PROP 32.2

Disseny d’un teclat

Facultat d’Informàtica de Barcelona
Curs 2023-2024
Quadrimestre Tardor

Daniel Cañizares
Jordi Otal
Pau Rambla

TAULA DE CONTINGUTS

1. Introducció....3
2. Capa persistència....4
CtrlPersistenciaFile....4
3. Capa domini....5
3.1 Model....5
Alfabet (abstracte)....5
BranchBound....6
CompleteAssignation....7
Generador....8
Genetic....8
Hungarian....10
Layout....11
Pair....12
Strategy....12
Teclat....12
Text....13
Words....13
3.2 Controladors domini....14
CtrlDomini....14
Factoria....15
4. Capa presentació....16
CtrlAfegirAlfabet....16
CtrlAfegirLayout....16
CtrlAfegirTeclat....16
CtrlEliminar....16
CtrlLlistaAlfabets....16
CtrlLlistaLayouts....16
CtrlLlistaTeclats....16
CtrlModificarTeclat....16
CtrlMostrarAlfabet....17
CtrlMostrarLayout....17
CtrlMostrarTeclat....17
CtrlPantallaInformativa....17
CtrlPantallaInici....17
CtrlPreMostrarTeclat....17
CtrlPresentacio....17
CtrlProvarTeclat....19
PantallaInformativa....19
Utils....19

1. Introducció

En aquest document es descriuen totes les classes del projecte. En primer lloc trobem les classes que pertanyen a la capa de persistència, seguides de la capa de domini i per finalitzar la capa de presentació. Cada classe té una petita descripció, seguida de les definicions dels seus atributs i les seves operacions (obviant els getters, els setters i algunes funcions no gaire rellevants de la classe).

2. Capa persistència

CtrlPersistenciaFile

És una classe que s'encarrega d'implementar la persistència dels Alfabets, Teclats i Layouts. Controla un directori, per cada tipus, on es guarden els diferents atributs dels Objectes existents en un fitxer **JSON**. Permet al Controlador de domini desar l'estat dels objectes Alfabet, Teclat i Layout creats per l'usuari.

3. Capa domini

A continuació es descriuen les classes de la capa domini. Les classes estan separades en 2 packages. El primer, el Model, són totes les classes que conformen el diagrama de classes del model del projecte. El segon package, el controlador domini, són les classes controladores de tot el

3.1 Model

Alfabet (abstracte)

Aquesta classe guarda tota la informació sobre un Alfabet i gestiona l'entrada d'un nou Alfabet. La classe Alfabet és abstracte donat que segons el tipus d'entrada es crearà una subclasse d'aquesta.

Atributs:

- *private String nom*
    - Atribut que identifica un Alfabet d'un altre (clau primària)
- *protected Map<Character, Double> characters*
    - Guarda la relació entre una lletra de l'Alfabet i la seva freqüència, és protected donat que les subclasses necessiten d'aquesta estructura
- *protected double[][] frequencies*
    - La informació relacionada amb la probabilitat que donada una lletra aparegui la següent, és protected donat que les subclasses necessiten d'aquesta estructura
- *private int size*
    - Mida de l'Alfabet (nombre de caràcters que té)
- *private Character[] abecedari*
    - Un array amb les lletres de l'abecedari

Operacions:

- *protected abstract void read(String path)*
    - Operació abstracte que implementen les subclasses. Cada subclasse està implementada diferent segons el tipus d'entrada que es doni per a llegir l'Alfabet
- *public void readInput (String path)*
    - Crida a l'operació read i cada subclasse farà les seves operacions
- *protected Map<Character, Double> processCharacters (String text, int length, Map<Character, Double> map)*
    - Calcula les aparicions de cada lletra en el text i les posa en el map que retorna
- *protected void processFrequencies (String text, int length)*
    - Calcula el nombre de vegades que donada una lletra aparegui la següent
- *private int findIndex (Character c)*
    - Retorna la posició de la lletra pasada com a paràmetre en l'abecedari
- *protected void calculateFrequencies ()*

    - Calcula les freqüències que donada una lletra aparegui la següent
- *protected void calculateCharacters (int length)*
    - Calcula les probabilitats de cada lletra en el Map characters

BranchBound

Classe encarregada d'executar l'algorisme de branching, i retornar una solució a partir de la qual es genera el teclat.

Atributs:

- *private double[] AbsoluteFreqs*
    - Array de freqüències absolutes de cada caràcter
- *private double[][] Frequencies*
    - Matriu de freqüències per a cada parella de caràcters (i,j)
- *private double[][] Distancies*
    - Matriu de distàncies per a cada parella de tecles (i,j)
- *private int n_size*
    - Nombre de caràcters = Nombre de tecles (l'array de freqüències és de mida n_size, i les matrius n_size*n_size)

Operacions:

- *private void printProgressBar(int percentage)*
    - Pinta una barra de progrés que es va actualitzant en funció del percentatge que rep com a paràmetre.
- *private double cost(int c1, int p1, int c2, int p2)*
    - Retorna el cost de l'aresta entre (c1,p1) i (c2,p2) on: c1 i c2 són els IDs de caràcter ; p1 i p2 són les posicions (tecles) dels respectius caràcters
- *private double[][] matrixC1(ArrayList<Integer> partialSol, ArrayList<Integer> missingChars)*
    - Retorna la matriu C1, necessària per al càlcul del Hungarian Algorithm. Matriu C1: Files [i] - Caràcters no emplaçats; Columnes [j] - Tecles (posicions) no emplaçades
- *private double[] vectorT(int char_id, ArrayList<Integer> missingChars)*
    - Retorna el vector T, necessari per al càlcul de la matriu C2. Vector T: Vector de freqüències entre caràcter [i] i la resta de caràcters no emplaçats. Ordre CREIXENT.
- *private double[] vectorD(int tecla_id, ArrayList<Integer> missingChars)*
    - Retorna el vector D, necessari per al càlcul de la matriu C2. Vector D: Vector de distàncies entre tecla [j] i la resta de tecles no emplaçades. Ordre DECREIXENT.
- *private double scalarProduct(double[] vecA, double[] vecB)*
    - Retorna el producte escalar de dos vectors de mida igual.

- *private double[][] matrixC2(ArrayList<Integer> partialSol, ArrayList<Integer> missingChars)*
    - Retorna la matriu C2, necessària per al càlcul del Hungarian Algorithm. Matriu C2: Files [i] - Caràcters no emplaçats; Columnes [j] - Tecles (posicions) no emplaçades
- *private double[][] matrixSum(double[][] A, double[][] B)*
    - Retorna la suma de dues matrius de mida igual.
- *private double calcF1(ArrayList<Integer> solParcial)*
    - F1 = Suma del cost de les arestes entre tots els caràcters ja emplaçats en una tecla.
- *private double costHungarian(ArrayList<Integer> solHungarian, double[][] matHungarian)*
    - Retorna el cost de l'assignació òptima calculada per Hungarian Algorithm. El ArrayList de la solució hungarian és de la forma: Index = Fila de la matriu (= index fila indica CARÀCTER); Value = Columna on hi ha el 0 de la solució (= index columna indica TECLA).
- *private double cotaGilmore(ArrayList<Integer> solParcial, ArrayList<Integer> charsMissing)*
    - Retorna el valor de la cota Gilmore, calculada en funció d'una solució parcial de la qual disposem. Es calcula com a la suma de tres termes, F1+F2+F3.
- *private int[] mostFrequentChars(int n_chars)*
    - Retorna un array que conté els índexs dels n_chars caràcters més freqüents. Serveix per a executar l'algoritme de branching de forma greedy, és a dir, que comencem amb una solució parcial on ja tenim alguns elements emplaçats.
- *private ArrayList<Integer> algorithm()*
    - Retorna un ArrayList amb el millor emplaçament possible aproximat per al Quadratic Assignment Problem, fent servir un algoritme de Branching similar al eager, amb un enfocament greedy per a millorar el temps d'execució, i una funció de bounding coneguda com a cota Gilmore, que es calcula amb l'ajuda del Hungarian Algorithm.
- *public ArrayList<Integer> generarTeclat(double[][] freq_matrix, double[] abs_frequencies, double[][] dist_matrix)*
    - Actualitza/estableix els valors dels atributs de la classe i retorna la solució aproximada per al Quadratic Assignment Problem.

CompleteAssignation

Classe encarregada de trobar la millor assignació possible donada una matriu, s'utilitza tant en passos intermitjos de l'Hungarian com per a trobar l'assignació completa final. La seva implementació és crítica per a l'eficiència de l'algorisme.

Atributs:

- *private double [][] matrix*
    - Matriu de la que volem trobar l'assignació més completa.
- *private int n*
    - mida de la matriu quadrada.

- *private boolean[] columnLabel*
    - Vector auxiliar del backtracking, per a marcar columnes amb zero.
- *private int[] currentAssig*
    - Assignació actual que esta processant l'algorisme.
- *private int[] bestAssig*
    - Millor assignació trobada per l'algorisme fins al moment.
- *private int bestLines*
    - Nombre més gran de línies.

Operacions:

- *public CompleteAssignation(double[][] mat)*
    - Creadora de la classe, per a inicialitzar tots els atributs necessaris per al càlcul de la millor assignació.
- *private void backtrackingAssig(int row, int currLines)*
    - Algorisme de backtracking que calcula la millor assignació possible = És el coll d'ampolla de totes les classes que l'implementen.
- *public int[] mostCompleteAssig()*
    - Retorna la millor assignació possible, és a dir, la millor combinació de zeros en files i columnes diferents, utilitzant un algorisme de backtracking.

Generador

La classe Generador "comunica" el Teclat i el Strategy que genera el teclat. Adapta les estructures de dades que demanen, i retorna el teclat generat.

Genetic

Classe encarregada de calcular la millor solució possible aproximada utilitzant un algorisme de tipus genètic.

Atributs:

- *private final int population_size = 3000*
    - Nombre d'individus (= mida de la població)
- *private final int generations = 1500*
    - Nombre de generacions
- *private final int mutation_percent = 30*
    - Percentatge màxim de parelles subjectes a mutació. No pot ser mai superior al 50%
- *private double[][] Frequencies*
    - Matriu de freqüències
- *private double[][] Distancies*

    - Matriu de distàncies
- *private int max_pair_mutation*
    - Nombre màxim de mutacions que pateix un individu (es calcula en temps d'execució)
- *private int problem_size*
    - Mida dels individus
- *private Random rand*
    - Instancia de la classe random per a cridar les funcions posteriorment
- *private ArrayList<ArrayList<Integer>> Population*
    - ArrayList que emmagatzema els individus de la població
- *private double[] Fitness*
    - Vector que emmagatzema els costos dels individus de la població

Operacions:

- *private double edgeCost(int c1, int p1, int c2, int p2)*
    - Retorna cost de l'aresta entre (c1,p1) i (c2,p2)
- *private double fitnessScore(ArrayList<Integer> perm)*
    - F1 = Suma del cost de les arestes entre tots els caràcters ja emplaçats en una tecla.
- *private int randInt(int range)*
    - Retorna un valor INT aleatori entre 0 i el valor de range.
- *private int randIntExcluding(int range, ArrayList<Integer> exclude)*
    - Retorna un valor INT aleatori entre 0 i el valor de range, complint la condició de que el valor de retorn no pot ser mai un valor que ja existeix dins del ArrayList exclude.
- *private int randIntExcluding(int range, int exclude)*
    - Retorna un valor INT aleatori entre 0 i el valor de range, complint la condició de que el valor de retorn no pot ser mai igual al valor de exclude.
- *private ArrayList<Integer> randomIndividual(int size)*
    - Genera un individu de la població de forma aleatoria, per tant retorna un ArrayList amb valors entre 0 i size, ordenats aleatòriament i sense repeticions.
- *private void setInitialPopulation()*
    - Genera una població inicial aleatoria.
- *private void updateFitness()*
    - Actualiza el vector Fitness amb els costos de la població actual.
- *private int[] obtainSelection(int top_percentage)*
    - Retorna un vector que conté els índexs del top_percentage% d'individus amb menor cost dintre la població actual.
- *private ArrayList<Integer> performCrossover(ArrayList<Integer> A, ArrayList<Integer> B)*
    - Realitza la funció de Crossover entre dos individus A i B, i retorna un individu fruit de la mescla dels dos pares.
- *private ArrayList<Integer> randomMutation(ArrayList<Integer> input_individual)*

    - Realitza la funció de mutació d'un individu, intercanviant la posició d'entre [0 i n_pars] parelles de caràcters escollides aleatòriament.
- *private ArrayList<ArrayList<Integer>> nextGenPopulation()*
    - Crea la següent generació de població a partir de la població actual, utilitzant les funcions de crossover i mutació.
- *private ArrayList<Integer> algorithm()*
    - Calcula la millor solució possible, a partir d'una població inicial aleatoria i l'evolució d'aquesta població durant el nombre definit de generacions.
- *public ArrayList<Integer> generarTeclat(double[][] freq_matrix, double[] abs_frequencies, double[][] dist_matrix)*
    - Retorna el millor teclat generat utilitzant un algorisme genetic.

Hungarian

Classe encarregada d'executar l'Hungarian Algorithm, algorisme el qual a partir d'una matriu ens ofereix una assignació òptima, a partir de la qual obtenim una aproximació del càlcul dels termes 2 i 3 de la cota de Gilmore. Requereix de la classe CompleteAssignation.

Atributs: *(No n'hi ha)*

Operacions:

- *private static boolean[][] calcMinimumLines(double[][] mat)*
    - Retorna un array que conté dues arrays, una que indica les files cobertes i una altra que indica les columnes cobertes. (Descriu els passos 1-4 de l'algorisme)
- *private static int numberOfLines(boolean[][] mat)*
    - Retorna el nombre de línies necessàries per a cobrir tots els zeros de la matriu. La matriu d'entrada ha de ser la que ens retorna calcMinimumLines !!!
- *private static double minValRow(double[][] mat, int row)*
    - Retorna el valor mínim de la fila row en la matriu mat.
- *private static double minValColumn(double[][] mat, int col)*
    - Retorna el valor mínim de la columna col en la matriu mat.
- *private static double minNonCovered(double[][] mat, boolean[][] covLines)*
    - Retorna el valor mínim no cobert en la matriu mat.
- *private static double minMatrix(double[][] mat)*
    - Retorna el valor mínim de la matriu mat.
- *public static ArrayList<Integer> hungarianAlgorithm(double[][] mat)*
    - Retorna l'assignació òptima calculada per l'Hungarian Algorithm. (Descriu els passos de l'algorisme)

Layout

Aquesta classe representa els layouts d'un teclat, les tecles, quina id tenen, a on van en la distribució i a quina distància estan les unes de les altres.

Atributs:

- *private int mida*
    - La mida del Layout és la seva clau primària, no pot haver dos amb la mateixa mida. Representa el número de posicions disponibles.
- *private int[] mov_x*
    - Un array d'enters que indica les coordenades de l'eix x en el cercle inicial de la generació de la distribució.
- *private int[] mov_y*
    - Un array d'enters que indica les coordenades de l'eix y en el cercle inicial de la generació de la distribució.
- *private double[][] distancies*
    - Una matriu de double que guarda les distàncies entre dues ids.

      (distancies[id1][id2] → distancia entre id1 i id2).
- *private List<Pair<Integer, Integer>> coordenades*
    - Una llista de Pair<Integer, Integer> indica per cada id les coordenades a la distribució del layout.

      (coordenades.get(id1) → dona el Pair<Integer, Integer> que representen les dues coordenades de id1)
- *private int[][] distribucio*
    - Una matriu d'enters que guarda la id que hi ha a cada parell de coordenades.
- *private int ncol*
    - El nombre de columnes que té la distribució del Layout.
- *private int nfil*
    - El nombre de files que té la distribució del Layout.

Operacions:

- *private void inicialitzarDistribucio()*
    - Inicia les diferents estructures de dades de la classe.
- *private void omplirCoordenades()*
    - Relaciona una id amb unes coordenades, aixo es fa de forma que les ids mes petites estiguin mes al centre de la distribució i a mesura que creixen es van allunyant.
- *private int distanciaManhattan(Pair<Integer,Integer> c1, Pair<Integer, Integer> c2)*
    - Calcula la distancia Manhattan entre dos punts c1 i c2.
- *private void calcularDistancies()*
    - Calcula les distancies entre totes les ids una vegada posicionades.

- *private String saveData()*
    - Retorna un String amb la mida del Layout en format JSON

Pair

Classe que serveix d'estructura de dades auxiliar com a un parell d'un tipus (int, double, char, etc.)

Strategy

Classe Interface per aplicar el patró Strategy. Les classes que l'implementen són BranchBound i Genetic, les dues classes per a generar un Teclat de diferents maneres.

Teclat

Aquesta classe guarda tota la informació sobre un Teclat, principalment d'atributs simples hi ha un nom: String per identificar al Teclat, un L: Layout, un A: Alfabet i un G: Generador que serveixen per a generar el Teclat. D'atributs amb estructures de dades més complexes hi ha:

Atributs:

- *private String nom*
    - Nom que identifica un Teclat de la resta de Teclats
- *private Layout L*
    - Layout amb el qual està creat aquest Teclat
- *private Alfabet A*
    - Alfabet amb el qual està creat aquest Teclat
- *private Generador G*
    - Generador amb el que està creat el Teclat
- *private Map<Character, Integer> teclat*
    - Guarda la relació entre una lletra de l'Alfabet i la id d'una posició del Layout
- *private char[][] distribucioCharacters*
    - La matriu on es guarda el teclat (es guarden les lletres de l'Alfabet)

Operacions:

- *public void crearTeclat ()*
    - Crida a generador per a que generi un teclat a partir de les dades del Layout i de l'Alfabet
- *private void omplirDistribucio()*
    - S'omple la matriu distribucioCharacters amb les lletres de l'Alfabet i la seva id segons la generació donada pel Generador
- *public void modificarTeclat(Character a, Character b)*
    - Modificar el Teclat intercanviant les posicions de dues lletres de l'Alfabet donades
- *private void swapLetters(Character a, Character b)*

- Intercanvia dues lletres a, b

Text

La classe Text és una subclasse de la classe abstracte Alfabet. Gestiona l'entrada d'un Alfabet quan és un text.

Operacions:

- *public void read (String path)*
  - Fa els càlculs necessaris per tenir les dades necessàries d'un Alfabet
- *private String getText (String path)*
  - Llegeix el text que es troba en el path i retorna la lectura

Words

La classe Words és una subclasse de la classe abstracte Alfabet. Gestiona l'entrada d'un Alfabet quan és una llista de paraules amb probabilitats.

Operacions:

- *public void read (String path)*
  - Fa els càlculs necessaris per tenir les dades necessàries d'un Alfabet
- *private HashMap<String, Double> getWords (String path)*
  - Llegeix la llista de paraules amb probabilitats que es troba en el path i retorna la lectura

3.2 Controladors domini

CtrlDomini

Classe que gestiona les classes del model i es comunica amb altres capes quan fa falta (Presentació i Persistencia)

Atributs:

- *private Generador generador*
    - Generador que s'utilitza per a crea
- *private int[] midesInicials*
    - Representa unes mides inicials per crear uns Layouts
- *private String strategy*
    - Cadena de caràcters on es guarda quina estratègia s'està utilitzant
- *private HashMap<String, Teclat> Teclats*
    - Llista que guarda tots els Teclats del sistema
- *private HashMap<String, Alfabet> Alfabet*
    - Llista que guarda tots els Alfabets del sistema
- *private HashMap<Integer, Layout> Layouts*
    - Llista que guarda tots els Layouts del sistema
- *private static CtrlDomini singletonDomini*
    - La pròpia instància del controlador de domini (una sola)

Operacions:

- *public void init()*
    - Inicialitza les instàncies del model i variables del controlador.
- *public void inicialitzarLayoutsBase()*
    - Crea els Layouts inicials generats per defecte.
- *public void crearNouTeclat(String nt, String na, Integer idL)*
    - Crea un nou teclat amb nom nt, alfabet na i layout idL.
- *public void modificarTeclat(String nt, ArrayList<Pair<Character, Character>> canvis)*
    - Li aplica al Teclat nt els canvis entre els caràcters de la llista canvis.
- *public Map<String, Object> visualitzaTeclat(String nt)*
    - Retorna les dades necessàries per mostrar el teclat nt.
- *public void esborrarTeclat(String nt)*
    - Esborra el teclat nt.
- *public void afegirAlfabet(String na, String ta, String pf)*
    - Crea un alfabet amb nom na, de tipus ta, i amb les dades del path pf.
- *public Map<String, Object> visualitzarAlfabet(String na)*
    - Mostra les dades necessàries per mostrar l'alfabet na.

- *public void esborrarAlfabet(String na)*
    - Esborra l'Alfabet na.
- *public void afegirLayout(Integer idL)*
    - Crea un Layout de mida idL.
- *public Map<String, Object> visualitzarLayout(Integer idL)*
    - Mostra les dades necessàries per mostrar el Layout idL.
- *public void esborrarLayout(Integer idL)*
    - Esborra el Layout idL.
- *public void saveState()*
    - Guarda l'estat dels diccionaris Alfabets, Layouts i Teclats.
- *public void restoreState()*
    - Restaura l'estat dels diccionaris Alfabets, Layouts i Teclats.

Factoria

Classe que s'encarrega de crear els controladors de les 3 capes i de retornar les seves instàncies.

4. Capa presentació

En el següents punts es detallen totes les classes que pertanyen a la capa de presentació.

CtrlAfegirAlfabet

Vista per afegir un nou Alfabet. Un camp per posar el nom, un radio button per indicar el tipus d'entrada i una finestra per seleccionar el fitxer d'entrada.

CtrlAfegirLayout

Vista per afegir un nou Layout. Un camp per posar la mida del layout.

CtrlAfegirTeclat

Vista per afegir un nou Layout. Un camp per posar el nom del Teclat, un radio button per seleccionar el Generador i una llista per seleccionar l'Alfabet.

CtrlEliminar

Vista que apareix quan es vol eliminar un element (Teclat, Alfabet o Layout) amb un missatge per a confirmar.

CtrlLlistaAlfabets

Vista que mostra tots els Alfabets en el sistema i a més apareix un botó per a afegir un de nou. Per cada Alfabet que es mostra es pot veure més informació (pantalla per mostrar Alfabet) i també hi ha un botó per eliminar-ne algun.

CtrlLlistaLayouts

Vista que mostra tots els Layouts en el sistema i a més apareix un botó per a afegir un de nou. Per cada Layout que es mostra es pot veure més informació (pantalla per mostrar Layout) i també hi ha un botó per eliminar-ne algun (només si el Layout no ha estat creat pel sistema)

CtrlLlistaTeclats

Vista que mostra tots els Teclats en el sistema i a més apareix un botó per a afegir un de nou. Per cada Teclat que es mostra es pot veure més informació (pantalla per mostrar Teclat), també hi ha un botó per eliminar-ne algun i també un botó per editar algun Teclat. A més a la part inferior apareixen dos botons per anar a la llista de Layouts i Alfabets respectivament. Finalment hi ha un botó per sortir i tots els canvis realitzats es guardaran.

CtrlModificarTeclat

Vista per modificar el Teclat, es mostra el Teclat i es seleccionen les lletres que es volen intercanviar. Per cada parell es confirma per veure el canvi. Quan s'han fet tots els canvis es confirma tot el que s'ha modificat o es cancela.

CtrlMostrarAlfabet

Vista que mostra un Alfabet. Inicialment es mostren les freqüències de cada caràcter de l'Alfabet i a sobre de cada caràcter es pot veure quina és la probabilitat que hi ha que aparegui el següent caràcter. A més hi ha un botó per eliminar l'Alfabet.

CtrlMostrarLayout

Vista que mostra el Layout, la forma que té. També apareix un botó per eliminar-lo.

CtrlMostrarTeclat

Vista per mostrar el Teclat. Es mostra la forma del Layout amb els caràcters de l'Alfabet. A més hi ha tres botons, un per modificar el Teclat, un segon per provar el Teclat i un tercer per esborrar-lo.

CtrlPantallaInformativa

Vista que apareix quan es vol donar una informació addicional com per exemple un missatge d'error.

CtrlPantallaInici

Vista inicial que mostra el títol, els integrants del grup i un botó START per començar.

CtrlPreMostrarTeclat

Vista per previsualitzar la creació d'un Teclat (pantalla posterior a la creació).

CtrlPresentacio

Classe que controla la capa de presentació. Gestiona quina finestra es mostra i es comunica amb la capa inferior (domini) per a demanar alguna dada.

Atributs:

- *private CtrlDomini cd*
    - Instància del controlador del domini
- *private CtrlPresentacio ctrlPresentació*
    - Instància del controlador de presentació, és un atribut estàtic i és null fins que es fa un getInstance()

Operacions:

- *private CtrlPresentacio()*
    - Creadora de la classe, instancia el controlador de domini, restaura les dades que hi ha a la capa de presentació d'altres execucions anteriors i mostra la primera pantalla.
- *public String toggleStrategy()*
    - Canviar d'estratègia per a generar el Teclat

- *public void crearNouTeclat(String nt, String na, String generador)*
    - Crear un nou Teclat amb nom nt, Alfabet amb nom na amb un Layout de la mateixa mida que l'alfabet a partir d'un generador
- *public void modificarTeclat(String nt, ArrayList<Pair<Character, Character>> canvis)*
    - Modificar el Teclat identificat amb nom nt, fent un intercanvi en totes les lletres en la Llista canvis
- *public void visualitzarTeclat(String nt)*
    - Veure el Teclat amb nom nt
- *public void esborrarTeclat(String nt)*
    - Esborrar el Teclat amb nom nt
- *public void afegirAlfabet(String na, String ta, String pf)*
    - Afegir un Alfabet amb nom na, del tipus ta (text o llista-paraules) amb l'entrada al path pf
- *public String visualitzarAlfabet(String na)*
    - Visualitzar l'Alfabet na
- *public void esborrarAlfabet(String na)*
    - Esborrar l'Alfabet amb nom na
- *public void afegirLayout(Integer idL)*
    - Afegir un Layout amb la mida idL
- *public String visualitzarLayout(Integer idL)*
    - Visualitzar el Layout amb la mida idL
- *public void esborrarLayout(Integer idL)*
    - Esborrar el Layout de mida idL
- *public void guarda()*
    - Guarda tots els canvis realitzats durant l'execució del programa
- *public String lastSave()*
    - Retorna la data de l'últim accés a fitxer que s'ha fet
- *public void canviVista(String vista, String elementAMostrar)*
    - Canvi a la vista que s'ha passat com a paràmetre, elementAMostrar és un String identificador de la instància de Teclat, Alfabet o Layout per si la vista necessita aquesta dada.
- *public void elimina(String tipus, String clau, JFrame pantalla, String pantallaRetorn)*
    - Elimina una instància segons el tipus (Teclat, Alfabet o Layout) a partir del nom que l'identifica (clau). El paràmetre pantalla és la pantalla actual que s'està mostrant i pantallaRetorn fa referència a la pantalla que es vol retornar una vegada s'ha eliminat la instància.
- *public void Excepcio(JFrame pantalla, String title, String msg)*
    - Mostra un missatge amb l'execpció que ha aparegut. El paràmetre pantalla és la pantalla actual sobre la que es mostra el missatge, title és el nom de l'excepció i msg és el missatge que es mostra.

CtrlProvarTeclat

Es mostra el Teclat com a la pantalla de mostrar Teclat, però cada tecla és un botó que pot ser premut i en un camp de text es veu el resultat.

PantallaInformativa

Pantalla que mostra informació rellevant sobre els canvis del sistema.

Utils

Classe auxiliar per a evitar tenir el mateix codi entre les diferents classes de la capa presentació.