Diseño de un teclado

Proyecto de PROP

Mariona Aguilera Folqué - mariona.aguilera@estudiantat.upc.edu
Eneko Sabaté Iturgaiz - eneko.sabate@estudiantat.upc.edu
Miguel Angel Montero Flores - miguel.angel.montero@estudiantat.upc.edu
Pol Ribera Moreno - pol.ribera@estudiantat.upc.edu

Documentación del proyecto presentada por
el subgrupo 41.2

Segunda entrega
Version: 1.0
12/12/2023

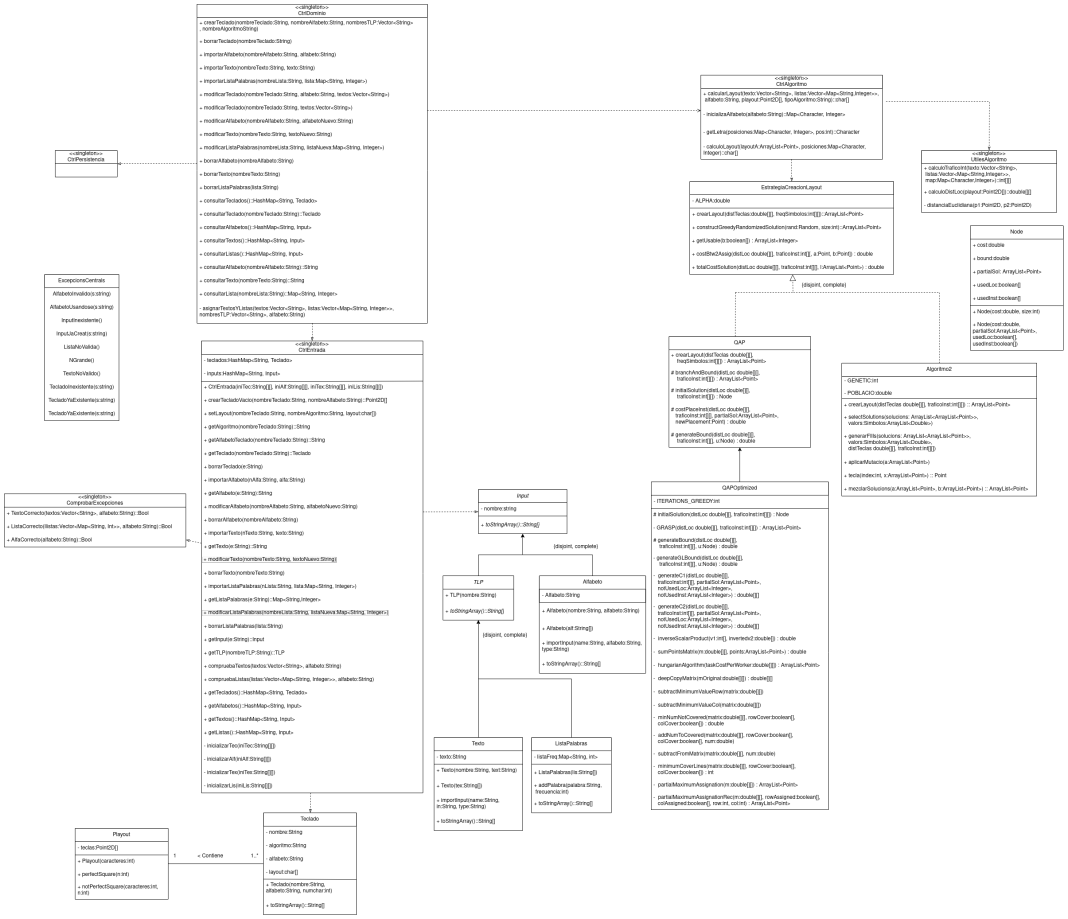
Index

1

Descripción y Diagramas UML

1.1. Capa de Dominio

1.1.1. Diagrama UML

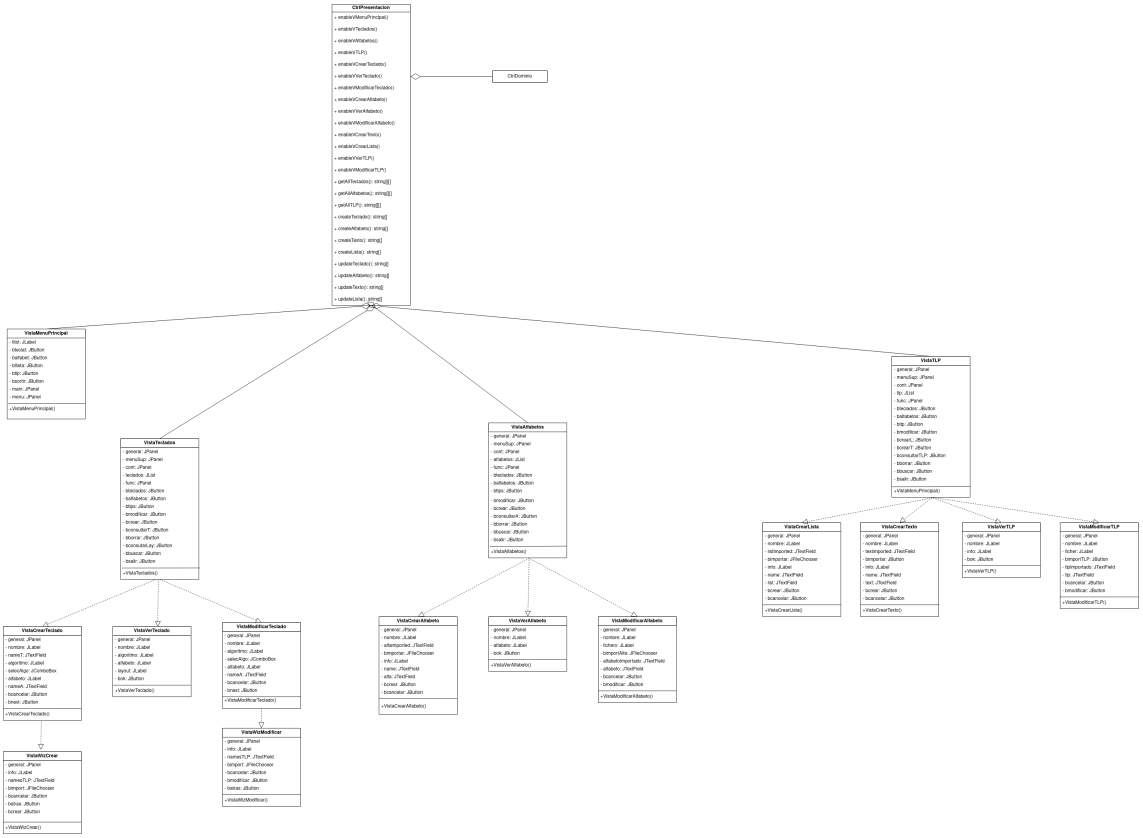


1.1.2. Descripción de métodos y atributos

Hemos realizado la documentación de los métodos y atributos mediante Javadoc. Por lo tanto, para acceder a la documentación abrir Javadoc/index.html o mirar los comentarios encima de la implementación de cada método/atributo.

1.2. Capa de Presentación

1.2.1. Diagrama UML



1.2.2. Descripción de clases

1.2.2.1. VistaMenuPrincipal

Esta vista es la encargada de mostrar la pantalla inicial al ejecutar el programa. Está compuesta por 4 botones, 3 de los cuales nos llevan a vistas distintas para poder ver y gestionar los teclados, alfabetos y textos y listas de palabras. El último boton nos permite cerrar el programa.

1.2.2.2. VistaTeclados

Esta vista se encarga de mostrar todos los teclados creados hasta el momento y las funcionalidades correspondientes a un teclado. En la parte superior de la vista, hay un menú con tres botones, uno para teclados, otro para alfabetos y el último para textos y listas de palabras. También hay un botón en la parte inferior para salir del programa.

En la parte izquierda de la ventana, vemos una barra de búsqueda con una lista de teclados a los que, si accedemos, vemos todas las funcionalidades asociadas a la derecha de la ventana. Cada una de estas funcionalidades tiene asociada un botón que transportará a otra ventana.

1.2.2.3. VistaMenuPrincipal

Esta vista es la encargada de mostrar la pantalla inicial al ejecutar el programa. Está compuesta por 4 botones, 3 de los cuales nos llevan a vistas distintas para poder ver y gestionar los teclados, alfabetos

y textos y listas de palabras. El último boton nos permite cerrar el programa.

1.2.2.4. VistaCrearTeclado

Esta vista se encarga de mostrar los campos necesarios a rellenar para poder crear un teclado correctamente. En concreto, el nombre, el algoritmo y el alfabeto. De las listas y los textos se encarga otra vista, a la cual se accede clicando el botón siguiente. También hay un botón para cancelar la operación, que nos devuelve a la vista teclado.

1.2.2.5. VistaWizCrear

Esta vista se encarga de acabar de recoger los textos y/o listas de palabras necesarios para crear el teclado. Tiene un botón para cancelar la operación, otro para volver a la vista de CrearTeclado y otro para crear el teclado una vez ya esté todo.

1.2.2.6. VistaVerTeclado

Esta vista es la encargada de mostrar toda la información relacionada con el teclado especificado.

1.2.2.7. VistaModificarTeclado

Esta vista se encarga de mostrar los campos necesarios a rellenar para poder modificar el teclado correctamente. En concreto, el nombre, el algoritmo y el alfabeto. De las listas y los textos se encarga otra vista, a la cual se accede clicando el botón siguiente. También hay un botón para cancelar la operación, que nos devuelve a la vista teclado.

1.2.2.8. VistaWizModificar

Esta vista se encarga de acabar de recoger los textos y/o listas de palabras necesarios para modificar el teclado. Tiene un botón para cancelar la operación, otro para volver a la vista de ModificarTeclado y otro para modificar el teclado una vez ya esté todo.

1.2.2.9. VistaAlfabetos

Esta vista se encarga de mostrar todos los alfabetos creados hasta el momento y las funcionalidades correspondientes a un alfabeto. En la parte superior de la vista, hay un menú con tres botones, uno para teclados, otro para alfabetos y el ultimo para textos y listas de palabras. También hay un botón en la parte inferior para salir del programa.

En la parte izquierda de la ventana, vemos una barra de búsqueda con una lista de alfabetos a los que, si accedemos, vemos todas las funcionalidades asociadas a la derecha de la ventana. Cada una de estas funcionalidades tiene asociada un botón que transportará a otra ventana.

1.2.2.10. VistaCrearAlfabeto

Esta vista se encarga de recoger toda la información necesaria para crear un alfabeto, es decir, el nombre y el alfabeto. El alfabeto a crear puede ser importado desde un archivo csv o escrito a mano desde la aplicación.

En la parte inferior derecha de la ventana, hay un botón para crear el alfabeto y otro para cancelar la operación.

1.2.2.11. VistaVerAlfabeto

Esta vista nos muestra toda la información del alfabeto seleccionado, es decir, el nombre y los caracteres. En la parte inferior hay un botón para salir de la ventana, que nos lleva a la ventana de Alfabetos.

1.2.2.12. VistaModificarAlfabeto

Esta vista se encarga de recoger toda la información necesaria para modificar un alfabeto, es decir, el nombre y el alfabeto. El alfabeto a modificar puede ser importado desde un archivo csv o escrito a mano desde la aplicación.

En la parte inferior derecha de la ventana, hay un botón para crear el alfabeto y otro para cancelar la operación.

1.2.2.13. VistaTLP

Esta vista se encarga de mostrar todos los textos y listas de palabras creados hasta el momento y las funcionalidades correspondientes a estos. En la parte superior de la vista, hay un menú con tres botones, uno para teclados, otro para alfabetos y el ultimo para textos y listas de palabras. También hay un botón en la parte inferior para salir del programa.

En la parte izquierda de la ventana, vemos una barra de búsqueda con una lista de textos y listas de palabras a los que, si accedemos, vemos todas las funcionalidades asociadas a la derecha de la ventana. Cada una de estas funcionalidades tiene asociada un botón que transportará a otra ventana.

1.2.2.14. VistaCrearLista

Esta vista se encarga de recoger toda la información necesaria para la creación de una lista de palabras con frecuencia. En el extremo inferior tiene dos botones, uno para crear la lista una vez ya estén los datos introducidos y otro para cancelar la operación y volver a la vista TLP.

1.2.2.15. VistaCrearTexto

Esta vista se encarga de recoger toda la información necesaria para la creación de texto. En el extremo inferior tiene dos botones, uno para crear el texto una vez ya estén los datos introducidos y otro para cancelar la operación y volver a la vista TLP.

1.2.2.16. VistaVerTLP

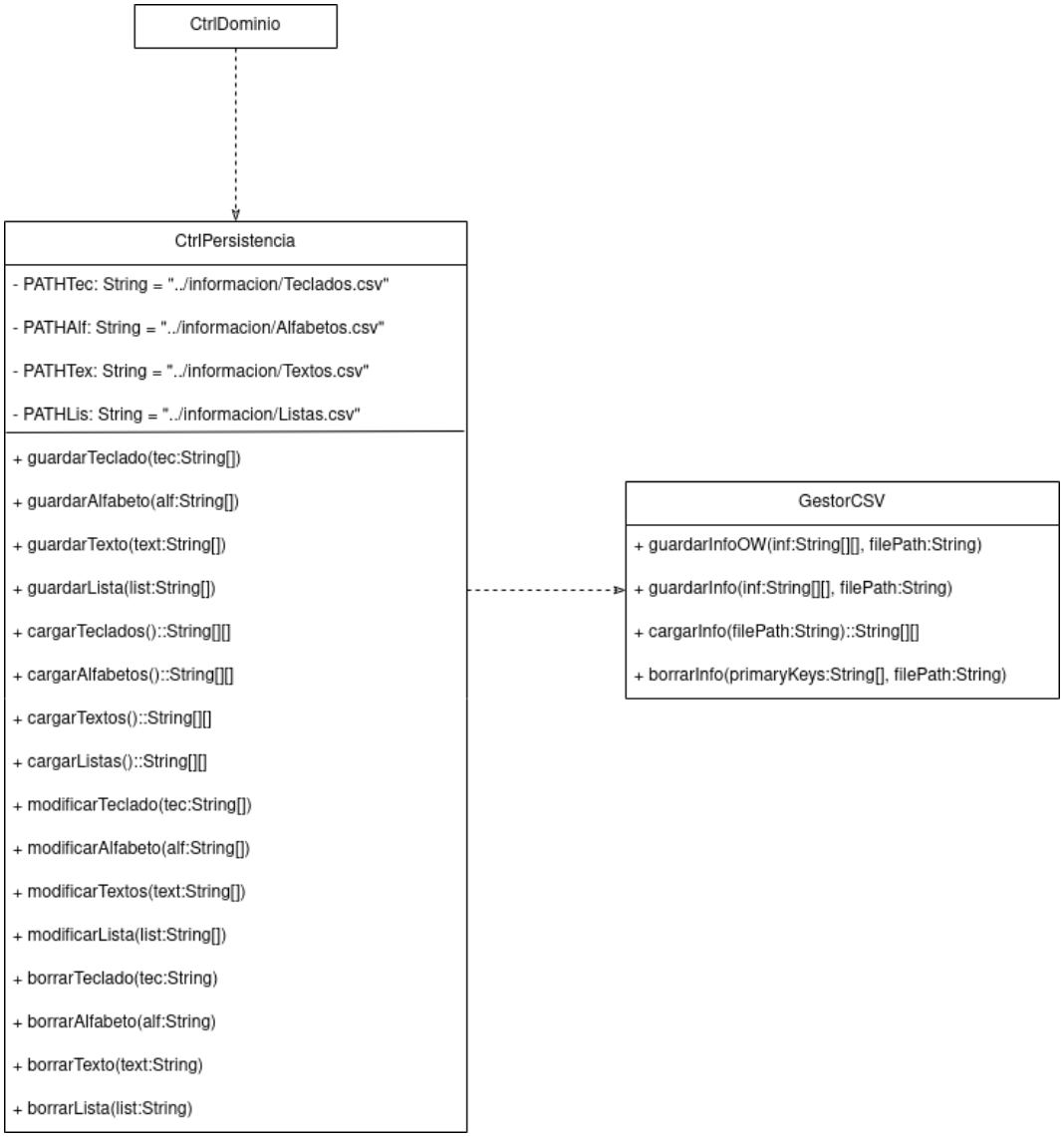
Esta vista nos muestra toda la información del texto o lista de palabras seleccionado, es decir, el nombre y el contenido. En la parte inferior hay un botón para salir de la ventana, que nos lleva a la ventana de TLP.

1.2.2.17. VistaModificarTLP

Esta vista nos recoge toda la información necesaria para modificar un texto o una lista de palabras previamente seleccionado. En el extremo de la ventana hay dos botones, uno para cancelar la operación y el otro para aceptarla.

1.3. Capa de Persistencia

1.3.1. Diagrama UML



1.3.2. Descripción de clases

1.3.2.1. CtrlPersistencia

El controlador de persistencia es el encargado de proporcionar el flujo de información del controlador del dominio y el gestor de archivos CSV con el fin de poder almacenar los datos de la capa de dominio.

1.3.2.2. GestorCSV

La clase GestorCSV se encarga de gestionar el cargado, guardado y borrado de datos, con el objetivo de la persistencia de datos en distintas ejecuciones del programa. Como su nombre indica, este administrará los datos recibidos en archivos csv (guardados en la carpeta FONTS/informacion).

2

Estructuras de Datos y Algoritmos

2.1. Descripción algoritmos

En el siguiente apartado describiremos los algoritmos principales implementados para el funcionamiento del programa.

2.1.1. Branch And Bound

El algoritmo Branch And Bound es el algoritmo principal que usa nuestro programa a la hora de crear layouts. Este algoritmo construye un árbol de soluciones factibles y, cuando llega a las hojas del árbol, compara la solución final de la hoja con la mejor solución hasta el momento. Con el objetivo de mejorar la eficiencia, se introduce el concepto de Bounding: Para cada solución parcial se hace el cálculo de una cota inferior para todas las soluciones completas conseguibles a partir de esta y, si esta cota es mayor a la mejor solución encontrada hasta el momento, se descarta la solución. Aun así, el algoritmo es de coste peor exponencial, lo que lo hace inviable para casos grandes.

Los siguientes algoritmos son los que hemos implementado con el fin de calcular estas cotas inferiores.

2.1.2. GRASP

GRASP, o Greedy randomized adaptive search procedure, es el algoritmo utilizado para econtrar una primera cota inferior como preproceso del BranchAndBound. Este coniste en una serie de iteraciones, donde para cada una se construye una solución greedy randomizada para el problema QAP. Se va guardando la mejor solución encontrada hasta al momento y, cuando termina el bucle, devuelve la mejor solución encontrada de todas las construidas.

2.1.3. Gilmore-Lawler Bound

Calcula una cota inferior para una solución parcial del árbol de soluciones generado por el BranchAndBound. Dada una solución parcial, la cota de Gilmore-Lawler se puede calcular como la suma del coste de las instalaciones ya emplazadas, el coste de las instalaciones que qudean por colocar respecto a las ya colocadas y el coste entre instalaciones aún no emplazadas. Estos dos últimos términos no se puedes saber con exactitud, por eso este algoritmo los intenta aproximar mediante una cota inferior de esos costes. Este algoritmo proporcionará una cota inferior que se usa en el BranchAndBound, con el objetivo de podar las ramas que no pueden proporcionar soluciones óptimas.

2.1.4. Hungarian Algorithm

El Hungarian Algorithm es un algoritmo que resuelve problemas de asignación lineal. Para nuestro caso concreto, lo utilizamos para calcular de una manera más precisa la cota inferior generada por Gilmore-Lawler Bound. Esto se traduce en un mejor funcionamiento del algoritmo BranchAndBound en cuanto a eficiencia de computo.

2.1.5. Genetic Algorithm

Se ha implementado el algoritmo genético adaptado a nuestro problema con el objetivo de crear layouts grandes, ya que el QAP es un algoritmo computacionalmente muy costoso. Para implementarlo hemos generado aleatoriamente n soluciones posibles de teclado desde las cuales hemos ido seleccionando m veces las n/2 mejores soluciones y mezclandolas para conseguir n/2 hijos los cuales quizás sufren de alguna mutación. Una vez terminadas las m iteraciones, el mejor teclado de los que nos quedan será nuestra solución.

2.2. Descripción estructuras de datos

En el siguiente apartado describiremos las estructuras de datos implementadas en las clases principales del programa.

2.2.1. Node

La clase Node implementada representa un nodo en el árbol de soluciones generado por el BranchAndBound. Así, un nodo tiene información sobre el coste de la solución parcial actual, una cota inferior para la solución parcial, la solución en sí, y las instalaciones y ubicaciones utilizadas hasta el momento. Esta estructura de datos es imprescindible para el correcto funcionamiento del BranchAndBound, y por lo tanto, para resolver el QAP.

Se ha implementado la solución parcial en Node como un ArrayList¡Point¿, donde cada punto representa una asignación. Se ha elegido utilizar la estructura de datos Point ya que nos era útil al ser análoga a un pair¡int, int¿, y esto era justo lo que necesitabamos para expresar asignaciones. En cuanto a la ArrayList, hemos decidido usarla en vez de un Array ya que los ArrayList proporcionan un uso dinámico de la memoria, cosa que con los Arrays es imposible ya que estos proporcionan memoria estática. Por otro lado, la implementación de las ubicaciones/instalaciones usadas la hemos hecho mediante Arrays, ya que en este caso sí que nos vale simplemente con memoria estática

2.2.2. QAP

Las dos estructuras de datos más importantes para implementar la clase QAP (y QAPOptimized) son la PriorityQueue y Node. La PriorityQueue es usada en el algoritmo BranchAndBound para almacenar y poder coger las nodos del árbol de soluciones parciales. Esta estructura nos es ideal ya que tratamos el problema de QAP mediante una estrategia .^{ea}ger", y por lo tanto necesitamos una estructura de datos mediante la que guardar las soluciones parciales y tratar las "mejores"soluciones cuanto antes (por eso es de prioridad). La segunda estructura es el Node, la cual utilizamos como elementos contenidores de la cola de prioridad. Esta estructura es totalmente imprescindible ya que es la forma que tenemos de guardar la información de la solución actual para un cierto nodo del árbol de soluciones.

2.2.3. AG

Para implementar el algoritmo hemos usado dos estructuras de datos: una ArrayList¡ArrayList¡Point donde guardaremos todas las soluciones posibles a nuestro problema (poblacion) y un ArrayList¡Double¿donde guardaremos la calidad de cada solucion para no tener que calcularla cada vez que la necessitamos.

En la representacion de las soluciones hemos usado un arraylist de arraylist de point ya que el teclado que necessitamos devolver tiene que tener la estructura ArrayList¡Point¿, entonces como queremos una lista de teclados los hemos puesto en otra array para poder manipularlos facilmente. En consecuencia, para mantener la cohesion del codigo, tambien hemos usado una arraylist para representar los valores (el primer valor siempre corresponde a la primera solucion y asi succesivamente)

2.2.4. CtrlEntrada

Con el objetivo de la persitencia de datos entre la ejecución de comandos del programa, se han creado estructuras de datos que mantengan el estado del programa cuando se realiza un cambio en el sistema.

Para implementar el guardado de teclados hemos utilizado un HashMap¡String, Teclado¿, ya que nos permite operar con teclados usando su nombre como identificador de una manera eficiente. En cuanto a los Inputs hemos utilizado también un HashMap¡String, Input¿, por la misma razón que los teclados. Además nunca tendremos dos inputs con el mismo nombre, así que usar esta estructura no dará error.

2.2.5. Playout

Hemos utilizado un array de Point2D (Point2D[]) para guardar las teclas. La idea inicial era utilizar una matriz que contuviese los puntos que iban a ocupar las teclas en el layout fisco. Sin embargo, para layouts con un número impar de caracteres, la matriz quedaba con espacios en null. Para evitar ocupar más espacio del necesario, decidimos que el atributo de la clase Playout sería un array con el tamaño exacto que se requería.

2.2.6. Teclado

Para guardar el Layout que tiene el teclado, hemos utilizado un array de caracteres (char[]). Utilizamos un char[] frente a un String debido a que nos parece más sencillo la asignación del layout con el Playout, puesto que al iterar por un String se debe llamar a las funciones charAt() o toCharArray().

2.2.7. ListaPalabras

Para tener la lista de palabras con frecuencia, hemos usado un Map¡String, Integer¿, para poder enlazar fácilmente cada palabra con su frecuencia.

2.2.8. ComprobarExepciones

Hemos usado un HashSet¡Character¿en la función de comprobar si un alfabeto es válido, ya que en él ponemos todos los caracteres vistos hasta el momento y si alguno se repite sabremos que el alfabeto no es válido.