Generador de teclats

Grupo 14.3

Participantes:

Alèxia Mayor Giménez

Víctor Moreno Villanueva

Júlia Tena Domingo

Oriol Ramos Puig

Versió del lliurament: 2.0

2ª Entrega PROP

Índice

1. DIAGRAMA DE LAS CLASES Y CONTROLADORES DE LA CAPA DE PRESENTACIÓN

A continuación mostramos el diseño del diagrama de clases y controladores de la capa de presentación. El diagrama se puede encontrar más detallado en la carpeta DOCS.

Figura 1: Diagrama de capa presentación

1.1 Explicación de las clases de presentación

1.1.1 VistaMenuPrincipal

Esta vista es la encargada de mostrar la primera pantalla que aparece al ejecutar nuestro sistema que consiste en un menú con un texto con el título del programa y 5 botones, 4 de los cuáles tienen cada uno una lista desplegable con dos elementos, que nos llevan al resto de vistas donde se puede agregar, borrar, consultar y modificar alfabetos, textos, asociaciones y teclados. El quinto botón sirve para salir y cerrar el programa.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVista: Etiqueta que muestra el título de la ventana, en este caso, "Generador de teclados PROP".
- JPopupMenu popuMenuA: Menú emergente/desplegable que incluye las opciones de funcionalidades que se pueden hacer con el elemento alfabeto.
- JPopupMenu popuMenuTxt: Menú emergente/desplegable que incluye las opciones de funcionalidades que se pueden hacer con el elemento texto.
- JPopupMenu popuMenuAT: Menú emergente/desplegable que incluye las opciones de funcionalidades que se pueden hacer con el elemento asociación de textos.
- JPopupMenu popuMenuT: Menú emergente/desplegable que incluye las opciones de funcionalidades que se pueden hacer con el elemento teclado.
- JMenuItem agregarA: Ítem del menú desplegable popuMenuA para agregar un nuevo alfabeto
- JMenuItem borrarA: Ítem del menú desplegable popuMenuA para borrar un alfabeto existente
- JMenuItem consultarA: Ítem del menú desplegable popuMenuA para consultar el contenido de un alfabeto existente.
- JMenuItem modificarA: Ítem del menú desplegable popuMenuA para modificar el contenido de un alfabeto existente.
- JMenuItem agregarTxt: Ítem del menú desplegable popuMenuTxt para agregar un nuevo texto.
- JMenuItem borrarTxt: Ítem del menú desplegable popuMenuTxt para borrar un texto existente.
- JMenuItem consultarTxt: Ítem del menú desplegable popuMenuTxt para consultar el contenido de un texto existente.
- JMenuItem modificarTxt: Ítem del menú desplegable popuMenuTxt para modificar el contenido de un texto existente.
- JMenuItem agregarAT: Ítem del menú desplegable popuMenuAT para agregar una nueva asociación de textos.

- JMenuItem borrarAT: Ítem del menú desplegable popuMenuAT para borrar una asociación de textos existente.
- JMenuItem consultarAT: Ítem del menú desplegable popuMenuAT para consultar la lista de textos que forman una asociación de textos existente.
- JMenuItem modificarAT: Ítem del menú desplegable popuMenuAT para modificar la lista de textos que forman una asociación de textos existente.
- JMenuItem agregarT: Ítem del menú desplegable popuMenuT para agregar un nuevo teclado.
- JMenuItem borrarT: Ítem del menú desplegable popuMenuT para borrar un teclado existente
- JMenuItem consultarT: Ítem del menú desplegable popuMenuT para consultar el contenido de un teclado existente.
- JMenuItem modificarT: Ítem del menú desplegable popuMenuT para modificar el contenido de un teclado existente.
- JButton bAlfabeto: Botón que el usuario puede presionar para ver las opciones de funcionalidades que puede elegir para hacer con el elemento alfabeto.
- JButton bTexto: Botón que el usuario puede presionar para ver las opciones de funcionalidades que puede elegir para hacer con el elemento texto.
- JButton bAsociacion: Botón que el usuario puede presionar para ver las opciones de funcionalidades que puede elegir para hacer con el elemento asociación de textos.
- JButton bTeclado: Botón que el usuario puede presionar para ver las opciones de funcionalidades que puede elegir para hacer con el elemento teclado.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y cerrar la ventana.

Métodos:

- VistaMenuPrincipal(): Constructora de la ventana del menú principal. Ofrece 4 botones distintos, cada uno hace referencia a un elemento: alfabeto, texto, asociación de textos y teclado.
- Se inicializan los componentes de la interfaz de usuario, se establece el diseño y las acciones de los botones, y se configura el comportamiento general de la vista.
- Dentro del propio constructor, se añaden ActionListeners a los botones para que cuando se clique uno se despliegue un menú con las posibles opciones de funcionalidades que se pueden hacer con ese elemento.
- Se configura el comportamiento de cambiar de ventana y también el de cerrar la ventana.

1.1.2 VistaAlfabetoA

Esta vista es la encargada de agregar un alfabeto. Contiene todo lo necesario para agregar. Se indican 3 campos: nombre, contenido y path, los cuáles son los parámetros necesarios para crear un alfabeto. Podemos crear mediante nombre y contenido o mediante nombre y path. Hay un botón disponible (agregar) para confirmar la acción.

Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaAA: Etiqueta que muestra el título de la ventana, en este caso, "Agregar alfabeto".
- JButton bAgregarAlfabeto: Botón que el usuario puede presionar para iniciar el proceso de agregar un nuevo alfabeto.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreAA: Etiqueta que indica al usuario dónde ingresar el nombre del alfabeto.
- JTextArea areaNomAA: Área de texto donde el usuario puede escribir el nombre del nuevo alfabeto.
- JLabel txtContenidoAA: Etiqueta que indica al usuario dónde ingresar el contenido del alfabeto.
- JTextArea areaContenidoAA: Área de texto donde el usuario puede escribir el contenido del nuevo alfabeto.
- JLabel txtPathAA: Etiqueta que indica al usuario dónde ingresar el path del archivo del alfabeto (si se desea cargar desde un archivo).
- JTextArea areaPathAA: Área de texto donde el usuario puede escribir el path del archivo del alfabeto.

Métodos:

- VistaAlfabetoA(): Constructor de la clase VistaAlfabetoA. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para agregar un nuevo alfabeto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para crear el alfabeto.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.3 VistaAlfabetoB

Esta vista es la encargada de borrar un alfabeto. Tiene un desplegable con todos los nombres de alfabetos para que el usuario pueda seleccionar uno a eliminar. Al seleccionarlo se escribe el nombre del alfabeto seleccionado en un área de texto. Tiene un botón para eliminar alfabeto y otro para cancelar.

Atributos:

- JFrame frame: Marco principal utilizado como ventana para las interfaces gráficas relacionadas con la eliminación de un alfabeto.
- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaAB: Etiqueta que muestra el título de la ventana, en este caso, "Borrar alfabeto".
- JButton bBorrarAlfabeto: Botón que el usuario puede presionar para iniciar el proceso de borrar un nuevo alfabeto.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtDesplegableAB: Etiqueta que indica al usuario dónde está la lista de nombres de alfabetos.
- JComboBox<String> nombresAB : Desplegable que contiene todos los nombres de los alfabetos del conjunto de alfabetos.
- JLabel txtNombreAB: Etiqueta que indica al usuario dónde ingresar el nombre del alfabeto.
- JTextArea areaNomAB: Área de texto donde el usuario puede escribir el nombre del nuevo alfabeto.

Métodos:

- VistaAlfabetoB(): Constructor de la clase VistaAlfabetoB. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para borrar un alfabeto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para borrar el alfabeto.
- Se despliega una lista con los alfabetos existentes para que se pueda seleccionar uno para borrar.
- Si la eliminación es exitosa, se muestra un diálogo informando al usuario que el alfabeto se ha borrado correctamente.

- Si el alfabeto no se puede eliminar (por ejemplo, si no existe), se muestra un diálogo de error.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.4 VistaAlfabetoC

Esta vista es la encargada de consultar un alfabeto. Contiene todo lo necesario para consultar. Tenemos un desplegable dónde salen todos los nombres de los alfabetos, para que el usuario pueda seleccionar uno. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido del alfabeto seleccionado. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaAC: Etiqueta que muestra el título de la parte superior de la ventana, "Consultar alfabeto".
- JLabel txtDesplegableAC: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String> nombresAC: Desplegable que contiene la lista de nombres de los alfabetos, permite seleccionar uno de ellos.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreAC: Etiqueta que indica al usuario dónde se mostrará el nombre del alfabeto consultado.
- JTextArea areanomAC: Área de texto donde se muestra el nombre del alfabeto consultado. Está configurada para ser no editable.
- JLabel txtContenidoAC: Etiqueta que indica al usuario dónde se mostrará el contenido del alfabeto consultado.
- JTextArea areacontenidoAC: Área de texto donde se muestra el contenido del alfabeto consultado. Está configurada para ser no editable y contiene un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaAlfabetoC(): Constructor de la clase VistaAlfabetoC. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.

- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para consultar un alfabeto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para consultar el alfabeto.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.5 VistaAlfabetoM

Esta vista es la encargada de modificar un alfabeto. Contiene todo lo necesario para modificar. Tenemos un desplegable dónde salen todos los nombres de los alfabetos, para que el usuario pueda seleccionar uno. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido del alfabeto seleccionado. Estos campos se pueden modificar y se guarda la modificación del alfabeto. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que contiene los elementos de la interfaz de usuario.
- JLabel tituloVistaAM: Etiqueta que muestra el título de la ventana, en este caso, "Modificar alfabeto".
- JLabel txtDesplegableAM: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String> nombresAM: Desplegable que contiene la lista de nombres de los alfabetos, permite seleccionar uno de ellos.
- JButton bModificarAlfabeto: Botón que, al ser presionado, inicia el proceso de modificar el alfabeto seleccionado.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreAM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el nombre del alfabeto.
- JTextArea areanomAM: Área de texto donde se muestra o se puede modificar el nombre del alfabeto.
- JLabel txtContenidoAM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el contenido del alfabeto.
- JTextArea areacontenidoAM: Área de texto donde se muestra o se puede modificar el contenido del alfabeto. Incluye un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaAlfabetoM(): Constructor de la clase VistaAlfabetoM. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.

- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para modificar un alfabeto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para modificar el alfabeto.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.6 VistaTextoA

Esta vista es la encargada de agregar un texto. Contiene todo lo necesario para agregar, ya sea mediante palabras o mediante frecuencias. Se indican 3 campos en la izquierda y 3 en la derecha: nombre, contenido y path, los cuáles son los parámetros necesarios para crear un texto, en la izquierda mediante palabras y en la derecha mediante frecuencias. Hay un botón disponible (agregar) tanto para palabras cómo para frecuencias para confirmar la acción.

Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaTxtA: Etiqueta que muestra el título de la parte izquierda de la ventana, en este caso, "Agregar texto de palabras".
- JLabel tituloVistaTxtA1: Etiqueta que muestra el título de la parte derecha de la ventana, en este caso, "Agregar texto de frecuencias".
- JButton bAgregarTexto: Botón que el usuario puede presionar para iniciar el proceso de agregar un nuevo texto con el formato de palabras.
- JButton bAgregarTextoFrec: Botón que el usuario puede presionar para iniciar el proceso de agregar un nuevo texto con el formato de frecuencias.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.

Introducir un texto en formato texto de palabras

- JLabel txtNombreTxtA: Etiqueta que indica al usuario dónde ingresar el nombre del texto.
- JTextArea areaNomTxtA: Área de texto donde el usuario puede escribir el nombre del nuevo texto.
- JLabel txtContenidoTxtA: Etiqueta que indica al usuario dónde ingresar el contenido del texto.
- JTextArea areaContenidoTxtA: Área de texto donde el usuario puede escribir el contenido del nuevo texto (si se desea escribir el contenido manualmente).

- JLabel txtPathTxtA: Etiqueta que indica al usuario dónde ingresar el path del archivo del texto.
- JTextArea areaPathTxtA: Área de texto donde el usuario puede escribir el path del archivo del texto (si se desea cargar desde un archivo).
- JLabel txtInstruccionesTxtA: Etiqueta que indica al usuario el formato en el cuál hay que entrar el contenido del texto a agregar.

Introducir un texto en formato texto de frecuencias

- JLabel txtNombreTxtA1: Etiqueta que indica al usuario dónde ingresar el nombre del texto.
- JTextArea areaNomTxtA1: Área de texto donde el usuario puede escribir el nombre del nuevo texto.
- JLabel txtContenidoTxtA1: Etiqueta que indica al usuario dónde ingresar el contenido del texto.
- JTextArea areaContenidoTxtA1: Área de texto donde el usuario puede escribir el contenido del nuevo texto (si se desea escribir el contenido manualmente).
- JLabel txtPathTxtA1: Etiqueta que indica al usuario dónde ingresar el path del archivo del texto.
- JTextArea areaPathTxtA1: Área de texto donde el usuario puede escribir el path del archivo del texto (si se desea cargar desde un archivo).
- JLabel txtInstruccionesTxtA1: Etiqueta que indica al usuario el formato en el cuál hay que entrar el contenido del texto a agregar.
- JLabel txtInstruccionesTxtA2: Etiqueta que indica al usuario el formato en el cuál hay que entrar el contenido del texto a agregar.

Métodos:

- VistaTextoA(): Constructor de la clase VistaTextoA. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Además se divide la ventana en dos partes, parte izquierda destinada a agregar un texto con el formato palabras y parte derecha para agregar un texto con el formato frecuencias.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario. Además de pequeños textos informativos de las normas de formato que debe tener el contenido del texto a agregar.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para agregar un nuevo texto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para crear el texto.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.7 VistaTextoB

Esta vista es la encargada de borrar un texto. Se abre la carpeta donde se guardan todos los textos para que el usuario pueda seleccionar uno a eliminar. Esta carpeta tiene un botón para confirmar y otro para cancelar,

Atributos:

- JFrame frame: Marco principal utilizado como ventana para las interfaces gráficas relacionadas con la eliminación de un texto.
- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaTxtB: Etiqueta que muestra el título de la ventana, en este caso, "Borrar texto".
- JButton bBorrarTexto: Botón que el usuario puede presionar para iniciar el proceso de borrar un nuevo texto.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtDesplegableTxtB: Etiqueta que indica al usuario dónde está la lista de nombres de textos.
- JComboBox<String> nombresTxtB : Desplegable que contiene todos los nombres de los textos del conjunto de textos.
- JLabel txtNombreTxtB: Etiqueta que indica al usuario dónde ingresar el nombre del texto.
- JTextArea areaNomTxtB: Área de texto donde el usuario puede escribir el nombre del nuevo texto.

Métodos:

- VistaTextoB(): Constructor de la clase VistaTextoB. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para borrar un texto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para borrar el texto.
- Se despliega una lista con los textos existentes para que se pueda seleccionar uno para borrar.
- Si la eliminación es exitosa, se muestra un diálogo informando al usuario que el texto se ha borrado correctamente.
- Si el texto no se puede eliminar (por ejemplo, si no existe), se muestra un diálogo de error.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.8 VistaTextoC

Esta vista es la encargada de consultar un texto. Contiene todo lo necesario para consultar. Tenemos un desplegable dónde salen todos los nombres de los textos, para que el usuario pueda seleccionar uno. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido del texto seleccionado. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaTxtC: Etiqueta que muestra el título de la parte superior de la ventana, "Consultar texto".
- JLabel txtDesplegableTxtC: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String> nombresTxtC: Desplegable que contiene la lista de nombres de los textos, permite seleccionar uno de ellos.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreTxtC: Etiqueta que indica al usuario dónde se mostrará el nombre del texto consultado.
- JTextArea areanomTxtC: Área de texto donde se muestra el nombre del texto consultado. Está configurada para ser no editable.
- JLabel txtContenidoTxtC: Etiqueta que indica al usuario dónde se mostrará el contenido del texto consultado.
- JTextArea areacontenidoTxtC: Área de texto donde se muestra el contenido del texto consultado. Está configurada para ser no editable y contiene un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaTextoC(): Constructor de la clase VistaTextoC. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para consultar un texto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para consultar el texto .
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.9 VistaTextoM

Esta vista es la encargada de modificar un texto. Contiene todo lo necesario para modificar. Tenemos un desplegable dónde salen todos los nombres de los textos, para que el usuario pueda seleccionar uno. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido del texto seleccionado. Estos campos se pueden modificar y se guarda la modificación del texto. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que contiene los elementos de la interfaz de usuario.
- JLabel tituloVistaTxtM: Etiqueta que muestra el título de la ventana, en este caso, "Modificar texto".
- JLabel txtDesplegableTxtM: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String>nombresTxtM: Desplegable que contiene la lista de nombres de los textos, permite seleccionar uno de ellos.
- JButton bModificarTexto: Botón que, al ser presionado, inicia el proceso de modificar el texto seleccionado.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreTxtM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el nombre del texto.
- JTextArea areanomTxtM: Área de texto donde se muestra o se puede modificar el nombre del texto.
- JLabel txtContenidoTxtM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el contenido del texto.
- JTextArea areacontenidoTxtM: Área de texto donde se muestra o se puede modificar el contenido del texto. Incluye un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaTextoM(): Constructor de la clase VistaTextoM. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para modificar un texto o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para modificar el texto.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.10 VistaAsociacionTextosA

Esta vista es la encargada de crear una asociación de textos. Contiene todo lo necesario para crearla. Se indican 2 campos: nombre, textos a agregar. El primero es necesario para crear una asociación y el segundo se utiliza para añadir los textos que el usuario desee a la lista de textos que contiene esa asociación.

Primero hay un botón para confirmar la creación de la asociación con el nombre escrito. Por otro lado, con el botón agregar texto asociación se abre una ventana correspondiente con todos los nombres de textos para seleccionar. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JFileChooser chooser: Ventana de selección del archivo que se quiere cargar.
- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaATA: Etiqueta que muestra el título de la parte izquierda de la ventana, en este caso, "Agregar asociación de textos".
- JButton bAgregarTexto: Botón que el usuario puede presionar para que le aparezca la ventana de selección de textos y pueda elegir uno.
- JButton bAgregarAsociacion: Botón que el usuario puede presionar para iniciar el proceso de crear una nueva asociación de textos.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreATA: Etiqueta que indica al usuario dónde ingresar el nombre de la asociación.
- JTextArea areanomATA: Área de texto donde el usuario puede escribir el nombre de la nueva asociación de textos.
- JLabel txtTextosAgregarATA: Etiqueta que indica al usuario dónde ingresar el nombre del texto que se quiere agregar a la asociación.
- JTextArea areaTextosAgregarATA: Área de texto donde quedará seleccionado el archivo que el usuario haya elegido de la ventana de selección de textos.
- JFrame NomFrame: Marco utilizado para mostrar un mensaje de error en caso de que el usuario intente agregar una asociación sin proporcionar un nombre.

Métodos:

- VistaAsociacionTextosA(): Constructor de la clase VistaAsociacionTextosA. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.

- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para agregar una nueva asociación de textos o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para crear la asociación.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.11 VistaAsociacionTextosB

Esta vista es la encargada de borrar una asociación de textos. Se abre la carpeta donde se guardan todas las asociaciones para que el usuario pueda seleccionar una a eliminar. Esta carpeta tiene un botón para confirmar y otro para cancelar,

Atributos:

- JFrame frame: Marco principal utilizado como ventana para las interfaces gráficas relacionadas con la eliminación de una asociación de textos.
- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaATB: Etiqueta que muestra el título de la ventana, en este caso, "Borrar asociación".
- JButton bBorrarAsociacion: Botón que el usuario puede presionar para iniciar el proceso de borrar una nueva asociación.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtDesplegableATB: Etiqueta que indica al usuario dónde está la lista de nombres de asociaciones.
- JComboBox<String> nombresATB : Desplegable que contiene todos los nombres de las asociaciones del conjunto de asociaciones.
- JLabel txtNombreATB: Etiqueta que indica al usuario dónde ingresar el nombre de la asociación.
- JTextArea areaNomATB: Área de texto donde el usuario puede escribir el nombre de la nueva asociación.

Métodos:

- VistaAsociacionTextosB(): Constructor de la clase VistaAsociacionTextosB. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para borrar una asociación o salir de la vista.

- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para borrar la asociación.
- Se despliega una lista con las asociaciones existentes para que se pueda seleccionar una para borrar.
- Si la eliminación es exitosa, se muestra un diálogo informando al usuario que la asociación se ha borrado correctamente.
- Si la asociación no se puede eliminar (por ejemplo, si no existe), se muestra un diálogo de error.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.12 VistaAsociacionTextosC

Esta vista es la encargada de consultar una asociación. Contiene todo lo necesario para consultar. Tenemos un desplegable dónde salen todos los nombres de las asociaciones , para que el usuario pueda seleccionar una. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido de la asociación seleccionada. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaATC: Etiqueta que muestra el título de la parte superior de la ventana, "Consultar asociación".
- JLabel txtDesplegableATC: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String>nombresATC: Desplegable que contiene la lista de nombres de las asociaciones, permite seleccionar uno de ellos.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreATC: Etiqueta que indica al usuario dónde se mostrará el nombre de la asociación consultada.
- JTextArea areanomATC: Área de texto donde se muestra el nombre de la asociación consultada. Está configurada para ser no editable.
- JLabel txtContenidoATC: Etiqueta que indica al usuario dónde se mostrará el contenido de la asociación consultada.
- JTextArea areacontenidoATC: Área de texto donde se muestra el contenido de la asociación consultada. Está configurada para ser no editable y contiene un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaAsociacionTextosC(): Constructor de la clase VistaAsociacionTextosC. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para consultar una asociación o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para consultar la asociación.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.13 VistaAsociacionTextosM

Esta vista es la encargada de modificar una asociación. Contiene todo lo necesario para modificar. Tenemos un desplegable dónde salen todos los nombres de las asociaciones, para que el usuario pueda seleccionar una. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido de la asociación seleccionada, siendo el contenido el nombre de los textos asociados. Estos campos se pueden modificar y se guarda la modificación de la asociación. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que contiene los elementos de la interfaz de usuario.
- JLabel tituloVistaATM: Etiqueta que muestra el título de la ventana, en este caso, "Modificar asociación".
- JLabel txtDesplegableATM: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String>nombresATM: Desplegable que contiene la lista de nombres de las asociaciones, permite seleccionar una de ellas.
- JButton bModificarAsociación: Botón que, al ser presionado, inicia el proceso de modificar la asociación seleccionada.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreATM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el nombre de la asociación.
- JTextArea areanomATM: Área de texto donde se muestra o se puede modificar el nombre de la asociación.

- JLabel txtContenidoATM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el contenido de la asociación.
- JTextArea areacontenidoATM: Área de texto donde se muestra o se puede modificar el contenido de la asociación. Incluye un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaAsociacionTextosM(): Constructor de la clase VistaAsociacionTextosM. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para modificar una asociación o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para modificar la asociación.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.14 VistaTecladoA

Esta vista es la encargada de crear un teclado. Contiene todo lo necesario para crearlo. Se indican 3 campos: nombre, alfabeto y asociación, los cuáles son los parámetros necesarios para crear un teclado. Con los botones seleccionar alfabeto y seleccionar asociación se abren ventanas correspondientes con todos los nombres de alfabetos y asociaciones para seleccionar uno de cada. Alternativamente podemos escribir directamente en el área de texto. Hay también un botón para confirmar la creación del teclado. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JFileChooser chooser: Ventana de selección del archivo que se quiere cargar.
- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaTA: Etiqueta que muestra el título de la parte izquierda de la ventana, en este caso, "Agregar teclado".
- JButton bAgregarTecladoBB: Botón que el usuario puede presionar para iniciar el proceso de crear un nuevo teclado mediante el algoritmo Branch & Bound.
- JButton bAgregarTecladoSA: Botón que el usuario puede presionar para iniciar el proceso de crear un nuevo teclado mediante el algoritmo Simulated Annealing.
- JButton bSeleccionarAlfabeto: Botón que el usuario puede presionar para que le aparezca la ventana de selección de alfabetos y pueda elegir uno.

- JButton bSeleccionarAsociacion: Botón que el usuario puede presionar para que le aparezca la ventana de selección de asociaciones de texto y pueda elegir una.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreTA: Etiqueta que indica al usuario dónde ingresar el nombre del teclado.
- JTextArea areaNomTA: Área de texto donde el usuario puede escribir el nombre del nuevo teclado.
- JLabel txtNombreAlfabetoTA: Etiqueta que indica al usuario dónde ingresar el nombre del alfabeto vinculado a ese nuevo teclado.
- JTextArea areaContenidoAlfabetoTA: Área de texto donde quedará seleccionado el archivo alfabeto que el usuario haya elegido de la ventana de selección de alfabetos.
- JLabel txtNombreAsociacionTA: Etiqueta que indica al usuario dónde ingresar el nombre de la asociación vinculada a ese nuevo teclado.
- JTextArea areaContenidoAsociacionTA: Área de texto donde quedará seleccionado el archivo asociación de textos que el usuario haya elegido de la ventana de selección de asociaciones.
- txtbuttonBB: Texto para indicar que el botón bAgregarTecladoBB sirve para agregar un teclado mediante el método Branch & Bound.
- txtbuttonSA: Texto para indicar que el botón bAgregarTecladoSA sirve para agregar un teclado mediante el método heurístico Simulated Annealing.
- JFrame NomFrame: Marco utilizado para mostrar un mensaje de error en caso de que el usuario intente agregar un teclado sin proporcionar un nombre.
- JFrame CPframe: Marco utilizado para mostrar un mensaje de error en caso de que el usuario intente crear un teclado sin proporcionar un alfabeto y una asociación de textos a vincular.
- JFrame frame: Marco utilizado para otros mensajes de error potenciales.

Métodos:

- VistaTecladoA(): Constructor de la clase VistaTecladoA. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para agregar un nuevo teclado o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para crear el teclado.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.15 VistaTecladoB

Esta vista es la encargada de borrar un teclado. Se abre la carpeta donde se guardan todos los teclados para que el usuario pueda seleccionar uno a eliminar. Esta carpeta tiene un botón para confirmar y otro para cancelar,

Atributos:

- JFrame frame: Marco principal utilizado como ventana para las interfaces gráficas relacionadas con la eliminación de un teclado.
- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaTB: Etiqueta que muestra el título de la ventana, en este caso, "Borrar teclado".
- JButton bBorrarTeclado: Botón que el usuario puede presionar para iniciar el proceso de borrar un nuevo teclado.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtDesplegableTB: Etiqueta que indica al usuario dónde está la lista de nombres de teclados.
- JComboBox<String> nombresTB : Desplegable que contiene todos los nombres de los teclados del conjunto de teclados.
- JLabel txtNombreTB: Etiqueta que indica al usuario dónde ingresar el nombre del teclado.
- JTextArea areaNomTB: Área de texto donde el usuario puede escribir el nombre del nuevo teclado.

Métodos:

- VistaTecladoB(): Constructor de la clase VistaTecladoB. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para borrar un teclado o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para borrar el teclado.
- Se despliega una lista con los teclados existentes para que se pueda seleccionar uno para borrar.
- Si la eliminación es exitosa, se muestra un diálogo informando al usuario que el teclado se ha borrado correctamente.
- Si el teclado no se puede eliminar (por ejemplo, si no existe), se muestra un diálogo de error.

- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.16 VistaTecladoC

Esta vista es la encargada de consultar un teclado. Contiene todo lo necesario para consultar. Tenemos un desplegable dónde salen todos los nombres de los teclados, para que el usuario pueda seleccionar uno. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido del teclado seleccionado. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que sirve como contenedor para los demás componentes de la interfaz de usuario en esta vista.
- JLabel tituloVistaTC: Etiqueta que muestra el título de la parte superior de la ventana, "Consultar teclado".
- JLabel txtDesplegableTC: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String> nombresTC: Desplegable que contiene la lista de nombres de los teclados, permite seleccionar uno de ellos.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreTC: Etiqueta que indica al usuario dónde se mostrará el nombre del teclado consultado.
- JTextArea areanomTC: Área de texto donde se muestra el nombre del teclado consultado. Está configurada para ser no editable.
- JLabel txtContenidoTC: Etiqueta que indica al usuario dónde se mostrará el contenido del teclado consultado.
- JTextArea areacontenidoTC: Área de texto donde se muestra el contenido del teclado consultado. Está configurada para ser no editable y contiene un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaTecladoC(): Constructor de la clase VistaTecladoC. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.
- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para consultar un teclado o salir de la vista.

- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para consultar el teclado.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.1.17 VistaTecladoM

Esta vista es la encargada de modificar un teclado. Contiene todo lo necesario para modificar. Tenemos un desplegable dónde salen todos los nombres de los teclados, para que el usuario pueda seleccionar uno. Al seleccionar uno se muestra en los campos nombre y contenido el nombre y contenido del teclado seleccionado. Estos campos se pueden modificar y se guarda la modificación del teclado. Por último tenemos el botón atrás con el cuál volvemos a la pantalla principal.

Atributos:

- JPanel lámina: Panel principal que contiene los elementos de la interfaz de usuario.
- JLabel tituloVistaTM: Etiqueta que muestra el título de la ventana, en este caso, "Modificar teclado".
- JLabel txtDesplegableTM: Texto indicativo para facilitar la comprensión del desplegable, descrito justo debajo.
- JComboBox<String> nombresTM: Desplegable que contiene la lista de nombres de los teclados, permite seleccionar uno de ellos.
- JButton bModificarTeclado: Botón que, al ser presionado, inicia el proceso de modificar el teclado seleccionado.
- JButton bsalir: Botón que permite al usuario salir de la vista actual y regresar al menú principal.
- JLabel txtNombreTM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el nombre del teclado.
- JTextArea areanomTM: Área de texto donde se muestra o se puede modificar el nombre del teclado.
- JLabel txtContenidoTM: Etiqueta que indica al usuario dónde se muestra o se puede modificar el contenido del teclado.
- JTextArea areacontenidoTM: Área de texto donde se muestra o se puede modificar el contenido del teclado. Incluye un scroll para facilitar la visualización de contenido extenso.

Métodos:

- VistaTecladoM(): Constructor de la clase VistaTecladoM. Inicializa los componentes de la interfaz de usuario, establece el diseño y las acciones de los botones, y configura el comportamiento general de la vista.
- Dentro del constructor, se establece el tamaño y la disposición de los elementos de la interfaz de usuario.

- Se añaden ActionListeners a los botones para manejar eventos de clic, como la lógica para modificar un teclado o salir de la vista.
- Se añade también el control de errores que tiene en cuenta que se llenen todos los campos necesarios para modificar el teclado.
- Se configura el comportamiento de la ventana (la acción al cerrar la ventana).

1.2 Explicación del controlador de presentación

1.2.1 CtrlPresentació

El controlador de Presentación se encarga de hacer la comunicación entre las vistas de la capa de presentación. También es el que transmite a la capa de presentación los datos de las capas inferiores.

Atributos:

- CtrlDominio cd: Instancia del controlador de dominio.

Métodos:

- iniPresentacion(): Muestra la pantalla de la ventana del menú principal.
- vistaAlfabetoA(): Muestra la pantalla de la ventana de agregar alfabeto.
- vistaAlfabetoB(): Muestra la pantalla de la ventana de borrar alfabeto.
- vistaAlfabetoC(): Muestra la pantalla de la ventana de consultar alfabeto.
- vistaAlfabetoM(): Muestra la pantalla de la ventana de modificar alfabeto.
- vistaTextoA(): Muestra la pantalla de la ventana de agregar texto.
- vistaTextoB(): Muestra la pantalla de la ventana de borrar texto.
- vistaTextoC(): Muestra la pantalla de la ventana de consultar texto.
- vistaTextoM(): Muestra la pantalla de la ventana de modificar texto.
- vistaAsociacionTextosA(): Muestra la pantalla de la ventana de agregar asociación de textos.
- vistaAsociacionTextosB(): Muestra la pantalla de la ventana de borrar asociación de textos.
- vistaAsociacionTextosC(): Muestra la pantalla de la ventana de consultar asociación de textos.
- vistaAsociacionTextosM(): Muestra la pantalla de la ventana de modificar asociación de textos.
- vistaTecladoA(): Muestra la pantalla de la ventana de agregar teclado.
- vistaTecladoB(): Muestra la pantalla de la ventana de borrar teclado.
- vistaTecladoC(): Muestra la pantalla de la ventana de consultar teclado.
- vistaTecladoM(): Muestra la pantalla de la ventana de modificar teclado.
- agregarAlfabetoManual(String nomA, String entradaCaracteres, String path): Llama a la función agregarAlfabeto de CtrlDominio pasándole como parámetro el nombre del alfabeto a agregar y el contenido de este.

- agregarAlfabetoPath(String nomA, String path): Llama a la función agregarAlfabeto de CtrlDominio pasándole como parámetro el nombre del alfabeto a agregar y el contenido de este.
- getNombresAlfabetos(): Llama a la función getNombresAlfabetos de CtrlDominio.
- existeAlfabeto(String nomA): Llama a la función existeAlfabeto de CtrlDominio pasándole como parámetro el nombre del alfabeto a consultar.
- consultarContenidoAlfabeto(String nomA): Llama a la función consultarContenidoAlfabeto de CtrlDominio pasándole como parámetro el nombre del alfabeto a consultar.
- modificarContenidoAlfabeto(String nomA, ArrayList<Character> entrada): Llama a la función modificarContenidoAlfabeto de CtrlDominio.
- borrarAlfabeto(String nomA): Llama a la función borrarAlfabeto de CtrlDominio pasándole como parámetro el nombre del alfabeto a borrar.
- agregarTextoPalabras(String nomTxt, String texto, String path): Llama a la función agregarTextoPalabras de CtrlDominio pasándole como parámetro el nombre del texto a agregar y el contenido de este.
- agregarTextoFrecuencias(String nomTxt, String frecuenciasPalabras, String path): Llama a la función agregarTextoFrecuencias de CtrlDominio pasándole como parámetro el nombre del texto a agregar y el contenido de este.
- getNombresTextos(): Llama a la función getNombresTextos de CtrlDominio.
- existeTexto(String nomTxt): Llama a la función existeTexto de CtrlDominio pasándole como parámetro el nombre del texto a consultar.
- consultarContenidoTexto(String nomTxt): Llama a la función consultarContenidoTexto de CtrlDominio pasándole como parámetro el nombre del texto a consultar.
- modificarContenidoTexto(String nomTxt, String entrada): Llama a la función modificarContenidoTexto de CtrlDominio.
- borrarTexto(String nomTxt): Llama a la función borrarTexto de CtrlDominio pasándole como parámetro el nombre del texto a borrar.
- agregarAsociacionNombre(String nomA): Llama a la función agregarAsociacionNombre de CtrlDominio pasándole como parámetro el nombre de la asociación a agregar.
- agregarTextoAsociacion(String nomAT, String nomTxt): Llama a la función agregarTextoAsociacion de CtrlDominio pasándole como parámetro el nombre de la asociación donde hay que añadir el texto y el nombre del texto a añadir.
- getNombreAsociaciones(): Llama a la función getNombresAsociaciones de CtrlDominio.
- existeAsociacion(String nomAT): Llama a la función existeAsociacion de CtrlDominio pasándole como parámetro el nombre de la asociación a consultar.
- consultarCjtTextosAsociacion(String nomAT): Llama a la función consultarContenidoAsociacion de CtrlDominio.

- borrarTextoAsociacion(String nomAT, String nomTxt): Llama a la función borrarTextoAsociacion de CtrlDominio pasándole como parámetro el nombre de la asociación donde hay que borrar el texto y el nombre del texto a borrar de la lista de textos de esa asociación.
- borrarAsociacionTextos(String nomAT): Llama a la función borrarAsociacionTextos de CtrlDominio pasándole como parámetro el nombre de la asociación a borrar.
- compatibles(Alfabeto alfabeto, AsociacionTextos asociacionTextos): Retorna si el alfabeto y la asociacion de textos tiene caracteres parecidos y por lo tanto son compatibles.
- agregarTeclado(String nomT, String nomA, String nomAT): Llama a la función agregarTeclado de CtrlDominio pasándole como parámetro el nombre del teclado a agregar y el nombre del alfabeto y la asociación a vincular.
- consultarContenidoTeclado(String nomT): Llama a la función consultarContenidoTeclado de CtrlDominio pasándole como parámetro el nombre del teclado a consultar.
- consultarAlfabetoAsociadoTeclado(String nomT): Llama a la función consultarAlfabetoAsociadoTeclado de CtrlDominio pasándole como parámetro el nombre del teclado a consultar.
- consultarAsociacionAsociadoTeclado(String nomT): Llama a la función consultarAsociacionAsociadoTeclado de CtrlDominio pasándole como parámetro el nombre del teclado a consultar.
- getListaTeclados(): Llama a la función getListaTeclados de CtrlDominio.
- borrarTeclado(String nomT): Llama a la función borrarTeclado de CtrlDominio pasándole como parámetro el nombre del teclado a borrar.

2. DIAGRAMA DE LAS CLASES Y CONTROLADORES DE LA CAPA DE DOMINIO

A continuación mostramos el diseño del diagrama de las clases y controladores de la capa de dominio. El diagrama se puede encontrar más detallado en la carpeta DOCS.

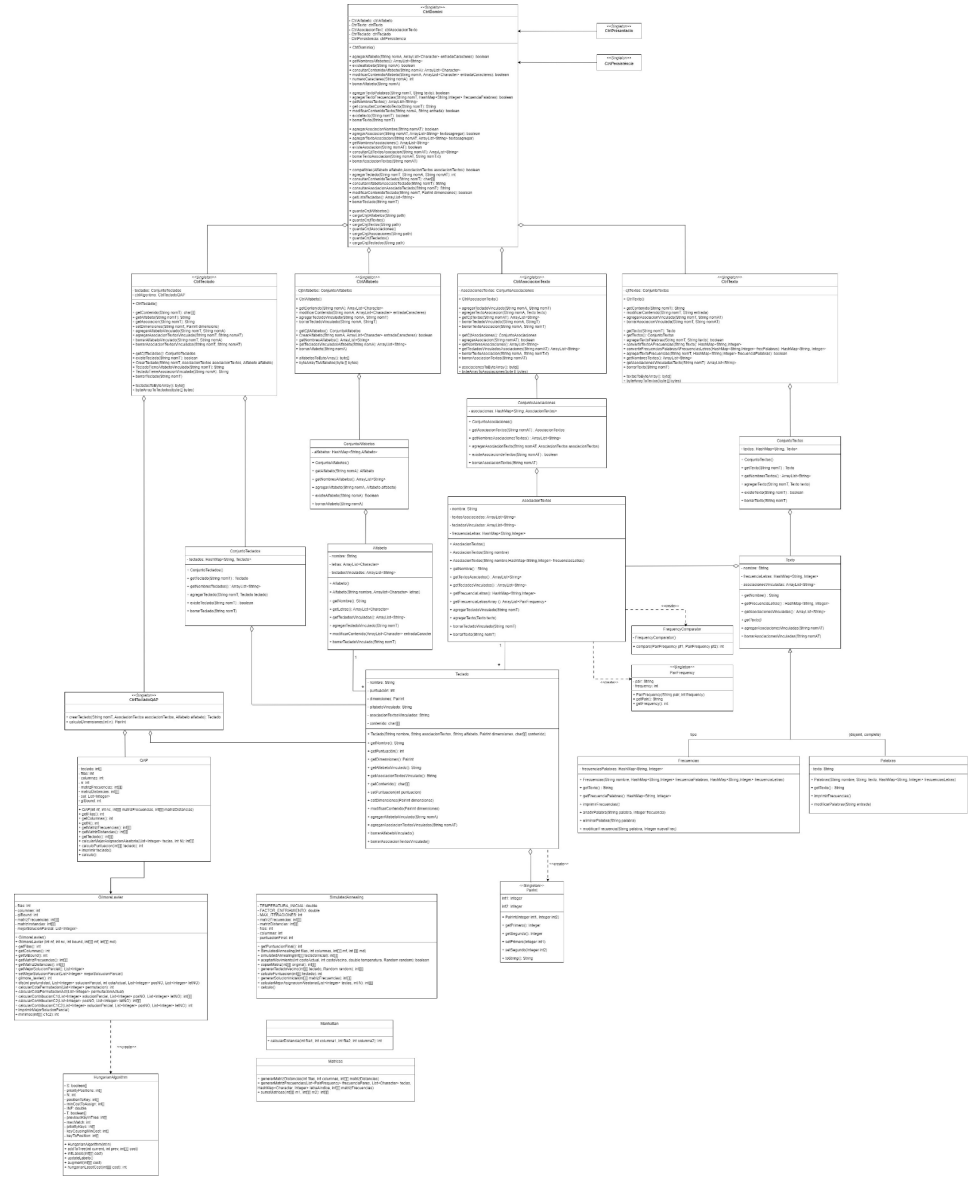

Figura 2: Diagrama de modelo conceptual

Restricciones textuales:

- Claves externas: (ConjuntoAlfabetos, alfabetos), (ConjuntoAsociaciones, asociaciones), (ConjuntoTextos, textos), (ConjuntoTeclados, teclados), (Alfabeto, nombre), (AsociacionTextos, nombre), (Texto, nombre), (Teclado, nombre)

2.1 Descripción de las clases de dominio

2.1.1 ConjuntoAlfabetos

Esta clase recoge todos los alfabetos que se han añadido y, por tanto, están presentes en el sistema. La información se almacena en un HashMap<String, Alfabeto>, donde el string hace referencia al nombre del alfabeto y Alfabeto es el objeto que define a cada alfabeto en particular. La clase tiene una operación constructora, distintos getters que retornan información de un alfabeto en específico y el conjunto de todos los alfabetos existentes. Además, también se puede editar este conjunto, ya que hay una función para agregar un alfabeto nuevo y otra para borrarlo en caso de que exista y ya no se quiera. Por último, existe otra para determinar si un alfabeto está registrado o no.
A continuación describiremos los atributos y métodos de manera un poco más concreta.

Atributos:

- HashMap<String, Alfabeto> alfabetos: Es el map donde se guardan todos los alfabetos agregados. Este está ordenado por la clave de cada alfabeto que es su nombre. El segundo valor es una instancia del propio Alfabeto

Métodos:

- ConjuntoAlfabetos(): Constructora
- getAlfabeto(String nomA): Retorna la instancia del alfabeto que tiene como clave única el nombre nomA
- getNombresAlfabetos(): Retorna una lista con todos los nombres(identificadores de los alfabetos) de todos los alfabetos existentes.
- agregarAlfabeto(String nomA, Alfabeto alfabeto): Añade un alfabeto al map de conjunto de alfabetos donde se encuentran todos los alfabetos existentes. Este alfabeto se identifica con el nombre nomA.
- existeAlfabeto(String nomA): Busca en el map de conjunto de alfabetos si está entre los existentes el alfabeto con nombre nomA.
- borrarAlfabeto(String nomA): Borra el alfabeto con identificador nomA del conjunto de alfabetos existentes.

2.1.2 Alfabeto

Esta clase sirve para guardar toda la información relacionada con un alfabeto. Se identifica por el *nombre* (clave primaria) pero también tiene otros atributos como *letras*, que es un *ArrayList* con los carácteres de los cuales se compone el alfabeto, y otro *ArrayList<String> tecladosVinculados* con los teclados que se han creado usando ese alfabeto. Esta clase contiene todos los getters necesarios para obtener los valores de los atributos y los setters para agregar o borrar un teclado vinculado a la lista. Aparte de esto, hay constructoras para el caso de querer crear un alfabeto vacío o con el nombre y su respectivo contenido.

A continuación describiremos los atributos y métodos de manera un poco más concreta.

Atributos:

- String nombre: Nombre y clave única del alfabeto
- ArrayList<Character> letras: lista de carácteres que forman el contenido del alfabeto.
- ArrayList<String> tecladosVinculados: lista de nombres (identificadores) de los teclados creados a partir de ese alfabeto

Métodos:

- Alfabeto(): Constructora.
- Alfabeto(String nombre, ArrayList<Character> letras): Constructora del alfabeto donde tiene un nombre y una lista de carácteres concreta y definida por el usuario, por otro lado tiene una lista de tecladosVinculados vacía inicialmente.
- getNombre(); Retorna el nombre de dicho alfabeto.
- getLetras(): Retorna el contenido del teclado que está guardado en la lista letras.
- getTecladosVinculados():Retorna la lista de nombres de los teclados vinculados a ese alfabeto.
- agregarTecladoVinculado(String nomT): Añade a la lista de nombres de teclados vinculados el nombre del teclado nomT.
- modificarContenido(ArrayList<Character> entradaCaracteres): Actualiza el contenido del alfabeto con el nuevo contenido
- borrarTecladoVinculado(String nomT): Borra de la lista de nombres de teclados vinculados el nombre del teclado nomT.

2.1.3 ConjuntoAsociaciones

La clase ConjuntoAsociaciones se encarga de tener un registro de todas las asociaciones existentes. El atributo de esta clase es un HashMap<String, AsociacionTextos>, definido por el nombre de la asociación y esta misma. Existen los getters necesarios para obtener la información particular del atributo y también de una asociación de textos en concreto. Además hay funciones para determinar si una asociación de textos está presente en el map, para añadir y para borrar una asociación del conjunto. Cuenta con los siguientes atributos y métodos.

Atributos:

- HashMap<String, AsociacionTextos> asociaciones: Es el map donde se guardan todas las asociaciones agregadas. Este está ordenado por la clave de cada asociación que es su nombre. El segundo valor es una instancia de la propia asociación.
-

Métodos:

- ConjuntoAsociaciones(): Constructora

- getAsociacionTextos(String nomAT): Devuelve la asociación de textos nomAT introducida si existe.
- getNombresAsociacionesTextos(): Devuelve el nombre de las asociaciones del conjunto.
- agregarAsociacionTexto(String nomAT, AsociacionTextos asociacionTextos): Añade una asociación de textos al conjunto de asociaciones de textos.
- existeAsociaciondeTextos(String nomAT): Devuelve si existe la asociación de textos con el nombre dado.
- borrarAsociacionTextos(String nomAT): Borra la asociación de textos con nombre nomAT del conjunto de asociaciones.

2.1.4 AsociacionTextos

AsociacionTextos tiene como atributos el nombre, que la identifica, los textos usados para esta asociación (*textosAsociados*) y los teclados que usan esa asociación (*tecladosVinculados*). Además tiene también un HashMap que guarda todas las parejas de letras (presentes en todos los textos de la asociación) con la frecuencia en la que aparecen (*frecuenciaLetras*). Existen unas funciones constructoras para el caso de querer crear una asociación vacía, con solo el nombre o con *nombre* y *frecuenciaLetras*. Por otro lado, hay getters para obtener el contenido de todos los atributos, setters para agregar teclados vinculados y textos. Por último, se encuentran las funciones auxiliares para borrar de *textosAsociados* y de *tecladosVinculados* un texto o un teclado, respectivamente.

Esta clase cuenta con los siguientes atributos y métodos:

Atributos:

- String nombre: Nombre introducido por el usuario
- ArrayList<String> textosAsociados: Lista de textos de los que está compuesta la asociación
- ArrayList<String> tecladosVinculados: Lista de teclados que se han creado a partir de la asociación.
- HashMap<String, Integer> frecuenciaLetras: Map que asocia pares de letras con sus frecuencias

Métodos:

- AsociaciónTextos(): Constructora.
- AsociacionTextos(String nombre, HashMap<String, Integer> frecuenciaLetras): Constructora inizializando las variables como vacías menos los que paso como parámetros.
- AsociaciónTextos(String nombre): Constructora inicializando las variables como vacías menos el nombre.
- getNombre(): Devuelve el nombre introducido por el usuario.
- getTextosAsociaciados(): Devuelve el nombre de los textos asociados.
- getTecladosVinculados(): Devuelve el nombre de los teclados vinculados a la asociación.
- getFrecuenciaLetras(): Devuelve el map de pares de letras con sus frecuencias.

- getFrecuenciaLetrasArray(): Devuelve los pares String,Integer de los pares de letras con sus frecuencias ordenadas.
- agregarTecladoVinculado(String nomT): Añade a la lista de nombres de teclados vinculados el nombre del nuevo teclado nomT
- agregarTexto(Texto texto): Añade a la lista de nombres de los textos asociados el nuevo texto
- borrarTecladoVinculado(String nomT): Borra de la lista de nombres de los teclados vinculados el nombre del teclado nomT.
- borrarTexto(String nomT): Borra de la lista de textos asociados el nombre del texto nomT.

2.1.5 ConjuntoTextos

La clase ConjuntoTextos recoge todos los textos que se han agregado y, por tanto, están presentes en el sistema. Los guarda en un HashMap<String, Texto> donde el primer elemento del Hashmap es el nombre y sirve como clave única del texto al que define y como segundo elemento un objeto Texto. A continuación se puede observar los atributos y métodos que lo componen:

Atributos:

- HashMap<String, Texto> textos: Conjunto de textos introducidos por el usuario.

Métodos:

- ConjuntoTextos(): Constructora.
- getTexto(String nomT): Retorna el texto nomT (si existe).
- getNombresTextos(): Retorna el nombre de los textos del conjunto.
- agregarTexto(String nomT, Texto texto): Añade un texto al conjunto de textos.
- existeTexto(String nomT): Verifica la existencia del texto con el nombre dado.
- borrarTexto(String nomT): Borra el texto con nombre nomT del conjunto de textos.

2.1.6 Texto

Esta clase es de tipo *abstract*, es decir, que no puede ser instanciada directamente, y representa un texto. No tiene una función constructora pero sí *getters* para acceder al nombre del texto y para conseguir los pares de letras con sus frecuencias. También se puede acceder al contenido del texto y a la una lista de nombres de las asociaciones de texto a las que está asociado este mismo. Además, en esta clase es posible agregar AsociacionesVinculadas al texto que se elija o borrarlas. Las funciones y atributos de la clase son las siguientes:

Atributos:

- String nombre: Nombre introducido por el usuario.
- HashMap<String, Integer> frecuenciaLetras: Guarda la lista de pares de caracteres y su frecuencia ("ab", 5)
- ArrayList<String> asociacionesVinculadas: Guarda el nombre de las asociaciones de textos que tienen este texto en su asociación

Métodos:

- getNombre(): Devuelve el nombre introducido por el usuario.
- getFrecuenciaLetras(): Devuelve el map de pares de letras con sus frecuencias.
- getAsociacionesVinculadas() Devuelve una lista de nombres de las asociaciones de textos que contienen el texto.
- getTexto(): Devuelve el contenido del texto.
- agregarAsociacionesVinculadas(String nomAT): No devuelve nada. Añade a la lista de asociaciones vinculadas el nombre de la asociación de textos pasada por parámetro nomAT.
- borrarAsociacionesVinculadas(nomAT): No devuelve nada. Borra de la lista de asociaciones de textos vinculadas el nombre de la asociación pasada por parámetro nomAT.

2.1.7 Palabras

Palabras es una subclase de la clase abstract Texto, por lo que algo de esa clase también se atribuye a esta. Si el usuario entra el contenido del texto con frases o párrafos, guardaremos la información en esta subclase. En ella hay una constructora cuyos parámetros son nombre, texto y un HashMap<String, Integer> de frecuenciaLetras y dos getters para acceder al contenido del texto en sí o poder imprimir las frecuencias de las letras de este.

Atributos:

- String texto: Guarda el conjunto de palabras que conforman el texto.

Métodos:

- Palabras(String nombre, String texto, HashMap<String, Integer> frecuenciaLetras): Crea una instancia de la clase Palabras e inicializa los atributos de esta clase con los valores correspondientes.
- getTexto(): Devuelve el contenido, es decir, texto, del Texto de Tipo Palabras.
- imprimirFrecuencias(): Imprime los valores de frecuenciaLetras.
- modificarPalabras(String entrada): Actualiza el contenido del texto con el nuevo contenido.

2.1.8 Frecuencias

Al igual que la anterior, Frecuencias es también una subclase de texto, pero aquí se tiene en cuenta que el usuario entra el contenido del texto como palabras sueltas con sus respectivas frecuencias. Después será el propio programa quien se encargará de pasar estas frecuencias de palabras a frecuencias de pares de letras. Se cuenta con la opción de imprimir las Frecuencias explicadas anteriormente y, en el caso de que solo se quiera de las frecuencias de Palabras, existe la función *getFrecuenciasPalabras*.

Atributos:

- HashMap<String, Integer> frecuenciaPalabras: Guarda la lista de palabras que contienen el texto y su frecuencia.

Métodos:

- Frecuencias(String nombre, HashMap<String, Integer> frecuenciaPalabras, HashMap<String, Integer>) frecuenciaLetras: Crea una instancia de la clase Frecuencias e inicializa los atributos de esta clase con los valores correspondientes
- getTexto(): Devuelve el contenido, es decir, frecuenciaPalabras.
- getFrecuenciaPalabras(): Devuelve el map de pares de palabras con sus frecuencias
- imprimirFrecuencias(): No devuelve nada. Imprime los valores de frecuenciaPalabras y frecuenciaLetras.
- añadirPalabra(String palabra, Integer frecuencia): Añade la palabra con su frecuencia a frecuenciaPalabras.
- eliminarPalabra(String palabra): Eliminar la palabra de frecuenciaPalabras.
- modificarFrecuencia(String palabra, Integer nuevaFrec): Cambia la frecuencia de la palabra por la nueva frecuencia nuevaFrec.

2.1.9 ConjuntoTeclados

Esta clase se encarga de almacenar todos los teclados que existen en el sistema, es decir, los que el usuario ha agregado. Y lo hace en un HashMap<String, Teclado>, donde el primer parámetro hace referencia al nombre del teclado y el segundo es el objeto del Teclado en sí. La clase cuenta con distintas operaciones como, por ejemplo, una constructora con sus respectivos parámetros. Otras funciones que se pueden observar son los *getters* (que devuelven un teclado en concreto, un listado de todos o solo de sus nombres) o funciones para agregar un teclado. Para finalizar con la clase, existen métodos auxiliares para asegurarse de la presencia del Teclado escogido o para borrarlo.

Describimos los atributos y funciones de manera más concreta:

Atributos:

- HashMap<String, Teclado> teclados: Mapa donde se guardan todos los teclados. La clave es el nombre del teclado y el valor es el Teclado.

Métodos:

- conjuntoTeclados(): Constructora.
- getTeclado(String nomT): Retorna el teclado que tiene como identificador el nombre nomT.
- getNombresTeclados(): Retorna una lista que contiene los nombres de todos los teclados del conjunto.
- agregarTeclado(String nomT, Teclado teclado): Añade el Teclado teclado con nombre nomT al mapa que contiene todos los teclados del conjunto.
- existeTeclado(String nomT): Comprueba que el teclado con nombre nomT exista en el conjunto.

- borrarTeclado(String nomT): Elimina el teclado con nombre nomT del conjunto de teclados.

2.1.10 Teclado

Ésta guarda todo lo relacionado con los teclados y, como cualquier otra clase, tiene diversos métodos. En ella se cuenta con una constructora cuyos parámetros necesarios son un nombre para asignar el teclado, una asociación de textos, un alfabeto, una puntuación, un *PairInt* para definir las dimensiones y un char[][] para guardar el contenido. También hay *getters* con los que puedes acceder a distintos tipos de información que puede ser proporcionada por la clase. Esta información puede ser el nombre, la puntuación, las dimensiones, el contenido, el alfabeto vinculado o la asociación de texto vinculada. Finalmente, se cuenta con *setters* de los datos mencionados anteriormente (menos nombre) y funciones auxiliares para borrar cada una de los campos vinculados que puede tener un teclado.
Describimos los atributos y funciones de manera más concreta:

Atributos:

- String nombre: Nombre e identificador del teclado.
- int puntuación: Puntuación del teclado, generada por el algoritmo.
- PairInt dimensiones: Dimensiones del teclado guardadas en un pair. El first son las filas y el second las columnas.
- String alfabetoVinculado: Nombre del alfabeto vinculado al teclado.
- String asociaciónTextosVinculado: Nombre de la asociación de textos vinculada al teclado.
- char[][] contenido: Contenido del teclado, generado por el algoritmo. Se representan los carácteres y sus posiciones en una matriz de char.

Métodos:

- Teclado(String nombre, String asociacionTextos, String alfabeto, PairInt dimensiones, char[][] contenido): Constructora del teclado. Se crea con un alfabeto y asociación de textos asociados, unas dimensiones (filas y columnas), y una matriz de char vacía, donde pondremos el contenido una vez calculado.
- getNombre(): Retorna el nombre del teclado.
- getPuntuación(): Retorna la puntuación del teclado.
- getDimensiones(): Retorna el PairInt de las dimensiones (filas y columnas).
- getAlfabetoVinculado(): Retorna el nombre del alfabeto vinculado al teclado.
- getAsociaciónTextosVinculado(): Retorna el nombre de la asociación de textos vinculada.
- getContenido(): Retorna el contenido del teclado, en forma de matriz de char.
- setPuntuación(int puntuación): Se setea la posición del teclado al valor enviado por parámetro.
- setDimensiones(PairInt dimensiones): Se setean las dimensiones del teclado al par de filas y columnas pasado por parámetro.

- modificarContenido(PairInt dimensiones): Actualiza el contenido del teclado con el nuevo contenido
- agregarAlfabetoVinculado(String nomA): Se setea el valor del alfabeto vinculado al teclado con el atributo pasado por parámetro (sobreescribimos el anterior).
- agregarAsociaciónTextosVinculado(String nomAT): Se setea el valor de la asociación de textos vinculada al teclado con el atributo pasado por parámetro (sobreescribimos el anterior).
- borrarAlfabetoVinculado(): Se pone el atributo alfabetoVinculado a null.
- borrarAsociaciónTextosVinculada(): Se pone el atributo asociaciónTextosVinculada a null.

2.1.11 Gilmore-Lawler

Clase que implementa el cálculo de la cota de Gilmore-Lawler. Lo hacemos mediante un dfs, con la técnica lazy, y usamos el hungarian para tener una mejor cota. En el dfs, va calculando la cota de Gilmore-Lawler, y la va actualizando, a la misma vez que la solución parcial. Al acabar tendremos la mejor solución para las matrices que le pasamos a la creadora, ya calculadas. Explicamos con más detalle la elección de las estructuras en el apartado 4 de la documentación (ED&A).
Ahora describimos los atributos y funciones de manera explícita.

Atributos:

- int filas: Filas del teclado, parámetro auxiliar para facilidad en los bucles.
- int columnas: Columnas del teclado, parámetro auxiliar para facilidad en los bucles.
- int glBound: Cota de Gilmore-Lawler. Tiene un valor inicial dado por la clase QAP al generar la solución inicial.
- int [][] matrizFrecuencias: Matriz de frecuencias generada por la clase QAP antes de llamar a Gilmore
- int [][] matrizDistancias: Matriz de distancias generada por la clase QAP antes de llamar a Gilmore
- List<Integer> mejorSolucionParcial: Lista de enteros donde iremos guardando la mejor solución siempre que actualicemos la cota, para tener así siempre la mejor solución actual guardada.

Métodos:

- Gilmore-Lawler(): Creadora vacía de la clase de Gilmore-Lawler.
- Gilmore-Lawler(int nf, int nc, int bound, int[][] mf, int[][] md): Creadora de Gilmore-Lawler que recibe por parámetro las filas, columnas, cota inicial y las matrices de frecuencia y distancias.
- getFilas(): Retorna el atributo filas.
- getColumnas(): Retorna el atributo filas.
- getGlBound(): Retorna la cota actual.
- getMatrizFrecuencias(): Retorna la matriz de frecuencias.

- getMatrizDistancias(): Retorna la matriz de frecuencias.
- getMejorSoluciónParcial(): Retorna la mejor solución parcial hasta el momento.
- setMejorSoluciónParcial(list<Integer> mejorSolucionParcial): Setea la mejor solución parcial a la lista pasada por parámetro, de tamaño filas*columnas.
- gilmore-lawler(): Función que se ocupa de inicializar las estructuras necesarias para llamar al dfs y empezar el cálculo de la cota. Se explica en el apartado 4 de la documentación (ED&A).
- dfs(int profundidad, List<Integer> solucionParcial, int cotaActual, List<Integer> posNO, List<Integer> letNO): Recorrido en profundidad para encontrar la mejor solución en la generación del teclado. Se poda en función de la cota y se van explorando todos los teclados posibles hasta encontrar el mejor. Se explica en el apartado 4 de la documentación (ED&A).
- calcularCotaPermutación(List<Integer> permutacion): Calcula la puntuación de la permutación actual, haciendo frecuencias*distancias para todo par de letras.
- calcularCotaPermutaciónAct(List<Integer> permutacionActual): Calcula la puntuación de la permutación actual, haciendo frecuencias*distancias para todo par de letras y teniendo en cuenta que la permutación puede no ser completa. Nos sirve para calcular parte de la cota de Gilmore-Lawler.
- calcularContribuciónC1(List<Integer> solucionParcial, List<Integer> posNO, List<Integer> letNO): Calcula el coste de colocar la i-ésima tecla en la k-ésima posición respecto a las teclas ya emplazadas. Se explica en el apartado 4 de la documentación (ED&A).
- calcularContribuciónC2(List<Integer> posNO, List<Integer> letNO): Calcula el coste de colocar la i-ésima tecla en la k-ésima posición respecto a las teclas no emplazadas. Se explica en el apartado 4 de la documentación (ED&A).
- calcularContribuciónC1C2(List<Integer> solucionParcial, List<Integer> posNO, List<Integer> letNO): Función que llama a las funciones que calculan C1 y C2 y se ocupa de sumarlas y calcular la cota de manera exacta llamando al algoritmo Húngaro con la matriz C1+C2.
- imprimirMejorSolucionParcial(): Función que imprime la lista de mejor solución parcial en forma de teclado con sus filas y columnas correspondientes.
- minimos(int[][] c1c2): Función usada anteriormente en sustitución al algoritmo húngaro. Sirve para calcular la cota de Gilmore-Lawler a partir de la matriz C1+C2, siendo un poco menos exacta que con el algoritmo húngaro.

2.1.12 Hungarian Algorithm

Clase que se encarga de implementar el algoritmo húngaro con tal de poder asignar de manera óptima las teclas que se deben colocar en el teclado generado. Este mismo algoritmo cuenta con funciones que utilizan técnicas de reducción y búsqueda para encontrar la solución óptima. Esta clase cuenta con estos atributos y métodos:

Atributos:

- int N: Número de elementos (teclas o posiciones).
- int maxcoincidencia: Máximo número de coincidencias encontradas.
- int[] prioridadteclas: Prioridades asignadas a las teclas.
- int[] prioridadPos: Prioridades asignadas a las posiciones.
- int[] teclastoPos: Asignaciones de teclas a posiciones.
- int[] Postoteclas: Asignaciones de posiciones a teclas.
- boolean[] S: Conjunto de teclas seleccionadas.
- boolean[] T: Conjunto de posiciones seleccionadas.
- int[] minimCostassignar: Coste mínimo para asignar una tecla a una posición
- int[] TeclascostMin: Tecla que causa el costo mínimo.
- int[] teclaprevia: Tecla previa en el árbol de asignación.

Métodos:

- HungarianAlgorithm(int n): Constructora.
- addToTree(int act, int prev, int[] [] cost): Añade una tecla al árbol de expansión.
- initetiq(int[][] cost): Inicializa las prioridades de las teclas y las posiciones.
- actualitzaretiquetas(): Actualiza las prioridades de las teclas y las posiciones.
- augment(int[][] cost): Aumenta el número máximo de asignaciones posibles.
- hungarianLeastCost(int[][] cost): Función main. Calcula la asignación con el menor costo total.

2.1.13 Manhattan

Clase que sirve para calcular la distancia de Manhattan entre dos puntos (x, y) y (x', y'). Esta distancia la usamos para calcular la distancia entre teclas en el algoritmo de generación de teclado.
Describimos la función.

Métodos:

- calcularDistancia(int fila1, int columna1, int fila2, int columna2): Devuelve la distancia de Manhattan entre los puntos (fila1, columna1) y (fila2, columna2).

2.1.14 Matrices

Clase que calcula las diferentes matrices y la operación suma entre dos matrices. Son los cálculos de matrices necesarios para el algoritmo de generación de teclado.
Ahora describimos las funciones de manera explícita.

Métodos:

- generarMatrizDistancias(int filas, int columnas, int[][] matrizDistancias): Genera la matriz de distancias de Manhattan entre todos los pares de emplazamientos (teclas el caso de nuestro problema). Las filas y columnas se pasan de manera auxiliar para hacer los bucles.
- generarMatrizFrecuencias(List<PairFrequency> frecuenciasPares, List<Character> teclas, HashMap<Character, Integer> letraAIndice, int[][] matrizFrecuencias): Genera la matriz de frecuencias a partir de la lista frecuenciasPares, donde tenemos, cómo se explica mejor en el apartado 4, de Estructuras de Datos y Algoritmos, las frecuencias ordenadas. La matriz de frecuencias representa las frecuencias entre todos los pares de letras. La estructura letraAIndice se explica también en la documentación apartado 4, sirve para indexar las letras del alfabeto.
- sumaMatrices(int[][] m1, int[][] m2): Función sencilla que recibe cómo entrada un par de matrices y simplemente retorna en m3 la suma de ellas (m1+m2).

2.1.15 QAP

La clase QAP es la clase que se ocupa de enviar la información necesaria a Gilmore-Lawler para que esta otra clase calcule la cota de Gilmore-Lawler y la solución para las matrices de frecuencias y distancias que enviaremos desde QAP. Se ocupa también de generar la solución inicial y calcular su puntuación, para enviar una cota inicial lo más buena posible, y de esta manera, reducir el tiempo del cálculo de la cota. Tiene también una función auxiliar para imprimir el teclado. Todos los detalles de las estructuras de datos y algoritmos seleccionadas se explican con más detalle en el apartado 4 de la documentación, Estructura de Datos y Algoritmos, dónde damos la justificación de la elección de estas estructuras y explicamos con más detalle todos los cálculos y decisiones tomadas en la generación del teclado mediante el algoritmo.

Ahora describimos los atributos y funciones de manera explícita.

Atributos:

- int[][] teclado: Matriz de enteros que representa un teclado indexado, es decir, en lugar de representar las letras del alfabeto representa sus índices correspondientes.
- int filas: Filas del teclado, parámetro auxiliar para facilitar los recorridos en los bucles
- int columnas: Columnas del teclado, paràmetro auxiliar para facilitar los recorridos en los bucles.
- int n: Filas y columnas de las matrices de frecuencias y distancias, calculado como filas*columnas.
- int[][] matrizFrecuencias: Matriz calculada anteriormente, que representa las frecuencias entre todas las letras.
- int[][] matrizDistancias: Matriz calculada anteriormente, que representa la distancia de Manhattan entre todas las posiciones del teclado.

- List<Integer> sol: Solución para la generación del teclado. En lugar de representarse como una matriz se representa como una lista de filas*columnas, la cuál se tendrá que descomponer a posteriori en un matriz de filas y columnas.
- int glBound: Entero que representa una cota para Gilmore-Lawler, le daremos valor cuando calculemos la solución inicial y el coste del teclado resultante en esa solución.

Métodos:

- QAP(int nf, int nc, int[][] matrizFrecuencias, int[][] matrizDistancias): Creadora de la clase QAP, recibe filas, columnas y las dos matrices. Se ocupa de inicializar los atributos y de llamar a la función que genera la solución inicial, calcular su puntuación, y llamar a la clase Gilmore-Lawler con los parámetros necesarios. Explicación más detallada en el apartado 4 de la documentación (ED&A).
- getFilas(): Retorna el atributo filas.
- getColumnas(): Retorna el atributo columnas.
- getN(): Retorna el atributo n.
- getMatrizFrecuencias(): Retorna una copia de la matriz de frecuencias (para evitar su modificación)
- getMatrizDistancias(): Retorna una copia de la matriz de distancias (para evitar su modificación).
- getTeclado(): Retorna una copia de la matriz que representa el teclado (para evitar su modificación).
- calcularMejorAsignacionAleatoria(List<Integer> teclas, int N): Cálculo de la solución inicial. Consiste en calcular N asignaciones aleatorias (en nuestro caso, la N es 100, la explicación está en el apartado 4 de la documentación). Las va calculando y se queda con la mejor solución de estas 100 en coste.
- calculoPuntuacion(int[][] teclado): Recorre la matriz teclado y calcula su puntuación, determinada por frecuencia*distancia de todos los pares de letras del teclado.
- imprimirTeclado(): Función auxiliar que imprime el teclado por terminal.
- calculo(): Función auxiliar que llama a la clase Gilmore-Lawler y a su función que genera una lista de enteros que representa el teclado indexado, mediante el cálculo de la cota y el uso del algoritmo Húngaro. Se recupera la solución y se modifica el atributo teclado.

2.1.16. SimulatedAnnealing

Comentario importante: No lo tenemos bien puesto en el diagrama, en la Entrega 3 lo tendremos, ya que está implementado pero no conectado con el resto del código.

Clase que se encarga de implementar el algoritmo de Simulated Annealing, segundo algoritmo del proyecto. Su objetivo es calcular una asignación lo más cercano a la óptima posible del teclado. Esta clase cuenta con estos atributos y métodos:

Atributos:

- TEMPERATURA_INICIAL: Temperatura inicial que nos determina si podemos escoger soluciones peores a la actual. Al principio tendremos más libertad, y a medida que se enfríe seremos más restrictivos. Valor = 1000000.
- FACTOR_ENFRIAMIENTO: Factor de enfriamiento mencionado en la temperatura; nos determina a qué velocidad nos volvemos más restrictivos a la hora de escoger soluciones. Valor = 0.999.
- MAX_ITERACIONES: Número máximo de iteraciones del algoritmo. Valor alto (1000000) para poder explorar el espacio de soluciones de manera como más completa mejor
- matrizFrecuencias: Matriz de frecuencias entre las letras del alfabeto, la misma que en el primer algoritmo.
- matrizDistancias: Matriz de distancias (de Manhattan) entre las posiciones del teclado, la misma que en el primer algoritmo.
- filas: Número de filas del teclado.
- columnas: Número de columnas del teclado.
- puntuacionFinal: Puntuación final del teclado, al acabar la ejecución del algoritmo.

Métodos:

- getPuntuacionFinal(): Getter del atributo puntuación final.
- SimulatedAnnealing(int filas, int columnas, int[][] mf, int [][] md): Creadora de la clase, le pasamos las matrices de frecuencias y distancias correctamente calculadas, junto con los parámetros auxiliares filas y columnas.
- simulatedAnnealing(int[][] tecladoInicial): Función principal que ejecuta SimulatedAnnealing con nuestro teclado generado por la solución inicial.
- aceptarMovimiento(int costoActual, int costoVecino, double temperatura, Random random): Función auxiliar para comprobar si aceptamos la solución propuesta, dependiendo de la mejora/empeoramiento y de la temperatura.
- copiarMatriz(int[][] original): Función auxiliar para copiar una matriz.
- generarTecladoVecino(int[][] teclado, Random random): Genera un sucesor mediante nuestro operador intercambiar de manera random.
- calculoPuntuacion(int[][] teclado): Calcula la puntuación del teclado pasado por parámetro, igual que en el primer algoritmo (suma frec*dist).
- generarSolucionInicial(int[][] matrizFrecuencias): Genera la solución inicial greedy, mencionada en el apartado de ED&A.
- calcularMejorAsignacionAleatoria(List<Integer> teclas, int N): Genera la solución inicial habitual, igual que en el primer algoritmo, de quedarse con la mejor solución de N soluciones random.

- calculo(): Se ocupa de generar la solución inicial, calcular su puntuación, y preparar el teclado para llamar a la función simulatedAnnealing y empezar la ejecución del algoritmo.

2.1.17 PairInt

Clase que implementa un par de enteros (Integer, Integer), para simplificar el uso de las dimensiones del teclado.

Ahora describimos los atributos y funciones de manera explícita.

Atributos:

- Integer int1: First del par, un entero.
- Integer int2: Second del par, un entero.

Métodos:

- PairInt(Integer int1, Integer int2): Creadora del par de enteros, recibe como parámetro el first y second, enteros.
- getPrimero(): Getter del first del par.
- getSegundo(): Getter del second del par.
- setPrimero(Integer int1): Setea el first al entero pasado por parámetro.
- setSegundo(Integer int2): Setea el second al entero pasado por parámetro.
- toString(): Convierte el par de enteros a un string.

2.1.18 PairFrequency

Clase que implementa un par de string y entero (String, int), usada para simplificar el uso de las frecuencias entre letras de los textos y las asociaciones de textos.

Ahora describimos los atributos y funciones de manera explícita.

Atributos:

- String pair: First del par, un string,
- int frequency: Second del par, un int.

Métodos:

- PairFrequency(String pair, int frequency): Creadora del par, recibe como parámetros el first y el second.
- getPair(): Getter del first del par.
- getFrequency(): Getter del second del par.

2.2 Descripción de los controladores de dominio

2.2.1 CtrlDominio

Este controlador tiene como constructora los demás controladores explicados a continuación. Al ser un Controlador "padre", para hacer todas las funciones que se ven en el código se llaman a los demás controladores.

Atributos:

- CtrlAlfabeto ctrlAlfabeto: Crea una instancia del controlador alfabeto
- CtrlTexto ctrlTexto: Crea una instancia del controlador texto
- CtrlAsociacionTexto ctrlAsociacionTexto: Crea una instancia del controlador asociación de textos
- CtrlTeclado ctrlTeclado: Crea una instancia del controlador teclado
- CtrlPersistencia ctrlPersistencia: Comunica el dominio con la capa de persistencia

Métodos:

- CtrlDominio(): Inicialización de la instancia controlador dominio
- agregarAlfabeto (String nomA, ArrayList<Character> entradaCaracteres): Retorna si se ha creado bien el alfabeto con el nombre y el contenido dados.
- getNombresAlfabetos(): Retorna la lista de nombres de los alfabetos existentes.
- existealfabeto(String nomA): Retorna si existe el alfabeto con nomA.
- consultarContenidoAlfabeto(String nomA): Retorna el contenido del alfabeto con nomA.
- modificarContenidoAlfabeto(String nomA, ArrayList<Character> entradaCaracteres): Modifica el contenido del alfabeto con nomA.
- numeroCaracteres(String nomA): Retorna el número de caracteres que tiene el contenido del alfabeto.
- borrarAlfabeto(String nomA): Borra el alfabeto con nomA y borra en cascada los teclados vinculados
- agregarTextoPalabras(String nomT, String texto): Retorna si se ha creado bien el texto.
- agregarTextoFrecuencias(String nomT. HashMap<String, Integer> frecuenciaPalabras): Retorna si se ha creado bien el texto.
- getNombresTextos(): Retorna la lista de nombres de los textos existentes.
- consultarContenidoTexto(String nomT): Devuelve el contenido del texto con nombre nomT.
- modificarContenidoTexto(String nomA, String entrada): Modifica el contenido del alfabeto con nomA.
- existetexto(String nomT): Retorna si existeel texto con nombre nomT.
- borrarTexto(String nomT): Retorna si el texto nomT ha sido borrado correctamente.
- agregarAsociacionNombre(String nomAT): Retorna si se ha creado bien la asociación con nombre nomAT.

- agregarAsociacion(String nomAT): Retorna si se ha creado bien la asociación con nombre nomAT.
- agregarTextoAsociacion (String nomAT, String nomTxt): Añade un texto a una asociación ya existente.
- getNombresAsociaciones(): Retorna la lista de nombres de las asociaciones existentes.
- existeAsociacion(String nomAT): Retorna si existe la asociación con nombre nomAT.
- consultarCjtTextosAsociacion(String nomAT): Retorna la lista de textos asociados a esa asociación.
- borrarTextoAsociacion (String nomAT, String nomTxt): Borra un texto a una asociación ya existente.
- borrarAsociacionTextos(String nomT): Retorna si el texto nomT ha sido borrado correctamente.
- compatibles(Alfabeto alfabeto, AsociacionTextos asociacionTextos): Retorna si el alfabeto y la asociación de textos tiene caracteres parecidos y por lo tanto son compatibles.
- agregarTeclado(String nomT, String nomA, String nomAT): Retorna un valor para comprobar si el teclado ha sido agregado correctamente o ha habido algún fallo.
- consultarContenidoTeclado(String nomT): Retorna el contenido del teclado nomT.
- consultarAlfabetoAsociadoTeclado(String nomT): Retorna el nombre del alfabeto asociado al teclado con nombre nomT.
- consultarAsociacionAsociadoTeclado(String nomT): Retorna el nombre de la asociación de textos asociada al teclado nomT.
- modificarContenidoTeclado(String nomT, PairInt dimensiones): Retorna si se ha modificado bien el contenido del teclado con nombre nomT.
- getListaTeclados(): Retorna la lista de los nombres de los teclados existentes.
- borrarTeclado(String nomT): Borra el teclado con nombre nomT. También desvincula el alfabeto y la asociación de textos que tenía vinculados
- guardaCnjtAlfabetos(): Almacena el conjunto de alfabetos en una localización específica.
- cargaCnjtAlfabetos(String path): Retorna el conjunto de alfabetos que se encuentra en path.
- guardaCnjtTextos(): Guarda el conjunto de textos en una ubicación particular.
- cargaCnjtTextos(String path): Devuelve el conjunto de textos que se halla en la ruta especificada.
- guardaCnjtAsociaciones(): Guarda en un lugar determinado el conjunto de asociaciones de textos.
- cargaCnjtAsociaciones(String path): Entrega el conjunto de asociaciones de textos ubicado en la ruta indicada.
- guardaCnjtTeclados(): Deposita el conjunto de teclados en una ubicación designada.

- cargaCnjtTeclados(String path): Recupera el conjunto de teclados presente en la ruta especificada.

2.2.2 CtrlAlfabeto

El controlador de alfabeto cuenta con un objeto de la clase ConjuntoAlfabetos como función constructora. Por tal de cumplir su función como *controller*, utiliza funciones de diversas clases creadas en el proyecto. Primeramente se pueden observar funciones de Alfabeto para conseguir acceder al contenido del propio o para vincularle un teclado. Además, también trata métodos de ConjuntoAlfabetos como devolver el objeto *cjtalfabetos* pedido o una lista de los nombres de los alfabetos existentes en ese objeto. Más también permite crear un alfabeto y almacenarlo en ConjuntoAlfabetos o borrar uno de los existentes.

Atributos:

- ConjuntoAlfabetos Cjtalfabetos: Representa el conjunto de alfabetos dentro del controlador para gestionar y manipular los textos.

Métodos:

- CtrlAlfabeto(): Constructora.
- getContenido(String nomA): Devuelve el contenido del alfabeto con nombre nomA.
- modificarContenido(String nomA, ArrayList<Character> entradaCaracteres): Actualiza el contenido del alfabeto con el nuevo contenido.
- agregarTecladoVinculado (String nomA, String nomT): Añade el nombre del teclado nomT a la lista de teclados vinculados del alfabeto nomA.
- borrarTecladoVinculado(String nomA, String nomT): Borra el nombre del teclado nomT de la lista de teclados vinculados del alfabeto nomA.
- getCjtAlfabetos(): Devuelve el objeto cjtAlfabetos.
- CrearAlfabeto(String nomA, ArrayList<Character> entradaCaracteres): Devuelve si se ha creado bien el alfabeto con el nombre nomA.
- getNombresAlfabetos(): Devuelve la lista de nombres de alfabetos existentes.
- getTecladosVinculadosAlfabeto(String nomA): Devuelve la lista de nombres de teclados vinculados al alfabeto con nombre nomA.
- borrarAlfabeto(String nomA): Borra el alfabeto con nombre nomA.
- alfabetosToByteArray(): Convierte un conjunto de alfabeto en ByteArray con el fin de almacenarlos.
- byteToAlfabetos(byte [] bytes): Transforma un conjunto de alfabetos para almacenar en un array de bytes al formato original de Alfabeto.

2.2.3 CtrlAsociacionTextos

Éste es el encargado de gestionar los diferentes métodos de asociaciones de texto. Tiene como constructora un cjtAsociacionesTextos y, como los demás, cuenta con métodos de diferentes clases. En este caso la AsociacionTexto proporciona las opciones de vincular un teclado a la asociación y agregar un texto a la misma. Después utilizamos funciones de conjunto de asociaciones para acceder a una asociación de textos y para agregar o borrar una nueva asociación a la clase.

Atributos:

- ConjuntoAsociaciones AsociacionesTextos: Representa el conjunto de asociaciones de textos dentro del controlador para gestionar y manipular los textos

Métodos:

- CtrlAsociacionTextos(): Constructora.
- agregarTecladoVinculado(String nomA, String nomT): Manda añadir el nuevo teclado al conjunto de teclados vinculados.
- agregarTextoAsociacion(String nomA, Texto texto): Manda añadir un texto al conjunto de textos de la asociación con nombre nomA.
- getCjtTextos(String nomAT): Devuelve la lista de nombres de los textos asociados a la asociación con nombre nomAT.
- borrarTecladoVinculado(String nomA, String nomT): Manda borrar el teclado del conjunto de teclados vinculados.
- borrarTextoAsociacion(String nomA, String nomT): Borra un texto de la asociación existente.
- getCjtAsociaciones(): Devuelve el conjunto de asociaciones de textos.
- agregarAsociacion(String nomAT): Agrega una nueva asociación de textos al conjunto.
- getNombresAsociaciones(): Devuelve la lista de nombres de asociaciones existentes.
- getTecladosVinculadosAsociacion(String nomAT): Devuelve la lista de nombres de teclados vinculados a la asociación con nombre nomA.
- borrarTextoAsociacion(String nomA, String nomTxt): Manda borrar un texto al conjunto de textos de la asociación con nombre nomA.
- borrarAsociacionTextos(String nomAT): Borra una asociación de textos ya existente.
- asociacionesToByteArray(): Convierte un conjunto de asociación de textos en ByteArray con el fin de almacenarlos.
- byteArrayToAsociaciones(byte[] bytes): Transforma un conjunto de asociación de textos para almacenar en un array de bytes al formato original de AsociacionTextos.

2.2.4 CtrlTexto

CtrlTexto proporciona un Conjunto de Textos como constructora y, como a todo controlador, diversas clases le proporcionan funcionalidades para poder cumplir con su objetivo. En este caso, la clase Texto y ConjuntoTextos cogen importancia. La primera ayuda a proporcionar el contenido del

propio texto que se pida y ayuda a vincularlo a una asociación. Con la segunda , de la misma manera que en el controlador Alfabeto, se puede conseguir el contenido de un texto en específico y un listado del conjunto de textos. Además con ConjuntoTextos podemos convertir el texto de entrada a frecuencias de letras con funciones como *agregarTextoPalabras, convertirTextoaFrecuencias* y *convertirFrecuenciasPalabrasAFrecuenciasLetras*. Y todas estas funciones se agregan a un texto utilizable para la creación del teclado con *agregarTextoFrecuencias*.

Atributos:

- ConjuntoTextos CjtTextos: Representa un conjunto de textos dentro del controlador para gestionar y manipular los textos.

Métodos:

- CtrlTexto(): Inicializa el conjunto de textos.
- getContenido(String nomT): Obtiene el contenido de un texto dado su nombre.
- modificarContenido(String nomT, String entrada): Actualiza el contenido del texto con el nuevo contenido.
- agregarAsociacionVinculada(String nomT, String nomAT): No devuelve nada. Agrega una asociación vinculada a un texto dado su nombre
- borrarAsociacionVinculada(String nomT, String nomAT): No devuelve nada. Borra una asociación vinculada a un texto dado su nombre
- getTexto(String nomT): Obtiene un objeto Texto dado su nombre
- getTextos(): Obtiene el conjunto de textos
- agregarTextoPalabras(String nomT, String texto): Convierte el texto de entrada a frecuencias de letras y agrega un nuevo texto del tipo Palabras
- convertirTextoAFrecuencias(String texto): Convierte el texto de entrada en frecuencia de letras(pareja letras con frecuencia respectiva)
- convertirFrecuenciasPalabrasAFrecuenciasLetras(HashMap<String,Integer> frecPalabras): Convierte las frecuencias de palabras en frecuencias de letras
- agregarTextoFrecuencias(String nomT, HashMap<String,Integer> frecuenciaPalabras): Convierte las frecuencias de palabras a frecuencias de letras y agrega un nuevo texto del tipo Frecuencias.
- getNombresTextos(): Devuelve la lista de nombres de textos existentes.
- getAsociacionesVinculadasTexto(String nomT): Devuelve la lista de nombres de teclados vinculados al alfabeto con nombre nomA.
- borrarTexto(nomT): Elimina un text del conjunto del textos que tiene el sistema.
- textosToByteArray(): Convierte un conjunto de textos en ByteArray con el fin de almacenarlos.
- byteArrayToTextos(byte[] bytes): Transforma un conjunto de textos almacenar en un array de bytes al formato original de Texto.

2.2.5 CtrlTeclado

Este controlador se encarga de gestionar los métodos de añadir, modificar y borrar del conjunto de teclados, teclado y el controlador del Algoritmo.

Utiliza una HashMap<String, Teclado> para almacenar los teclados cuyo nombre único se utiliza como clave y el segundo parámetro es el objeto en sí. En el código se encuentra un constructor sin parámetros para inicializar como también unas operaciones que permiten establecer datos (como puntuación o dimensiones) a un teclado en específico. También hay la posibilidad de agregar Alfabetos o Asociaciones de Texto vinculados a ese objeto. Para finalizar, se pueden observar métodos de las clases Conjunto de Teclados (un *getter*, uno para asegurar que el teclado pedido existe y uno que crea un objeto de teclado utilizando el controlador de algoritmo) y algunos auxiliares.

Describimos atributos y métodos de manera explícita.

Atributos:

- ConjuntoTeclados teclados: crea una instancia del conjunto de teclados.
- CtrlTecladoQAP ctrlAlgoritmo: crea una instancia del controlador del algoritmo.

Métodos:

- CtrlTeclado(): constructora del controlador del teclado.
- getContenido(String nomT): Retorna el atributo contendio del teclado con nombre nomT.
- getAlfabeto(String nomT): Retorna el atributo alfabetoVinculado del teclado con nombre nomT.
- getAsociacion(String nomT): Retorna el atributo asociaciónTextoVinculada del teclado con nombre nomT.
- setDimensiones(String nomT, PairInt dimensiones): Setea el atributo dimensiones al valor pasado por parámetro del teclado con nombre nomT.
- agregarAlfabetoVinculado(String nomT, String nomA): Setea el atributo alfabetoVinculado del teclado con nombre nomT al nombre de alfabeto pasado por parámetro.
- agregarAsociacionTextosVinculada(String nomT, String nomAT): Setea el atributo asociaciónTextosVinculada del teclado con nombre nomT al nombre de asociación pasado por parámetro.
- borrarAlfabetoVinculado(String nomT, String nomA): Manda borrar el alfabeto nomA de alfabeto vinculado del teclado con nombre nomT.
- borrarAsociacionTextosVinculados(String nomT, String nomAT): Manda borrar la asociación nomAT de asociación vinculada del teclado con nombre nomT.
- getCjtTeclados(): Retorna el atributo teclados del controlador.
- existeTeclado(String nomT): comprueba si existe el teclado con nombre nomT dentro del conjunto de teclados.

- CrearTeclado(String nomT, AsociacionTextos asociacionTextos, Alfabeto alfabeto): crea un teclado con clave nomT y con atributos alfabetoVinculado = alfabeto y asociaciónTextosVinculada = asociaciónTextos.
- TecladoTieneAlfabetoVinculado(String nomT): Obtiene el nombre del alfabeto vinculado a un teclado específico.
- TecladoTieneAsociacionVinculada(String nomA): Obtiene el nombre de la asociación de textos vinculada a un teclado específico.
- borrarTeclado(String nomT): Borra Teclado con nombre nomT.
- tecladosToByteArray(): Convierte un conjunto de teclados en ByteArray con el fin de almacenarlos.
- byteArrayToTeclados(byte[] bytes): Transforma un conjunto de teclados almacenar en un array de bytes al formato original de Teclado.

2.2.6 CtrlTecladoQAP

Este controlador se encarga de crear un teclado. Tiene una única función que se ocupa de crear un teclado, la cuál es llamada por el controladorTeclado. Se encarga de, a partir de el nombre, alfabeto, asociación y dimensiones necesarias, generar el teclado y crear una nueva instancia de teclado con sus parámetros correspondientes, calculados por el algoritmo usado en cuestión.
Describimos este método, el cual tiene métodos auxiliares:

Métodos:

- crearTeclado(String nomT, AsociacionTextos asociacionTextos, Alfabeto alfabeto): esta función se encarga de crear el teclado y poner los valores correspondientes. Se genera a partir de las letras del alfabeto y de las frecuencias de pares de letras de la asociación de textos. La función genera las matrices de frecuencias y distancias, llamando a la clase Matrices, y a posteriori llama a la clase QAP, pasándole las matrices cómo parámetros, para que genere el teclado. Finalmente recupera el contenido y puntuación del teclado para asignárselo al teclado.
- calculaDimensiones(int n): función que a partir del número de teclas se ocupa de generar las dimensiones del teclado.

2.3 Drivers

2.3.1 InOut

Este controlador tiene como función principal controlar y gestionar la entrada/salida del proyecto en sí, pero solo en Terminal. Ahora describimos los atributos y funciones de manera explícita.

Atributos:

- Scanner scanner: Atributo necesario para la entrada salida de la terminal.
- InOut(): Constructor de la clase InOut. Inicializa un objeto Scanner para la entrada estándar.

Métodos:

- leerString(): Lee una línea de texto desde la entrada estándar.
- leerEntero(): Lee un número entero desde la entrada estándar.
- leerCaracteresDeTerminal(String line): Convierte una cadena de texto en una lista de caracteres.
- leerCaracteresDeArchivo(String nombreArchivo): Lee caracteres de un archivo y los almacena en una lista. Puede lanzar excepciones si hay un error al leer el archivo o si el contenido no es válido.
- leerTextoFrecuenciasPalabras(int n): Lee líneas de texto que contienen palabras y sus frecuencias y almacena la información en un HashMap.
- leerPalabrasDeArchivo(String nombreArchivo): Lee todas las líneas de un archivo y concatena el contenido en un solo String. Lanza una excepción en el caso que haya un error en el archivo que se desea leer.
- alfabetoValido(String entrada): Verifica si la cadena de entrada representa una secuencia de caracteres válida, es decir, caracteres individuales separados por espacios.
- cerrarScanner(): Cierra el objeto Scanner utilizado para la entrada estándar si aún no ha sido cerrado.

2.3.2 DriverPresentacion

2.3.3 DriverPersistencia

El encargado de probar todas las funcionalidades de lo que envuelve la persistencia del proyecto, es decir, del proceso de almacenamiento y carga. Los atributos y métodos necesarios para esto son los siguientes:

Atributos:

- InOut inOut: Crea una instancia del driver InOut.

- CtrlDominio ctrlDominio: Inicializa un objeto CtrlDominio con el fin de comunicar este driver con lo que controla el proyecto en sí

Métodos:

- DriverPersistencia(): Constructora del driver de la funcionalidad principal.
- agregarAlfabetoPorTerminal(): Muestra por pantalla mensajes para introducir información por terminal del alfabeto que se quiere crear.
- borrarAlfabeto(): Borra el Alfabeto elegido por el usuario (lo escribe por terminal).
- agregarAlfabetoPorArchivo(): Muestra por pantalla mensajes para introducir información por archivo del alfabeto que se quiere crear y se crea.
- imprimirNombresAlfabetos(): Imprime por pantalla una lista de los alfabetos que hay en el sistema.
- consultarContenidoAlfabeto(): Imprime por pantalla el contenido del teclado que se desea consultar, si existe.
- agregarTextoPorTerminal(): Muestra por pantalla mensajes para introducir información por terminal del texto que se quiere crear y se crea.
- agregarTextoPorArchivo(): Muestra por pantalla mensajes para introducir información por archivo del texto que se quiere crear y se crea.
- borrarTexto(): Borra el Texto elegido por el usuario que lo elige escribiéndolo por terminal.
- imprimirNombresTextos(): Imprime por pantalla una lista de los textos que hay en el sistema.
- imprimirNombresAsociaciones(): Imprime por pantalla una lista de las asociaciones de textos que hay en el sistema.
- consultarContenidoTexto(): Imprime por pantalla el contenido del texto que se desea consultar, si existe.
- crearAsociacion(): Crea una asociación nueva con textos existentes que haya en el sistema.
- agregarTeclado(): Muestra por pantalla mensajes para introducir información por terminal del teclado que se quiere crear y se crea.
- imprimirNombresTeclados(): Imprime por pantalla una lista de los teclados que hay en el sistema.
- consultarContenidoTeclado(): Imprime por pantalla el contenido del teclado que se desea consultar, si existe.
- imprimirPosiblesDimensiones(String nomA): Genera y devuelve un HashMap que representa las posibles dimensiones para organizar un alfabeto en una matriz.
- main(String[] args): Función principal, es lo que ve el usuario al tratar con el proyecto y las diferentes opciones posibles
- borrarAsociacionTexto(): Borra la Asociación de Textos elegido por el usuario (lo escribe por terminal).
- borrarTeclado(): Borra el Teclado elegido por el usuario (lo escribe por terminal).

- muestraMetodos(): Imprime por pantalla todos los métodos que el usuario es capaz de realizar
- volverMenu(): Imprime la acción que el usuario debe realizar para volver al menú principal

3. DIAGRAMA DE LAS CLASES Y CONTROLADORES DE LA CAPA DE PERSISTENCIA

A continuación mostramos el diseño del diagrama de las clases y controladores de la capa de persistencia. El diagrama se puede encontrar también en la carpeta DOCS.

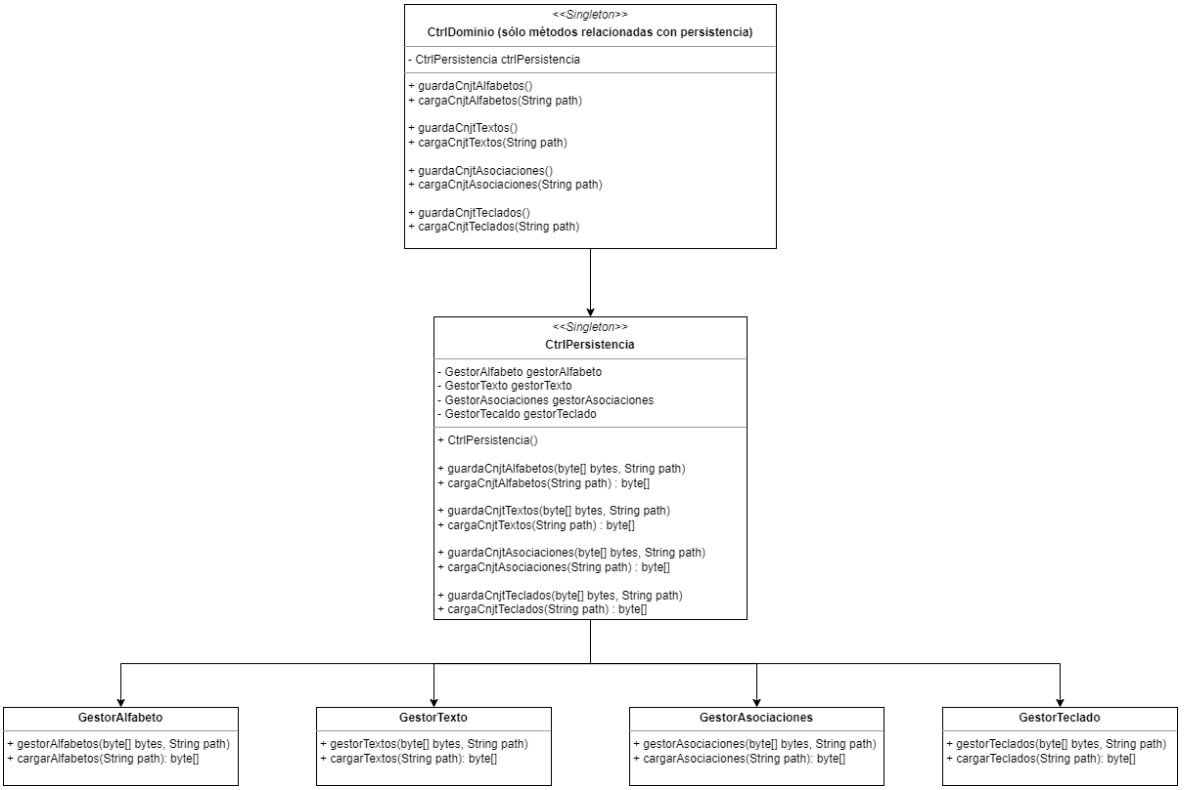

Figura 3: Diagrama de capa persistencia

3.1 Explicación de las clases de persistencia

3.1.1 GestorAlfabeto

Esta clase se encarga de cargar y volcar la información guardada en conjuntoAlfabetos. La clase no cuenta con ningún atributo, sin embargo, hay los siguientes métodos:

Métodos:

- gestorAlfabetos(byte[] bytes, String path): Guarda los datos del byte array en disco en el path marcado por el atributo path.
- cargarAlfabetos(String path): Devuelve un byte array con los datos leídos del fichero marcado por el atributo path.

3.1.2 GestorTexto

Esta clase recoge los datos en relación con conjuntoTextos. A continuación se encuentran los métodos, no hay atributos:

Métodos:

- gestorTextos(byte[] bytes, String path): Guarda la información contenida en el conjunto de bytes (byte array) en un archivo en la ubicación indicada por el atributo "path".
- cargarTextos(String path): Retorna un conjunto de bytes (byte array) que contiene los datos leídos desde el archivo cuya ubicación está especificada por el atributo "path".

3.1.3 GestorAsociacion

Esta clase se encarga de manejar todas las operaciones que afectan al conjuntoAsociaciones al cargar, modificar o eliminar información. No contiene atributos, pero incluye los siguientes métodos:

Métodos:

- gestorAsociaciones(byte[] bytes, String path): Almacena los datos del conjunto de bytes (byte array) en un archivo, guardándolos en la ubicación especificada por el atributo "path".
- cargarAsociaciones(String path): Genera y devuelve un conjunto de bytes (byte array) que corresponde a los datos obtenidos mediante la lectura del archivo ubicado en la ruta indicada por el atributo "path".

3.1.4 GestorTeclado

Esta categoría se ocupa de gestionar todo lo relacionado con la manipulación de información en el conjuntoTeclados, ya sea al cargar, modificar o eliminar datos. Sus funciones comprenden lo siguiente:

Métodos:

- gestorTeclados(byte[] bytes, String path): Esta función tiene la responsabilidad de leer el contenido de un archivo localizado en la ruta indicada por el atributo "path" y luego devuelve esa información como un conjunto de bytes, conocido como byte array.
- cargarTeclados(String path): Esta función se encarga de leer el contenido de un archivo situado en la ruta especificada por el atributo "path" y luego devuelve esa información en forma de byte array.

3.2 Explicación del controlador de persistencia

3.2.1 CtrlPersistencia

Es el encargado de controlar toda acción que involucre los gestores que componen toda la unidad de persistencia del proyecto, la que se encarga de los procesos de guardar y volcar de las

diferentes clases del dominio. En nuestro caso y, como ya explicamos en los anteriores puntos, nos centramos en los conjuntos (ya sea de Alfabeto, Texto, Asociación de Texto y Teclado).

Atributos:

- GestorAlfabeto gestorAlfabeto: Crea una instancia del gestor perteneciente a Alfabeto
- GestorTexto gestorTexto: Crea una instancia del gestor perteneciente a Texto
- GestorAsociaciones gestorAsociaciones: Crea una instancia del gestor perteneciente a Asociación de Textos.
- GestorTeclado gestorTeclado: Crea una instancia del gestor perteneciente a Teclado.

Métodos:

- CtrlPersistencia(): Constructora, inicializa los atributos.
- guardaCnjtAlfabetos(byte [] bytes, String path): Almacena el conjunto de Alfabetos en path.
- cargaCnjtAlfabetos(String path): Retorna el conjunto de Alfabetos cuya localización es path.
- guardaCnjtTextos(byte [] bytes, String path): Almacena el conjunto de Textos en path.
- cargaCnjtTextos(String path): Retorna el conjunto de Textos cuya localización es path.
- guardaCnjtAsociaciones(byte [] bytes, String path): Almacena el conjunto de Asociaciones de Textos en path.
- cargaCnjtAsociaciones(String path): Retorna el conjunto de Asociaciones de Textos cuya localización es path.
- guardaCnjtTeclados(byte [] bytes, String path): Almacena el conjunto de Teclados en path.
- cargaCnjtTeclados(String path): Retorna el conjunto de Teclados cuya localización es path.

4. ESTRUCTURA DE DATOS Y ALGORITMOS UTILIZADOS

4.1 Estructuras de datos

A continuación explicaremos cada una de las estructuras de datos que hemos considerado con tal de crear nuestro generador de teclados, justificando la elección.

4.1.1 Alfabeto

Estructura básica de un alfabeto. Está identificado por un nombre como clave única.

Sus otros atributos son:

→ ArrayList<Character> letras:

Es una lista que contiene, para todos los alfabetos creados, las letras que forman parte del contenido de ese alfabeto.

→ ArrayList<String> tecladosVinculados:

Es una lista que contiene, para todos los alfabetos creados, los nombres de los teclados asociados a ese alfabeto. Puede estar vacía, dependerá de si se ha usado el alfabeto para

generar un teclado. Usamos esta estructura para poder buscar y clasificar por nombre, la clave primaria tanto de alfabeto como de teclado.

Hemos usado el nombre cómo identificador ya que no queremos tener más de un alfabeto guardado con el mismo nombre, ya que eso confundiría al usuario a la hora de seleccionar un alfabeto ya existente.

Por otra parte, hemos escogido que la estructura para la representación de los caracteres sea un ArrayList por varias razones:

- ArrayList nos permite acceder aleatoriamente a sus elementos en tiempo constante O(1). Esto es útil cuando necesitamos obtener caracteres con frecuencia, basándonos en su posición dentro del alfabeto. Es nuestro caso ya que en cualquier momento podemos querer consultar/modificar/eliminar un alfabeto.
- A diferencia de los arrays nativos en Java, ArrayList puede cambiar su tamaño dinámicamente. Esto es muy útil si el número de caracteres en el alfabeto no es conocido o podría cambiar durante la ejecución del programa. De nuevo, al poder consultar/modificar/eliminar, debemos ir consultando el número de caracteres actual, por lo que el ArrayList nos resulta ser una buena opción.
- Por último, ArrayList proporciona una amplia gama de métodos que facilitan las operaciones comunes, como agregar, eliminar y buscar elementos,que son justamente las operaciones que utilizaremos, y además puede hacer que el código sea más legible y conciso.

Finalmente, mencionamos que otras opciones habíamos contemplado y mencionamos algún aspecto que nos haya hecho valorarlas pero que el ArrayList:

- Array nativo (char[]): Tiene un tamaño fijo y no puede expandirse o contraerse dinámicamente, lo que lo hace menos flexible. Además, realizar operaciones como insertar o eliminar elementos es más laborioso y menos eficiente.
- LinkedList: Aunque LinkedList proporciona inserciones y eliminaciones de elementos más eficientes (O(1)), el acceso aleatorio es más lento (O(n)), lo que la hace menos adecuada para operaciones de lectura frecuente.
- HashSet/TreeSet: Mientras que HashSet y TreeSet son útiles para garantizar la unicidad y, en el caso de TreeSet, el ordenamiento, no son ideales para representar un alfabeto donde el orden y la posición son importantes y donde las operaciones de acceso aleatorio son comunes.

4.1.2 Texto

Estructura básica de un texto. Está identificado por un nombre como clave única. Es una clase abstracta que tiene como subclases frecuencias y palabras, las cuales describiremos después del texto. Estas dos subclases representan las dos diferentes maneras de entrar un texto. La primera es un texto (secuencia de palabras) y la segunda es entrar pares de letras con sus frecuencias. Describimos la clase y sus subclases y luego justificamos la elección.

Los otros atributos de texto son:

→ HashMap<String, Integer> frecuenciasLetras:

Independientemente del formato de la entrada, guardaremos en un mapa los pares de letras con sus frecuencias, ya que nos interesará tenerlo guardado de cara a generar el teclado. La clave en el mapa es un String que representa el par de carácteres y el Integer la frecuencia del par de carácteres correspondiente.

→ ArrayList<String> asociacionesVinculadas

Nuestro teclado se generará a partir de asociaciones de textos, que son asociaciones con uno o más textos de los que tenemos disponibles. Estas se describirán más adelante. Esta es una lista que contiene, para todos los textos creados, los nombres de las asociaciones de textos a las que pertenece ese texto, o dicho de otro modo, las asociaciones vinculadas al texto.

Palabras:

Estructura básica de un texto que se ha creado mediante una lista de palabras, o dicho de otra manera, un texto convencional. Su único atributo es el contenido, un String texto, además de los atributos heredados de la clase texto, al ser subclase de ella.

Las funciones son una constructora, un getter del atributo y una función auxiliar para imprimir las frecuencias que se construyen a partir del texto.

Frecuencias:

Estructura básica de un texto que se ha creado mediante una lista de pares de caracteres y la frecuencia de esos caracteres. Más explícitamente, lo que representa es la frecuencia que tiene en un texto normal el par de caracteres guardados. De manera similar a palabras, solo tiene un atributo representando a su contenido, en este caso un HashMap<String,Integer> donde String es el par de caracteres e Integer la frecuencia.

La elección de una clase abstracta Texto con dos subclases Palabras y Frecuencias se justifica por la necesidad de tratar diferentes tipos de entradas textuales bajo un mismo concepto general, permitiendo un manejo polimórfico y una amplia reutilización de código.

1. HashMap<String, Integer> para pares de caracteres y sus frecuencias (en Texto):
   - Eficiencia en Acceso y Búsqueda: HashMap proporciona acceso rápido a las frecuencias a partir de pares de caracteres. Esto es crucial para el algoritmo ya que requiere consultar frecuencias de manera eficiente.
   - Clave-Valor: La naturaleza de clave-valor de HashMap es ideal para asociar pares de caracteres con sus respectivas frecuencias.

- Flexibilidad: Permite agregar, eliminar y actualizar pares de caracteres y sus frecuencias con facilidad.

2. ArrayList<String> para nombres de teclados vinculados (en Texto):
   - Eficiencia en la Lista: Ofrece operaciones eficientes para agregar, eliminar y acceder a elementos.

3. String para representar el texto (en Palabras):

   - Simplicidad: Un String es una forma natural y directa de representar un texto. Facilita operaciones como la lectura, escritura y manipulación del texto.

4. HashMap<String, Integer> para frecuencias (en Frecuencias):

   - Consistencia: Usar la misma estructura que en Texto para pares de caracteres y sus frecuencias mantiene la consistencia en el manejo de datos entre diferentes tipos de entrada.
   - Eficiencia y Flexibilidad: Al igual que en Texto, permite un manejo eficiente y flexible de las frecuencias.

En comparación con otras estructuras como TreeMap o LinkedHashMap, HashMap es generalmente más rápido en las operaciones de búsqueda y actualización, aunque no mantiene el orden de los elementos. ArrayList se prefiere sobre LinkedList cuando el acceso aleatorio es más común que las inserciones y eliminaciones en medio de la lista.

Esta combinación de HashMap y ArrayList, junto con el uso de String, ofrece un equilibrio adecuado entre eficiencia, facilidad de uso y adaptabilidad a los requisitos del proyecto.

4.1.3 Asociación de textos

Estructura que define una asociación de textos, la cual está compuesta por uno o más textos. La usamos porque para generar el teclado, que es nuestra funcionalidad principal, lo hacemos mediante una asociación de textos, para poder representar grupos de textos que usen el mismo alfabeto, como por ejemplo, catalán básico (varios textos diferentes de catalán básico).
Está identificado por un nombre como clave única.
Sus otros atributos son:
→ ArrayList<String> textosAsociados:

Es una lista que contiene los nombres de los textos por los cuáles está compuesta la asociación de textos correspondiente. Se elige esta estructura para, de nuevo, tener facilidad para acceder mediante la clave primaria, el nombre.

→ ArrayList<String> tecladosVinculados:

Es una lista que contiene, para todas las asociaciones creadas, los nombres de los teclados asociados a esa asociación. Puede estar vacía, dependerá de si se ha usado la asociación para generar un teclado. Usamos esta estructura para poder buscar y clasificar por nombre, la clave primaria tanto de asociación como de teclado.

→ HashMap<String, Integer> frecuenciasLetras:

Mapa que representa la misma estructura que teníamos en texto para representar los pares de frecuencias, pero para todos los textos. Usamos la clase auxiliar FrequencyComparator para sumar los distintos pares de frecuencias de todos los textos asociados y ponerlos de manera correcta en nuestra estructura. La decisión es la misma que en texto, el mapa es para tener los pares de letras como clave guardados con su frecuencia como valor y poder acceder a ellos fácilmente.

Justificación de las elecciones:

1. ArrayList<String> para textos asociados:
   - Eficiencia en la Gestión de Tamaños Dinámicos: ArrayList maneja eficientemente el cambio en el tamaño de la lista (por ejemplo, al agregar o eliminar textos), lo que es importante ya que la cantidad de textos varía frecuentemente.
2. ArrayList<String> para teclados vinculados:
   - Flexibilidad y Rendimiento en Acceso: Permite un acceso eficiente a los nombres de los teclados, así como facilidades para agregar o eliminar teclados.
3. HashMap<String, Integer> para frecuencias acumuladas:
   - Eficiencia en Búsqueda y Actualización: Al almacenar las frecuencias acumuladas de todos los textos en un HashMap, se garantiza un acceso rápido a las frecuencias para cualquier par de caracteres. Esto es esencial ya que la preparación del algoritmo requiere consultar frecuencias frecuentemente.
   - Facilidad para Agregar Nuevas Frecuencias: Cuando se añade un nuevo texto a la asociación, es fácil actualizar el HashMap con las nuevas frecuencias, manteniendo una visión agregada de todas las frecuencias.

En comparación con otras estructuras, ArrayList y HashMap ofrecen un equilibrio óptimo entre rendimiento y facilidad de uso en este contexto. Otras estructuras, como LinkedList o TreeMap, podrían no ser tan eficientes en términos de acceso aleatorio o actualizaciones rápidas, respectivamente. Por lo tanto, la elección de ArrayList y HashMap para AsociacionTextos está bien fundamentada y alineada con las necesidades de la generación de teclados.

4.1.4 Teclado

Estructura para guardar un teclado. Nuestra funcionalidad es la generación de teclados. Cómo hemos ido diciendo anteriormente, el teclado se genera a partir de una asociación de textos y un alfabeto, usando el algoritmo, el cuál se describe después del teclado, y tiene un contenido.

Está identificado por un nombre como clave única.

Sus otros atributos son:

→ Puntuación: Generada por el algoritmo, nos sirve para generar un teclado lo mejor posible.

→ Dimensiones: Estructura determinada por una clase que hemos implementado, llamada PairInt, que es básicamente un par de dos enteros, que son las filas y columnas del teclado.

→ String alfabetoVinculado: nombre del alfabeto mediante el cuál se ha creado el teclado.

→ String asociaciónTextosVinculado: nombre de la asociación mediante la cuál se ha generado el teclado.

→ char[][] contenido: matriz de char que es la representación del teclado, con las letras asignadas en sus posiciones correspondientes al teclado con mejor puntuación generado por el algoritmo.

Las funciones de la clase son la constructora, los getters y un par de setters, y un par de funciones para agregar o eliminar el alfabeto/asociación vinculados.

Justificación de las elecciones, justificamos las dimensiones y el contenido ya que son los dos atributos que no son básicos:

1. Dimensiones (PairInt): Utilizar una clase como PairInt para representar las dimensiones (filas y columnas) del teclado proporciona una forma clara y estructurada de manejar estas medidas. Esto es más intuitivo y seguro que manejar dos variables separadas para filas y columnas.

2. char[][] contenido: Representar el teclado como una matriz bidimensional de caracteres es una manera natural y directa de mapear la disposición física de un teclado real. Facilita la visualización y la interacción con la disposición del teclado.

4.1.5 Conjuntos

Estructura que representa un conjunto de alfabetos, textos, asociaciones o teclados.. Para todos los conjuntos hemos usado la misma estructura, por simplicidad y para facilitar los accesos y búsquedas.

→ HashMap<String, Alfabeto> alfabetos.

→ HashMap<String, Texto> textos.

→ HashMap<String, AsociacionTextos> asociaciones.

→ HashMap<String, Teclado> teclados.

Es un map que contiene todos los Alfabetos, Textos, Asociaciones o Teclados creados. La clave identificadora es el nombre de cada alfabeto (string) y el segundo parámetro es el tipo (Alfabeto, Texto, AsociaciónTextos, Teclado) con sus atributos y métodos. Hemos escogido

esta estructura porque nos permite almacenar todos los objetos y buscar uno por su clave primaria, el nombre.

Justificación:

1. Acceso Rápido y Búsqueda Eficiente: El uso de un HashMap permite un acceso y búsqueda muy rápidos por clave, que en este caso es el nombre del objeto (alfabeto, texto, asociación, teclado). Esto es especialmente útil ya que necesitamos recuperar o modificar información frecuentemente y de manera eficiente.

2. Unicidad de Claves: El nombre de cada objeto sirve como una clave única, lo que garantiza que no haya duplicados en cada conjunto. Esto es crucial para mantener la integridad de los datos y evitar confusiones.

3. Flexibilidad y Escalabilidad: HashMap puede adaptarse fácilmente a cambios en la cantidad de datos almacenados. El conjunto de objetos crece con el tiempo, y HashMap se maneja bien con el aumento de elementos, manteniendo un rendimiento constante en términos de tiempo de acceso.

4. Simplicidad y Consistencia: Utilizar una misma estructura de datos para todos los conjuntos nos brinda consistencia en el manejo de los diferentes tipos de objetos. Esto simplifica el diseño del software y facilita la comprensión y el mantenimiento del código.

4.2 Algoritmo

Para la generación de teclados, hemos tenido que resolver el *Quadratic Assignment Problem*. Para nuestro primer algoritmo hemos usado *branch and bound*, y hemos calculado la cota de *Gilmore-Lawler*, usando también el algoritmo *Húngaro*. Ahora explicaremos de manera extensa el funcionamiento del algoritmo y justificaremos la elección de las técnicas y estructuras de datos utilizadas.

Para empezar, explicaremos el proceso desde el inicio del algoritmo, para entender todos los pasos y detalles.

Recibiremos un nombre para el teclado, un alfabeto y una asociación de textos. Usaremos el contenido (caracteres) del alfabeto y la lista de frecuencias (almacenada en un HashMap), para generar nuestro teclado.

Para poder usar el algoritmo debemos de generar las matrices de frecuencias y de distancias. La matriz de frecuencias representa las frecuencias entre todos los pares de letras de las asociaciones de textos. La matriz de distancias representa la distancia de Manhattan entre todas las teclas. A partir de estas matrices llamaremos al algoritmo.

Para el algoritmo, usamos de Branch & Bound, con la técnica Lazy, y lo hacemos mediante un dfs.

Las razones por las que hacemos esto son:

Branch & Bound:

- Eficiencia en la Poda: Esta técnica es eficaz para reducir el espacio de búsqueda al podar ramas del árbol de búsqueda que no conducen a una solución mejor que la mejor solución conocida hasta el momento. Esto disminuye significativamente el número de configuraciones que deben examinarse.
- Enfoque Dirigido: Al comparar los límites de las soluciones parciales con la mejor solución actual, se enfoca la búsqueda hacia áreas más prometedoras del espacio de soluciones, evitando el desperdicio de recursos en opciones menos óptimas.

Técnica Lazy:

- Reducción de Cálculos Innecesarios: Al posponer ciertos cálculos hasta que son absolutamente necesarios (es decir, ser "perezoso"), se evita el trabajo innecesario en ramas del árbol de búsqueda que pueden ser podadas más tarde. Esto ahorra tiempo y recursos computacionales.

DFS (Depth-First Search):

- Uso de Memoria Optimizado: A diferencia del BFS (Breadth-First Search), DFS requiere menos memoria ya que sólo necesita almacenar una rama del árbol de búsqueda a la vez.
- Encontrar Soluciones Profundas Rápidamente: DFS es útil para llegar a soluciones profundas en el árbol de búsqueda rápidamente, lo cual es valioso para establecer un buen límite superior temprano en el proceso de Branch & Bound.
- Simplicidad en la Implementación: DFS es conceptualmente más simple y, por lo general, más fácil de implementar en problemas de búsqueda de este tipo.

Para el segundo algoritmo, hemos decidido usar el Simulated Annealing,. La razón principal es porque es un método heurístico y aunque no conseguirá el resultado óptimo como un método exacto, los resultados en tiempo no serán comparables, serán exponencialmente mejores. Explicamos más adelante en el apartado correspondiente un resumen de lo implementado, ya que está implementado de forma sencilla, en la documentación de la siguiente entrega estará mejor estudiado, especialmente a nivel de constantes y probablemente con operadores o soluciones iniciales añadidas.

4.2.1 QAP

Atributos:

→ int [][] teclado: organización del teclado generada por el algoritmo.

→ filas

→ columnas

→ int n = filas*columnas.

→ int [][] matrizFrecuencias: representa la frecuencia de todos los pares de caracteres.

→ int [][] matrizDistancias: representa todas las distancias entre las teclas del teclado.
→ list<Integer> sol: solución generada por gilmore-lawler, para luego generar el teclado a partir de ella.
→ int glBound: valor inicial para calcular la cota de Gilmore-Lawler.
Usaremos la cota glBound cómo cota inicial de Gilmore-Lawler, y el cálculo de las matrices a partir del alfabeto, las frecuencias del conjunto de asociaciones y la distribución del teclado, usando la clase auxiliar Matrices. Con las matrices calculadas llamaremos a Gilmore-Lawler para que continúe con el algoritmo.
Además en QAP tenemos un par de funciones para calcular la puntuación y generar la solución inicial (cálculo de glBound). La solución inicial se basa en calcular N (100) asignaciones aleatorias del teclado, y nos quedamos con la mejor solución de estas N.
Hemos escogido 100 ya que con números más grandes ya no había un cambio significativo en la puntuación inicial y además no hay problemas de tiempo con este número.

Solución Inicial:

- Generación de N Asignaciones Aleatorias: Se calculan N asignaciones aleatorias del teclado y se selecciona la mejor. Esta estrategia tiene varias ventajas:
    - Exploración Diversa: Al generar múltiples configuraciones aleatorias, el algoritmo explora una amplia gama de posibles soluciones, aumentando la probabilidad de encontrar una disposición eficiente desde el principio.
    - Punto de Partida Sólido: La mejor solución entre estas asignaciones aleatorias proporciona un punto de partida sólido para el algoritmo de optimización, lo que puede acelerar la convergencia hacia la solución óptima.
    - Factibilidad y Simplicidad: Es una técnica sencilla y factible para generar soluciones iniciales.

En resumen, estos atributos y la estrategia de solución inicial están diseñados para abordar de manera efectiva el problema del QAP en el contexto de la optimización del teclado, equilibrando la exploración del espacio de soluciones con la eficiencia computacional.

4.2.2 Gilmore-Lawler

→ int[][] matrizFrecuencias : Matriz que determina la frecuencia entre los pares de letras del alfabeto. (por ejemplo, “ab”, 60). Generada por la clase QAP.
→ int[][]matrizDistancias: Matriz que determina la distancia de Manhattan, implementada en la clase Manhattan, entre dos teclas del teclado. Genera la distancia entre todas las teclas.
→ int glBound: Cota mínima, inicializada al generar la solución inicial para tener una poda eficaz.

→ List<Integer> mejorSoluciónParcial. Solución parcial que vamos actualizando a medida que hacemos el dfs en nuestro algoritmo para que al salir tengamos el mejor teclado encontrado (el de menor puntuación).
En esta clase queremos calcular la cota de Gilmore-Lawler. Hacemos el dfs según posición y letra. En Gilmore-Lawler calculamos la matriz de contribución C1+C2 con la que generamos la cota. Se llama al algoritmo Húngaro, pasándole como parámetro la matriz C1+C2.

Matrices C1 y C2:

> Matriz C1: En C1 el elemento (i, k) representa el coste de colocar la i-ésima instalación no emplazada en la k-ésima ubicación, con respecto a las instalaciones ya emplazadas. Los valores de esta matriz se pueden calcular de forma exacta ($O(m(N-m)2) \subseteq O(N\ 3)$).
> Matriz de Distancias (C2): En C2 el elemento (i, k) representa el coste de colocar la i-ésima instalación no emplazada en la k-ésima ubicación, con respecto a las instalaciones aún no emplazadas. Sin embargo, los elementos de esta matriz no se pueden calcular de manera exacta y se han de aproximar. Sea T el vector de tráficos desde la i-ésima instalación al resto (de las no emplazadas) y sea D el vector de distancias desde la k-ésima ubicación al resto (de las no ocupadas); la cota inferior del coste de emplazar i en k resulta del producto escalar del T ordenado crecientemente con D ordenado decrecientemente ($O((N-m)3) \subseteq O(N\ 3)$).

Algoritmo DFS (Búsqueda en Profundidad) y Cálculo de la Cota:

- DFS (Búsqueda en Profundidad): El DFS es utilizado para explorar todas las posibles configuraciones del teclado. En cada nivel del árbol de búsqueda, se coloca una letra en una posición específica del teclado y se explora esta decisión de manera profunda antes de retroceder y probar una alternativa. Esta técnica asegura que se exploren todas las configuraciones posibles.
- Cálculo de la Cota con Gilmore-Lawler: La cota de Gilmore-Lawler se calcula para podar el espacio de búsqueda y descartar aquellas configuraciones del teclado que no sean óptimas. Se utiliza la suma de las contribuciones de las matrices C1 y C2 para cada posible asignación de tecla. Esto proporciona una estimación de la puntuación o coste de una configuración particular del teclado. Si esta cota supera la mejor solución encontrada hasta el momento, la configuración actual se descarta, lo que ahorra tiempo de cálculo.
- Matriz de Contribución C1+C2: Esta matriz combina las contribuciones de la matriz C1 y la matriz C2. Al sumar estas dos matrices, se obtiene una representación del coste de tener ciertas letras en posiciones específicas del teclado.
- Algoritmo Húngaro: Una vez calculada la matriz C1+C2, se utiliza el Algoritmo Húngaro para encontrar la asignación óptima de teclas que minimiza el costo total. Este algoritmo es

efectivo para resolver problemas de asignación y asegura que se encuentre la mejor disposición posible del teclado dadas las frecuencias de letras y las distancias entre teclas.

En resumen, la combinación de estas técnicas y estructuras de datos en la clase Gilmore-Lawler permite realizar un análisis exhaustivo y eficiente de las posibles disposiciones del teclado. El uso de DFS asegura una exploración completa, mientras que la cota de Gilmore-Lawler, apoyada por las matrices C1 y C2 y el Algoritmo Húngaro, garantiza una poda eficaz del espacio de búsqueda y una selección óptima de la disposición del teclado. Todo esto se orienta a crear un teclado que sea ergonómicamente eficiente y que reduzca el esfuerzo de escritura.

4.2.3 Hungarian Algorithm

El algoritmo Húngaro nos sirve para hacer la cota de Gilmore-Lawler más exacta, y de esta manera, hacer una poda más eficaz y mejorar la velocidad de ejecución del algoritmo. Explicamos de manera exhaustiva la clase.

Constantes y Variables de Clase:

- INF: Representa un número extremadamente grande, utilizado para garantizar que ciertos valores sean efectivamente considerados como infinitos. Es fundamental para manejar casos donde ciertas combinaciones deben ser efectivamente descartadas.
- N: El tamaño de la matriz determina la escala del problema de asignación. Esta variable es crucial ya que la dimensión del problema dicta la complejidad del algoritmo.
- maxMatch: Cuenta el número de asignaciones realizadas. Es un indicador clave para determinar cuándo se ha encontrado una solución completa, justificando su terminación.
- Arreglos como priorityKeys, priorityPositions, keyToPosition, etc.: Estas estructuras almacenan información esencial para el proceso iterativo del Algoritmo Húngaro, permitiendo un seguimiento eficiente de las asignaciones y los costos asociados.

  Constructor y Métodos de Inicialización:
- HungarianAlgorithm(int n): Establece las bases del problema de asignación. La inicialización adecuada de estas estructuras es fundamental para el correcto funcionamiento del algoritmo.
- initLabels(int[][] cost): Establece las prioridades iniciales para teclas y posiciones basadas en los costos. Es un paso esencial para comenzar el proceso de asignación y garantizar que se consideren todas las posibles combinaciones.

  Métodos de Búsqueda y Actualización:
- addToTree(...): Al expandir el árbol de asignación, este método asegura que el algoritmo explore todas las posibles asignaciones, actualizando la información de 'slack' que es vital para determinar los siguientes pasos del algoritmo.

- updateLabels(): Este método recalcula las prioridades, permitiendo que el algoritmo ajuste su búsqueda de una solución óptima basada en los resultados anteriores. Es un mecanismo clave para la eficiencia del algoritmo, permitiendo evitar repeticiones innecesarias y concentrarse en las asignaciones más prometedoras.

  Métodos de Asignación:
- augment(int[][] cost): Este método es el corazón del algoritmo, donde intenta encontrar y mejorar las asignaciones. El uso de la recursividad aquí permite al algoritmo explorar eficientemente todas las combinaciones posibles, asegurando que se encuentre la mejor solución.
- hungarianLeastCost(int[][] cost): Calcula la solución final del problema de asignación. La conversión inicial de costos y el ajuste final son cruciales para adaptar el algoritmo a un marco de maximización, lo que es más natural para este tipo de problemas.

Conclusión

Cada componente de HungarianAlgorithm ha sido diseñado para manejar específicamente los desafíos que presenta este tipo de problema, desde la inicialización y actualización de prioridades hasta la exploración exhaustiva de todas las asignaciones posibles. La estructura y los métodos del algoritmo garantizan que se encuentre una solución óptima de manera eficiente, haciendo que sea una elección adecuada para problemas de asignación en una variedad de contextos.

4.2.4 Simulated Annealing

Hemos considerado varias razones para el uso de este algoritmo en el problema de QAP:

- Exploración Eficiente del Espacio de Soluciones: SA es particularmente bueno para explorar espacios de soluciones grandes y complejos, como los que se encuentran en QAP. Su capacidad para aceptar temporalmente soluciones peores le permite escapar de óptimos locales y explorar más ampliamente el espacio de soluciones.
- Manejo de Complejidad del Problema: QAP es un problema NP-difícil, lo que significa que encontrar una solución óptima puede ser extremadamente difícil y consumir mucho tiempo para instancias de gran tamaño, como nos pasaba en el algoritmo anterior. SA proporciona un enfoque práctico para obtener soluciones de buena calidad en un tiempo razonable.
- Ajuste de Parámetros: Los parámetros clave de SA, que son la temperatura inicial, la tasa de enfriamiento y el número de iteraciones, pueden ajustarse para equilibrar entre la exploración del espacio de soluciones y la convergencia a una solución. Esto permite un control más fino sobre el proceso de búsqueda en comparación con otros métodos.

- Robustez en la Obtención de Soluciones: A diferencia de los métodos deterministas, SA tiene una naturaleza estocástica que le permite obtener diferentes soluciones en diferentes ejecuciones, aumentando las posibilidades de encontrar una solución de alta calidad.
- Simplicidad y Facilidad de Implementación: A pesar de su poder y eficacia, el algoritmo de SA es conceptualmente simple y relativamente fácil de implementar y adaptar a problemas específicos como el nuestro, QAP.

En resumen, el uso de Simulated Annealing para resolver QAP es una elección sólida debido a su capacidad para manejar la complejidad del problema, explorar eficientemente el espacio de soluciones, y su flexibilidad para adaptarse a diferentes variantes del problema, todo esto manteniendo un balance entre la calidad de la solución y el tiempo de cómputo.

Nuestras constantes actualmente, a falta de experimentación, son:

1. Temperatura inicial = 1000000
2. Factor de enfriamiento = 0.999
3. Máximo de iteraciones = 1000000

Son constantes que nos han dado valores suficientemente buenos, alrededor del 96/97% del óptimo global, y nos permiten explorar con firmeza el espacio de soluciones.

Nuestra solución inicial es la misma que en el otro algoritmo actualmente, generar N soluciones random y escoger la mejor, pero estamos en proceso de implementación de un greedy que hasta el momento da resultados muy similares, y se basa en por cada tecla que seleccionamos, la siguiente a seleccionar será la tecla con más frecuencia con la tecla actual que quede por seleccionar.

La heurística se basa en el cálculo de la solución, ya que solo tratamos soluciones completas y queremos minimizarla.

Por último, nuestro único operador actualmente es intercambiar teclas, que intercambia un par de teclas de manera aleatoria, y según una función de aceptación determina si aceptamos el cambio o no.

En resumen, este algoritmo ha sido sencillo y eficaz, y nos ha sacado del principal problema de los métodos exactos, el tiempo de ejecución.