Documentació 2a Entrega PROP

Creador de Teclats

Quadrimestre de tardor, curs 2023/24

Versió lliurament: 2.0

Grup 31.3

Marc Mostazo

Arnau Tajahuerce

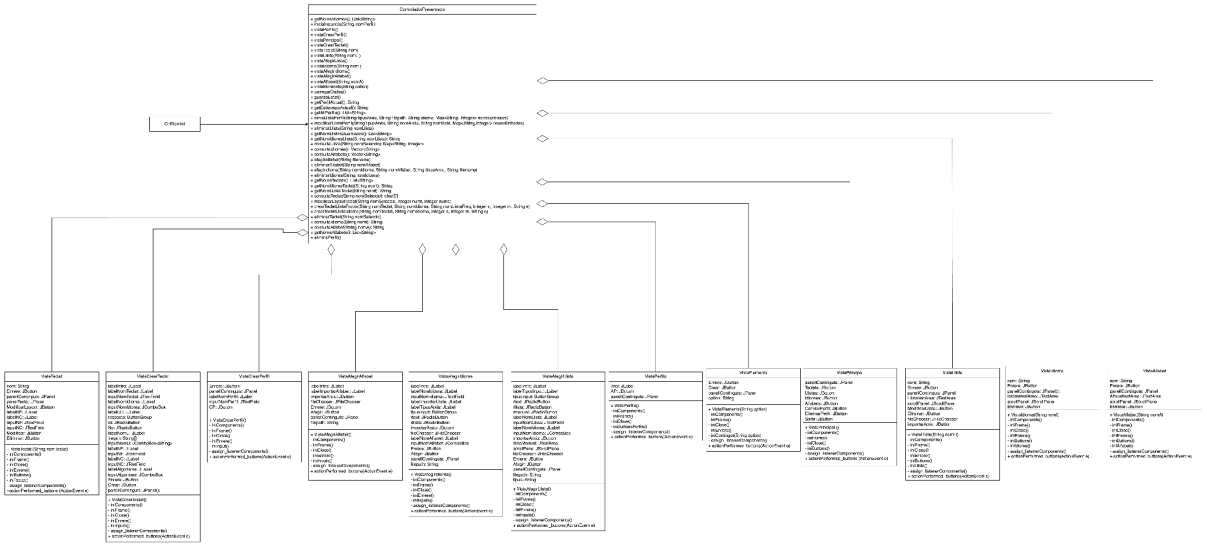
Agustí Costabella

Francisco Torredemer

Taula de continguts

1. Diagrama de les classes i controladors de la capa de presentació

A continuació mostrem el disseny del diagrama de les classes i controladors de la capa de presentació. El diagrama es pot trobar detallat a la carpeta DOCS.



1.1 Explicació de les classes de presentació:

Vistes:

1.1.1 VistaPerfils

Aquesta vista és la primera que apareix a l'executar el nostre sistema. Consisteix en un menú amb el títol del nostre sistema i tot seguit una llista horitzontal de botons que són els perfils que hi ha al sistema. També apareix un botó per afegir un perfil nou. Ens hem inspirat en aplicacions com Netflix i Prime Video a l'hora de crear aquesta vista, ja que partim d'un compte i arran d'aquest es poden crear diversos perfils, cadascun independent de l'altre.

- **Atributs:**
  - **Títol: JLabel.** Títol del sistema.
  - **AP: JButton.** Botó per canviar a la vista *VistaCrearPerfil*.
  - **panellContinguts: JPanel.** Panell de continguts.
- **Mètodes:**
  - **VistaPerfils().**
    - Constructora de la vista.
  - **iniComponents().**
    - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
  - **iniFrame().**
    - Inicialitza el marc.
  - **iniClose().**
    - Inicialitza el botó per sortir del programa.
  - **iniButtonsPerfils().**
    - Inicialitza els botons dels perfils i el d'afegir un nou perfil.
  - **assign_listenerComponents().**
    - Assigna els listeners als botons.
  - **actionPerformed_buttons(ActionEvent e).**
    - Dirigeix les accions basant-se en el botó clicat.

1.1.2 VistaCrearPerfil

Aquesta vista és l'encarregada de crear un perfil. Demana que s'introdueixi el nom i consta d'un botó per a crear-lo. Després de crear-lo, retorna a la pantalla inicial amb tots els perfils.

- **Atributs:**
  - **Enrere: JButton.** Botó per tornar enrere.
  - **panellContinguts: JPanel.** Panell de continguts.
  - **labelNomPerfil: JLabel.** Text que demana el nom del perfil.
  - **inputNomPerfil: JTextField.** Camp de text per introduir el nom del perfil.
  - **CP: JButton.** Botó per crear el nou perfil.
- **Mètodes:**
  - **VistaCrearPerfil()**
    - Constructora de la vista.
  - **iniComponents()**
    - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
  - **iniFrame()**
    - Inicialitza el marc.
  - **iniClose()**
    - Inicialitza el botó per sortir del programa.
  - **iniEnrere()**
    - Inicialitza el botó per tornar enrere.
  - **iniInput()**
    - Inicialitza els textos i camps d'entrada.
  - **assign_listenerComponents().**
    - Assigna els listeners als botons.
  - **actionPerformed_buttons(ActionEvent e).**
    - Dirigeix les accions basant-se en el botó clicat.

1.1.3 VistaPrincipal

Aquesta vista és l'encarregada de mostrar el menú principal del sistema i apareix just després de triar un perfil. Aquesta està formada per un conjunt de 6 botons. Els 4 primers et porten a vistes diferents per gestionar (crear/eliminar/modificar) les diferents dades del sistema (Teclats, Llistes de freqüències, Idiomes i Alfabets), el cinquè permet tornar a la vista dels perfils per canviar de perfil, el sisè per a eliminar el perfil l i l'últim permet sortir de l'aplicació.

- **Atributs:**
  - **panellContinguts: JPanel.** Panell de continguts.
  - **Teclats: JButton.** Botó per anar a la vista de teclats.
  - **Llistes: JButton.** Botó per anar a la vista de llistes.
  - **Idiomes: JButton.** Botó per anar a la vista d'Idiomes
  - **Alfabets: JButton.** Botó per anar a la vista d'alfabets.
  - **CanviarPerfil: JButton.** Botó per canviar de perfil.
  - **Sortir: JButton.** Botó per sortir del sistema.
- **Mètodes:**
  - **VistaPrincipal()**
    - Constructora de la vista
  - **iniComponents()**
    - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
  - **iniFrame()**
    - Inicialitza el marc
  - **iniClose()**
    - Inicialitza el botó per sortir del programa
  - **iniButtons()**
    - Inicialitza els sis botons del menú principal
  - **assign_listenerComponents().**
    - Assigna els listeners als botons.
  - **actionPerformed_buttons(ActionEvent e).**
    - Dirigeix les accions basant-se en el botó clicat.

1.1.4 VistaAlfabet

Aquesta vista és l'encarregada de mostrar la informació de l'alfabet amb el nom indicat a l'atribut de la classe. Aquesta està formada per un àrea de text la qual es mostra el nom de l'alfabet, el número de lletres de l'alfabet i aquestes. També inclou un botó per poder eliminar aquest alfabet del sistema i un altre per tornar a la vista anterior.

- **Atributs:**
    - **nom: String.** És el nom de l'alfabet, per a dur a terme la seva consulta.
    - **Enrere: JButton.** Botó per tornar enrere.
    - **panellContinguts: JPanel.** Panell de continguts.
    - **AlfabettextArea: JTextArea.** Àrea de text que mostra l'alfabet.
    - **scrollPanel: JScrollPane.** Panell que conté l'àrea de text de l'alfabet.
    - **Eliminar: JButton.** Botó per eliminar l'alfabet
- **Mètodes:**
    - **VistaAlfabet(String nomA)**
        - Constructora de la vista
    - **iniComponents()**
        - Funció que inicialitza els components de la vista
    - **iniFrame()**
        - Inicialitza el marc
    - **iniClose()**
        - Inicialitza el botó per sortir del programa
    - **iniEnrere()**
        - Inicialitza el botó per tornar enrere.
    - **iniButtons()**
        - Inicialitza el botó d'eliminar alfabet
    - **iniAlfabet()**
        - Inicialitza l'àrea de text que mostra l'alfabet
    - **assign_listenerComponents().**
        - Assigna els listeners als botons.
    - **actionPerformed_buttons(ActionEvent e).**
        - Dirigeix les accions basant-se en el botó clicat.

1.1.5 VistaAfegirAlfabet

Aquesta vista és l'encarregada de crear un alfabet. Aquesta està formada per un botó que permet seleccionar l'arxiu que conté les lletres de l'alfabet que es vol crear. Un cop importat l'arxiu, es pot fer click a un altre botó per crear l'alfabet. També s'inclou un botó que permet tornar a la vista anterior.

- **Atributs:**
  - **labelIntro: JLabel.** Text que demana que s'introdueixin les dades
  - **labelNomAlfabet: JLabel.** Text que demana el nom de l'alfabet
  - **inputNomAlfabet: JTextField.** Camp de text per introduir el nom de l'alfabet
  - **labelImportarAlfabet: JLabel.** Text que demana importar un alfabet
  - **importarArxiu: JButton.** Botó per importar un arxiu.
  - **fileChooser: JFileChooser.** Vista del directori home per importar un arxiu.
  - **Enrere: JButton.** Botó per tornar enrere
  - **Afegir: JButton.** Botó per afegir l'alfabet
  - **panellContinguts: JPanel.** Panell de continguts.
  - **filepath: String.** Filepath de l'arxiu importat
- **Mètodes:**
  - **VistaAfegirAlfabet()**
    - Constructora de la vista
  - **iniComponents()**
    - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
  - **iniFrame()**
    - Inicialitza el marc
  - **iniClose()**
    - Inicialitza el botó per sortir del programa
  - **iniEnrere()**
    - Inicialitza el botó per tornar enrere.
  - **iniInputs()**

        - Inicialitza els camps d'entrada.
    - **assign_listenerComponents().**
        - Assigna els listeners als botons.
    - **actionPerformed_buttons(ActionEvent e).**
        - Dirigeix les accions basant-se en el botó clicat.

1.1.6 Vistaldioma

Aquesta vista és l'encarregada de mostrar la informació de l'idioma amb el nom indicat a l'atribut de la classe. Aquesta està formada per un àrea de text la qual es mostra el nom de l'idioma, el nom de l'alfabet de l'idioma i el nom de la llista de freqüències predeterminada de l'idioma. També inclou un botó per poder eliminar aquest idioma del sistema i un altre per tornar enrere.

- **Atributs:**
    - **nom: String.** És el nom de l'idioma, per a dur a terme la seva consulta.
    - **Enrere: JButton.** Botó per tornar enrere.
    - **panellContinguts: JPanel.** Panell de continguts.
    - **IdiomatextArea: JTextArea.** Àrea de text que mostra l'idioma.
    - **scrollPanel: JScrollPane.** Panell que conté l'àrea de text de l'idioma.
    - **Eliminar: JButton.** Botó per eliminar l'Idioma
- **Mètodes:**
    - **Vistaldioma(String noml)**
        - Constructora de la vista
    - **iniComponents()**
        - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
    - **iniFrame()**
        - Inicialitza el marc
    - **iniClose()**
        - Inicialitza el botó per sortir del programa
    - **iniEnrere()**
        - Inicialitza el botó per tornar enrere.

- **iniButtons()**
  - Inicialitza el botó per eliminar l'idioma
- **iniIdioma()**
  - Inicialitza l'àrea de text que mostra l'idioma.
- **assign_listenerComponents().**
  - Assigna els listeners als botons.
- **actionPerformed_buttons(ActionEvent e).**
  - Dirigeix les accions basant-se en el botó clicat.

1.1.7 VistaAfegirIdioma

Aquesta vista és l'encarregada de crear un idioma. S'indica un camp clarament per introduir el nom de l'idioma que es vol crear. També mitjançant un botó es permet importar l'arxiu que conté la llista de freqüències predeterminada de l'idioma i seleccionar amb 2 radiobuttons si es de tipus "Llista" o "Text". A més a més s'inclou un desplegable per poder triar l'alfabet de l'idioma.Finalment, es pot fer click a un altre botó per crear l'idioma. També s'inclou un botó que permet tornar a la vista anterior.

- **Atributs:**
  - **labelIntro: JLabel.** Text que demana que s'introdueixin les dades
  - **labelNomIdioma: JLabel.** Text que demana el nom de l'idioma
  - **inputNomIdioma: JTextField.** Camp de text per introduir l'idioma
  - **labelImportarLlista: JLabel.** Text que demana que s'importi la llista de freqüències predeterminada de l'idioma
  - **labelTipusArxiu: JLabel.** Text que pregunta de quin tipus és l'arxiu que s'importarà.
  - **tipusInput: ButtonGroup.** Conjunt de botons d'opció.
  - **rtext: JRadioButton.** Botó d'opció text.
  - **rllista: JRadioButton.** Botó d'opció llista.
  - **importarArxiu: JButton.** Botó per importar l'arxiu
  - **fileChooser: JFileChooser.** Vista del directori home per importar un arxiu.
  - **labelNomAlfabet: JLabel.** Text que demana l'alfabet de l'idioma
  - **inputNomAlfabet: JComboBox.** Selector per a seleccionar l'alfabet.

- **Enrere: JButton.** Botó per tornar enrere.
- **Afegir: JButton.** Botó per afegir l'idioma.
- **panellContinguts: JPanel.** Panell de continguts.
- **filepath: String.** Filepath de l'arxiu importat

- **Mètodes:**
  - **VistaAfegirIdioma()**
    - Constructora de la vista
  - **iniComponents()**
    - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
  - **iniFrame()**
    - Inicialitza el marc
  - **iniClose()**
    - Inicialitza el botó per sortir del programa
  - **iniEnrere()**
    - Inicialitza el botó per tornar enrere.
  - **iniInputs()**
    - Inicialitza els camps d'entrada.
  - **assign_listenerComponents().**
    - Assigna els listeners als botons.
  - **actionPerformed_buttons(ActionEvent e).**
    - Dirigeix les accions basant-se en el botó clicat.

1.1.8 VistaLlista

Aquesta vista és l'encarregada de mostrar la informació de l'idioma amb el nom indicat a l'atribut de la classe. Aquesta està formada per un àrea de text la qual es mostren les totes les paraules i les seves freqüències de la llista de freqüències. També inclou un botó per poder eliminar aquesta llista de freqüències del sistema i un altre per tornar enrere.

- **Atributs:**
  - **nom: String.** Nom de la llista
  - **Enrere: JButton.** Botó per tornar enrere.

- **panellContinguts: JPanel.** Panell de continguts.
- **LlistatextArea: JTextArea.** Àrea de text que mostra la llista.
- **scrollPanel: JScrollPane.** Panell que conté l'àrea de text de la llista.
- **ModificarLlista: JButton.** Botó per modificar la llista
- **Eliminar: JButton.** Botó per eliminar la llista

- **Mètodes:**
  - **VistaLlista(String nomLl)**
    - Constructora de la vista
  - **iniComponents()**
    - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
  - **iniFrame()**
    - Inicialitza el marc
  - **iniClose()**
    - Inicialitza el botó per sortir del programa
  - **iniEnrere()**
    - Inicialitza el botó per tornar enrere.
  - **iniButtons()**
    - Inicialitza els botons d'eliminar i modificar llista
  - **iniLlista()**
    - Inicialitza el panell i l'àrea de text que mostra la llista
  - **assign_listenerComponents().**
    - Assigna els listeners als botons.
  - **actionPerformed_buttons(ActionEvent e).**
    - Dirigeix les accions basant-se en el botó clicat.

1.1.9 VistaAfegirLlista

Aquesta vista és l'encarregada de crear una llista de freqüències. Aquesta està formada per un desplegable que permet seleccionar l'idioma de la llista de freqüències. També s'ha de seleccionar amb 3 radiobuttons si és de tipus "Llista", "Text" o "Manual". En cas de seleccionar manual, s'obra un quadre de text per escriure la llista de les paraules i freqüències. A més a més s'indica un camp clarament per introduir el nom de la llista de freqüències que es vol crear. Finalment,

es pot fer click a un altre botó per crear la llista de freqüències. També s'inclou un botó que permet tornar a la vista anterior.

- **Atributs:**
    - **labelIntro: JLabel.** Text que demana que s'introdueixin les dades.
    - **labelTipusInput: JLabel.** Text que demana el tipus d'entrada de la llista.
    - **tipusInput: ButtonGroup.** Conjunt de botons d'opció.
    - **rtext: JRadioButton.** Botó d'opció text.
    - **rllista: JRadioButton.** Botó d'opció llista.
    - **rmanual: JRadioButton.** Botó d'opció manual.
    - **labelNomLlista: JLabel.** Text que demana el nom de la llista si és entrada manual.
    - **inputNomLlista: JTextField.** Camp de text per introduir el nom de la llista si és entrada manual.
    - **labelNomIdioma: JLabel.** Text que demana el nom de l'idioma.
    - **inputNomIdioma: JComboBox.** Selector de l'idioma.
    - **importarArxiu: JButton.** Botó per a importar l'arxiu si és entrada tipus text o llista.
    - **llistaManual: JTextArea.** Àrea de text per introduir la llista manualment.
    - **scrollPane: JScrollPane.** Panell que mostra la llista manual.
    - **fileChooser: JFileChooser.**Vista del directori home per importar un arxiu.
    - **Enrere: JButton.** Botó per tornar enrere.
    - **Afegir: JButton.** Botó per afegir la llista.
    - **panellContinguts: JPanel.** Panell de continguts.
    - **filepath: String.** Filepath de l'arxiu importat
    - **tipus: String.** Indica el tipus d'entrada de la llista (llista, text o manual).
- **Mètodes:**
    - **VistaAfegirLlista()**
        - Constructora de la vista
    - **iniComponents()**

        - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
    - **iniFrame()**
        - Inicialitza el marc
    - **iniClose()**
        - Inicialitza el botó per sortir del programa
    - **iniEnrere()**
        - Inicialitza el botó per tornar enrere.
    - **iniInputs()**
        - Inicialitza els camps d'entrada.
    - **assign_listenerComponents().**
        - Assigna els listeners als botons.
    - **actionPerformed_buttons(ActionEvent e).**
        - Dirigeix les accions basant-se en el botó clicat.

1.1.10 VistaTeclat

Aquesta vista és l'encarregada de mostrar la informació del teclat amb el nom indicat a l'atribut de la classe. En aquesta es mostra la disposició del teclat (formada per un conjunt de botons col·locats per n files i m columnes on cadascun dels botons té una lletra de l'alfabet de l'idioma del teclat. També inclou un botó per poder eliminar aquest teclat del sistema, un altre per modificar el layout i un altre per tornar enrere.

- **Atributs:**
    - **nom: String.** És el nom del teclat, per a dur a terme la seva consulta.
    - **Enrere: JButton.** Botó per tornar enrere.
    - **panellContinguts: JPanel.** Panell de continguts.
    - **panelTeclat: JPanel.** Panell que mostra el teclat
    - **ModificarLayout: JButton.** Botó per a modificar el layout del teclat
    - **labelNF: JLabel.** Text que demana el número de files del teclat
    - **labelNC: JLabel.** Text que demana el número de columnes del teclat
    - **inputNF: JTextField.** Camp de text per introduir el número de files del teclat

    - **inputNC: JTextField.** Camp de text per introduir el número de columnes del teclat
    - **Modificar: JButton.** Botó per modificar l'Idioma
    - **Eliminar: JButton.** Botó per eliminar l'Idioma
- **Mètodes:**
    - **VistaTeclat (String nomTeclat)**
        - Constructora de la vista
    - **iniComponents()**
        - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
    - **iniFrame()**
        - Inicialitza el marc
    - **iniClose()**
        - Inicialitza el botó per sortir del programa
    - **iniEnrere()**
        - Inicialitza el botó per tornar enrere.
    - **iniButtons()**
        - Inicialitza els botons d'eliminar i modificar el layout del teclat
    - **iniTeclat()**
        - Inicialtiza el panell que representa el teclat en una matriu de botons.
    - **assign_listenerComponents().**
        - Assigna els listeners als botons.
    - **actionPerformed_buttons(ActionEvent e).**
        - Dirigeix les accions basant-se en el botó clicat.

1.1.11 VistaCrearTeclat

Aquesta vista és l'encarregada de crear un teclat. S'indica un camp clarament per introduir el nom del teclat que es vol crear. També hi ha un desplegable que permet seleccionar l'idioma del teclat. També s'ha de seleccionar amb 2 radiobuttons es vol crear el teclat amb una llista de freqüències pròpia o no. En cas de seleccionar Sí, apareix un desplegable que permet seleccionar la llista de freqüències. A continuació hi han dos camps per introduir el número de files i columnes del teclat.

També hi ha un desplegable que permet seleccionar quins dels dos algorismes utilitzar per a la creació del teclat. Finalment, es pot fer click a un altre botó per crear el teclat. També s'inclou un botó que permet tornar a la vista anterior.

- **Atributs:**
    - **labelIntro: JLabel.** Text que demana que s'introdueixin les dades.
    - **labelNomTeclat: JLabel.** Text que demana el nom del teclat.
    - **inputNomTeclat: JTextField.** Camp de text per introduir el nom del teclat.
    - **labelNomIdioma: JLabel.** Text que demana seleccionar l'idioma del teclat.
    - **inputNomIdioma: JComboBox.** Desplegable que permet seleccionar un dels idiomes.
    - **labelULL: JLabel.** Text que demana si es vol utilitzar una llista de freqüències propia.
    - **resposta: ButtonGroup.** Conjunt de botons d'opció.
    - **Si: JRadioButton.** Botó d'opció Sí.
    - **No: JRadioButton.** Botó d'opció No.
    - **labelNomLI: JLabel.** Text que demana la llista si ha respost que Sí.
    - **inputNomLI: JComboBox<String>.** Desplegable per seleccionar la llista de freqüències si ha respost que Sí.
    - **labelNF: JLabel.** Text que demana el número de files del teclat
    - **inputNF: JTextField.** Camp de text per introduir el número de files del teclat
    - **labelNC: JLabel.** Text que demana el número de columnes del teclat
    - **inputNC: JTextField.** Camp de text per introduir el número de columnes del teclat
    - **labelAlgorisme: JLabel.** Text que demana seleccionar un algorisme
    - **inputAlgorisme: JComboBox.** Desplegable que permet seleccionar un dels dos algorismes
    - **Enrere: JButton.** Botó per tornar enrere.
    - **Crear: JButton.** Botó per crear el teclat.
    - **panellContinguts: JPanel.** Panell de continguts.
- **Mètodes:**

- **VistaCrearTeclat()**
  - Constructora de la vista
- **iniComponents()**
  - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
- **iniFrame()**
  - Inicialitza el marc
- **iniClose()**
  - Inicialitza el botó per sortir del programa
- **iniEnrere()**
  - Inicialitza el botó per tornar enrere.
- **iniInputs()**
  - Inicialitza els camps d'entrada.
- **assign_listenerComponents().**
  - Assigna els listeners als botons.
- **actionPerformed_buttons(ActionEvent e).**
  - Dirigeix les accions basant-se en el botó clicat.

1.1.12 VistaElements

Aquesta vista serveix per crear i llistar verticalment els elements de les classes: Teclats, LlistaFreqüències, Idiomes i Alfabets de manera que es mostren tants botons com elements de la classe hi hagi. Llista i crea els elements d'una determinada classe segons el paràmetre *option*.

- **Atributs:**
  - **Enrere: JButton.** Botó per tornar enrere.
  - **Crear: JButton.** Botó per crear un nou element del tipus corresponent.
  - **panellContinguts: JPanel.** Panell de continguts.
  - **option: String.** Tipus d'element (teclat, llista, alfabets, idiomes)
- **Mètodes:**
  - **VistaElements(String option)**
    - Constructora de la vista
  - **iniComponents()**

        - Defineix i afegeix els components i els seus contenidors, les característiques del JFrame i associa els listeners corresponents.
    - **iniFrame()**
        - Inicialitza el marc
    - **iniClose()**
        - Inicialitza el botó per sortir del programa
    - **iniEnrere()**
        - Inicialitza el botó per tornar enrere.
    - **iniContingut(String option)**
        - Inicialitza la llista vertical dels botons que representen un element respectivament.
    - **assign_listenerComponents().**
        - Assigna els listeners als botons.
    - **actionPerformed_buttons(ActionEvent e).**
        - Dirigeix les accions basant-se en el botó clicat.

1.2 Explicació del controlador de presentació:

1.2.1 CtrlPresentació

El controlador de Presentació s'encarrega de fer de comunicador entre les vistes de la capa de presentació. També és el que transmet a la capa de presentació les dades de les capes inferiors.

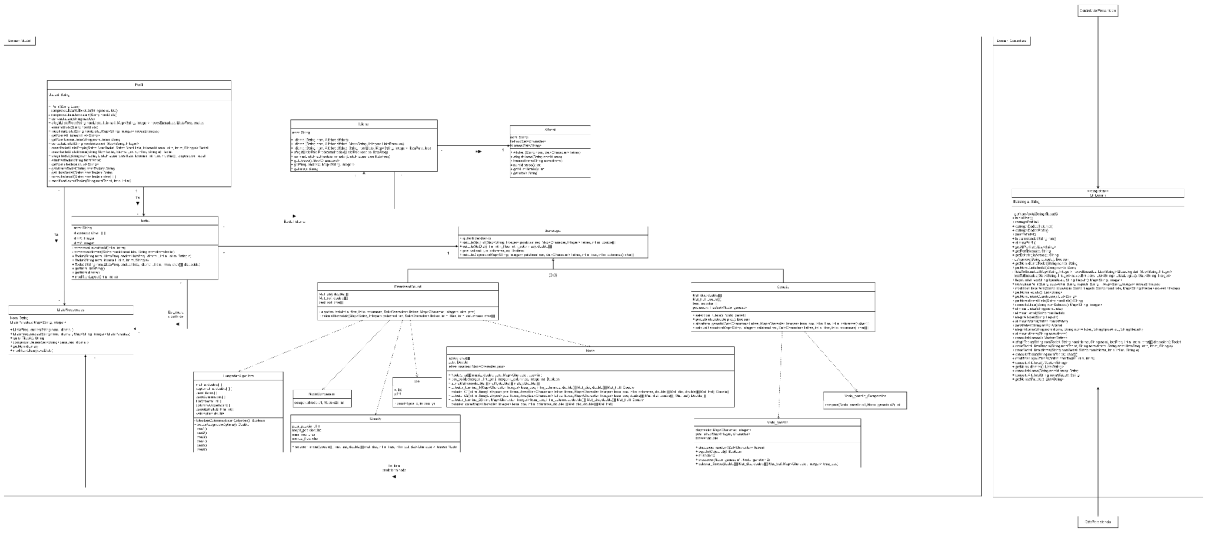
- **Mètodes:**
  - **vistaPerfils()**
    - Mostra en pantalla la finestra inicial dels perfils.
  - **vistaCrearPerfil()**
    - Mostra en pantalla la finestra per crear perfils.
  - **vistaPrincipal()**
    - Mostra en pantalla la finestra del menu principal.
  - **vistaCrearTeclat()**
    - Mostra en pantalla la finestra per crear teclats.
  - **vistaTeclat(String nom)**
    - Mostra en pantalla la finestra inicial dels teclats.
  - **vistaLlista(String nomLl)**
    - Mostra en pantalla la finestra inicial de les llistes..
  - **vistaAfegirLlista()**
    - Mostra en pantalla la finestra per afegir llistes.
  - **vistaIdioma(String nomI)**
    - Mostra en pantalla la finestra inicial dels idiomes.
  - **vistaAfegirIdioma()**
    - Mostra en pantalla la finestra per afegir idiomes.
  - **vistaAfegirAlfabet()**
    - Mostra en pantalla la finestra per afegir alfabets.
  - **vistaAlfabet(String nomA)**
    - Mostra en pantalla la finestra inicial dels alfabets.
  - **vistaElements(String option)**
    - Mostra en pantalla la finestra que mostra els elements de llistes, alfabets, idiomes o teclats, segons el paràmetre option.
  - **carregarDades()**
    - Crida als mètodes carregarDadesSistema i carregarDadesPerfil del de CtrlDomini que carrega totes les dades.

- **guardaEstat()**
  - Crida al mètode guardaEstat de CtrlDomini.
- **getPerfilActual()**
  - Crida al mètode getPerfilActual de CtrlDomini.
- **getEstrategiaActual()**
  - Crida al mètode getEstrategiaActual de CtrlDomini
- **getAllPerfils()**
  - Crida al mètode getAllPerfils de CtrlDomini.
- **novaLlistaPerfil(String tipusArxiu, String filepath, String i , Map<String,Integer> novesEntrades)**
  - Crida al mètode novaLlistaPerfil de CtrlDomini.
- **modificarLlistaPerfil(String tipusArxiu, String filename, String nomLlista, Map<String,Integer> novesEntrades)**
  - Crida al mètode modificarLlistaPerfil de CtrlDomini.
- **eliminarLlista(String nomLlista)**
  - Crida al mètode eliminarLlista de CtrlDomini.
- **getNomLlistesGuardades()**
  - Crida al mètodee getNomLlistesGuardades de CtrlDomini.
- **getNomIdiomaLlista(String nomllista)**
  - Crida al mètode getNomIdiomaLlista de CtrlDomini.
- **consultaLlista(String nomSeleccio)**
  - Crida al mètode consultaLlista de CtrlDomini.
- **consultaIdiomes()**
  - Crida al mètode consultaIdiomes de CtrlDomini.
- **consultaAlfabets()**
  - Crida al mètode consultaAlfabets de CtrlDomini.
- **afegirAlfabet(String filename)**
  - Crida al mètode afegirAlfabet de CtrlDomini.
- **eliminarAlfabet(String nomAlfabet)**
  - Crida al mètode eliminarAlfabet de CtrlDomini.
- **afegirIdioma(String nomIdioma, String nomAlfabet, String tipusArxiu, String filepath)**
  - Crida al mètode afegirIdioma de CtrlDomini.
- **eliminarIdioma(String nomIdioma)**

        - Crida al mètode eliminarIdioma de CtrlDomini.
    - **getNomsTeclats()**
        - Crida al mètode getNomsTeclats de CtrlDomini.
    - **getNomIdiomaTeclat(String nomt)**
        - Crida al mètode getNomIdiomaTeclat de CtrlDomini.
    - **getNomLListaTeclat(String nomt)**
        - Crida al mètode getNomLlistaTeclat de CtrlDomini.
    - **consultaTeclat(String nomTeclat)**
        - Crida al mètode consultaTeclat de CtrlDomini.
    - **modificarLayoutTeclat(String nomTeclat, int n, int m)**
        - Crida al mètode modificarLayoutTeclat de CtrlDomini.
    - **crearTeclatLlistaPropia(String nomTeclat, String nomIdioma, String nomLlistaFreq, int n, int m, String e)**
        - Crida al mètode crearTeclatLlistaPropia de CtrlDomini.
    - **crearTeclatLlistaIdioma(String nomTeclat, String nomIdioma, int n, int m, String e)**
        - Crida al mètode crearTeclatLlistaIdioma de CtrlDomini.
    - **eliminarTeclat(String nomTeclat)**
        - Crida al mètode eliminarTeclat de CtrlDomini.
    - **consultaIdioma(String nomIdioma)**
        - Crida al mètode consultaIdioma de CtrlDomini.
    - **consultaAlfabet(String nomAlfabet)**
        - Crida al mètode consultaAlfabet de CtrlDomini.
    - **getNomsAlfabets()**
        - Crida al mètode getNomsAlfabets de CtrlDomini.
    - **eliminaPerfil()**
        - Elimina el perfil actual i la seva informació relacionada

2. Diagrama de les classes i controladors de la capa de domini

A continuació mostrem el disseny del diagrama de les classes i controladors de la capa de domini. El diagrama es pot trobar detallat a la carpeta DOCS.



2.1 Explicació de les classes de domini:

2.1.1 Teclat

Aquesta classe fa referència als teclats que han estat creats pels usuaris, que es generen a partir d'una llista de freqüències i tenen un idioma determinat.

- **Atributs:**
  - **nom:** String
  - **disposició:** char[][]. Matriu de caràcters que representa la disposició del teclat. Cada posició la ocupa una lletra. La mida d'aquesta matriu la determinen dimX i dimY.
  - **dimX:** Integer. Representa el nombre de files del teclat
  - **dimY:** Integer. Representa el nombre de columnes del teclat
- **Mètodes:**
  - **comprovaLayoutValid(Integer n, Integer m)**
    - Si n*m < nombre de lletres de l'alfabet de l'idioma del teclat, aleshores retorna l'excepció [LayoutNoValid]
    - Si n*m > nombre de lletres de l'alfabet de l'idioma del teclat i la mida total deixa files o columnes totalment buïdes, aleshores retorna l'excepció [LayoutMassaGran]
  - **comprovaldiomes(String i1, String i2)**
    - Si l'idioma i1 != i2, aleshores retorna l'excepció [IdiomesDiferents]
  - **Teclat(String nom, LlistaFrequencies llistafreq, Idioma i , Integer n, Integer m)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **Teclat (String, Idioma, Integer, Integer)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **modificarLayout(Integer n, Integer m)**
    - Comprova que el layout (n*m) sigui vàlid i si ho és, modifica la disposició amb el nou layout.
- **Relacions:**

- Relació d'associació amb la classe Idioma. Associació "És de l'idioma" amb multiplicitat molts a 1, un Teclat és d'un únic Idioma mentre que un Idioma pot tindre molts Teclats. Navegabilitat unidireccional cap a Idioma.
- Relació d'associació amb la classe LlistaFreqüències. Associació "Es genera a partir de" amb multiplicitat molts a 1, un Teclat té és d'una única LlistaFreqüències mentre que una LlistaFreqüències pot tindre diversos teclats. Navegabilitat unidireccional cap a LlistaFreqüències.
- Relació d'associació amb la classe Perfil. Associació "Té" amb multiplicitat molts a 1, un Teclat és d'un únic Perfil mentre que un Perfil disposa de diversos teclats. No es necessita navegabilitat cap a aquesta classe.
- Relació d'agregació amb la classe Estrategia. Multiplicitat molts a 1. Navegabilitat unidireccional cap a la classe Estrategia.

2.1.2 Alfabet

Aquesta classe fa referència als alfabets que hi ha disponibles al sistema i que es generen a partir d'una llista de caràcters.

- **Atributs:**
  - **nom:** String. Identificador de l'alfabet
  - **lletres:** Set<Character>. Conjunt de lletres de l'alfabet
- Mètodes:
  - **Alfabet(String nom, Set<Character> lletres)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **afegirIdioma(String nomIdioma)**
    - S'afegeix a idiomes el nomIdioma
  - **treureIdioma(String nomIdioma)**
    - Es treu d'idiomes el nomIdioma
  - **numIdiomes()**
    - Retorna idiomes.size()
  - **getNumLletres()**
    - Retorna lletres

- **getInfo()**
  - Retorna un String amb una frase que inclou el nom, el número de lletres de l'alfabet i les lletres de l'alfabet

- **Relacions:**
  - Relació d'associació amb la classe Idioma. Associació "Té" amb multiplicitat 1 a molts, un Alfabet pot tindre diversos idiomes però un Idioma només té un únic alfabet. No necessita navegabilitat cap a la classe Idioma.

2.1.3 HungarianAlgorithm

Aquesta classe fa referència a l'algorisme hongarès. En aquest cas, s'utilitza per calcular a partir d'una matriu de costs, on cada fila és una lletra i cada columna és una posició del teclat, l'assignació òptima, de tal manera que el cost d'assignar a totes les lletres a una determinada posició, sigui mínim.

- **Atributs:**
  - **matriu:** double[][]. Matriu que s'utilitza per a fer els càlculs de l'assignació òptima. És una matriu quadrada.
  - **copiamatriu:** double[][]. Matriu original, s'utilitza per a obtenir els costs inicials de cada posició, ja que l'altre matriu es va modificant al llarg de l'execució.
  - **zeroFila:** int[].Vector de 0 o 1. Sent i la fila, zeroFila[i] indica si hi ha un zero a la fila, si és 1 hi ha un zero marcat, altrament no.
  - **zeroColumna:** int[].Vector de 0 o 1. Sent i la columna, zeroColumna[i] indica si hi ha un zero a la columna, si és 1 hi ha un zero marcat, altrament no.
  - **filaCoberta:** int[]. Vector de 0 o 1. Sent i la fila, filaCoberta[i] indica si està coberta, si és 1 està coberta, altrament no.
  - **columnaCoberta:** int[] Sent i la columna, columnaCoberta[i] indica si està coberta, si és 1 està coberta, altrament no.
  - **zerosEstrellaEnFila:** int[]. Guarda les posicions dels zeros "estrella". Sent i la fila, zerosEstrellaEnFila[i] indica la columna del zero "estrella.
  - **valoroptim:** double. S'obté a partir de la suma dels costos assignats de manera òptima.

- **Mètodes:**
  - **totesLesColumnesEstanCobertes():Boolean**
    - Retorna cert si totes les columnes estan cobertes, és a dir, columnaCoberta[i]=1 on i > 0 , i <columnaCoberta.length. Altrament, retorna fals. El seu cost és O(n).
  - **trobarAssignacioOptima():Double**
    - Calcula amb la resta de mètodes el valor òptim de l'assignació de costos donats per la matriu. El seu cost és $O(n^3)$.
  - **pas1()**
    - Redueix la matriu de manera que en cada fila i columna hi hagi com a mínim un zero. Resta el valor mínim de cada fila a cada element de la fila i resta el valor mínim de cada columna a cada element de la columna. El seu cost és $O(n^2)$.
  - **pas2()**
    - Marca cada zero que troba si no hi ha cap altre zero marcat a la mateixa fila i columna. El seu cost és $O(n^2)$.
  - **pas3()**
    - Cobreix totes les columnes que tenen un zero marcat. El seu cost és O(n).
  - **pas4()**
    - Troba el primer zero no cobert (zero "estrella) i guarda la seva posició al vector zeroEstrellaEnFila. Sent i la fila del zero "estrella", zeroEstrellaEnFila[i] és la columna d'aquest. El seu cost és $O(n^2)$.
  - **pas5()**
    - Comença amb un zeroPrincipal que és el primer zero no cobert. Després, crea una cadena K de "Zeros" i "0*" alternats. Si troba un camí de zeros marcats, alterna entre "Zeros" i "0*". Si no troba un nou zero marcat, acaba la seqüència. El seu cost és O(n).
  - **pas6()**
    - Troba el valor no cobert més petit de la matriu. Després, realitza operacions de resta i suma:

- Es resta a tots els elements no coberts.
- Es suma a tots els elements que estiguin coberts dues vegades.

El seu cost és $O(n^2)$.

- **Relacions:**
  - Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

2.1.4 Idioma

Aquesta classe fa referència als idiomes que hi ha disponibles al sistema i que es generen a partir d'un alfabet i una llista de freqüències predeterminada.

- **Atributs:**
  - **nom:** String. Identificador de l'idioma
- **Mètodes:**
  - **Idioma(String nom, Alfabet alfabet)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **Idioma(String nom, Alfabet alfabet, Map<String, Integer> llistaParaules)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents, creant una nova llista de freqüències amb nom "LlistaPrednomIdioma" a partir de la llistaParaules.
  - **afegirLlistaFreqPredeterminada(LlistaFrequencies llistaFreq)**
    - La llistaFreq passa a ser la nova llista de freqüències predeterminada de l'idioma si abans no en tenia cap.
  - **canviarLlistaFreqPredeterminada(LlistaFrequencies llistaFreq)**
    - La llistaFreq passa a ser la nova llista de freqüències predeterminada de l'idioma
  - **getLletres()**
    - Retorna el les lletres de l'alfabet de l'idioma
  - **getFrequencies()**
    - Retorna les paraules i les freqüències de la llista de freqüències predeterminada de l'idioma.

- **getInfo()**
  - Retorna un String amb una frase que inclou el nom de l'idioma, el nom de l'alfabet i el nom de la llista de freqüències predeterminada de l'idioma

- **Relacions:**
  - Relació d'associació amb la classe Alfabet. Associació "Té " amb multiplicitat molts a 1, un alfabet pot ser de molts idiomes mentre un Idioma només té un alfabet. Navegabilitat unidireccional cap a la classe Alfabet.
  - Relació d'associació amb la classe LlistaFreqüències. Associació "Té llista predeterminada" amb multiplicitat 1 a 1, un idioma té una llista de freqüències predeterminada i una llista de freqüències és d'un idioma. Navegabilitat bidireccional.
  - Relació d'associació amb la classe Teclat. Navegabilitat unidireccional de Teclat a Idioma.

2.1.5 Perfil

Aquesta classe fa referència als perfils que hi han donats d'alta al sistema.

- **Atributs:**
  - **Usuari:** String. Identificador del perfil
- **Mètodes:**
  - **comprovaLListaNoExisteix(String nomLlista)**
    - Comprova si la llista identificada per nomLlista existeix al perfil i si no llença una excepció
  - **comprovaLlistaJaExisteix(String nomLlista)**
    - Comprova si la llista identificada per nomLlista ja existeix al perfil i si ja existeix
  - **Perfil(String User)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **canviaUsuari(String newUs)**
    - Canvia l'usuari del perfil per newUs
  - **afegirLlistaFreq(String llista)**
    - Afegeix la llista de freqüències 'llista' al perfil

- **eliminaLlista(String nomLlista)**
  - elimina la llista de freqüències identificada per nomLlista del perfil
- **modificarLlista(String nomLlista, Map<String, Integer> novesEntrades)**
  - Modifica la llista de paraules de la llista identificada per nomLlista i ho substitueixi per novesEntrades\
- **getNomAllLlistes(): List<String>**
  - Obté el conjunt de noms de totes les llistes del perfil
- **consultaLlista(String nomSeleccio): Map <String,Integer>**
  - Consulta el contingut de la llista identificada per nomSeleccio
- **crearTeclatLlistaPropia(String NomTeclat, String NomLlista, idioma, int n, int m):Teclat**
  - Es crea un teclat amb NomTeclat a partir de la llista identificada per NomLLista, idioma 'idioma' i disposició n*m, i es guarda al perfil, retorna el teclat creat
- **crearTeclatLlistaIdioma(String NomTeclat, Idioma i, int n, int m):Teclat**
  - Es crea un teclat amb NomTeclat a partir de la llista predeterminada de l'idioma 'idioma' i disposició n*m, i es guarda al perfil, retorna el teclat creat
- **eliminarTeclat(String NomTeclat)**
  - S'elimina el teclat identificat per NomTeclat i s'elimina per tant de perfil
- **getNomsTeclats():List<String>**
  - obté el conjunt de noms dels teclats del perfil, retorna una llista amb els noms
- **consultaTecalt(String nomTeclat):char[][]**
  - Es consulta la disposició del teclat identificat per nomTeclat, retorna la dispocició obtinguda
- **modificarLayoutTeclat(String nomTeclat, int n, int m)**
  - Es modifica el layout del teclat nomTeclat que canvia a n*m

- **Relacions:**

- Relació d'associació amb la classe Perfil. Associació "Té" amb multiplicitat molts a 1, una LlistaFreqüències és d'un únic Perfil mentre que un Perfil pot tindre diverses LlistaFreqüències. Navegabilitat unidireccional cap a LlistaFreqüències.
- Relació d'associació amb la classe Teclat. Associació "Té" amb multiplicitat 1 a molts, un Perfil disposa de diversos teclats mentre que un Teclat és d'un únic Perfil. Navegabilitat unidireccional cap a la classe Teclat.

2.1.6 LlistaFreqüències

Aquesta classe fa referència al les llistes que ha creat un donat perfil al sistema, que es generen a partir d'un input de l'usuari (fitxers de llista, text, o entrada manual).

- **Atributs:**
  - **Nom:** String. Identificador de la LlistaFreqüències
  - **LlistaParaules:** Map<String, Integer>. Llista de paraules amb freqüències de la LlistaFrequencies
- **Mètodes:**
  - **LlistFrequencies(String nom, Idioma i)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **LlistaFrequencies(String nom, Idioma i,Map<String, Integer> LlistaParaules)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents
  - **quitarTilde(char c): String**
    - retorna el caracter c sense accent
  - **comprovarLletres(Set<String> paraules, Idioma i)**
    - Comprova que les lletres del conjunt de paraules estan incloses a l'alfabet de l'idioma
  - **modificarLlista(String novaLlista)**
    - Substitueix la llista de paraules de la classe per novaLLista
- **Relacions:**

- Relació d'associació amb la classe Idioma. Associació "Té llista predeterminada" amb multiplicitat 1 a 1, un idioma té una llista de freqüències predeterminada i una llista de freqüències és d'un idioma. Navegabilitat bidireccional.
- Relació d'associació amb la classe Perfil. Associació "Té" amb multiplicitat molts a 1, una LlistaFreqüències és d'un únic perfil mentre que un Perfil pot tenir diverses LlistaFreqüències. No necessita navegabilitat cap a la classe Perfil.
- Relació d'associació amb la classe Teclat. Associació "Es genera a partir de" amb multiplicitat 1 a molts, una LlistaFreqüències pot tindre molts teclats mentre que un Teclat es genera a partir d'una única LlistaFreqüències. No necessita navegabilitat cap a la classe Teclat.

2.1.7 BranchandBound

Aquesta classe fa referència al primer algoritme empleat per resoldre l'assignació de les lletres de l'abecedari al teclat segons les freqüències introduïdes.

- **Atributs:**
  - **Matriu de distàncies:** Double[][], guarda la distància de cada ubicació amb totes les altres ubicacions. La seva mida serà de n per n, amb n igual al nombre de lletres en el abecedari.
  - **Matriu de trànsit:** Double[][], guarda el trànsit que hi ha per cada parell de lletres. La seva mida serà també per tant de n per n, amb n igual al nombre de lletres de l'abecedari.
  - **Best_sol:** char[][], es el layout de la millor solució encontrada pel algoritme.
- **Mètodes:**
  - **algoritm_bab(int n_filas, int n_columnas, Set<Character> lletres, Map<Character, Integer> letra_pos)**

        - Es la funció que realitza l'algoritme branch and bound amb una estratègia eager per recorrer las posibles solucions i arribar a la més òptima. Quan arriba a la solució definitiva, iguala la solució encontrada al layout de la classe. Comença amb una solució inicial que obté amb el mètode solucion_inicial de la classe Greedy. El cost serà de n per el nombre de files per el nombre de columnes per el càlcul de la cota, per tant serà de $O(n^5)$.
    - **calculaDisposicio(Map<String, Integer> palabrasFrec, Set<Character> lletres, int n_filas, int n_columnas)**
        - Es el mètode que es crida per la realització del algoritme. És una funció abstracta a Estrategia, ja que es defineix el mètode a cada subclase d'aquesta. Retorna l'assignació òptima del teclat per els valors introduïts. El seu cost ve determinat per la funció algoritm_bab.
- **Relacions:**
    - Relació de dependència amb la classe HungarianAlgorithm, Greedy, Nodo, NodoComparator i pos.
    - Relació d'implementació amb la interfície Estrategia. BranchandBound implementa aquesta.

2.1.8 Pos

Identifica per una ubicació del teclat de 0 a n, sent n el nombre de lletres que té l'abecedari, la seva posició 'x' que fa referència a la seva fila al layout del teclat i la seva posició 'y' que fa referència a la seva columna al layout del teclat.

- **Atributs:**
    - **X:** int, la fila de la posició
    - **Y:** int, la columna de la posició.
- **Mètodes**
    - **pos(int pos_x, int pos_y)**
        - La creadora de la posició que assigna els paràmetres corresponents al objecte.
- **Relacions:**

- Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

2.1.9 Nodo

Representa cada solució parcial que explora l'algorisme, que segons la seva cota serà seleccionat per el procés de bounding per trobar altres nodes fins arribar al node o solució final.

- **Atributs:**
  - **Layout:** char[][] es la distribució del teclat de la solució parcial o final del node, amb un número n de lletres instal·lades a n posicions del layout.
  - **Cota:** Double es el valor de l'estimació del tràfic i les distàncies de les lletres instal·lades i les que no.
  - **Letras Usades:** Map<Character, pos>, conté cada lletra instalada al teclat (a la matriu layout del node) i la seva posició.
- **Mètodes:**
  - **Nodo(char[][] matriz, double cota, Map<Character, pos> ini)**
    - La creadora del node que assigna els paràmetres corresponents.
  - **pos_valida(Integer i, Integer j, Integer n_columnas, Integer n)**
    - Retorna fals per les posicions del layout de teclat que excedeixen del nombre de lletres de l'abecedari, i cert si es una posició utilitzable en el layout. Té cost O(1).
  - **sumaMatrices(double[][] mat1, double[][] mat2)**
    - Retorna una matriu de doubles amb la suma de les dos matrius dels paràmetres. El cost d'aquesta funció és del nombre de files per el nombre de columnes, i per tant, sent n el nombre de lletres total de l'abecedari, serà de O(n).
  - **calcular_termino_1(Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
    - Retorna un double amb el cost del trànsit entre les instal·lacions ja col·locades a la matriu layout del node. El seu cost serà del

nombre de files per el nombre de columnes per n, i per tant serà de: $O(n^2)$

- **calculo_C1(int m, ArrayList<pos> pos_libres, ArrayList<Character> letras_libres, Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
  - Aproxima el cost del trànsit entre les instal·lacions ja col·locades i les encara no col·locades. Retorna una matriu de doubles on per cada element i j de la matriu tenim el cost de col·locar la iéssima lletra lliure a la jéssima posició lliure amb respecte les instal·lacions ja fetes. El seu cost serà de n per el nombre de files per el nombre de columnes per n, i per tant serà de $O(n^3)$
- **calculo_C2(int m, ArrayList<pos> pos_libres, ArrayList<Character> letras_libres, Map<Character, Integer> letra_pos, double[][] Mat_dist, double[][] Mat_traf)**
  - Aproxima el cost de trànsit entre les instal·lacions no col·locades. Retorna una matriu de doubles on per cada element i j de la matriu tenim el cost de col·locar la iéssima lletra lliure a la jéssima posició lliure amb respecte les instal·lacions encara no emplaçades. El seu cost serà de n per n per n, i per tant de $O(n^3)$.
- **calcular_termino_2(int m, Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
  - Retorna un double que és el cost del terme 2, calculant les dues matrius C1 i C2 amb els dos mètodes anteriors, fent la suma d'aquestes dos matrius i posteriorment cridant al Hungarian Algorithm per resoldre l'assignació lineal òptima d'aquesta suma de matrius, que es el resultat que retorna. El seu cost serà la suma del càlcul de C1, més el càlcul de C2, més el cost de fer el hungarian, i per tant serà de $O(n^3) + O(n^3) + O(n^3)$, per tant $O(n^3)$.
- **calcular_cota(Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**

        - Assigna al paràmetre cota del node el valor de la suma del terme 1 i el terme 2, utilizant els mètodes que els calculen. El seu cost és igual al càlcul del primer terme més el càlcul del segon terme, i per tant serà de $O(n^2) + O(n^3)$, és dir, $O(n^3)$.
- **Relacions:**
    - Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

2.1.10 Greedy

Aquesta classe fa referència al algoritme greedy, que busca una solució inicial per la resolució del problema QAP amb el algorisme Branch and Bound.

- **Atributs:**
    - **Millor posició:** int, guarda la millor posició del layout donat, és dir, la que té una suma acumuluda de distàncies amb les altres posicions menor.
    - **Pitjor posició:** int, guarda la pitjor posició del layout donat, és dir, la que té una suma acumuluda de distàncies amb les altres posicions menor.
    - **Lletra més freqüent:** character, guarda la lletra més utilitzada.
    - **Lletra menys freqüent:** character, guarda la lletra menys utilitzada.
- **Mètodes:**
    - **solucion_inicial(double[][] mat_traf, double[][] mat_dist, int n_filas, int n_col, Set<Character> lletres)**
        - Primer busca la lletra més utilitzada i la menys, segons la matriu de freqüencies del paràmetre. Després busca també la pitjor i millor posició segons la matriu de distàncies que rep. Llavors el mètode retorna un node amb una matriu de layout buida, excepte a la millor posició calculada abans on posa la lletra més freqüent, i la pitjor posició on guarda la lletra menys freqüent. El seu cost és $O(n^2)$.
- **Relacions:**
    - Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

2.1.11 NodoComparator

Aquesta classe és utilitzada per l'algorisme Branch and Bound per comparar els nodes segons la seva cota i ordenar-los a una cola de prioritat de menor a major cota.

- **Mètodes:**
  - **compare(Nodo o1, Nodo o2)**
    - Si el node o1 te una cota menor que la del node o2, retornarà negatiu, si son iguals, retornarà 0 i si la cota del node o2 es superior a la del node o1 retornarà un nombre positiu.
- **Relacions:**
  - Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

2.1.12 Estrategia

Classe abstracta que utilitzem per aplicar el patró plantilla en el nostre disseny.

- **Mètodes:**
  - **quitarTilde(char c)**
    - Retorna un string amb el caràcter introduït sense accents ni signes.
  - **calculaMatTraf(Map<String, Integer> palabrasFrec, Map<Character, Integer> lletres, int n)**
    - Omple la matriu de trànsit calculant la freqüència per cada parell de lletres segons la llista de paraules i freqüències introduïda. El seu cost és $O(n^2)$.
  - **calculaMatDist(int n, int n_filas, int n_columnas)**
    - Omple la matriu de distàncies del Branch and Bound segons la distància Euclidiana per cada parell de posicions en el layout. El seu cost és $O(n^2)$.
  - **pos_valida(Integer i, Integer j, Integer n_columnas, Integer n):**
    - Retorna fals per les posicions del layout de teclat que excedeixen del nombre de lletres de l'abecedari, i cert si es una posició utilitzable en el layout.

- **calculaDisposicio(Map<String, Integer> palabrasFrec, Set<Character> lletres, int n_filas, int n_columnas)**
  - Mètode abstracte

- **Relacions:**
  - Relació d'implementació amb la classe BranchandBound. BranchandBound implementa aquesta interfície.

2.1.13 Genetic

Aquesta classe fa referència al segon algoritme empleat per resoldre l'assignació de les lletres de l'abecedari al teclat segons les freqüències introduïdes.

- **Atributs:**
  - **Matriu de distàncies:** Double[][], guarda la distància de cada ubicació amb totes les altres ubicacions. La seva mida serà de n per n, amb n igual al nombre de lletres en el abecedari.
  - **Matriu de trànsit:** Double[][], guarda el trànsit que hi ha per cada parell de lletres. La seva mida serà també per tant de n per n, amb n igual al nombre de lletres de l'abecedari.
  - **Best_sol:** char[][], es el layout de la millor solució encontrada pel algoritme.
  - **Població:** TreeSet<Nodo_genetic>, conjunt de solucions individuals
- **Mètodes:**
  - **seleccion_ruleta()**
    - Retorna un node_genetic de la població aleatori segons el seu fitness, els que tenen millor fitness tenen més probabilitats de ser escogits, i els que tenen menys fitness menys probabilitats. El seu cost és de 8 per n, per tant O(n).
  - **probabilidad(double prob)**
    - Rep un paràmetre de 0 a 1 que correspon a la probabilitat de que retorni cert, per tant si per exemple rep 0.1, i haurà un 10% de probabilitat de que retorni cert i un 90% de que retorni fals. Té cost(1).
  - **algoritme_genetic(Set<Character> lletres, Map<Character, Integer> letra_pos, int n_filas, int n_columnas)**

        - És la funció principal del algoritme, retornarà la solució definitiva. Genera primer una població aleatoria inicial de 4 vegades el tamany del abecedari, i després per un nombre n de generacions (hem posat 100 nosaltres) aniran millorant les solucions de la població. Per cada generació, sent m el nombre de lletres de l'abecedari, escogirem 2 progenitors diferents m/4 vegades amb el mètode ruleta, els quals creuarem obtenint 2 fills per cada parell de pares. Llavors per millorar la població substituïrem els m/4 pitjors membres de la població per els m/4 millors fills creats al procés d'encreuament. Després, també per cada generació, farem un procés de mutació amb una probabilitat del 10% per tots els membres de la població excepte el millor. Una vegada finalitzades les generacions, es retorna el layout del membre amb major fitness, en una matriu de characters. El seu cost és de 100 per n per 2 per el cost de calcular el fitness, i per tant serà de 200 per $O(n^3)$, és dir, $O(n^3)$.
    - **calculaDisposicio(Map<String, Integer> palabrasFrec, Set<Character> lletres, int n_filas, int n_columnas)**
        - Es el mètode que es crida per la realització del algoritme. És una funció abstracta a Estrategia, ja que es defineix el mètode a cada subclase d'aquesta. Retorna l'assignació òptima del teclat per els valors introduïts. El seu cost ve donat per la funció algoritme_genetic.

2.1.14 Nodo_genetic

- **Atributs:**
    - **disposicio:** Map<Character, Integer>, guarda per cada letra del abecedari la seva posició al layout del node. Sempre tindrem layouts complets, és dir, per cada lletra de l'abecedari tindrem la seva posició i per tant serà de tamany n, amb n igual al nombre de lletres de l'abecedari.

- **pos_letra:** Map<Integer, Character>, guarda per cada posició del teclat la lletra que conté, i com en el map anterior, serà de tamany n sent n el nombre de lletres de l'abecedari.
- **fitness:** double, mesura que tan bona és un node. Els millors nodes tindran fitness més alts, i els pitjors menys.

- **Mètodes:**
  - **disposicio_random(Set<Character> lletres)**
    - Iguala el layout del teclat a un totalment aleatori, assigna aleatoriament cada lletra de l'abecedari a una posició aleatoria del teclat. Té cost O(n).
  - **equals(Object obj)**
    - Retorna cert si un node és exactament igual a un altre node, i fals en cas contrari. Té cost O(1).
  - **mutacion()**
    - Modifica la disposició del node, agafa 2 posicions aleatòries diferents i les intercanvia. El seu cost és de O(1).
  - **crossover(Nodo_genetic n1, Nodo_genetic n2)**
    - Creua 2 nodes pares obtenint com a resultat de la funció un nou fill. Declara dos punts de cort aleatoris, el primer inferior al segon, i combina la part compresa entre els 2 punts del primer pare amb els elements no repetits del segon pare. El seu cost és de O(n).
  - **calcular_fitness(double[][] Mat_dist, double[][] Mat_traf, Map<Character, Integer> letra_pos)**
    - Iguala al valor del node el fitness que té, multiplicant la distància per la freqüència per cada posició i lletra. Com volem que el fitness per els millors nodes sigui el més gran possible, i la suma de distàncies per freqüències ha de ser el menor possible, igualarem el fitness a 1 entre aquesta suma. El seu cost és de $O(n^2)$.

2.1.15 Nodo_genetic_Comparator

- **Mètodes:**
    - **compare(Nodo_genetic o1, Nodo_genetic o2)**
        - Si el node o2 te una cota menor que la del node o1, retornarà negatiu, si son iguals, retornarà 1, per retornar el primer, i si la cota del node o1 es superior a la del node o2 retornarà un nombre positiu.

2.2 Explicació del controlador de domini:

2.2.1 CtrlDomini

El CtrlDomini és la classe que actua de controlador de les classes de domini. S'encarrega de la comunicació entre la capa de presentació / dades i la capa de domini.

- **Atributs:**
    - **Estrategia:** String, és el nom de l'estrategia que utilitza la creació del teclat.
- **Mètodes:**
    - **getNomArxiu(String filepath)**
        - Obté el nom de l'arxiu del filepath.
    - **inicialitzar()**
        - Inicialitza el controlador de domini i els controladors de persistència i estableix l'estratègia per defecte com "BranchAndBound".
    - **carregaPerfils()**
        - Carrega les dades de tots els perfils existents en el sistema a memoria.
    - **carregarDadesSistema()**
        - Carrega les dades de tots els alfabets i idiomes existents en el sistema a memòria.
    - **carregarDadesPerfil()**
        - Carrega les dades de totes les llistes de freqüències i teclats del perfil a memòria.
    - **guardaEstat()**
        - Guarda les dades de les llistes de freqüències i teclats del perfil a més a més dels alfabets i idiomes a persistencia.

- **iniciaInstancia(String nom)**
  - Inicia una instància amb el perfil donat pel nom, es guarden les llistes de freqüències i teclats del perfil actual i carreguen les llistes de freqüències i teclats del no perfil.
- **eliminaPerfil()**
  - Elimina el perfil actual, els seus teclats i llistes de freqüències.
- **getAllPerfils()**
  - Retorna el conjunt de noms d'usuari dels perfils del sistema.
- **getPerfilActual()**
  - Retorna l'usuari del perfil actual.
- **getEstrategiaActual()**
  - Retorna el nom de l'estratègia per fer la distribució de teclat.
- **esNumero(String paraula)**
  - Retorna TRUE si l'string és un número, FALESE en cas contrari.
- **getNomIdiomaTeclat(String nomt)**
  - Retorna el nom de l'idioma del teclat identificat per nomt.
- **getNomLListaTeclat(String nomt)**
  - Retorna el nom de la llista de freqüències del teclat identificat per nomt.
- **llistaToEntrades(Map<String, Integer> novesEntrades, List<String> LlistaLlegida)**
  - Converteix les dades llegides d'un arxiu de llista de freqüències a un Map de paraules i freqüències.
- **textToEntrades(Map<String, Integer> novesEntrades, List<String> LlistaLlegida)**
  - Converteix les dades llegides d'un arxiu de text a un Map de paraules i freqüències.
- **llegirLlistaFreq(String tipusArxiu, String filename)**
  - Llegeix una llista de freqüències de l'arxiu donat (ja sigui en format llista de freqüències o text) i la retorna com a Map.
- **novaLlistaPerfil(String tipusArxiu, String filepath, String i , Map<String,Integer> novesEntrades)**
  - Afegeix la informació de l'arxiu de llista de freqüències al perfil actual com a nova llista de freqüències.

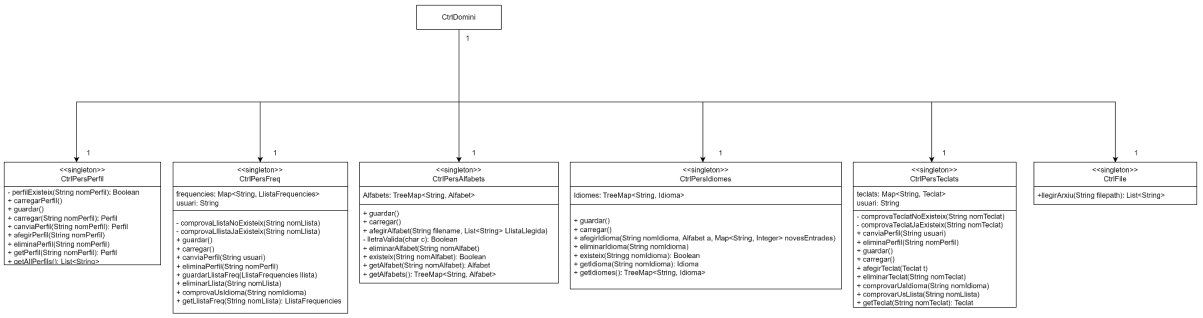
- **modificarLlistaPerfil(String tipusArxiu, String filename, String nomLlista, Map<String,Integer> novesEntrades)**
  - Modifica la informació de la llista de freqüències amb el nom especificat en el perfil actual amb les noves dades entrades.
- **getNomsTeclats()**
  - Retorna el conjunt de noms dels teclats del perfil actual.
- **getNomLlistesGuardades()**
  - Retorna el conjunt de noms de les llistes de freqüències del perfil actual.
- **getNomIdiomaLlista(String nomllista)**
  - Retorna el nom de l'idioma de la llista de freqüències identificada per nomllista.
- **consultaLlista(String nomSeleccio)**
  - Retorna el Map de paraules i freqüències de la llista de freqüències identificada per nomSeleccio.
- **eliminarLlista(String nomLlista)**
  - Si existeix, elimina la llista de freqüències identificada per nomLlista del perfil actual.
- **eliminarTeclat(String nomTeclat)**
  - Si existeix, elimina el teclat identificat per nomTeclat del perfil actual.
- **afegirAlfabet(String filename)**
  - Si no existeix en el sistema un alfabet identificat amb el nom de l'arxiu indicat, s'afageix un de nou amb aquest nom i les lletres que es troben dins de l'arxiu.
- **eliminarAlfabet(String nomAlfabet)**
  - Si existeix, s'elimina l'alfabet identificat per nomAlfabet del sistema.
- **getAlfabet(String nom)**
  - Si existeix, retorna l'alfabet identificat per nom.
- **afegirIdioma(String nomIdioma, String nomAlfabet, String tipusArxiu, String filepath)**

        - Si no existeix en el sistema un idioma identificat per nomIdioma, s'afegeix un de nou amb aquest nom, l'alfabet amb nomAlfabet i llista de freqüències creada amb les dades entrades.
    - **eliminarIdioma(String nomIdioma)**
        - Si existeix i aquest no està en ús per una llista de freqüències o teclat d'algun usuari, elimina l'idioma identificat per nomIdioma del sistema.
    - **consultaIdiomes()**
        - Retorna un vector d'strings amb informació de tots els idiomes del sistema.
    - **afegirTeclat(String nomTeclat, String nomIdioma, String nomLlistaFreq, int n, int m, char[][] disposicio)**
        - Si no existeix en el perfil actual un taclat identificat per nomTeclat, s'afegeix un de nou amb aquest nom, l'idioma identificat per nomIdioma, llista de freqüències identificat per nomLlistaFreq, n files, m columnes i disposició de lletres disposicio.
    - **crearTeclatLlistaPropia(String nomTeclat, String nomIdioma, String nomLlistaFreq, int n, int m, String e)**
        - Si no existeix en el perfil actual un teclat identificat per nomTeclat, s'afegeix i crea un de nou amb aquest nom, l'idioma identificat per nomIdioma, llista de freqüències identificat per nomLlistaFreq, n files, m columnes i utilitzant l'etrategia especificada a e.
    - **crearTeclatLlistaIdioma(String nomTeclat, String nomIdioma, int n, int m, String e)**
        - Si no existeix en el perfil actual un teclat identificat per nomTeclat, s'afegeix i crea un de nou amb aquest nom, l'idioma identificat per nomIdioma, la llista de freqüències predeterminada de l'idioma, n filas i m columnes i utilitzant l'estrategia especificada a e.
    - **consultaTeclat(String nomTeclat)**
        - Si existeix un teclat amb nomTeclat en el perfil actual, retorna una matriu de caràcters amb la disposició d'aquest.

- **modificarLayoutTeclat(String nomTeclat, int n, int m)**
  - Si existeix un teclat amb nomTeclat en el perfil actual, modifica el layout d'aquest amb n files i m columnes.
- **consultaAlfabets()**
  - Retorna un vector d'strings amb informació de tots els alfabets del sistema.
- **getNomsIdiomes()**
  - Retorna el conjunt de noms dels idiomes del sistema.
- **consultaIdioma(String nomIdioma)**
  - Si existeix un idioma identificat per nomIdioma en el sistema, retorna la informació d'aquest.
- **consultaAlfabet(String nomAlfabet)**
  - Si existeix un alfabet identificat per nomAlfabet en el sistema, retorna la informació d'aquest.
- **getNomsAlfabets()**
  - Retorna el conjunt de noms dels alfabets del sistema.

3. Diagrama de les classes i controladors de la capa de persistència

A continuació mostrem el disseny del diagrama de les classes i controladors de la capa de persistència. El diagrama es pot trobar detallat a la carpeta DOCS.



3.1 Explicació de les classes de persistència:

3.1.1 CtrlFile

El CtrlFile és un controlador que permet llegir fitxers de text i retornar les seves linies en una llista.

- **Mètodes:**
  - **llegirArxiu(String filepath)**
    - Llegeix l'arxiu ubicat a filepath i retorna una llista de strings on cada string conté cada línia de l'arxiu.

3.2 Explicació dels controladors de persistència:

3.2.1 CtrlPersPerfil

El CtrlPersPerfil és el controlador de persitència per gestionar perfils d'usuari del sistema.

- **Atributs:**
  - **perfilActual:** Perfil. Representa el perfil de l'usuari actual
  - **perfils:** HashMap<String, Perfil>**.** Conjunt de perfils carregats a memoria.
- **Mètodes:**
  - **perfilExisteix(String nomPerfil)**
    - Comprova si existeix el perfil identificat per nomPerfil
  - **guardar()**
    - Guarda els perfils carregats a memoria en un arxiu persistent .json.
  - **carregar()**
    - Carrega els perfils guardats a l'arxiu persistent .json a memoria.
  - **canviaPerfil(String nomPerfil)**
    - canvia de perfil al perfil identificat per nomPerfil i si no existeix el crea.
  - **eliminaPerfil(String nomPerfil)**
    - Elimina el perfil identificat per nomPerfil del conjunt de perfils.
  - **afegirPerfil(String nomPerfil)**
    - Afegeix un nou Perfil amb nom nomPerfil al conjunt de perfils i el guarda com perfilActual.
  - **getPerfil(String nomPerfil)**

        - Retorna el perfil identificat per nomPerfil, si no existeix llença una excepció
    - **getAllPerfils()**
        - Obté el conjunt de noms dels perfils

3.2.2 CtrlPersFreq

El CtrlPersIdiomes és el controlador de persitència per gestionar les llistes de paraules i freqüències de l'usuari actual.

- **Atributs:**
    - **frequencies:** HashMap<String, LlistaFrequencies>. Mapa de llistes de freqüències guardades per el perfil actual
    - **usuari:** String. Nom d'usuari del perfil actual
- **Mètodes:**
    - **comprovaLlistaNoExisteix(String nomLlista)**
        - Comprova que la llista identificada per nomLlista existeix i si no ho fa llença una excepció
    - **comprovaLlistaJaExisteix(String nomLlista)**
        - Comprova que la llista identificada per nomLlista ja existeix i si ho fa llença una excepció
    - **guardar()**
        - Guarda les llistes de freqüències del perfil actual al .json de persistencia
    - **carregar()**
        - Carrega les llistes de freqüències del perfil actual del .json de persistencia
    - **canviaPerfil(String usuari)**
        - Canvia el perfil actual per un altre.
        - Guarda el conjunt de llistes del perfil actual i carrega el del perfil nou.
    - **eliminaPerfil(String nomPerfil)**
        - Elimina de la capa de persistencia les llistes del perfil identificat per nomPerfil
    - **guardarLlistaFreq(LlistaFrequencies llista)**

        - Afegeix una llista de frequencies al conjunt de llistes del perfil actual.
    - **eliminarLlista(String nomLlista)**
        - Elimina la llista de freqüències identificada per nomLlista del conjunt de llistes del perfil actual.
    - **comprovarUsIdioma(String nomIdioma)**
        - Comprova que l'idioma identificat per nomIdioma no està en ús a cap llista de freqüències, i si ho està llença una excepció.
    - **getLlistaFreq(String nomLlista)**
        - Obté la llista de freqüències identificada per nomLlista del conjunt de llistes del perfil actual i si no existeix llença una excepció.

3.2.3 CtrlPersAlfabets

El CtrlPersAlfabets és el controlador de persitència per gestionar els alfabets del sistema.

- **Atributs:**
    - **Alfabets:** TreeMap<String, Alfabet>. Mapa dels alfabets del sistema
- **Mètodes:**
    - **guardar()**
        - Guarda les dades dels alfabets del sistema, el nom dels alfabets i les lletres de cada alfabet, en un document en format JSON.
    - **carregar()**
        - Carrega del fitxer amb format JSON corresponent tots els alfabets del sistema. S'afegeixen tots els alfabets al mapa d'Alfabets.
    - **afegirAlfabet(String filename, List<String> LlistaLlegida)**
        - En cas de que no existeixi un alfabet en el sistema amb nom filename, es crea un alfabet amb nom filename i les lletres de LlistaLlegida i s'afegeix al mapa d'alfabets.
    - **lletraValida(char c)**
        - Retorna TRUE si c és una lletra sense accent, FALSE si és un número, símbol o lletra amb accent.
    - **eliminarAlfabet(String nomAlfabet)**

        - Si existeix un alfabet en el sistema amb nomAlfabet, s'elimina.
    - **existeix(String nomAlfabet)**
        - Retorna TRUE si existeix un alfabet en el sistema amb nomAlfabet, FALSE en cas contrari.
    - **getAlfabet(String nomAlfabet)**
        - Si existeix, retorna l'alfabet amb nom nomAlfabet.
    - **getAlfabets()**
        - Retorna el mapa de tots els alfabets del sistema.

3.2.4 CtrlPersIdiomes

El CtrlPersIdiomes és el controlador de persitència per gestionar els idiomes del sistema.

- **Atributs:**
    - **Idiomes:** TreeMap<String, Idioma>. Mapa dels idiomes del sistema
- **Mètodes:**
    - **guardar()**
        - Guarda les dades dels idiomes del sistema, el nom dels idiomes, el nom de l'alfabet de l'idioma i la llista de paraules amb freqüències predeterminada de l'idioma, en un document en format JSON.
    - **carregar()**
        - Carrega del fitxer amb format JSON corresponent tots els idiomes del sistema. S'afegeixen tots els idiomes al mapa d'Idiomes.
    - **afegirIdioma(String nomIdioma, Alfabet a, Map<String, Integer> novesEntrades)**
        - En cas de que no existeixi un idioma en el sistema amb nomIdioma, es crea un idioma amb nom nomIdioma, alfabet a i llista de freqüències (que es crea amb el nom LlistaPrednomIdioma amb i llista de paraules novesEntrades) i s'afegeix al mapa d'idiomes.
    - **eliminarIdioma(String nomIdioma)**
        - Si existeix un idioma en el sistema amb nomIdioma, s'elimina.
    - **existeix(String nomIdioma)**

        - Retorna TRUE si existeix un idioma en el sistema amb nomIdioma, FALSE en cas contrari.
    - **getIdioma(String nomIdioma)**
        - Si existeix, retorna l'idioma amb nom nomIdioma.
    - **getIdiomes()**
        - Retorna el mapa de tots els idiomes del sistema.

3.2.5 CtrlPersTeclats

El CtrlPersIdiomes és el controlador de persitència per gestionar els teclats de l'usuari actual.

- **Atributs:**
    - **teclats:** Map<String, Teclat>. Mapa dels teclats de l'usuari actual
    - **usuari:** String. Nom d'usuari del perfil actual
- **Mètodes:**
    - **comprovaTeclatNoExisteix(String nomTeclat)**
        - Comprova que el teclat identificat per nomTeclat existeix i si no llença una excepció
    - **comprovaTeclatJaExisteix(String nomTeclat)**
        - Comprova que el teclat identificat per nomTeclat ja existeix i si ho fa llença una excepció
    - **canviaPerfil(String usuari)**
        - Canvia el perfil actual per un altre.
        - Guarda el conjunt de teclats del perfil actual i carrega la del perfil nou.
    - **eliminaPerfil(String nomPerfil)**
        - Elimina de la capa de persistencia els teclats del perfil identificat per nomPerfil
    - **guardar()**
        - Guarda els teclats del perfil actual al .json de teclats
    - **carregar()**
        - Carrega els teclats del perfil actual del .json de teclats
    - **afegirTeclat(Teclat t)**
        - Guarda el teclat t al conjunt de teclats

- **eliminarTeclat(String nomTeclat)**
  - Elimina el teclat identificat per nomTeclat de el conjunt de teclats, si no existeix llença una excepció
- **comprovarUsIdioma(String nomIdioma)**
  - Comprova que l'idioma identificat per nomIdioma no està en ús en cap teclat i si ho està llença una excepció
- **comprovarUsLlista(String nomLlista)**
  - Comprova que la llista identificada per nomLlista no està en ús en cap teclat i si ho està llença una excepció
- **getTeclat(String nomTeclat)**
  - Obté el teclat identificat per nomTeclat i si no existeix llença una excepció

4. Descripció dels algorismes i les estructures de dades

Finalment, aprofundirem en els algorismes i estructures de dades emprades en el projecte. No obstant, els costos concrets dels mètodes estan detallats en la documentació de les classes del domini.

4.1 Branch and Bound:

L'algorisme Branch and bound és el mètode amb el qual resoldrem el nostre problema NP-complet. Per resoldre aquest problema, primer necessitem tindre per cada parell d'ubicacions la seva distància i per cada parell de lletres la seva freqüència. El primer que farem serà, per tant, guardar la distància de cada parell de posicions al layout que rebi l'algoritme, i les freqüències de cada parell de lletres segons la llista de paraules i freqüències que es rep com paràmetre, en ambdós casos en una matriu de doubles.

La nostra estratègia per fer el branching serà Eager. Anirem construint un arbre de solucions, que s'aniran emmagatzemant en una cua de prioritat segons la seva cota. Començarem fent un algoritme Greedy que ens donarà una solució inicial a partir de la qual farem el branching i el bounding per arribar a la solució òptima. Quan trobem una solució final, és a dir, amb un total de lletres usades igual a les lletres de l'abecedari, haurem encontrat la solució. En cada procés de branching afegirem una lletra encara no utilitzada, i tindrem una solució parcial nova cadascuna amb aquesta lletra en una posició no ocupada diferent. Guardarem a la cua la millor solució d'aquestes.

Principals estructures de dades:

- **double[ ][ ] Mat_dist**
    - La matriu de distàncies,per cada element ‘i’ ‘j’ de la matriu tenim la distància de la posició ‘i’ del layout a la posició ‘j’. Per tant la diagonal d’aquesta matriu serà tot 0. El tamany serà sent n el nombre de lletres de l'abecedari, de n per n, i llavors si el layout introduït té més posicions que lletres en l'abecedari, deixarem en blanc les posicions sobrants, només omplirem les primeres n posicions del layout, és dir,

durant tota l'execució del algoritme mai no accedim a aquestes darreres posicions.

- **double[ ][ ] Mat_traf**
    - La matriu de trànsit ,per cada element 'i' 'j' de la matriu tenim el trànsit de la lletra 'i' de l'abecedari a la lletra 'j'. El tamany serà sent n el nombre de lletres de l'abecedari, de n per n. Per el càlcul d'aquesta matriu recorrem la llista de paraules i les seves freqüències. Haurem de treure tots els accents i signes de cada lletra de les paraules, perquè si no fallaria al buscar el caràcter al abecedari.
- **char[ ][ ] best_sol**
    - Quan l'algoritme tingui una solució definitiva, guardarem en aquest paràmetre el layout resultant. Serà del tamany introduït pel usuari, es dir, el nombre de files i el nombre de columnes passats per paràmetre.
- **Map<Character, Integer> letra_pos:**
    - Aquesta estructura l'utilitzarem per trobar el índex de cada lletra a les matrius de distància i trànsit. L'omplirem al principi de l'execució, posant per cada lletra en ordre de l'alfabet el seu índex, és dir, per la primera lletra de l'abecedari un 0, la segona 1, la tercera 2, i així fins la última que tindrà el n-1(n es el nombre de lletres).
- **PriorityQueue<Nodo> q = new PriorityQueue<>(new NodoComparator()):**
    - És la estructura que utilitzem per emmagatzemar las solucions que anem explorant. S'ordena segons NodoComparator: segons la cota del node, de forma que sempre surten primer les solucions amb menor cota, i últimes les que tenen major cota.

4.2 Solucions parcials(Nodo) i càlcul de la cota:

Cada solució que obtenim del procés de branching es un objecte de la classe Nodo. Per cada node calcularem la seva cota segons el mètode de Gilmore-Lawler. Aquest mètode estima el cost d'una solució segons tres termes. El primer terme l'obtindrem sumant la distància per la freqüència de cada parell de lletres col·locades a la solució actual. Per tant per cada lletra utilitzada recorrem la matriu solució i sumem al total el trànsit per la distància amb les instal·lacions que hi hagin.

Sigui m el nombre de lletres encara no utilitzades:

Per el segon terme retornarem una matriu C1 de tamany m per m, on cada element ‘i’ ‘j’ es el cost que implica colocar la lletra no utilitzada ‘i’ a la posició no ocupada ‘j’, respecte les instalacions ja fetes.

Llavors, recorrem les lletres no utilitzades, i per cadascuna, recorrem totes les posicions lliures a la matriu, i ara que tenim la lletra no utilitzada ‘i’ i la posició no ocupada ‘j’, recorrem cada instal·lació feta per veure el cost distància per freqüència acumulat que suposaria afegir la lletra a aquesta posició.
Per el tercer terme retornarem una matriu C2 de m per m, on cada element ‘i’ ‘j’ es el cost que implica colocar la lletra no utilitzada ‘i’ a la posició no ocupada ‘j’, respecte les instalacions no fetes. Aquest cost ho calcularem com el producte escalar de dos vectors, T i D. T és un vector creixent del trànsit de cada lletra lliure amb les altres lletras lliures, de tamany m-1. D es un vector decreixent de les distàncias de cada posició no utilitzada amb les altres, de tamany m-1.

Llavors, recorrem les lletres no utilitzades, i per cadascuna, primer calculem el seu vector T recorrent les altres lletres lliures i després recorrem totes les posicions lliures a la matriu. Ara per cada posició lliure calculem el seu vector D recorrent les altres posicions lliures, i calculem el producte escalar de D i T, que serà l’element que guardarem a la posició ‘i’ ‘j’ de la matriu.

Per retornar la cota total, ara primer tindrem que calcular la suma de les matrius C1 i C2, i calcular la seva assignació lineal òptima que farem amb el Hungarian Algorithm, i el resultat de la cota serà la suma del terme 1 més el resultat del Hungarian Algorithm.

Principals estructures de dades:

- **char[][] layout:**
    - Cada node guarda la seva solució actual, que és una matriu de tamany n_filas per n_columnes (dades introduïdes pel usuari), amb un nombre m de lletres ubicades a m posicions, i tot la resta buida.
- **Map<Character, pos> letres_usades:**

    - Cada node guarda les lletres que ha instal·lat i les ubicacions on han sigut emplaçades, de forma que podem consultar fàcilment quines lletres ja tenim i la seva posició.
- **Double[][] Mat_C1**
    - És la matriu on guardem el càlcul de C1. Amb m igual al nombre de lletres no instal·lades, té tamany m per m.
- **Double[][] Mat_C2**
    - És la matriu on guardem el càlcul de C2. Amb m igual al nombre de lletres no instal·lades, té tamany m per m.
- **ArrayList<pos> pos_libres:**
    - És un vector de posicions lliures del layout actual. Aprofitem que hem d'accedir a les posicions lliures tant en el càlcul de C1 com de C2 per omplir aquest vector quan calculem C1 i així aprofitar-ho quan calculem C2.
- **ArrayList<Character> letras_libres:**
    - És un vector de lletres lliures del layout actual. Com amb el vector anterior, aprofitem que en C1 hem d'accedir a les lletres lliures per omplir el vector i després utilitzar lo en C2.
- **PriorityQueue<Double> T:**
    - És la estructura que utilitzarem per guardar el vector T en ordre creixent. Per després fer el producte vectorial, ho recorrem amb un iterador, i accedim a cada element ordenat de menor a major.
- **PriorityQueue<Double>D=new PriorityQueue<>(Comparator.reverseOrder());**
    - És la estructura que utilitzarem per guardar el vector D en ordre decreixent. Per després fer el producte vectorial, ho recorrem amb un iterador, i accedim a cada element ordenat de major a menor.

4.3 Greedy

L'algoritme Greedy serveix per trobar una bona cota i solució inicial per començar el algoritme de branch and bound. En aquest cas el que fem és buscar quina és la millor i la pitjor posició, i quina és la lletra més freqüent i la menys. La millor posició serà la que tingui menys distància en total a les altres, i la pitjor serà la que tingui més. Per tant el que fem és primer recórrer la matriu de trànsit i encontrar la lletra

més i menys freqüent i després el mateix per les posicions en la matriu de distàncies. Hem de tindre en compte que l'algoritme fallaria en cas de tenir una matriu en el cual la pitjor i millor posició sigui la mateixa o totes les lletres amb les mateixes freqüències. Per això fem que el primer if en els dos casos la comparació sigui només '>' i en el segon if sigui '<=', per tal de que en el cas de per exemple una matriu 2 per 2, no s'iguali la pitjor i millor posició a la mateixa.

Després retornarem un node amb layout igual a una matriu buida de n_files per n_columnes, amb la lletra més freqüent a la millor posició i la lletra menys freqüent a la pitjor posició. Aquest node ha de tenir el camp letres_usades actualitzat amb les lletres posades al Greedy i les seves posicions.

Principals estructures de dades:

- **char[][] matriz_inicial:**
    - La matriu del node que retornem, és de n-files per n-columnes (dades introduïdes pel usuari) que tindrà la solució inicial del algoritme.
- **Map<Character, pos> ini:**
    - Utilitzem aquesta estructura per inicialitzar el camp lletres_usades del node inicial amb les lletres i posicions utilitzades pel algoritme Greedy.

4.4 Hungarian Algorithm:

L'algorisme hongarès, també conegut com l'algorisme de l'assignació de Munkres, és una potent tècnica d'optimització. Va ser presentat per Harold W. Kuhn el 1955 i més endavant revisat per James Munkres el 1957. Aquest algorisme és utilitzat per resoldre problemes d'assignació en temps $O(n^3)$.

Mitjançant una sèrie d'iteracions, l'algorisme hongarès redueix una matriu de costos, identifica zeros independents i ajusta les assignacions per trobar la millor combinació possible entre elements de dues llistes. Aquest procés permet trobar la correspondència òptima entre els elements, essent una eina valuosa en la resolució de problemes d'assignació amb diverses aplicacions pràctiques.

En aquest projecte, el farem servir per a calcular l'assignació òptima de les posicions del teclat a un conjunt determinat de lletres. Donada una matriu quadrada de costs, calcularà el cost mínim d'assignar cada lletra a una posició concreta del teclat, de tal manera que totes tinguin una posició assignada i la suma dels costs sigui la mínima possible.

Principals estructures de dades:

- **double[ ][ ] matriu:**
    - Aquesta és la matriu inicial que conté els costos o valors associats a l'assignació d'elements. En el context d'aquest algorisme, representa els costos d'assignar elements d'una llista a una altra.
- **double[ ][ ]copiamatriu:**
    - És una còpia de la matriu original. Es crea per preservar els valors originals mentre es realitzen les operacions d'optimització. Aquesta còpia és útil per a la resta de càlculs i per mantenir els valors inicials intactes.
- **int[ ] zeroFila i int[ ]zeroColumna**:
    - Aquestes llistes indiquen les posicions dels zeros marcats a la matriu. Cada posició d'aquests vectors correspon a una fila o columna, i si una fila o columna té un zero marcat, aquestes llistes enregistren la seva posició.
- **int [ ] filaCoberta i int[ ] columnaCoberta**:
    - Aquestes llistes es fan servir per determinar quines files i columnes estan cobertes durant els càlculs. Si una fila o columna està coberta, el seu valor corresponent en aquests vectors serà diferent de zero.
- **int [ ] zerosEstrellaEnFila:**
    - Aquest vector emmagatzema informació sobre els zeros marcats ('0*') a cada fila específica de la matriu. Cada posició d'aquesta llista està associada a una fila i guarda la columna on es troba el zero marcat.
- **Set<Int[ ]> K:**

- L'estructura de dades "K" és un conjunt de coordenades que s'utilitza durant el pas 5 de l'algorisme d'assignació hongarès. Aquesta estructura emmagatzema les posicions dels zeros i zeros marcats trobats seguint un patró específic a través de la matriu de costos. Funciona com a guia per eliminar zeros marcats innecessaris, actualitzar les assignacions vàlides i modificar les estructures de control, permetent una manipulació eficient de la matriu per trobar la millor assignació entre els elements de les llistes.

4.5 Genetic Algorithm i selecció ruleta:

El Genetic Algorithm és el segon algoritme que utilitzarem per resoldre aquest problema NP-complet. Aquest algoritme està inspirat en el procés de selecció natural que ocurreix a la natura. Per tant partirem d'una població inicial aleatòria menys bona (amb pitjor fitness o cota) i mitjançant processos com l'encreuament o la mutació anirem arribant a poblacions més bones, amb mijor fitness, generació a generació. Per fer aquests processos d'encreuament i mutació necessitarem també fer un procés de selecció i de reemplaçament.

La nostra població serà sempre de 4 vegades el nombre de lletres de l'abecedari, i partirem d'una població inicial totalment aleatòria. Farem un total de 100 generacions, per tant farem una iteració d'un bucle per cada generació. Per cada iteració, hem declarat un nombre m de iteracions que correspon a un quart del nombre de la població (per tant equival al nombre n de lletres de l'abecedari), i farem un bucle de m iteracions. Per cada iteració d'aquest bucle intern, seleccionarem 2 membres de la població diferents que anomenarem pares, i els creuarem 2 vegades (en ordre diferent) per obtenir 2 fills, que guardarem a un Treeset (auxiliar). Per tant al finalitzar aquest bucle intern tindrem un total de 2 * m fills, guardats a un Treeset. Llavors el que hem fet és reemplaçar els m pitjors nodes de la població per els m millors nodes fills creats al bucle anterior.

Després, amb un factor de probabilitat del 10%, aplicarem per tots els membres de la població excepte per el millor una mutació. Quan un membre es selecciona per ser mutat, s'esborra de la població i s'afegeix el membre mutat a un

Treeset(auxiliar2). Una vegada ha acabat el procés de mutació, s'afegeixen els membres mutats del Treeset(auxiliar2) a la població.
Llavors una vegada finalitzades les 100 generacions, escogirem el primer membre del Treeset població, per tant, el membre amb major fitness, és dir, el millor trobat, i retornarem la seva disposició en una matriu de characters.

Per el procés de selecció dels pares per l'encreuament, hem utilitzat la selecció per ruleta. Aquest mètode consisteix en que es seleccionarà mitjançant l'atzar un membre de la població, amb més probabilitats de ser seleccionats els nodes amb un fitness major i menys probabilitats els que sigui menor. Per fer això primer sumem la suma total dels fitness de tota la població, i després obtenim un nombre aleatori de 0 fins aquest valor de la suma. Després anirem iterant membres de la població en ordre de major a pitjor fitness, i anirem sumant la suma parcial dels seus fitness. Quan la suma parcial sigui major o igual al nombre aleatori, retornarem el node actual.

Principals estructures de dades:

- **private double[][] Mat_dist;**
    - La matriu de distàncies,per cada element 'i' 'j' de la matriu tenim la distància de la posició 'i' del layout a la posició 'j'. Per tant la diagonal d'aquesta matriu serà tot 0. El tamany serà sent n el nombre de lletres de l'abecedari, de n per n, i llavors si el layout introduït té més posicions que lletres en l'abecedari, deixarem en blanc les posicions sobrants, només omplirem les primeres n posicions del layout, és dir, durant tota l'execució del algoritme mai no accedim a aquestes darreres posicions.
- **private double[][] Mat_traf;**
    - La matriu de trànsit ,per cada element 'i' 'j' de la matriu tenim el trànsit de la lletra 'i' de l'abecedari a la lletra 'j'. El tamany serà sent n el nombre de lletres de l'abecedari, de n per n. Per el càlcul d'aquesta matriu recorrem la llista de paraules i les seves freqüències. Haurem de treure tots els accents i signes de cada lletra de les paraules, perquè si no fallaria al buscar el caràcter al abecedari.

- **private char[][] best_sol;**
    - Quan l'algoritme tingui una solució definitiva, guardarem en aquest paràmetre el layout resultant. Serà del tamany introduït pel usuari, es dir, el nombre de files i el nombre de columnes passats per paràmetre.
- **TreeSet<Nodo_genetic> poblacion;**
    - En aquesta estructura guardarem els nodes de la població. S'ordena segons NodoGeneticComparator: segons el fitness del node, de forma que s'ordenen primers els membres amb major fitness, i últims els que tenen pitjor fitness. Amb aquesta estructura, podem accedir als millors nodes amb un iterador convencional, i també accedir als pitjors(els darrers) mitjançant un iterador descendent. A més, també utilitzarem el mètode addall per afegir tots els elements de un Treeset a un altra (amb auxiliar2). El seu tamany és de, sent n el nombre de lletres, 4*n.
- **TreeSet<Nodo_genetic> auxiliar = new TreeSet<>(new Nodo_genetic_Comparator());**
    - Utilitzarem aquest Treeset per enmagatzemar els nodes, membres de la població, obtinguts del procés de encreuament, és dir, els fills. També està ordenat segons nodoGeneticComparator. Serà de tamany 2* per el nombre n de lletres.
- **TreeSet<Nodo_genetic> auxiliar2 = new TreeSet<>(new Nodo_genetic_Comparator());**
    - Utilitzarem aquest Treeset per enmagatzemar els nodes, membres de la població, obtinguts del procés de mutació. També està ordenat segons nodoGeneticComparator. No sabem el seu tamany degut a que és aleatori, peró serà com a molt, sent n el nombre de lletres, 4*n - 1.

4.6 Membres de la població (nodo_genetic)

Cadascun dels membres de la població és un objecte de la classe Nodo_genetic. Per cadascun calcularem el seu fitness, amb el mètode que vam utilitzar per el primer terme de la cota del algoritme anterior, només que ara les solucions sempre son completes. Per tant, l'obtindrem sumant la distància per la freqüència de cada parell de lletres a la distribució actual. Per tant per cada posició recorrem la resta i sumem al total el trànsit per la distància de tota la resta de posicions.

4.6.1 Mutació

Per el procés de mutació el que farem es intercanviar 2 posicions de la disposició actual del teclat. Per tant, el que farem primer és seleccionar aleatòriament 2 posicions diferents del teclat, i esborrarem de la disposició actual les dues posicions(tant al mapa disposicio com al mapa pos_letra), i després afegirem la primera posició amb la lletra que contenia la segona posició, i la segona posició amb la lletra que contenia la primera posició.

4.6.2 Encreuament

Per el procés d'encreuament, el que farem primer es seleccionar 2 punts de tall, amb el primer menor al segon. Per obtenir fills més variats el que hem fet és que els 2 punts de tall s'obtinguin de forma aleatòria amb un nombre de 0 a n-1, amb n igual al nombre de lletres. Després, guardarem cada lletra amb cada posició que tingui el primer pare del punt de tall 1 al punt de tall 2, ambdós inclosos.

Després el que farem es recórrer a partir del punt 2 fins al final, i després desde la posició 0 fins el punt 1 el segon pare. El que farem és per cada posició en ordre del punt 2 al final, i seguit de 0 fins al punt 1, anar guardant cada lletra que trobem en el recorregut anterior en el pare2 que encara no hem guardat. D'aquesta forma no guardarem lletres repetides i farem una fusió del contingut que té el pare 1 entre els dos punts de tall i la resta de lletres al segon pare.

Principals estructures de dades:

- **public Map<Character, Integer> disposicio;**
    - En aquesta estructura guardem per cada lletra la seva posició al teclat de 0 a n -1 (sent n el nombre de lletres de l'abecedari), de forma que podem obtenir la posició a partir d'una lletra. El seu tamany serà de n, sent n el nombre de lletres de l'abecedari.
- **public Map<Integer, Character> pos_letra;**
    - En aquesta estructura guardem per cada posició de 0 a n -1 (sent n el nombre de lletres de l'abecedari) la lletra que conté al teclat, de forma que podem obtenir a partir d'una posició la lletra que conté. El seu tamany serà de n, sent n el nombre de lletres de l'abecedari.
- **List<Character> lista_random = new ArrayList<>(lletres);**

    - Estructura que utilitzarem a l'hora de crear les disposicions aleatòries per la població inicial, degut a que amb el mètode collections.shuffle podem desordenar les lletres de l'abecedari i utilitzar aquesta llista per obtindre disposicions aleatòries. El seu tamany serà, sent n el nombre de lletres de l'abecedari, de n.
- **Map<Character, Integer> disposicio_fill = new HashMap<>();**
    - Aquesta estructura l'utilitzem en l'encreuament per guardar cada lletra amb la seva posició del fill. Ens serveix també per comprovar en el recorregut si una lletra del segon pare l'hem guardada ja o no. Tindrà tamany n, sent n el nombre de lletres.
- **Map<Integer, Character> pos_letra_fill = new HashMap<>();**
    - Aquesta estructura l'utilitzem per guardar per cada posició la lletra que conté del fill. Tindrà tamany n, sent n el nombre de lletres.
- **Queue<Integer> cola = new LinkedList<>();**
    - Utilitzem aquesta cua per guardar les posicions que hem d'omplir encara en el fill. Primer guardarem les posicions del segon punt de tall fins al final, i després les de 0 fins al primer punt de tall. Quan trobem una lletra no guardada al recorregut, farem poll de la cua per obtenir i lliurar aquesta posició. D'aquesta forma, si trobem una lletra ja guardada, la primera posició de la cua seguirà sent la mateixa, i no es farà poll fins trobar una lletra no guardada. El seu tamany serà, sent n el nombre de lletres, de n - (punt de tall 2 - punt de tall 1 + 1).

4.7 Altres estructures de dades rellevants del sistema:

També hi ha altres estructures de dades rellevants que s'usen en diferents classes del sistema (la majoria, dins dels controladors de persistència de la capa de Dades), i són les que s'esmenten a continuació:

- **TreeMap<String, Alfabet> Alfabets**
    - És un map on la clau és el nom d'un alfabet en minúscula i el valor és l'alfabet en qüestió
- **TreeMap<String, Idioma> Idiomes**
    - És un map on la clau és el nom d'un idioma i el valor és l'idioma en qüestió
- **HashMap<String, LlistaFrequencies> frequencies**

    - És un map on la clau és el nom d'una llista de freqüències i el valor és la llista de freqüències en qüestió, utilitzat a Perfil i a CtrlPersFreq
- **HashMap<String, Teclat> teclats**
    - És un map on la clau és el nom del teclat i el valor és el teclat en qüestió, utilitzat a Perfil i a CtrlPersTeclats
- **HashMap<String, Perfil> PerfilsActius**
    - És un map on la clau és el nom d'un perfil i el valor és el perfil en qüestió.

Per tal d'emmagatzemar les dades del nostre sistema hem decidit utilitzar Maps, ja que en tots els casos ens interessa buscar els objectes de les classes per la seva clau primària, i els maps ofereixen búsquedes eficients per les seves claus primàries. En els casos dels TreeMap les operacions bàsiques (inserció, búsqueda ii eliminació) tenen un rendiment de O(log n), on n és el nombre d'elements del mapa i aquest permet mantenir un ordre ascendent dels objectes a partir de les seves claus. Per altra banda, en els HashMap les operacions bàsiques tenen de promig, un rendiment constant O(1) i no garantitzen cap ordre específic dels elements.