaryName))
                {
                    if(!Dictionary.createDictionary(dictionaryName))
                    {
                        JOptionPane.showMessageDialog(this, "The program cannot create
the dictionary.\n"+
                        "Ensure that MySQL Server is running or review the Database
```

```
Options and try again.",
                        "Keys4Free", JOptionPane.ERROR_MESSAGE);
                        return;
                    }
                    comboBoxDictionaries.addItem(dictionaryName);
                    comboBoxDictionaryDelete.addItem(dictionaryName);
                    JOptionPane.showMessageDialog(this,
                                                  "The dictionary "+dictionaryName+" has been
correctly created.",
                                                  "Keys4Free",
                                                  JOptionPane.INFORMATION_MESSAGE);
                    textFieldName.setText("");
                    dialogCreateDictionary.setVisible(false);
                }
                else
                {
                    JOptionPane.showMessageDialog(this,
                                                  "The dictionary "+dictionaryName+" is already
created.",
                                                  "Keys4Free",
                                                  JOptionPane.INFORMATION_MESSAGE);
                }
            }
            else
            {
                JOptionPane.showMessageDialog(this,
                                              "You must enter a dictionary name.",
                                              "Keys4Free",
                                              JOptionPane.WARNING_MESSAGE);
            }
        }

        /**
         * S'executa quan seleccionem l'opció de menú de carregar diccionaris.
         * @param evt
         */
        private void menuItemLoadDictionariesActionPerformed(java.awt.event.ActionEvent evt)
{
            loadDictionaries();
        }

        /**
         * Carrega la llista de la predicció amb les paraules que se li passen.
         * @param words Llista de paraules que s'ha de posar a la llista de la
         *                  predicció.
         */
        public void sendWords(ArrayList<String> words)
        {
            listModel.clear();

            int i = 0;
            for(String word : words)
            {
```

```
                listModel.add(i++, word);
            }
        }

        /**
         * Serveix per carregar els elements a dintre d'un combo box
         * @param combo El combo box que volem carregar.
         * @param dictionaries Noms de diccionaris a carregar.
         */
        public void chargeComboBoxDictionaries(JComboBox combo, ArrayList<String>
dictionaries)
        {
            int i = 0;
            for(String dictionary : dictionaries)
            {
                combo.insertItemAt(dictionary, i++);
            }
        }

        /**
         * Carrega els diccionaris de la base de dades.
         */
        public void loadDictionaries()
        {
            if(!Dictionary.loadDictionaries())
            {
                JOptionPane.showMessageDialog(this, "The program cannot load the
dictionaries.\n"+
                        "Ensure that MySQL Server is running or review the Database
Options and try again.",
                        "Keys4Free", JOptionPane.ERROR_MESSAGE);
                return;
            }
            menuItemLoadDictionaries.setEnabled(false);
            chargeComboBoxDictionaries(comboBoxDictionaries,Dictionary.dictionariesName);
            chargeComboBoxDictionaries(comboBoxDictionaryDelete,Dictionary.dictionariesName
);
        }

        // Variables declaration - do not modify
        private javax.swing.JButton buttonCancelCreateAction;
        private javax.swing.JButton buttonCancelDelete;
        private javax.swing.JButton buttonOKCreateAction;
        private javax.swing.JButton buttonOKDelete;
        private javax.swing.JCheckBoxMenuItem checkBoxMenuItemLearningMode;
        private javax.swing.JCheckBoxMenuItem checkBoxMenuItemPredictCatalan;
        private javax.swing.JCheckBoxMenuItem checkBoxMenuItemStrictGrammar;
        private javax.swing.JComboBox comboBoxDictionaries;
        private javax.swing.JComboBox comboBoxDictionaryDelete;
        private javax.swing.JDialog dialogCreateDictionary;
        private javax.swing.JDialog dialogDeleteDictionary;
        private javax.swing.JLabel labelMessage;
        private javax.swing.JLabel labelMessageDelete;
```

```
private javax.swing.JMenuBar menuBarPrediction;
private javax.swing.JMenu menuDictionary;
private javax.swing.JMenuItem menuItemCreateDictionary;
private javax.swing.JMenuItem menuItemDeleteDictionary;
private javax.swing.JMenuItem menuItemLoadDictionaries;
private javax.swing.JMenu menuLanguage;
private javax.swing.JScrollPane scrollPaneWordsList;
private javax.swing.JPopupMenu.Separator separatorMenuDictionary;
private javax.swing.JTextField textFieldName;
private javax.swing.JList wordsList;
// End of variables declaration

/**
 * S'executa quan detectem l'entrada d'un caràcter a la caixa de text de la
 * pantalla principal.
 * Depenent de si és un separador de paraula o no, executem la funció
 * corresponent del predictor. En ambdós casos actualitzem la llista de
 * paraules de la predicció.
 * @param e
 */
@Override
public void insertUpdate(DocumentEvent e)
{
    if(dictionaryActive())
    {
        try
        {
            int changeLength = e.getLength();
            Document doc = (Document)e.getDocument();
            String inserted = doc.getText(e.getOffset(), e.getLength());

            inserted = inserted.toLowerCase();

            System.out.println(inserted);

            if(inserted.length() < 2 &&
                    currentDictionary.predictor.separators.contains(inserted))
            {
                currentDictionary.predictor.addSeparator(inserted);
                sendWords(currentDictionary.predictor.getCandidateWords());
            }
            else
            {
                currentDictionary.predictor.newCharacter(inserted);
                sendWords(currentDictionary.predictor.getCandidateWords());
            }

        }
        catch (BadLocationException ex)
        {

        }
    }
```

```
    }

    /**
     * S'executa quan eliminem un caràcter de la caixa de text de la pantalla
     * principal.
     * @param e
     */
    @Override
    public void removeUpdate(DocumentEvent e)
    {
        if(dictionaryActive())
        {
            currentDictionary.predictor.processRemove();
            sendWords(currentDictionary.predictor.getCandidateWords());
        }
    }

    @Override
    public void changedUpdate(DocumentEvent e)
    {

    }

    /**
     * Indica si hi ha cap diccionari utilitzant-sa
     * @return Si s'utilitza cap diccionari.
     */
    public boolean dictionaryActive()
    {
        return currentDictionary != null;
    }

}
```

6.1.16. TernarySearchTree

```
package keysforfree;

import java.util.ArrayList;
import java.util.LinkedList;

/**
 * Implementació de l'estructura de dades d'arbre ternari de cerca per guardar
 * les paraules d'un diccionari durant la predicció.
 * @author eloicreus
 */
public class TernarySearchTree
{
    /**
     * Conté el primer node de l'arbre
     */
    Node root;
    /**
     * Indica el node de l'arbre on estem actualment
     */
    Node currentNode;
    /**
     * Conté un apuntador al primer node de l'arbre per facilitar la implementació
     */
    Node cent;

    /**
     * Indica el nombre de paraules de l'arbre.
     */
    int numberOfWords;

    /**
     * Conté una llista de nodes representant la part de paraula que s'ha
     * escrit fins el moment. Serveix per pujar nivells de l'arbre.
     */
    LinkedList<Node> previousLevels;

    /**
     * Serveix per representar cada un dels nodes de l'arbre.
     */
    public class Node
    {
        /**
         * Indica la freqüència de la paraula que s'acaba en el node, si es que
         * se'n i acaba cap.
         */
        int frequency = 0;
        /**
         * Caràcter que té associat el node.
         */
        char character;
        /*
```

```
     * Apuntadors als fills del node.
     */
    Node leftChild, middleChild, rightChild;
    /**
     * Indica si en el node s'hi acaba una paraula o no.
     */
    boolean isWord = false;

    /**
     * Constructor per crear nous nodes.
     * @param character Caràcter del node.
     * @param isWord Indica si s'hi acaba una paraula o no.
     */
    public Node(char character, boolean isWord) {
        this.character = character;
        this.isWord = isWord;
        leftChild = middleChild = rightChild = null;
    }

    /**
     * Serveix per construir nous nodes però només indicant el caràcter.
     * @param character Caràcter del node.
     */
    public Node(char character) {
        this.character = character;
        leftChild = middleChild = rightChild = null;
    }

    /**
     * Indica si el node té fill esquerra.
     * @return Si existeix fill esquerra.
     */
    public boolean existLeft()
    {
        return leftChild != null;
    }

    /**
     * Indica si el node té fill del mig.
     * @return Si existeix fill del mig.
     */
    public boolean existMiddle()
    {
        return middleChild != null;
    }

    /**
     * Indica si el node té fill dret.
     * @return Si existeix fill dret.
     */
    public boolean existRight()
    {
        return rightChild != null;
```

```
        }

        /**
         * Indica si el node té algun fill.
         * @return Si existeix algun fill.
         */
        public boolean hasChilds()
        {
            return leftChild != null || middleChild != null || rightChild != null;
        }

    }

    /**
     * Constructor bàsic de l'arbre ternari de cerca. Inicialitza els atributs
     * necessaris.
     */
    private TernarySearchTree()
    {
        root = null;
        currentNode = null;
        numberOfWords = 0;
        previousLevels = new LinkedList<Node>();
    }

    /**
     * Constructor de l'arbre inserint en ell les parelles de paraula i
     * freqüència que li passen.
     * Inicialitzem l'atribut cent perquè estiguem situats a dalt de l'arbre
     * per començar a baixar quan l'usuari comenci a escriure.
     * @param wordList Llista de paraules i freqüències a inserir en l'arbre.
     */
    public TernarySearchTree(ArrayList<WordInfo> wordList)
    {
        this();
        balancedInsertion(wordList,0,wordList.size()-1);
        cent = new Node('0');
        cent.middleChild = root;
        currentNode = cent;
    }

    /**
     * S'utilitza en la funció anterior per inserir les diferents parelles de
     * paraula i freqüència.
     * Inserint d'aquesta manera, l'arbre queda equilibrat, permetent tenir la
     * màxima eficiència en les operacions d'aquest.
     * @param wordList Llista de parelles a introduir.
     * @param first Indica la posició de la primera paraula de la llista.
     * @param last  Indica la posició de l'última paraula de la llista.
     */
    private void balancedInsertion(ArrayList<WordInfo> wordList, int first, int last)
    {
        if(last >= first)
```

```
        {
            int middle = ((last - first)/2)+first;

            WordInfo wordInfo = wordList.get(middle);
            insert(wordInfo.word,wordInfo.frequency);
            System.out.println(wordInfo.word);

            balancedInsertion(wordList,first,middle-1);
            balancedInsertion(wordList,middle+1,last);
        }
    }

    /**
     * Funció que insereix una parella de paraula i freqüència dintre de l'arbre.
     * Es tracta d'anar inserint un node per cada caràcter de la paraula, a la
     * posició que li pertoca de l'arbre, per anar-lo creant.
     * @param word Paraula que es vol inserir.
     * @param frequency Freqüència de la paraula a inserir.
     */
    public void insert(String word,int frequency)
    {
        numberOfWords++;
        int level = 0;
        int wordLength = word.length();
        if(root == null)
        {
            root = new Node(word.charAt(level),level+1 == wordLength);
            if(level == wordLength-1)
            {
                root.frequency = frequency;
                root.isWord = true;
                return;
            }
        }

        char character;

        Node travesalNode;
        travesalNode = root;

        boolean continueDown = true;
        character = word.charAt(level);

        while(continueDown)
        {
            if(character < travesalNode.character)
            {
                if(!travesalNode.existLeft())
                {
                    travesalNode.leftChild = new Node(character,level+1 == wordLength);
                    continueDown = false;
                }
                travesalNode = travesalNode.leftChild;
```

```
            }
            else if(character == travesalNode.character)
            {
                if(level == wordLength-1)
                {
                    travesalNode.frequency = frequency;
                    travesalNode.isWord = true;
                    return;
                }
                if(!travesalNode.existMiddle())
                {
                    travesalNode.middleChild = new Node(word.charAt(level+1),level+1 ==
                      wordLength-1);
                    continueDown = false;
                }
                travesalNode = travesalNode.middleChild;
                level++;
                character = word.charAt(level);
            }
            else
            {
                if(!travesalNode.existRight())
                {
                    travesalNode.rightChild = new Node(character,level+1 == wordLength);
                    continueDown = false;
                }
                travesalNode = travesalNode.rightChild;
            }

        }


        while(level < wordLength-1)
        {
            travesalNode.middleChild = new Node(word.charAt(level+1),level+1 ==
              wordLength-1);
            level++;
            travesalNode = travesalNode.middleChild;
        }
        travesalNode.frequency = frequency;

    }

    /**
     * Serveix per baixar per l'arbre tants nivells com caràcters tingui el
     * paràmetre que ens passin.
     * @param character Cadena da caràcters a baixar per l'arbre.
     */
    public void positioning(String character)
    {
        int length = character.length();
        for(int i = 0; i < length; i++)
        {
```

```
            nextLevel(character.charAt(i));
        }
    }

    /**
     * Serveix per baixar un sol nivell de l'arbre. Es busca la posició del
     * caràcter donat al següent nivell de l'arbre, i si no existeix, es crea
     * un nou node amb el caràcter donat.
     * Actualitzem la posició de currentNode per poder predir les paraules
     * correctament.
     * @param character Indica el caràcter escrit per l'usuari.
     */
    public void nextLevel(char character)
    {
        if(root == null)
        {
            root = new Node(character);
            cent.middleChild = root;
            currentNode = root;
            return;
        }

        if(currentNode.existMiddle())
        {
            Node travesalNode;
            previousLevels.push(currentNode);
            travesalNode = currentNode.middleChild;
            boolean continueDown = true;

            while(continueDown)
            {
                if(character < travesalNode.character)
                {
                    if(!travesalNode.existLeft())
                    {
                        travesalNode.leftChild = new Node(character);
                        continueDown = false;
                    }
                    travesalNode = travesalNode.leftChild;
                }
                else if(character == travesalNode.character)
                {
                    continueDown = false;
                }
                else
                {
                    if(!travesalNode.existRight())
                    {
                        travesalNode.rightChild = new Node(character);
                        continueDown = false;
                    }
                    travesalNode = travesalNode.rightChild;
                }
```

```
            }
            currentNode = travesalNode;
        }
        else
        {
            currentNode.middleChild = new Node(character);
            previousLevels.push(currentNode);
            currentNode = currentNode.middleChild;
        }
    }

    /**
     * Serveix per anar al nivell anterior de l'arbre quan l'usuari esborra
     * l'últim caràcter de la paraula que esta escrivint.
     * @return Cert si hem anat al nivell anterior, fals si estàvem al primer
     *             nivell.
     */
    public boolean previousLevel()
    {
        if(!previousLevels.isEmpty())
        {
            Node previousNode = previousLevels.pop();

            if(!currentNode.hasChilds() && !currentNode.isWord)
            {
                char character = currentNode.character;

                Node travesalNode;
                travesalNode = previousNode.middleChild;
                boolean continueDown = true;

                while(continueDown)
                {
                    if(character < travesalNode.character)
                    {
                        if(travesalNode.leftChild.character == character)
                        {
                            travesalNode.leftChild = null;
                            break;
                        }else
                        {
                            travesalNode = travesalNode.leftChild;
                        }
                    }
                    else if(character == travesalNode.character)
                    {
                        previousNode.middleChild = null;
                        break;
                    }
                    else
                    {
                        if(travesalNode.rightChild.character == character)
```

```
                    {
                        travesalNode.rightChild = null;
                        break;
                    }
                    else
                    {
                        travesalNode = travesalNode.rightChild;
                    }
                }

            }
        }
        currentNode = previousNode;

        return true;
    }
    return false;
}

/**
 * Serveix per eliminar l'última paraula entrada a l'arbre en cas que no es
 * vulguin aprendre paraules durant la predicció.
 */
public void deleteNodes()
{
    while(!currentNode.hasChilds() && !currentNode.isWord && previousLevel())
    {
        ;
    }
    if(!root.hasChilds())
    {
        root = null;
    }
}

/**
 * S'executa quan l'usuari acaba d'escriure una paraula.
 * Si l'últim node ja era el final d'una paraula, actualitzem la
 * freqüència d'aparició d'aquesta paraula. En cas contrari, si volem
 * aprendre la paraula indiquem que en l'últim node s'hi acaba una paraula,
 * si no en volem aprendre, eliminem la paraula que acabem d'entrar.
 * @param learning Indica si volem aprendre la paraula escrita o no.
 */
public void confirmWord(boolean learning)
{
    if(currentNode.isWord)
    {
        currentNode.frequency++;
    }
    else
    {
        if(learning)
        {
```

```
                currentNode.isWord = true;
            }
            else
            {
                deleteNodes();
            }
        }
        currentNode = cent;
    }

    /**
     * Funció que obté totes les paraules i freqüències que hi ha a l'arbre sota
     * d'un determinat node. S'utilitza per obtenir la llista de paraules que es
     * mostraran a l'usuari en la predicció.
     * @param subtreeRoot Node on s'inicia la cerca de paraules.
     * @param word Prefix comú de les paraules que es trobin.
     * @param map Serveix per guardar les paraules i freqüències que es vagin
     *          trobant.
     */
    public void getSubtree(Node subtreeRoot, String word, ArrayList<WordInfo> map)
    {
        if( subtreeRoot != null)
        {
            Node p = subtreeRoot;

            if(p.existLeft()) getSubtree(p.leftChild,word,map);

            if(p.isWord)
            {
                map.add(new WordInfo(word+p.character, p.frequency));
            }

            if(p.existMiddle()) getSubtree(p.middleChild,word+p.character,map);

            if(p.existRight()) getSubtree(p.rightChild,word,map);
        }
    }
}
```

6.1.17. TextAdjustEvent

```
package keysforfree;

import java.util.EventObject;

/**
 * Classe que defineix els esdeveniments d'ajustament del text.
 * @author eloicreus
 */
public class TextAdjustEvent extends EventObject
{
    /**
     * Constructor d'aquest tipus d'esdeveniments.
     * Utilitza el constructor de la superclasse.
     * @param source
     */
    public TextAdjustEvent(Object source)
    {
        super(source);
    }
}
```

6.1.18. TextAdjustEventListener

```
package keysforfree;

import java.util.EventListener;

/**
 * Interfície que defineix el mètode que han de implementar les classes que
 * vulguin rebre esdeveniments de tipus TextAdjustEvent.
 * @author eloicreus
 */
public interface TextAdjustEventListener extends EventListener
{
    /**
     * Funció a implementar que canviara la mida del text de les tecles.
     */
    public void adjustText();
}
```

6.1.19. WordInfo

```
package keysforfree;

/**
 * Classe per guardar parelles de paraula i freqüència
 * @author eloicreus
 */
public class WordInfo
{
    /*
     * Guardem la paraula en un String i la seva freqüència
     */
    String word;
    int frequency;

    /**
     * Constructor que inicialitza els valors de la parella
     * @param word Paraula que volem guardar
     * @param frequency Freqüència de la paraula que volem guardar
     */
    public WordInfo(String word, int frequency) {
        this.word = word;
        this.frequency = frequency;
    }

}
```

6.2. Codi natiu per cada sistema operatiu

6.2.1. Codi natiu per Windows

NativeIssue.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class NativeIssue */

#include <windows.h>

#ifndef _Included_keysforfree_NativeIssue
#define _Included_keysforfree_NativeIssue
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     AbstractKeyboard
 * Method:    setKeyAtComponent
 * Signature: (Ljava/lang/String)V
 */
JNIEXPORT void JNICALL
Java_keysforfree_AbstractKeyboard_setKeyAtComponent
(JNIEnv *, jobject, jstring);

/*
 * Class:     AbstractKeyboard
 * Method:    setTextModifier
 * Signature: (Ljava/lang/String)V
 */
JNIEXPORT void
JNICALL Java_keysforfree_AbstractKeyboard_setTextModifier
(JNIEnv *, jobject, jstring);

HWND getFocusedWindow();

#ifdef __cplusplus
}
#endif
#endif
```

NativeIssue.c

```
#include  "NativeIssue.h"

/*
 S'executa per escriure al component actiu de l'aplicació activa del sistema
 operatiu.
 Obtenim aquest component actiu i per cada caràcter que tinguem d'escriure
 simulem la pulsació de la tecla corresponent.
 */
```

```
JNIEXPORT void JNICALL Java_keysforfree_AbstractKeyboard_setKeyAtComponent
  (JNIEnv * jnienv, jobject object, jstring keyTyped)
{
    HWND focusedWindow = getFocusedWindow();

    const TCHAR *character;
    character = (* jnienv)->GetStringUTFChars(jnienv,keyTyped, NULL);
    if(character == NULL)
    {
        printf("Memory error");
        return;
    }

    int i;
    int length = strlen(character);
    for(i=0;i<length;i++)
    {
        UINT VKCode = LOBYTE(VkKeyScan(character[i]));
        SendMessage(focusedWindow, WM_KEYDOWN, VKCode, 0);
        SendMessage(focusedWindow, WM_CHAR, character[i], 0);
        SendMessage(focusedWindow, WM_KEYUP, VKCode, 0);
    }

    (* jnienv)->ReleaseStringUTFChars(jnienv,keyTyped,character);
}

/*
 Serveix per obtenir la referència del component que té el cursor
 d'escriptura.
 Obtenim l'aplicació activa i enganxem l'entrada del nostre procés
 amb l'entrada del procés de l'aplicació activa per obtenir la
 referència buscada.
 */
HWND getFocusedWindow()
{
    HWND foreHwnd;
    foreHwnd = GetForegroundWindow();

    DWORD foreWindowThread = GetWindowThreadProcessId(foreHwnd,NULL);

    AttachThreadInput(GetCurrentThreadId(),foreWindowThread,TRUE);

    HWND focusedWindow = GetFocus();

    AttachThreadInput(GetCurrentThreadId(),foreWindowThread,FALSE);

    return focusedWindow;
}

/*
 S'executa quan volem escriure un modificador de text a l'aplicació activa.
 Novament obtenim la referència del component amb el cursor d'escriptura, i
 simulem la pulsació de la tecla corresponent amb el codi virtual del
```

```
 modificador de text.
 */
JNIEXPORT JNIEXPORT void JNICALL Java_keysforfree_AbstractKeyboard_setTextModifier
  (JNIEnv * jnienv, jobject object, jstring modifierName)
{
    HWND focusedWindow = getFocusedWindow();

    const TCHAR *modifier;
    modifier = (* jnienv)->GetStringUTFChars(jnienv,modifierName, NULL);
    if(modifier == NULL)
    {
        printf("Memory error");
        return;
    }

    UINT VKCode;
    if (strcmp(modifier,"Tab") == 0)
        VKCode = VK_TAB;
    else if (strcmp(modifier,"Backspace") == 0)
        VKCode = VK_BACK;
    else if (strcmp(modifier,"Enter") == 0)
        VKCode = VK_RETURN;
    else
        VKCode = VK_SPACE;

    SendMessage(focusedWindow, WM_KEYDOWN, VKCode, 0);
    SendMessage(focusedWindow, WM_CHAR, VKCode, 0);
    SendMessage(focusedWindow, WM_KEYUP, VKCode, 0);

    (* jnienv)->ReleaseStringUTFChars(jnienv,modifierName,modifier);
}
```

6.2.2. Codi natiu per Linux

NativeIssue.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class NativeIssue */

#include <X11/Xlib.h>
#include <X11/keysym.h>

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#ifndef _Included_NativeIssue
#define _Included_NativeIssue
#ifdef __cplusplus
extern "C" {
#endif

/*
 * Class:     AbstractKeyboard
 * Method:    setKeyAtComponent
 * Signature: (Ljava/lang/String)V
 */
JNIEXPORT void JNICALL Java_keysforfree_NativeIssue_setKeyAtComponent
  (JNIEnv *, jobject, jstring);

/*
 * Class:     NativeIssue
 * Method:    setTextModifier
 * Signature: (Ljava/lang/String)V
 */
JNIEXPORT void JNICALL Java_keysforfree_NativeIssue_setTextModifier
  (JNIEnv *, jobject, jstring);

XKeyEvent createKeyEvent(Display *display,
                         Window window,
                         Window winRoot,
                         Bool press,
                         int keyCode,
                         int modifiers);

#ifdef __cplusplus
}
#endif
#endif
```

NativeIssue.c

```
#include "NativeIssue.h"

/**
 * Funció que serveix per crear els esdeveniments necessaris per a
 * la simulació de la pulsació d'una tecla.
 * @param display Indica el monitor on es produeix l'esdeveniment.
 * @param window Indica el component que té el cursor d'escriptura.
 * @param winRoot Indica la pantalla de l'aplicació activa que conté
 *                el component anterior.
 * @param press Indica si es un esdeveniment de prémer o deixar.
 * @param keyCode Indica el codi del sistema de la tecla que simulem.
 * @param modifiers Indica els modificadors existents quan simulem la
 *                  pulsació.
 * @return L'esdeveniment creat segons els paràmetres passats.
 */
XKeyEvent createKeyEvent(Display *display,
                         Window window,
                         Window winRoot,
                         Bool press,
                         int keyCode,
                         int modifiers)
{
    XKeyEvent event;

    event.display = display;
    event.window = window;
    event.root = winRoot;
    event.subwindow = None;
    event.time = CurrentTime;
    event.x = 1;
    event.y = 1;
    event.x_root = 1;
    event.y_root = 1;
    event.same_screen = True;
    event.keycode = XKeysymToKeycode(display,keyCode);
    event.state = modifiers;

    (press) ? (event.type = KeyPress) : (event.type = KeyRelease);

    return event;
}
/*
 * Class:     AbstractKeyboard
 * Method:    setKeyAtComponent
 * Signature: (Ljava/lang/String)V
 * S'utilitza per escriure el caràcter o paraula que li passem a la
 * pantalla que té el cursor d'escriptura. Ens hem de connectar al
 * servidor, obtenir la referència del component que té el cursor
 * i simular la pulsació de cada un dels caràcters. Per fer-ho hem
 * d'enviar al servidor l'esdeveniment corresponent de prémer i de
 * deixar.
```

```
 */
JNIEXPORT void JNICALL Java_keysforfree_AbstractKeyboard_setKeyAtComponent
  (JNIEnv * jnienv, jobject object, jstring keyTyped)
{
    Display *display = XOpenDisplay(0);

    if(display == NULL)
    {
        return;
    }

    Window winRoot = XDefaultRootWindow(display);

    Window winFocus;
    int state;
    XGetInputFocus(display, &winFocus, &state);

    const char * character = (*jnienv)->GetStringUTFChars(jnienv,keyTyped,NULL);
    if(character == NULL)
    {
        return;
    }

    XKeyEvent event;
    int i;
    for(i=0;i<strlen(character);i++)
    {
        event = createKeyEvent(display, winFocus, winRoot, True, character[i], 0 );
        XSendEvent(event.display, event.window, True, KeyPressMask, (XEvent *)&event);

        event = createKeyEvent(display, winFocus, winRoot, False, character[i], 0 );
        XSendEvent(event.display, event.window, True, KeyReleaseMask, (XEvent
*)&event);
    }

    (*jnienv)->ReleaseStringUTFChars(jnienv,keyTyped,character);

    XCloseDisplay(display);
}

/*
 * Class:     AbstractKeyboard
 * Method:    setTextModifier
 * Signature: (Ljava/lang/String)V
 * S'executa quan volem simular la pulsació d'un modificador de text.
 * A partir del nom del modificador de text que ens passen obtenim el
 * codi del sistema per aquest modificador. Llavors simulem la seva
 * pulsació.
 */
JNIEXPORT void JNICALL Java_keysforfree_AbstractKeyboard_setTextModifier
  (JNIEnv * jnienv, jobject object, jstring modifierName)
{
    Display *display = XOpenDisplay(0);
```

```
    if(display == NULL)
    {
        return;
    }

    Window winRoot = XDefaultRootWindow(display);

    Window winFocus;
    int state;
    XGetInputFocus(display, &winFocus, &state);

    const char * modifier = (*jnienv)->GetStringUTFChars(jnienv,modifierName,NULL);
    if(modifier == NULL)
    {
        return;
    }

    XKeyEvent event;
    KeySym keySym;

    if(strcmp(modifier,"Tab")==0)
        keySym = XK_Tab;
    else if(strcmp(modifier,"Backspace")==0)
        keySym = XK_BackSpace;
    else if(strcmp(modifier,"Enter")==0)
        keySym = XK_Return;
    else
        keySym = XK_space;

    event = createKeyEvent(display, winFocus, winRoot, True, keySym, 0 );
    XSendEvent(event.display, event.window, True, KeyPressMask, (XEvent *)&event);

    event = createKeyEvent(display, winFocus, winRoot, False, keySym, 0 );
    XSendEvent(event.display, event.window, True, KeyReleaseMask, (XEvent *)&event);

    (*jnienv)->ReleaseStringUTFChars(jnienv,modifierName,modifier);

    XCloseDisplay(display);
}
```

6.2.3. Codi natiu per Mac

NativeIssue.h

```
/*
 *   NativeIssue.h
 *   Macintosh
 *
 *   Created by eloicreus on 06/09/11.
 *
 */
#import <Cocoa/Cocoa.h>

#import <JavaNativeFoundation/JavaNativeFoundation.h> // helper framework for Cocoa
and JNI development
#import <AppKit/AppKit.h>
#import <JavaVM/jawt_md.h>
#import <ApplicationServices/ApplicationServices.h>
#import <Carbon/Carbon.h>

NSView * GetViewFromComponent(jobject object,JNIEnv *jnienv);

JNIEXPORT void JNICALL Java_keysforfree_MainWindow_setWindowInactivated(JNIEnv *
jnienv,jobject object);

JNIEXPORT void JNICALL Java_keysforfree_PredictorWindow_setWindowInactivated(JNIEnv
* jnienv,jobject object);

void setWindowInactivated(JNIEnv * jnienv,jobject object);

void simulateKeyStroke(CGEventRef keyDown, CGEventRef keyUp, UniChar character);

JNIEXPORT void JNICALL Java_keysforfree_AbstractKeyboard_setKeyAtComponent(JNIEnv
* jnienv, jobject object, jstring keyTyped);

JNIEXPORT void JNICALL Java_keysforfree_AbstractKeyboard_setTextModifier(JNIEnv *
jnienv, jobject object, jstring modifierName);
```

NativeIssue.m

```
/*
 *   NativeIssue.c
 *   Macintosh
 *
 *   Created by eloicreus on 06/09/11.
 *
 */

#import "NativeIssue.h"

/*
 * Serveix per obtenir la vista que li ha assignat el sistema operatiu una pantalla
```

```
 * del programa. Això ens permetrà obtenir la finestra que se l'hi la assignat.
 */
NSView * GetViewFromComponent(jobject object,JNIEnv *jnienv)
{
        JAWT awt;
        JAWT_DrawingSurface* ds;
        JAWT_DrawingSurfaceInfo* dsi;
        JAWT_MacOSXDrawingSurfaceInfo* dsi_mac;
        jboolean result;
        jint lock;

        awt.version = JAWT_VERSION_1_4;
        result = JAWT_GetAWT(jnienv,&awt);
        assert(result != JNI_FALSE);

        ds = awt.GetDrawingSurface(jnienv,object);
        assert(ds != NULL);

        lock = ds->Lock(ds);
        assert((lock & JAWT_LOCK_ERROR) == 0);

        dsi = ds->GetDrawingSurfaceInfo(ds);

        dsi_mac = (JAWT_MacOSXDrawingSurfaceInfo*)dsi->platformInfo;

        NSView *view = dsi_mac->cocoaViewRef;

        ds->FreeDrawingSurfaceInfo(dsi);

        ds->Unlock(ds);

        awt.FreeDrawingSurface(ds);

        return view;
}

/*
 * Amb la vista que té assignada la finestra del teclat virtual o de la
 * pantalla de predicció obtenim la finestra que té assignada.
 * Amb aquesta finestra indiquem al sistema que volem un estil de
 * finestra que no agafi el focus quan la polsin.
 */
void setWindowInactivated(JNIEnv * jnienv, jobject object)
{
    JNF_COCOA_ENTER(jnienv);

    NSView *view = GetViewFromComponent(object,jnienv);
    NSWindow *window = [view window];

    [JNFRunLoop performOnMainThreadWaiting:YES withBlock:^() {

    [window setStyleMask:[window styleMask]|NSNonactivatingPanelMask];
    [window setAcceptsMouseMovedEvents:YES];
```

```
    }];

    NSLog(@"fet");

    JNF_COCOA_EXIT(jnienv);
}


/*
 * Class:     MainWindow
 * Method:    setWindowInactivated
 * Signature: ()V
 * L'executa la pantalla principal del teclat virtual, per posar-se en un estat
 * en que no rebi el focus.
 */
JNIEXPORT void JNICALL
Java_keysforfree_MainWindow_setWindowInactivated(JNIEnv * jnienv, jobject object)
{
    setWindowInactivated(jnienv, object);
}


/*
 * Class:     PredictorWindow
 * Method:    setWindowInactivated
 * Signature: ()V
 * L'executa la pantalla de predicció del teclat virtual, per posar-se en un estat
 * en que no rebi el focus.
 */
JNIEXPORT void JNICALL
Java_keysforfree_PredictorWindow_setWindowInactivated(JNIEnv * jnienv,jobject object)
{
    setWindowInactivated(jnienv, object);
}


/*
 * Simula la pulsació d'un determinat caracter.
 */
void simulateKeyStroke(CGEventRef keyDown, CGEventRef keyUp, UniChar character)
{
    CGEventKeyboardSetUnicodeString(keyDown, 1, &character);
    CGEventPost(kCGAnnotatedSessionEventTap, keyDown);
    CGEventKeyboardSetUnicodeString(keyUp, 1, &character);
    CGEventPost(kCGAnnotatedSessionEventTap, keyUp);
}

/*
 * Class:     AbstractKeyboard
 * Method:    setKeyAtComponent
 * Signature: (Ljava/lang/String;)V
 * S'utilitza per escriure un determinat caràcter o una paraula al component
```

```
 * que té el cursor d'escriptura.
 * En aquest cas simulem la pulsació informant al sistema de la pulsació
 * d'una tecla i aquest ja l'enviarà a l'aplicació activa.
 */
JNIEXPORT void JNICALL
Java_keysforfree_AbstractKeyboard_setKeyAtComponent(JNIEnv * jnienv, jobject object,
jstring keyTyped)
{
        JNF_COCOA_ENTER(jnienv);

    const jchar * character = (*jnienv)->GetStringChars(jnienv, keyTyped, NULL);
        NSString *charsTyped = [NSString stringWithCharacters:(UniChar *)character
                                                        length:(*jnienv)-
>GetStringLength(jnienv, keyTyped)];
        (*jnienv)->ReleaseStringChars(jnienv, keyTyped, character);

    CGEventRef keyDown = CGEventCreateKeyboardEvent(NULL, 1, TRUE);
    CGEventRef keyUp = CGEventCreateKeyboardEvent(NULL, 1, FALSE);
    UniChar charc;

    for(int i=0; i < [charsTyped length]; i++)
    {
        [charsTyped getCharacters:&charc range:NSMakeRange(i,1)];
        simulateKeyStroke(keyDown,keyUp,charc);
    }
    CFRelease(keyDown);
    CFRelease(keyUp);

        JNF_COCOA_EXIT(jnienv);

        return;
}

/*
 * Class:       AbstractKeyboard
 * Method:      setTextModifier
 * Signature: (Ljava/lang/String;)V
 * S'utilitza per simular la pulsació d'un modificador de text.
 * Amb el nom d'aquest modificador que ens passen obtenim el seu codi
 * virtual i indiquem al sistema la pulsació d'aquest caràcter.
 */
JNIEXPORT void JNICALL
Java_keysforfree_AbstractKeyboard_setTextModifier(JNIEnv * jnienv, jobject object, jstring
modifierName)
{
        JNF_COCOA_ENTER(jnienv);

    const jchar * modifier = (*jnienv)->GetStringChars(jnienv, modifierName, NULL);
        NSString *modifierTyped = [NSString stringWithCharacters:(UniChar *)modifier
                                                        length:(*jnienv)-
>GetStringLength(jnienv, modifierName)];
        (*jnienv)->ReleaseStringChars(jnienv, modifierName, modifier);
```

```
    CGKeyCode modifierCode;
    if([modifierTyped isEqualToString:@"Tab"])
        modifierCode = kVK_Tab;
    else if ([modifierTyped isEqualToString:@"Backspace"])
        modifierCode = kVK_Delete;
    else if ([modifierTyped isEqualToString:@"Enter"])
        modifierCode = kVK_Return;
    else
        modifierCode = kVK_Space;

    CGEventRef keyDown = CGEventCreateKeyboardEvent(NULL, modifierCode, TRUE);
    CGEventRef keyUp = CGEventCreateKeyboardEvent(NULL, modifierCode, FALSE);

    CGEventPost(kCGAnnotatedSessionEventTap, keyDown);
    CGEventPost(kCGAnnotatedSessionEventTap, keyUp);

    CFRelease(keyUp);
    CFRelease(keyDown);

    JNF_COCOA_EXIT(jnienv);
}
```