Gestor de Documents

Projecte de Programació 22-23 Q1

Identificador del grup: 14.4

Joan Vazquez Gonzalez	joan.vazquez.gonzalez
Paula Grau Dominguez	paula.grau
Neus Mayol Alcaraz	neus.mayol
Tomàs Calaf Martí	tomas.calaf.marti

Sumari

1. Introducció

Aquest és l'escrit que acompanya el codi de la primera entrega de l'assignatura Projectes de Programació, impartida a la Facultat d'Informàtica de Barcelona el quadrimestre de tardor del curs 2022-23. L'objectiu final del projecte és la creació d'un entorn de manipulació de documents, on puguem obrir-los, modificar-los, llistar-los etc.
Durant aquesta entrega, hem programat el que és necessari perquè el programa funcioni. Això sí, deixant de banda la capa de presentació, que ens permetrà veure el programa en un format més atractiu, i la capa de persistència, que s'ocuparà de guardar aquells elements del programa que volem mantenir cada vegada que el tanquem. Així doncs, el programa resultant de la primera entrega conté la implementació i el *testeig* dels casos d'ús demanats a l'enunciat.

2. Metodologia

Per començar, vam llegir conjuntament l'enunciat i vam explotar tots els possibles casos d'ús que acabaríem necessitant per crear un entorn amb totes es funcionalitats que se'ns demanava. A partir d'aquests casos d'ús vam procedir a pensar quines classes necessitava el nostre programa, com l'organitzaríem. Per fer-ho vam acabar creant un diagrama d'implementació on es mostren les classes i com s'associen entre elles. A partir d'allà vam pensar quines eren les funcions i les estructures de dades que havíem de programar a cada classe perquè aquesta funcionés correctament i es pogués comunicar amb les seves classes asssociades. Arribats a aquest punt vam acabar de completar el diagrama amb aquelles funcions que relacionaven els elements d'una mateixa classe i els algorismes de cerca.

Havent concretat l'estructura del nostre projecte, vam començar a programar les classes. Tal i com es demana a l'assignatura, tot el codi ha estat programat en llenguatge *Java*. Tots quatre integrants del grup hem fet servir l'entorn *IntelliJ* per programar les nostres classes individualment. Per combinar el codi de cada membre amb el de la resta del grup hem fet servir el repositori de *GitLab* proporcionat pels professors de l'assigantura.

Per programar les classes s'ha assignat un cert nombre de classes i estructures de dades a cada membre del grup, els quals podreu trobar a la següent relació. El repartiment es va fer tenint en compte que no totes les classes comporten la mateixa càrrega de treball, i intentant que cada membre del grup es quedés amb classes que estiguessin relacionades, ja que vam creure que així podríem completar les nostres tasques amb més agilitat.

A la taula a continuació es mostra la relació entre les classes i el membre del grup que l'ha programat. La taula mostra només el nom de la classe, pot ser que aquest no deixi clar quina és la finalitat d'aquesta, això ho explicarem detalladament més endavant.

Repartiment de les classes del programa.

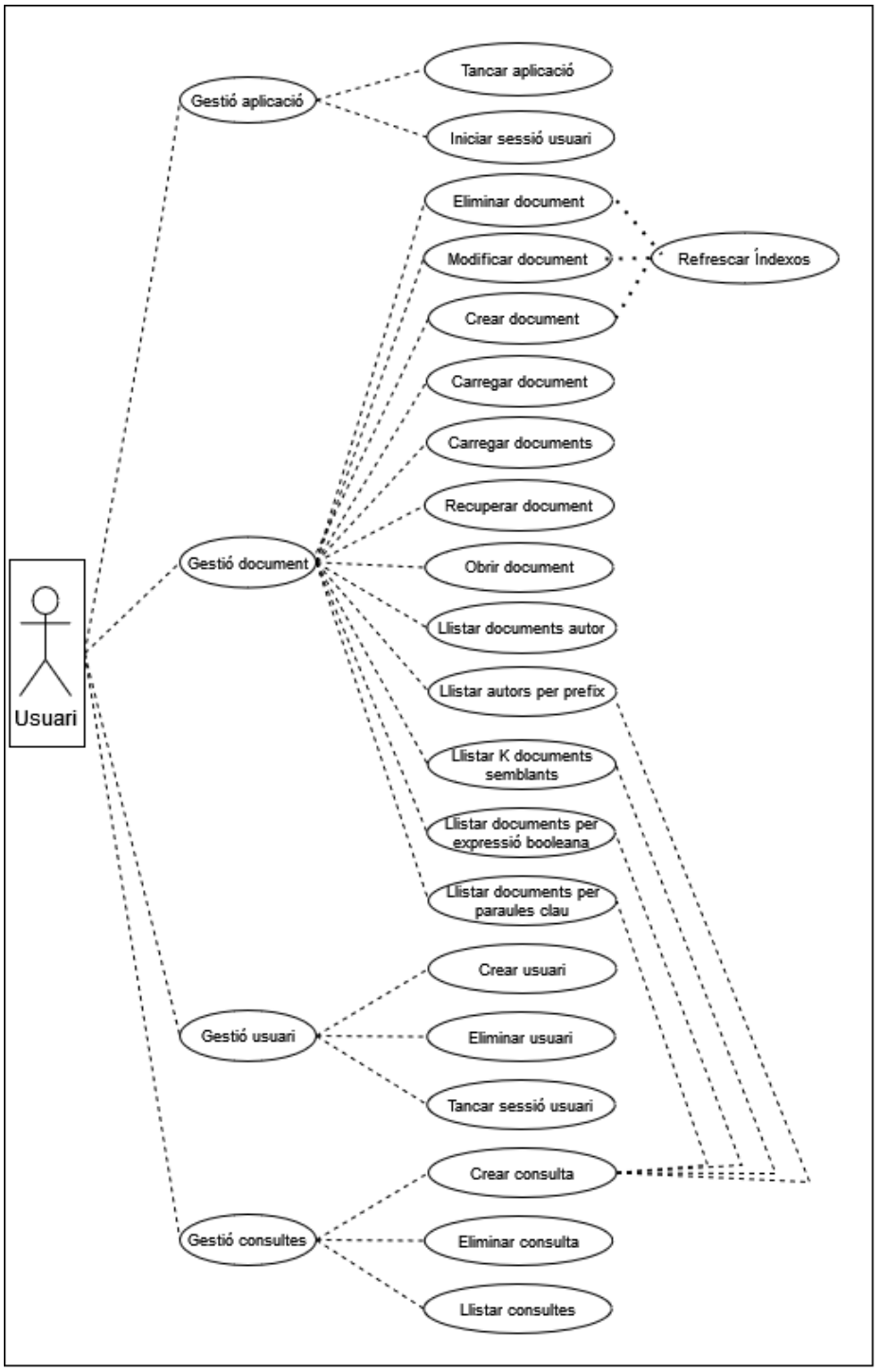
Classe	**Autor grup**
CtrlConsulta	Neus
CtrlDocument	Joan
CtrlDomini	Tomas
CtrlUsuari	Tomas
Consulta	Tomas
Document	Joan
Expressio booleana	Neus
ÍndexAutor	Paula
ÍndexDocumentsAutor	Paula
ÍndexExpressiósBooleana	Neus
Usuari	Tomas
DriverCtrlDomini	Tomas
KeyP	Joan
PairP	Paula
SortedValue	Joan
TipusConsulta	Tomas
Tree	Neus
Trie	Paula

Al acabar de programar vam procedir a fusionar conjuntament les classes i a corregir els errors fruit de la fusió i els que no havíem tingut en compte fins al moment. Per acabar de comprovar que tot funcionés, vam fer ús de *JUnit* per crear *tests* i comprovar que totes les funcions ens donaven el resultat esperat. També vam programar els *drivers* pertinents i vam explotar tots aquells casos que no ens permetrien continuar executant el programa: les excepcions.

Com que les classes principals estan agregades als controladors, els controladors estan agregats al controlador domini, les classes i la resta de classes són classes simples o estructures de dades, hem decidit prescindir dels *stubs*.

3. Diagrama de casos d'ús

L'actor del diagrama, tal i com es pot veure a l'esquema, és l'usuari: que pot interaccionar amb l'aplicació de varies maneres, que hem acabat dividint en quatre grups: Gestió d'aplicació, d'usuaris, de documents i de consultes.



Gestió aplicació

Els casos d'ús d'aquest grup corresponent a tot el que podem fer amb l'aplicació oberta i que comporten canvis al programa en general.

TANCAR APLICACIÓ	Es tanca l'aplicació
ACTOR	Usuari
PRECONDICIÓ	L'aplicació ha d'estar oberta
COMPORTAMENT	- L'usuari selecciona l'opció de tancar el gestor de documents. - El sistema tanca la finestra gràfica i para l'execució de l'aplicació.

EXTENSIONS i ERRORS	- Si hi ha canvis no guardats, el sistema avisa a l'usuari i li dona diverses opcions:
	  - Tancar sense guardar
	  - Guardar els canvis i tancar
	  - Cancel·lar l'acció

INCLUDES	Gestió aplicació

INICIAR SESSIÓ USUARI	S'inicia sessió d'un usuari
ACTOR	Usuari
PRECONDICIÓ	No ha d'haver cap usuari que ja tingui la sessió iniciada
COMPORTAMENT	- L'usuari inicia la sessió d'un usuari fent servir la seva contrasenya. - Sobre l'editor amb l'estat de l'usuari guardat.
EXTENSIONS I ERRORS	Errors: - Ja hi ha un usuari actiu - No existeix l'usuari(nom) - Contrasenya incorrecta
INCLUDES	Gestió aplicació

Gestió documents

Aquest grup inclou tots aquells casos d'ús que s'apliquen sobre un document o el conjunt de documents. També s'inclouen les funcions de llistar els documents que, tal i com es pot veure al diagrama, també estan relacionades amb un cas d'ús del grup de gestió de consultes. S'ha decidit estructurar així el diagrama perquè les cerques es fan sobre el conjunt de documents, però es criden des de les consultes.

CREAR DOCUMENT	Es crea un document amb el títol, autor, contingut i format especificats
ACTOR	Usuari
PRECONDICIÓ	- No pot haver un document ja existent amb el mateix identificador (autor i títol) que el document que volem crear - El format especificat ha de ser .txt, .xml, .prop - L'idioma en el que es crea el document és català, castellà o anglès
COMPORTAMENT	- L'usuari selecciona l'opció crear document - S'obrirà una pestanya i haurà de indicar el títol i autor del document - També haurà d'indicar el format i, si n'hi ha, indicar el contingut - Si no hi ha cap error es modificarà la informació.
EXTENSIONS I ERRORS	- Si existeix un document ja existent amb el mateix identificador o el format o idioma no són correctes, el document no es crea i apareix un missatge en una pestanya auxiliar indicant l'error - Si les claus primàries de document (autor, títol i format) són null el document no es crea i apareix un missatge en una pestanya auxiliar indicant l'error.
INCLUDES	Gestió documents

MODIFICAR DOCUMENT	Canviar la informació d'un document. És a dir: El títol, l'autor i/o el contingut
ACTOR	Usuari
PRECONDICIÓ	El document amb identificador (títol, autor) ha d'existir
COMPORTAMENT	- L'usuari selecciona l'opció de modificar informació, que sortirà fent click dret sobre el document. - S'obrirà una pestanya auxiliar amb els paràmetres actuals del document, disposats en quadres de text. - L'usuari modificarà el text dels camps que vulgui modificar i apretarà un botó de confirmació en acabat. - Si no hi ha cap error es modificarà la informació.
EXTENSIONS I ERRORS	- No es pot fer la modificació del contingut, si al fer-ho aquest passa a tenir el mateix títol i autor que un altre document ja existent a l'aplicació. Apareix una pestanya auxiliar amb un missatge indicant l'error - No es pot fer la modificació del contingut, si al fer-ho aquest passa a tenir el títol, l'autor o el format null. Apareix una pestanya auxiliar amb un missatge indicant l'error
INCLUDES	Gestió documents

ELIMINAR DOCUMENT	Esborrar un document.
ACTOR	Usuari
PRECONDICIÓ	El document ha d'existir al gestor de documents
COMPORTAMENT	- L'usuari vol esborrar el document i fa click a la opció esborrar que apareixerà fent click dret o apretant la tecla suprimir. - El document s'esborrarà del directori de l'usuari
INCLUDES	Gestió documents

CARREGAR DOCUMENT	S'afegeix al gestor un document.
ACTOR	Usuari
PRECONDICIÓ	El document que es vol afegir ha d'existir a un altre lloc
COMPORTAMENT	- L'usuari vol carregar un document al gestor de documents. - L'usuari selecciona la opció de carregar un document des d'un menú al mateix gestor. - S'obre una pestanya on es permet seleccionar quin és el document que es vol carregar. - L'usuari selecciona el document en qüestió i confirma la seva selecció prement un botó. - Si no hi ha cap error, es carrega el document seleccionat al gestor. - Si ja existeix un document amb mateix títol i autor, es pregunta a l'usuari si el vol sobreescriure.
EXTENSIONS i ERRORS	- Ja existeix un document amb mateix títol, autor i format. - No existeix el document que es vol carregar, en aquest cas es podria donar l'opció de crear-ne un de nou.
INCLUDES	Gestió documents

CARREGAR DOCUMENTS	S'afegeix al gestor un conjunt de documents.
ACTOR	Usuari
PRECONDICIÓ	Els documents que es volen carregar han d'existir a un altre lloc.
COMPORTAMENT	- L'usuari vol carregar un document al gestor de documents. - L'usuari selecciona la opció de carregar documents des d'un menú al mateix gestor. - S'obre una pestanya on es permet seleccionar quins són els documents que es volen carregar. - L'usuari selecciona els documents en qüestió i confirma la seva selecció prement un botó. - Si no hi ha cap error, es carreguen els documents seleccionats al gestor. - Si ja existeix algun un document amb mateix títol i autor, es pregunta a l'usuari si el vol sobreescriure.
EXTENSIONS i ERRORS	- En algun cas, ja existeix un document amb mateix títol, autor i format. - En algun cas, no existeix el document que es vol carregar, en aquest cas es podria donar l'opció de crear-ne un de nou.
INCLUDES	Gestió documents

RECUPERAR DOCUMENT	Guardar el document de l'aplicació al directori (guardar como)
ACTOR	Usuari
PRECONDICIÓ	- El document ha d'existir al gestor de documents. - No pot haver un altre document amb el mateix títol i autor al directori de l'usuari
COMPORTAMENT	- L'usuari busca el document que vol recuperar i el copia al directori on el vulgui guardar. - L'usuari pot especificar el format.

EXTENSIONS i ERRORS	- Si ja existeix un document al directori amb el mateix títol i autor, apareix un missatge indicant-ho. L'usuari pot triar diferents opcions:
	  - cancel·lar l'operació
	  - canviar el títol i/o autor del document que volem recuperar
	  - sobreescriure el document que hi havia al document

INCLUDES	Gestió documents

OBRIR DOCUMENT	S'obre el document
ACTOR	Usuari
PRECONDICIÓ	El document que l'usuari vol obrir, ha d'existir al gestor de documents
COMPORTAMENT	- L'usuari busca al directori el document que vol visualitzar - Demana que vol obrir-lo fent doble clic sobre el document o clicant el botó dret del ratolí i després seleccionant la opció "obrir document". - Si no hi ha cap error, el document s'obre
INCLUDES	Gestió documents

REFRESCAR ÍNDEXS	Es dona lloc quan l'usuari ha creat, esborrat o modificat un document, i s'han d'actualitzar els índexs segons les modificacions.
ACTOR	Sistema
PRECONDICIÓ	S'ha creat, esborrat o modificat un document
COMPORTAMENT	- Es crida, per a cadascun dels índexs, un mètode que els refresqui. - S'actualitzen els índexs segons el nou conjunt de documents amb els canvis introduïts
INCLUDES	Gestió documents, gestió consultes

LLISTAR DOCUMENTS AUTOR	Es llisten els títols de tots els documents que de l'autor indicat
ACTOR	Usuari
PRECONDICIÓ	L'autor ha d'existir al gestor de documents, és a dir, hi ha d'haver algun document d'aquest
COMPORTAMENT	- L'usuari apreta la opció buscar en un menú. - De l'opció buscar, pot especificar de quina de les maneres demanades a l'enunciat es vol buscar, en aquest cas l'usuari seleccionarà "per autor". - Seguidament es demanarà a l'usuari que introdueixi el nom de l'autor. - L'usuari introdueix el nom de l'autor i apreta un botó per començar la cerca.
EXTENSIONS i ERRORS	- Opció de posar un error si l'autor demanat no existeix. Encara que també es podria mostrar una llista buida
INCLUDES	Gestió documents

LLISTAR AUTORS PER PREFIX	Es mostren tots els autors que comencen pel prefix introduït com a paràmetre de cerca
ACTOR	Usuari
COMPORTAMENT	- L'usuari apreta la opció buscar en un menú. - De l'opció buscar, pot especificar de quina de les maneres demanades a l'enunciat es vol buscar, en aquest cas l'usuari seleccionarà "per prefix". - Seguidament es demanarà a l'usuari que introdueixi el prefix i es retornarà el nom dels autors ordenats alfabèticament.
EXTENSIONS i ERRORS	- Pot ser que el prefix sigui buit, i en aquest cas es llistarien tots els autors existents al gestor de documents - Pot ser que no existeix cap autor amb aquest prefix, i en aquest cas apareixerà un missatge indicant-ho.
INCLUDES	Gestió documents

LLISTAR K DOCUMENTS SEMBLANTS	Es mostren els k (número enter) documents més semblants al document D.
ACTOR	Usuari
PRECONDICIO	- k ha de ser un nombre enter positiu o zero. - El document D ha d'existir al gestor de documents
COMPORTAMENT	- L'usuari apreta la opció buscar en un menú. - De l'opció buscar, pot especificar de quina de les maneres demanades a l'enunciat es vol buscar, en aquest cas l'usuari seleccionarà "per similitut". - Seguidament es demanarà a l'usuari que introdueixi el títol i autor que identifiquen al document D i el nombre k de documents a llistar. - Es llisten els títols i autors dels k documents més semblants a D
EXTENSIONS i ERRORS	- Si k no és un nombre enter positiu, no es pot realitzar la cerca - Si el document D no existeix al gestor de documents, apareix un missatge d'error indicant-ho i no es realitza la cerca
INCLUDES	Gestió documents

LLISTAR DOCUMENTS PER EXPRESSIÓ BOOLEANA	Es mostren els documents que satisfan l'expressió booleana especificada
ACTOR	Usuari
PRECONDICIÓ	L'expressió booleana donada ha de ser vàlida. Un exemple d'expressió booleana no vàlida és: "hola" & | "adeu"
COMPORTAMENT	- L'usuari apreta la opció buscar en un menú. - De l'opció buscar, pot especificar de quina de les maneres demanades a l'enunciat es vol buscar, en aquest cas l'usuari seleccionarà "per expressió". - Seguidament es demanarà a l'usuari que introdueixi l'expressió amb la que vol "filtrar" els documents. - Si l'expressió booleana és vàlida, es retorna el títol i autor dels documents que la satisfan
EXTENSIONS i ERRORS	- Si l'expressió booleana no és vàlida, apareix un missatge indicant-ho i no es realitza la cerca
INCLUDES	Gestió documents

LLISTAR DOCUMENTS PER PARAULES CLAU	Es mostren els k documents més rellevants segons la query de p paraules donades
ACTOR	Usuari
PRECONDICIO	- k ha de ser un nombre enter positiu o zero. - La query de p paraules donades ha de ser vàlida
COMPORTAMENT	- L'usuari apreta la opció buscar en un menú. - De l'opció buscar, pot especificar de quina de les quatre maneres demanades a l'enunciat es vol buscar, en aquest cas l'usuari seleccionarà "per paraules clau". - Seguidament es demanarà a l'usuari que introdueixi les paraules clau i l'enter k (que especifica el nombre de documents a llistar) amb les que es vol fer la cerca. - Si no hi ha cap error, es llistaran els k títols i autors dels documents més semblants a la query
EXTENSIONS i ERRORS	- Si k no és un nombre enter positiu, no es pot realitzar la cerca
INCLUDES	Gestió documents

Gestió usuari

Aquest subgrup agrupa tots els casos relatius a l'usuari i a la seva gestió.

CREAR USUARI	Es crea un nou usuari
ACTOR	Usuari
PRECONDICIÓ	- El nom de l'usuari que volem crear no ha d'existir
COMPORTAMENT	- L'usuari selecciona l'opció de crear un nou usuari - L'usuari introdueix el nom i contrasenya - Es crea un nou usuari amb un nom i contrasenya indicats
EXTENSIONS I ERRORS	- Si ja existeix un usuari amb el nom indicat, apareix un missatge indicant-ho i no es crea l'usuari
INCLUDES	Gestió usuaris

ELIMINAR USUARI	S'elimina un usuari
ACTOR	Usuari
PRECONDICIÓ	L'usuari ha d'existir
COMPORTAMENT	- L'usuari elimina un usuari existent. Ha d'introduir la contrasenya de l'usuari abans d'eliminar-lo. Si s'elimina a ell mateix torna a la pàgina de inici de sessió.
EXTENSIONS I ERRORS	- Si l'usuari que es vol eliminar no existeix, apareix un missatge indicant-ho - Si la contrasenya que s'introdueix de l'usuari que es vol eliminar no és correcte, apareix un missatge indicant-ho. En aquest cas es pot intentar tornar a introduir la contrasenya o cancel·lar l'operació
INCLUDES	Gestió usuaris

TANCAR SESSIÓ USUARI	Es tanca la sessió de l'usuari que tingui la sessió iniciada
ACTOR	Usuari
PRECONDICIÓ	- L'usuari ha d'existir - L'usuari ha de tenir la sessió iniciada
COMPORTAMENT	- L'usuari que tingui la sessió iniciada selecciona l'opció de tancar sessió - La sessió de l'usuari es tanca i tornem a la pantalla des de la qual es pot crear un nou usuari o iniciar sessió amb un usuari existent o crear-ne un de nou
EXTENSIONS I ERRORS	- Si l'usuari del qual es vol tancar la sessió no existeix, apareix un missatge indicant-ho - Si l'usuari no té la sessió iniciada no passa res al gestor
INCLUDES	Gestió usuaris

Gestió consultes

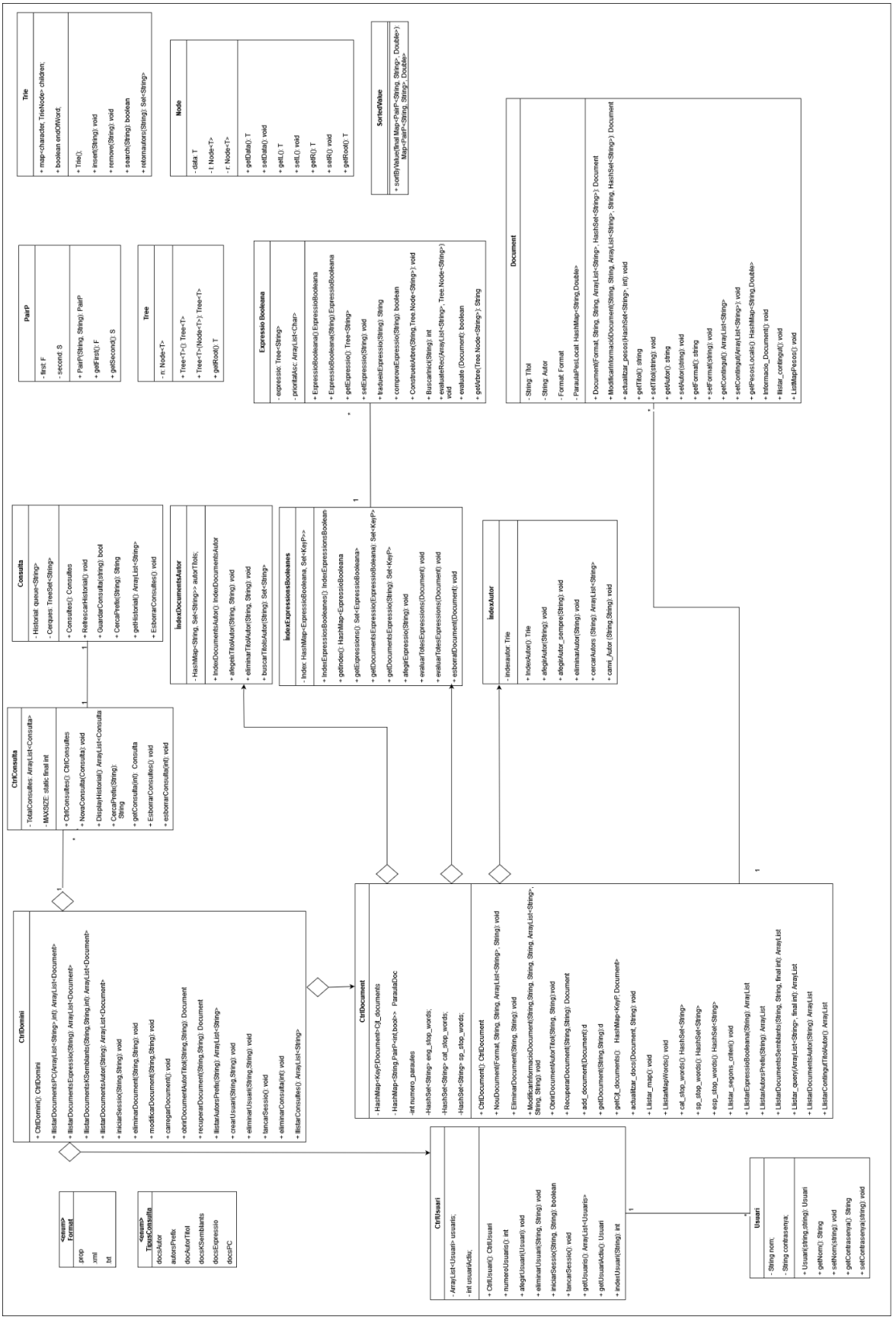
Aquest grup de casos gestiona les cerques en general, sense diferir en el tipus de consulta.

CREAR CONSULTA	Es crea una consulta feta per l'usuari
ACTOR	Usuari i sistema
PRECONDICIÓ	- La consulta ha de ser vàlida
COMPORTAMENT	- L'usuari fa una de les cerques de llistar i es crea la consulta
EXTENSIONS I ERRORS	- La consulta no era vàlida, no es crea
INCLUDES	Gestió consultes

ELIMINAR CONSULTA	S'elimina una consulta
ACTOR	Usuari
PRECONDICIÓ	- La consulta ha d'existir al conjunt de consultes existents fetes anteriorment per l'usuari
COMPORTAMENT	- La consulta seleccionada s'elimina de la llista de consultes
INCLUDES	Gestió consultes

LLISTAR CONSULTES	Es llisten les consultes realitzades per l'usuari fins el moment, que no han sigut eliminades
ACTOR	Usuari
PRECONDICIÓ	- La consulta ha de ser vàlida
COMPORTAMENT	- L'usuari fa una cerca i es crea la consulta
EXTENSIONS I ERRORS	- Si la consulta no era vàlida, no es crea
INCLUDES	Gestió consultes

4. Diagrama de classes



El diagrama també està a un *.png* a part.

Restriccions textuals UML

- el MAXSIZE de *CtrlConsulta* ha de ser un enter major o igual a 0
- el *numero_paraules* de *CtrlDocument* ha de ser un enter major a 0
- PRIMARY KEYS:

  Document (*Format*, *Autor*, *Títol*) :: (*Format*, *String*, *String*)

  Usuari (*Nom*) :: (*String*)

  KeyP (K1, K2) :: (*String*, *String*)

  PairP (F, S) :: (F,S)

  ExpressióBooleana(*expressió*)

  Consulta(*Parametres*)

Domini

A continuació s'explicarà amb més detall quin és el contingut de cada classe de la capa de domini.

Nom	**Consulta**
Descripció	Representa una consulta feta per l'usuari, formada per els paràmetres que ha introduït aquest per fer la cerca i el tipus de consulta realitzada
Atributs	- *parametres*: *ArrayList<String>* - *tipus*: *TipusConsulta*
Relacions	CtrlConsulta
Funcions	- *Consulta(ArrayList<String> parametres, TipusConsulta): void* - *getparametres(): String* - *setparametres(ArrayList<String>): void* - *nouParametre(String): void* - *getTipus(): TipusConsulta* - *setTipus(TipusConsulta): void* - *getExpressio(): String* - *checkNumParametres(ArrayList<String>, TipusConsulta): int*

Nom	**Document**
Descripció	Representa un document identificat amb un autor i títol. Té un format específic (.txt, .xml o .prop) i té un contingut.
Atributs	- *format*: *Format* - *títol*: *String* - *autor*: *String* - *ParaulesPesLocal*: *HashMap<String, Double>* - *contingut*: *ArrayList<String>* - *norma*: *double*
Relacions	CtrlDocument
Funcions	- *Document* (*Format*, *String*, *String*, *ArrayList<String>*, *HashSet<String>*): *void* - *ModificarInformacioDocument* (*String*, *String*, *ArrayList<String>*, *String*, *String*, *HashSet<String>*): *Document* - *actualitzar_pesos* (*HashSet<String>*, *int*): *void* - *getTitol*(): *String* - *getAutor*(): *String* - *getFormat*(): *Format* - *getContingut*(): *ArrayList<String>* - *getPesosLocal*(): *HashMap<String, Double>* - *setTitol*(*String*): *void* - *setAutor*(*String*): *void* - *setFormat*(*Format*): *void* - *setContingut*(*ArrayList<String>*): *void*

Nom	**Expressió Booleana**
Descripció	Representa una expressió booleana cercable a qualsevol document del sistema de fitxers. L'expressió, estructurada en un arbre binari, pot contenir les connectives lògiques *and* (&), *or* (|) i *not* (!), a més de subconjunts de paraules (delimitats entre claudàtors) i seqüències de paraules (delimitades entre cometes). La classe conté funcions per crear una expressió booleana i per evaluar-la donat un document.
Atributs	- *expressio*: *Tree<String>* - *prioritatAsc*: *ArrayList<Character>*
Relacions	IndexExpressionsBooleanes
Funcions	- *ExpressioBooleana*(): *void* - *ExpressioBooleana*(*String*): *void* - *getExpressio*(): *Tree<String>* - *setExpressio*(*String*): *void* - *tradueixExpressio*(*String*): *String* - *comprovaExpressio*(*String*): *boolean* - *ConstrueixArbre*(*String*, *Tree.node<String>*): *void* - *BuscarInici*(*String*): *int* - *evaluateRec*(*ArrayList<String>*, *Tree.node<String>*): *Boolean* - *evaluate*(*Document*): *Boolean* - *getArbre*(*Tree.node<String>*): *String*

Nom	**Índex Autor**
Descripció	Representa un arbre amb tots els autors que algun cop han tingut un document al gestor de documents. És un índex que facilita la cerca d'autors segons el seu prefix
Atributs	- *indexautor*: *Trie*
Relacions	Trie, CtrlDocument
Funcions	● *IndexAutor*(): *void* ● *afegirAutor_sempre*(*String*): *void* ● *afegirAutor*(*String*): *void* ● *eliminarAutor*(*String*): *void* ● *cercarAutors*(*String*): *ArrayList<String>* ● *canvi_Autor*(*String*, *String*): *void*

Nom	**Índex Documents Autor**
Descripció	Representa un HashMap amb els autors que algun cop han tingut un document al gestor de documents, i tots els títols de documents que actualment formen part del gestor. És un índex que facilita la cerca de títols segons el seu autor.
Atributs	- *autorTítols*: *HashMap<String, Set<String>>*
Relacions	CtrlDocuments
Funcions	- *IndexDocumentsAutor*(): *void* - *afageixTitolAutor*(*String*, *String*): *void* - *eliminarTitolAutor*(*String*, *String*): *void* - *buscarTitolsAutor*(*String*): *Set<String>*

Nom	**Índex Expressions Booleanes**
Descripció	Representa un *HashMap* amb les expressions booleanes creades i un *Set<KeyP>* que representen els identificadors (autor i títol) dels documents que satisfan l'expressió booleana. És un índex que facilita la cerca dels documents que satisfan una expressió i que es refrescarà cada vegada que s'afegeixi una nova expressio al sistema, o que es modifiqui un document.
Atributs	- *Index*: *HashMap<ExpressioBooleana, Set<KeyP>>*
Relacions	ExpressioBooleana, CtrlDocument
Funcions	- *IndexExpressionsBooleanes*(): *void* - *getIndex*(): *HashMap<ExpressioBooleana, Set<KeyP>>* - *getExpressions*(): *Set<ExpressioBooleana>* - *getDocumentsExpressio*(*ExpressioBooleana*): *Set<KeyP>* - *getDocumentsExpressio*(*String*): *Set<KeyP>* - *afegirExpressio*(*String*): *void* - *evaluarTotesExpressions*(*Document*): *void* - *esborratDocument*(*Document*): *void*

Nom	**Usuari**
Descripció	Representa un usuari que pot iniciar sessió al gestor de documents.
Atributs	- *nom*: *String* - *contrasenya*: *String*
Relacions	CtrlUsuari
Funcions	- *Usuari*(*String*, *String*): *void* - *getNom*(): *String* - *setNom*(*String*): *void* - *getContrasenya*(): *String* - *setConstrasenya*(): *void*

Controladors

Nom	**CtrlDomini**
Descripció	Controla la capa del domini
Atributs	- *ctrlConsultes: CtrlConsultes* - *ctrlDocument: CtrlDocument* - *ctrlUsuari: CtrlUsuari*
Relacions	CtrlConsultes, CtrlDocument, CtrlUsuari
Funcions	- CtrlDomini(): void - obrirDocumentAutorTitol(String, String): Document - recuperarDocument(String, String): Document - llistarDocumentsPC(ArrayList<String>,int): ArrayList<Document> - llistarDocumentsExpressio(String): ArrayList<Document> - llistarDocumentsKSemblants(String, String, int): ArrayList<Document> - llistarDocumentsAutor(String): ArrayList<Document> - iniciarSessio(String, String): void - eliminarDocument(String, String): void - modificarDocument(String, String): void - carregarDocument(): void - llistarAutorsPrefix(String): ArrayList<String> - crearUsuari(String, String): void - eliminarUsuaris(String, String): void - tancarSessio(): void - eliminarConsulta(int index): void - llistarConsultes(): ArrayList<Consulta> - reConsultar(int index): void

Nom	**CtrlDocuments**
Descripció	Controla els documents del gestor de documents
Atributs	- Cjt_documents: HashMap<KeyP, Document> - ParaulaDoc: HashMap<String, PairP<Integer, Boolean>> - numero_paraules: int - index: IndexAutor - indexDA: IndexDocumentsAutor - indexEB: IndexExpressionsBooleanes - eng_stop_words: HashSet<String> - ca_stop_words: HashSet<String> - sp_stop_words: HashSet<String>
Relacions	CtrlDomini, Document, IndexAutor, IndexDocumentsAutor, IndexExpressionsBooleanes
Funcions	- CtrlDocument(): void - EliminarDocument(String, String): void - NouDocument(Format, String, String, ArrayList<String>, String): void - ModificarInformacioDocument(String, String): void - RecuperarDocument(String, String): Document - ObrirDocumentAutorTitol(String, String): void - LlistarDocumentsSemblants(String, String, final int): HashMap<PairP<String, String>, Double> - LlistarDocumentsAutor(String): Set<String> - Llistar_query(ArrayList<String>, final int): HashMap<PairP<String, String>, Double> - add_document(Document): void - actualitzar_docs(Document, String): void - getDocument (String, String): Document - cat_stop_words(): HashSet<String> - eng_stop_words(): HashSet<String> - sp_stop_words(): HashSet<String> - LlistarAutorPrefix(String): ArrayList<String> - LlistarExpressioBooleana(String): Set<KeyP>

Nom	**CtrlUsuari**
Descripció	Controla els usuaris que utilitzen el gestor de documents
Atributs	- usuari: ArrayList<Usuaris> - usuariActiu: int
Relacions	CtrlDomini, Usuari
Funcions	- CtrlUsuari(): void - numeroUsuaris(): int - afegirUsuari(Usuari): void - eliminarUsuari(String, String): void - iniciarSessio(String, String): boolean - tancarSessio(): boolean - getUsuaris(): ArrayList<Usuari> - getUsuariActiu(): Usuari - indexUsuari(String): int

Nom	**CtrlConsulta**
Descripció	Controla les consultes realitzades per l'usuari. MAXSIZE: Nombre maxim de consultes
Atributs	- TotalConsultes: ArrayList<Consulta> - int MAXSIZE = 15
Relacions	CtrlDomini, Consulta
Funcions	- CtrlConsulta(): void - NovaConsulta(Consulta): void - DisplayHistorial(): ArrayList<Consulta> - EsborrarConsultes(): void - getConsulta(int): Consulta - esborrarConsulta(int): void

Utils

Nom	**KeyP**
Descripció	Representa una variable formada per dos Strings. Es diferencia de la classe PairP perquè pot ser comparat amb un altre KeyP.
Atributs	- k1: String - k2: String
Relacions	CtrlDocument, IndexExpressionsBooleanes
Funcions	- KeyP(String, String): void - getK1(): String - getK2(): String @Override: - equals(Object o): boolean - hashCode(): int

Nom	**PairP**
Descripció	Representa una variable formada per dos elements
Atributs	- first: F - second: S
Relacions	CtrlDocument
Funcions	- PairP(F, S): void - getFirst(): F - getSecond(): S

Nom	**SortedValue**
Descripció	És una classe que té com a funció ordenar un Map<PairP<String, String>, Double> segons el valor del Double
Atributs	No té atributs
Relacions	CtrlDocument
Funcions	- sortByValue(final Map<PairP<String, String>, Double> vec_ordenar): Map<PairP<String, String>, Double>

Nom	**Tree**
Descripció	Representa una estructura de dades en forma d'arbre. Cada node d'aquest guarda la informació pertinent i té dos fills.
Atributs	- root: Node<T> - class Node<T> La classe Node<T> representa un node de l'arbre de la classe Tree. Els atributs que té són: - data: T - l: Node<T> - r: Node<T> Està relacionada amb la classe Tree. Les funcions que té són: - getData(): T - setData(T): void - getL(): Node<T> - setL(Node<T>): void - getR(): Node<T> - setR(Node<T>): void
Relacions	ExpressioBooleana
Funcions	- Tree(T): void - Tree(): void - getRoot(): Node<T>

Nom	**Trie**
Descripció	Representa una estructura de dades en forma d'arbre. Cada node d'aquest és un map que conté characters, un punter que apunta al node següent i un booleà que indica si aquell node és un fi de paraula. Facilita la cerca de paraules segons un prefix, com en el cas de llistar autors per prefix.
Atributs	- root: TrieNode - class TrieNode La classe TrieNode representa un node de l'arbre de la classe Trie. Els atributs que té són: - children: HashMap<Character, TrieNode> - endOfWord: boolean Està relacionada amb la classe Trie. Les funcions que té són: - TrieNode(): void
Relacions	IndexAutor
Funcions	- Trie(): void - insert(String): void - removeautor(TrieNode, String, int): boolean - remove(String): boolean - search(String): boolean - search_prefix(String): boolean - rec(String, TrieNode): ArrayList<String> - retornautors(String): ArrayList<String>

Nom	**TipusConsulta: Enum**
Descripció	Enum: Un per cada cas d'us de llistar documents, autors i obrir document
Valors	- *docsAutor* - *autorsPrefix* - *docAutorTitol* - *docsKSemblants* - *docsExpressio* - *docsPC*

Nom	**Format: Enum**
Descripció	Enum: Un per cada format que pot tenir un document
Valors	- *txt* - *xml* - *prop*

5. Estructura de dades i algorismes

Domini

Consulta

ESTRUCTURES DE DADES I ATRIBUTS:

private ArrayList<String> parametres;

private TipusConsulta tipus;

La clase consta de dos atributs, els paràmetres i el tipus de consulta. El número màxim de paràmetres varia en funció del tipus de consulta.

EXCEPCIONS:

- *ExceptionConsultaLimitParametres*: No es pot superar el nombre màxim de paràmetres. Constructora i *nouParametre*.

MÈTODES I ALGORISMES EMPRATS:

- *Consulta*

  La constructora, crea una nova consulta. Els paràmetres d'entrada són un *ArrayList<String>* amb els paràmetres i un tipus de consulta, membre de l'enumeració *TipusConsulta*.
- *getParametres*

  getter de l'atribut *parametres*.
- *getTipus*

  getter, retorna l'atribut *tipus*.
- *setParametres*

  Assigna el paràmetre d'entrada, un *ArrayList<String>*, a l'atribut *parametres*.
- *setTipus*

  Assigna el paràmetre d'entrada a *tipus*
- *nouParametre*

  Aquesta funció té un únic *String par* com a paràmetre d'entrada. Primer es comprova que el número de parametres no es el màxim i, en cas afirmatiu salta

l'excepció *ExceptionConsultaLimitParametres*. Si el nombre de paràmetres no és el màxim es guarda el valor de *par* a l'atribut *parametres*.

- *getExpressio(): String*

  Retorna el *String* que s'ha de veure en pantalla al mostrar l'historial.

- *checkNumParametres*

  Pot retornar tres enters diferents segons el número de paràmetres per un cert tipus d'expressió.

  Possibles valors de retorn:

  0: si *parametres.size*() < MAX SIZE

  1: si *parametres.size*() = MAX SIZE

  2: si *parametres.size*() > MAX SIZE

- *Equals i hashCode*

  Funcions per comprovar les igualtats entre instàncies de la classe.

Totes les funcions d'aquesta classe tenen cost constant O(1).

RELACIONS AMB ALTRES CLASSES: Es relaciona amb *CtrlConsulta*. *CtrlConsulta* conté un conjunt de instàncies d'aquesta classe.

Document

ESTRUCTURES DE DADES I ATRIBUTS:

Els documents són un dels principals objectes d'aquest treball. Aquests són identificats per tres atributs fonamentals; el format, el títol i l'autor.

private Format format;

private String titol;

private String autor;

Totes tres són les *primary key* o claus primàries, per tant, no poden tindre valor null.

Els documents també tenen altres atributs, no menys importants, que ens emmagatzemen certa informació molt rellevant. Aquests són:

- *HashMap<String,Double> ParaulaPesLocal:* es tracta d'un HashMap on s'emmagatzema la freqüència unitària de cada paraula dins del contingut del document. Aquests valors seran molt importants sobretot en aquelles funcions o accions on s'hagi de comparar la similitud entre documents.
- *private ArrayList<String> contingut*: es tracta d'un conjunt de strings que ens permeten guardar el text que conté el nostre document. Aquest atribut pot ser buit.
- *double norma*: aquest atribut ens permet calcular la norma de les paraules del conjunt. Entenem com a norma, l'arrel de les potències quadràtiques de totes les freqüències unitàries del document.

EXCEPCIONS:

Per tal de respectar i conservar la integritat del nostre gestor, tenim certes excepcios que calen ser comprovades quan tractem amb documents.

- *ExceptionNotPrimaryKeys*

  Aquesta excepció salta quan s'ha fet un tractament amb un document i, les noves primary keys d'aquest document, no són correctes (no compleixen les propietats de claus primàries i tenen valor null). Bàsicament s'aplica quan es crea un document nou o quan es modifica un d'existent.

- *ExceptionFormatNoValid*

Aquesta excepció salta quan el format d'un document no pertany a cap tipus dels que existeixen; és a dir, el document no té cap format del enumeration de formats existent.

MÈTODES I ALGORISMES EMPRATS:

- *Document*

  aquesta funció és la constructora que ens permet crear un nou objecte document. Requereix de certs paràmetres (les claus primàries, el contingut i un map de stopwords) que seran assignats al nou objecte. El cost d'aquesta funció és la suma de costos d'assignar aquest valors i la crida a la funció *actualitzar_pesos*. Per tant és constant (O(1)) per a les assignacions de valors i O(n) on n és el nombre de paraules del contingut, per a actualitzar_pesos .

  És a dir, el cost és O(n).

- *ModificarInformacioDocument*

  Aquesta funció ens permet modificar els atributs del objecte sense necessitat de haver-lo d'eliminar i crear un de nou amb els atributs desitjats. Aquesta funció requereix de certs paràmetres que seran assignats als diferents atributs de l'objecte. Aquest són el nou títol, el nou autor, el nou contingut, l'idioma del nou document, els canvis que es volen fer i les *stopwords* de l'idioma del document.

  El cost de la funció en cas de modificar el títol o l'autor és constant. Si es modifica el contingut, es realitza una crida a *actualitzar_pesos* i el cost passa a ser O(n) on n és el nombre de paraules del contingut.

- *actualitzar_pesos*

  Aquesta funció és l'encarregada de conservar la coherència del valor de les freqüències unitàries de les paraules. És aplicada sempre que es faci una modificació en el contingut d'un document ja que és l'únic cas on s'han d'actualitzar aquestes freqüències. Rep com a parametres el llistat de stopwords de l'idioma del document, i el nombre de paraules de l'antic contingut. Aquest primer paràmetre ens permet saber quines paraules no són rellevants i no han de ser en el map ParaulaPesLocal, mentre que el segon serà utilitzat per a poder passar la freqüència unitària a freqüència respecte en nombre de paraules. El seu cost és O(n) on n és el nombre de paraules del contingut.

Tenim també els getters que ens permeten poder obtindre el valor dels atributs privats del document. El cost d'aquestes funcions es constant.

- *getTitol*
- *getAutor*
- *getFormat*
- *getContingut*
- *getPesosLocals*

Tenim els *setter* que ens permeten establir un valor als atributs privats del document. El cost d'aquestes funcions es constant.

- *setTitol*
- *setAutor*
- *setFormat*
- *setContingut;* en aquest cas el cost és O(n) on n és el nombre de paraules del contingut.

Printers:

- *Informacio_Document*

  Aquesta funció ens retorna per terminal els diferents atributs del document. El cost és la suma de obtindre les claus primàries (constant) i el de la crida a *llistar_contingut* (O(n) on n és el nombre de paraules del contingut).
- *llistar_contingut*

  Aquesta funció ens retorna el contingut del document per terminal. El cost és de O(n) on n és el nombre de paraules del contingut.

RELACIONS AMB ALTRES CLASSES:

Aquesta classe es relaciona amb la classe CtrlDocument, ja que de fet, té polimorfisme en algunes funcions com en la de ModificarInformacioDocument. La classe CtrlDocument accedeix a Document en algunes funcions i de fet té un conjunts d'instàncies d'aquesta classe.

Expressió Booleana

ESTRUCTURES DE DADES I ATRIBUTS:

L'atribut principal de la classe és un arbre binari anomenat *expressió*. És allà on s'emmagatzema el contingut de l'expressió booleana, preparat per ser evaluat. Els nodes poden ser o connectives lògiques (*and*, *or*, *not*) o bé paraules o seqüències de paraules. Els operands de cada node operador estan continguts als seus dos nodes fills, o dins el fill dret, en cas de l'operació *not*, que només s'aplica sobre un element.

Hem escollit utilitzar un arbre perquè aquesta estructura ens permet aprofitar la naturalesa de cada connectiva lògica per fer l'avaluació més eficient. Per exemple: si el primer dels operands d'una operació *and* resulta ser fals, podrem ignorar el segon dels operands, perquè ja sabem que el resultat de l'operació és *false*. A part, hem decidit que l'arbre fos binari, perquè totes les connectives lògiques tenen un màxim de dos operadors.

El segon atribut de la classe és un vector *prioritatAsc* que conté tots els operands ordenats segons el seu ordre de prioritat ascendent. És a dir: *or*,*and*,*not*,),(. Aquest vector ens ajudarà a construir l'arbre a la funció *buscarInici*. Explicarem la seva funció més endavant.

EXCEPCIONS:

- *ExpressióBooleanaIncorrecta*: Salta quan l'usuari introdueix una expressió que no és correcta. La correctesa de l'expressió s'ha avaluat a la funció *comprovaExpressio*.

MÈTODES I ALGORISMES EMPRATS:

- *getExpressio*

  Un simple *getter* per retornar l'atribut *expressio*.

- *setExpressio*

  L'únic paràmetre el la funció és un *string* corresponent a la cadena de caràcters que posa l'usuari al terminal, l'expressió. Però nosaltres volem guardar l'expressió a un arbre. El primer que farem, donat que l'usuari pot escriure l'expressió de moltes maneres diferents (amb espais entremig, sense, o bé posant-ne a vegades), serà adaptar-la a una manera estàndard: això es fa a la funció *tradueixExpressio*. Un cop traduïda procedirem a entrant a la funció *comprovaExpressio*, que comprova que l'expressió entrada sigui correcta. En cas afirmatiu, es construeix l'arbre i es situa a l'atribut *expressio* de la classe.

- *tradueixExpressio*

  Aquesta funció rep l'expressió que l'usuari entra el programa, i en retorna una d'adaptada per poder construir l'arbre. La funció recorre l'expressió caràcter per caràcter i busca tots els espais en blanc. Tots aquells espais que es trobin dins de claudàtors els substituirà per símbols *and* (&), els que es troben entre cometes els mantindrà igual, la resta els esborrarà. Havent recorregut tot el bucle, substituirà tots els claudàtors per parèntesis.

  En altres paraules, tots els subconjunts que entri l'usuari seran substituïts per seqüències d'operacions *and* entre parèntesis, que tenen significat equivalent al d'un subconjunt. Al recorrer tota la seqüència, la funció tindrà cost $O(n)$.

- *comprovaExpressio*

  Aquesta funció rep d'entrada una expressió ja traduïda i retorna *cert* si aquesta expressió és correcta, i fals altrament. S'ha assumit que una expressió NO és correcta en els següents casos:

  1. L'expressió conté un operador *and* o *or*, a l'inici o al final de l'expressió, o bé conté dos operadors *and* o *or* consecutius.
  2. L'expressió conté un operador not abans d'un operador, o al final de la seqüència.
  3. Els parèntesis concorden.

  Per fer-ho es recorre la seqüència caràcter per caràcter i cas per cas i s'actua segons el caràcter llegit. Si trobem un cas on la funció no és correcte la pararem de recórrer i retornarem fals.

  Com el lector ja haurà pogut deduïr, el cas pitjor de l'algorisme serà el que recorrem tota la seqüència de caràcters (el cas de l'expressió correcta). Per tant el cost d'aquesta funció serà $O(n)$, on n .

- *ConstrueixArbre*

  Aquesta funció serveix per construir els arbres d'expressions booleanes donada una *String* amb l'expressió i un node corresponent a l'arrel on començarem el nou l'arbre.

  El mètode per construir l'arbre és molt senzill. Comencem pensant com resoldríem nosaltres, les persones, una expressió donada. Des de petits se'ns ensenya que les resolucions d'aquest tipus es fan establint un ordre de prioritat entre els operadors. En cas de les expressions booleanes primer hem de resoldre

les expressions entre parèntesis, després aquelles expressions amb un operador *not*, després les operacions *and*, i per acabar les portes *or*.

Nosaltres estructurarem l'arbre perquè segueixi el mateix ordre. Ara bé, cal recordar que quan treballem amb arbres utilitzem funcions recursives que, per cada branca, analitzen primer les fulles, i acabem la funció analitzant l'arrel. Com que nosaltres volem que les operacions de prioritat alta siguin les primeres a ser analitzades, les haurem de posar als nivells més baixos de l'arbre, prop de les fulles, així seran les primeres a ser analitzades.

És per això que el primer que farà el nostre algorisme serà buscar l'operador de MENYS prioritat dins l'expressió començant per la dreta. Per a què comencem per la dreta? Doncs perquè quan nosaltres tenim una operació amb dos signes de prioritat equivalent, sempre donarem més prioritat al de l'esquerra, sent llavors l'element de la dreta el menys prioritari. La cerca de l'element menys prioritari es fa des de la funció *buscarInici*, que retornarà un enter *inici* equivalent a la posició de l'expressió amb l'operador de menys prioritat.

Sabent que l'inici es troba allà, siturem aquell caràcter al node arrel i dividirem la cadena de caràcters restant en dos: per una banda tindrem els caràcters situats abans que *inici* i per altra els posterios a *inici*. El següent pas serà cridar la mateixa funció *construirArbre* pels fills esquerre i dret passant com a paràmetre els caràcters anteriors i posteriors a *inici*, respectivament.

D'aquesta manera es repetirà el mateix procés fins que l'operació restant només contingui una variable. En aquell moment haurem arribat a una fulla i haurem acabat de construir la branca en qüestió. Com a apunt interessant, és interessant veure com les fulles d'un arbre seran sempre variables, mai operadors.

A part, mentre construim l'arbre anirem eliminant tots els parèntesis de l'expressió ja que només els fem servir per establir un ordre de prioritat, una vegada ja hem calculat *inici* ja no els necessitarem. Les cometes també les eliminarem, i situarem totes les paraules contingudes dins un mateix node. Cal recordar també que no ens hem de preocupar sobre com tractar els subconjunts, ja que hem "traduït" l'expressió i els subconjunts ara són expressions booleanes normals.

Pel que fa el cost, hem de revisar tants nodes com caràcters hi ha a l'expressió, per tant serà lineal.

- *buscarInici*

  Aquesta funció té un únic paràmetre d'entrada, l'expressió en forma de *String*. Retornarà quin és el caràcter de menys prioritat de l'expressió començant per la dreta. Per establir l'ordre de prioritat afegim tots els possibles operadors a l'atribut *prioritat*Asc. Després es recorrerà *prioritat*Asc i es buscarà cada operador a l'expressió començant per la dreta.

  El cost de *buscarinici* en cas pitjor (l'expressió és només una paraula i no té operadors), correspon a

  $O(prioritatAsc.length() * expressio.length()) = O(5 * expressio.length()) = O(expressio.length)$

- *evaluate*:

  Aquesta funció rep com a paràmetre un document i retorna *true* si el document compleix 'expressió booleana, *false* altrament. No s'entrarà gaire en el funcionament d'*evaluate* ja que l'avaluació en sí es fa a la funció recursiva *evaluateRec*, que es crida des d'aquesta funció. L'únic que fa *evaluate* és extreure el contingut del document i cridar *evaluateRec* perquè l'analitzi.

- evaluateRec: S'encarrega de fer l'avaluació recursiva de l'expressió donat el contingut d'un document en un array de *String*. La funció fa crides recursives fins a arribar a les fulles, recordem que a les fulles de l'arbre sempre hi haurà paraules. Per tant quan arribem a la fulla només haurem de comprovar que el document contingui aquesta paraula o seqüència de paraules. El resultat (*true* o *false*) de les fulles passarà al nivell superior de l'arbre, on hi haurà un operador que l'avaluarà, i així recursivament. Per tal d'estalviar-nos comprovacions, aprofitarem la naturalesa de les connectives lògiques per eliminar parts de l'arbre durant una avalucaió. Per exemple, si el fill esquerre d'un arbre és *true* i l'arrel és un operador *or*, no caldrà avaluar el fill dret de l'arbre, ja que el resultat de l'operació serà true independentment del valor del fill dret.

  El cost en cas pitjor d'aquest algorisme el trobem quan l'avaluació d'un arbre no ens permet ometre cap branca. En aquest cas seria $O(n)$, on *n* és el nombre de caràcters de l'expressió traduïda.

- *getArbre*: Retorna l'arbre en forma d'*String* i seguint una estructura de preordre.

  El cost de la funció és lineal.

RELACIONS AMB ALTRES CLASSES:

La classe només es relaciona amb la classe *IndexExpressionsBooleanes*, que conté un índex amb totes les Expressions que ha entrat l'usuari.

Índex Autor

ESTRUCTURES DE DADES I ATRIBUTS:

En aquesta classe només hem fet servir una estructura de dades, que és un Trie al qual hem anomenat indexautor. En aquest trie guardem tots els autors que algun cop han tingut un document guardat al gestor de documents. El funcionament d'aquesta estructura de dades l'explicarem de forma més desenvolupada a la classe Trie. Hem decidit utilitzar-la, ja que facilita la cerca d'autors per prefix en un termini de temps més curt que si haguéssim de recorrer el HashMap de Cjt_documents i comprovar per cada Document si el seu autor conté el prefix desitjat.

EXCEPCIONS:

- *ExceptionAutorJaExisteix*
  Salta quan l'usuari intenta fer una operació d'afegir un element nou al Trie, que ja existeix.
- *ExceptionAutorNoExisteix*
  Salta quan l'usuari intenta fer una operació d'eliminar o cercar un element al Trie, que no forma part d'aquest.

MÈTODES I ALGORISMES EMPRATS:

- *IndexAutor(): void*
  És la creadora de la classe IndexAutor. Té cost constant.
- *afegirAutor_sempre(String): void*
  Afegeix el string amb el nom d'un autor a l'estructura de dades.
  No fa la comprovació de que l'autor no existeixi ja, perquè es pot donar el cas en el que hi hagi dos documents amb el mateix autor i títol diferents. En aquest cas el document que voldríem afegir al gestor de documents no existiria, però l'autor sí i per tant no volem que salti cap excepció.
  Aquesta funció només crida a la funció insert del trie, per tant tindrà el mateix cost que aquesta. És a dir, el cost serà lineal O(m) on m és la mida de la paraula que volem inserir a l'índex.
- *afegirAutor(String): void*
  Afegeix el string amb el nom d'un autor a l'estructura de dades.
  Primerament comprova que l'autor que volem afegir no existeixi ja. Si ja existeix al nostre gestor, salta l'excepció *ExceptionAutorJaExisteix* que ho indica. En cas

contrari, el string amb el nom d'un autor és afegit a l'índex.

Aquesta funció primer fa una comprovació d'una excepció per comprovar que l'autor no existeixi ja a la funció. Per fer-ho crida a la funció *search* del trie que té cost lineal O(m) on m és la longitud del string que volem cercar a l'índex. Després crida la funció *insert* del trie, que té cost lineal O(m) on m és la mida de la paraula que volem inserir a l'índex.

El cost final d'aquesta funció és O(m+m), que podem igualar a O(2m), que és el mateix que O(m). Per tant, podem dir que aquesta funció té cost lineal O(m).

- *eliminarAutor(String): void*

  Elimina el string amb el nom d'un autor de l'estructura de dades.

  Primerament comprova que l'autor que volem eliminar existeixi ja a líndex. Si no existeix aquest autor al nostre gestor, salta l'excepció *ExceptionAutorNoExisteix* que ho indica. En cas contrari, el string amb el nom de l'autor és eliminat de l'índex. Per fer-ho crida a la funció *search* del trie que té cost lineal O(m) on m és la longitud del string que volem cercar a l'índex.

  Aquesta funció primer fa una comprovació d'una excepció per comprovar que l'autor existeixi ja a la funció. Per fer-ho crida a la funció *search* del trie que té cost lineal O(m) on m és la longitud del string que volem cercar a l'índex. Després crida la funció *remove* del trie, que té cost lineal O(m) on m és la mida de la paraula que volem eliminar de l'índex.

  El cost final d'aquesta funció és O(m+m), que podem igualar a O(2m), que és el mateix que O(m). Per tant, podem dir que aquesta funció té cost lineal O(m).

- *cercarAutors(String): ArrayList<String>*

  Cerca tots els autors que comencin amb el string del prefix indicat.

  Primerament comprova que existeixi el prefix a la nostra estructura de dades. Cal tenir en compte que el prefix no té perquè ser el nom d'un autor existent. Per tant buscarem que existeixi el string al trie, però no caldrà que el node al qual apunti l'última lletra del prefix indiqui que és un fi de paraula. Si no existeix el prefix al trie, saltarà l'excepció *ExceptionAutorNoExisteix*. En cas contrari, es retorna un ArrayList de Strings ordenat amb tots els noms dels autors que contenen aquest prefix.

  Aquesta funció primer fa una comprovació d'una excepció per comprovar que el prefix de l'autor existeixi ja a la funció. Per fer-ho crida a la funció *search* del trie

que té cost lineal O(p) on p és la longitud del string prefix que volem cercar a l'índex. Després crida la funció *retornautors* del trie, que té cost O(p+l) on *p* és la longitud del string prefix i *l* és el nombre d'autors que podem trobar a l'índex.

El cost final d'aquesta funció és O(2p+l). Per tant, ho podem igualar a O(p+l) i dir que aquest és el cost de la nostra funció.

- *canvi_Autor(String, String): void*

  Substitueix el nom d'un autor existent a la nostra estructura de dades, pel nom d'un nou autor no existent fins el moment.

  Primerament comprova si l'autor que volem eliminar del trie existeix. Si no existeix, salta l'excepció *ExceptionAutorNoExisteix* que ho indica.

  En segon lloc, comprova que el nou nom d'autor que volem afegir al trie no existeix ja a la nostra estructura de dades. Si ja existeix, salta l'excepció *ExceptionAutorJaExisteix*. En cas contrari, eliminem el nom antic de l'autor i afegim el nou nom d'aquest.

  Aquesta funció primer fa una comprovació d'una excepció per comprovar que l'autor que volem canviar existeixi ja a la funció. Per fer-ho crida a la funció *search* del trie que té cost lineal O(*m*) on m és la longitud del string que volem cercar a l'índex. En segon lloc, fa una comprovació d'una excepció per comprovar que l'autor que volem afegir no existeixi ja a la funció. Per fer-ho crida una altra vegada a la funció *search* del trie que té cost lineal O(*n*) on n és la longitud del string que volem cercar a l'índex. Si no salta cap de les excepcions, eliminem l'antic autor (cost O(*m*) on m és la longitud del string que volem eliminar de l'índex) i afegim el nou autor (cost O(n) on n és la longitud del string que volem afegir a l'índex).

  El cost final d'aquesta funció és O(2*m* + 2n), per tant és lineal O(m + n) on m és la longitud del string de l'antic nom de l'autor que volem modificar i n és la longitud del nou nom de l'autor que volem afegir.

RELACIONS AMB ALTRES CLASSES:

La funció d'aquesta classe és agilitzar la cerca d'autors per prefix.

És una classe agregada al *CtrlDocuments*, perquè es pugui fer servir com a índex pels autors.

Aquesta classe també està relacionada amb la classe Trie, ja que és on tenim implementades les operacions per poder afegir, eliminar i cercar elements de la nostra estructura de dades.

Índex Documents Autor

ESTRUCTURES DE DADES I ATRIBUTS:

En aquesta classe hem fet servir com a estructura de dades un HashMap on la clau és un String que identifica un autor del gestor de documents i el valor per cadascuna d'elles és un Set<String> per guardar tots els títols dels documents que existeixin d'aquest autor a l'aplicació.

L'objectiu d'aquesta classe és facilitar la cerca de títols de documents d'un autor i evitar haver de recórrer tot el conjunt de documents del gestor per trobar-los tots.

EXCEPCIONS:

- ExceptionAutorNoExisteix:
  Salta quan l'usuari intenta fer una operació d'eliminar un títol d'un autor o cercar els documents d'un autor, i aquest autor no existeix a la plataforma.

MÈTODES I ALGORISMES EMPRATS:

- IndexDocumentsAutor(): void
  És la creadora de la classe IndexDocumentsAutor. La creació d'un nou HashMap amb valor null és constant.
- afegeixTitolAutor(String, String): void
  Afegeix un títol al Set<String> de títols de l'autor seleccionat.
  En primer lloc comprova si l'autor en qüestió forma part del HashMap. En cas negatiu s'afegeix, crea el set de strings de l'autor i hi afegeix el títol. En cas afirmatiu afegeix directament el títol al set de l'autor.
  Aquesta funció té cost constant O(1), ja que és el cost de cercar i afegir elements en un HashMap.
- eliminarTitolAutor(String, String): void
  Elimina el títol i autor del gestor de l'índex.
  Primerament comprova si l'autor forma part de l'índex. Si no existeix al HashMap, salta l'excepció *ExceptionAutorNoExisteix* indicant-ho. En segon lloc, comprova si el títol forma part del Set<String> de l'autor en qüestió. Si no forma part del Set, salta l'excepció *ExceptionDocumentNoExisteix* indicant-ho. En cas contrari,

s'elimina el títol del set de strings de l'autor. Si l'autor ja no té més documents existents al gestor de documents, s'elimina també de l'índex.

Aquesta funció té cost constant O(1), ja que és el cost de cercar i eliminar elements d'un HashMap.

- buscarTitolsAutor(String): Set<String>

  Retorna un Set de strings amb tots els títols existents al gestor de documents al moment.

  Primerament es comprova que l'autor existeixi a l'índex, és a dir, que tingui algun document al gestor. Si no existeix, salta l'excepció *ExceptionAutorNoExisteix*. En cas contrari, es retorna el set amb els títols.

  Aquesta funció té cost constant O(1), ja que és el cost de retornar el punter al Set<Strings> de títols de l'autor.

RELACIONS AMB ALTRES CLASSES:

La funció d'aquesta classe és agilitzar la cerca dels títols dels documents d'un autor.

És una classe agregada al CtrlDocuments, perquè es pugui fer servir com a índex dels títols d'un autor.

Índex Expressions Booleanes

ESTRUCTURES DE DADES I ATRIBUTS:

L'únic atribut de la classe és un HashMap d'expressions booleanes i un *Set* de *KeyP*, que contindrà els documents que les compleixen. S'ha escollit aquesta estructura per aprofitar que té una funció per agafar totes les claus (les expressions booleanes), o tots els valors (les expressions que les compleixen).

EXCEPCIONS: Aquesta classe no té excepcions.

MÈTODES I ALGORISMES EMPRATS:

- *getIndex*

  Retorna l'atribut *Índex* amb la relació d'expressions booleanes i elements que les compleixen.
- *getExpressions*

  Retorna el conjunt d'Expressions de l'índex.
- *getDocumentsExpressio*

  Cerca lineal d'una expressio que acaba retornant tots els documents que la compleixen. El seu cost és $O(n)$.
- *afegirExpressio*

  S'afegeix una expressió a l'índex. L'únic paràmetre passat és una *String* corresponent a l'expressió en qüestió. La funció crea una nova expressió i busca si n'hi ha alguna d'idèntica a l'índex. En cas negatiu l'afegeix. El cost és lineal, ja que s'ha de revisar que l'expressió no coincideixi amb cap de les altres de l'índex.
- *evaluarTotesExpressions*

  Donat un document *d*, s'avaluen totes les expressions de l'índex en *d*. Per cada expressió que es compleixi, s'afegiran l'autor i el títol del document al *Set* que acompanya l'expressió en qüestió. Com que hem d'avaluar totes les expressions, el cost serà lineal.

  Aquesta funció es fa servir a l'hora de crear o modificar documents.
- *esborratDocument*

  Donat un document *d*, s'esborra totes les ocurrències dels identificadors del document en qüestió. El cost serà quadràtic perquè hem de cercar per a cada expressió dins el *Set* d'identificadors de documents.

RELACIONS AMB ALTRES CLASSES: Aquesta classe és una agregació de *CtrlDocument*, també es relaciona amb *ExpressióBooleana*.

Usuari

ESTRUCTURES DE DADES I ATRIBUTS:

private String nom

private String contrasenya

EXCEPCIONS: No hi ha excepcions en aquesta classe.

MÈTODES I ALGORISMES EMPRATS:

getNom(): String

setNom(String): void

getConstrasenya(): String

setConstrasenya(String): String

Fins a la capa de persistencia els usuaris només poden iniciar i tancar sessió. Només es necessiten els getters i setters.

RELACIONS AMB ALTRES CLASSES:

Es relaciona amb CtrlUsuari. CtrlUsuari conte el conjunt de totes les instàncies de la classe a l'aplicació.

Controladors

CtrlDomini

ESTRUCTURES DE DADES I ATRIBUTS:

No té atributs.

EXCEPCIONS:

En aquesta classe podem trobar totes les excepcions que son llançades per els tres controladors *CtrlDocument*, *CtrlUsuari*, *CtrlConsulta*:

ExceptionUsuariNoExisteix, *ExceptionSessioJaActiva*, *ExceptionContrasenyaIncorrecta*, *ExceptionDocumentNoExisteix*, *ExceptionNoSessioActiva*, *ExceptionAutorNoExisteix*, *ExceptionFormatNoValid*, *ExceptionAutorJaExisteix*, *ExceptionConsultaLimitParametres*, *ExceptionExpressioBooleanaIncorrecta*, *ExceptionNoEnterValid*, *ExceptionDocumentJaExisteix*, *ExceptionAutorJaExisteix*, *ExceptionUsuariJaExisteix*, *ExceptionIndexConsultaNoValid.*

MÈTODES I ALGORISMES EMPRATS:

Per a tots els mètodes que requereixen una sessió activa es comprova i es llença *ExceptionNoSessioActiva* si és necessari. Això són tots els mètodes menys iniciar i tancar sessió i crear i eliminar usuaris.

- *IniciarSessio*

  Es crida a la funció Iniciar Sessió de el controlador de usuaris.

  ExceptionUsuariNoExisteix, *ExceptionSessioJaActiva*, *ExceptionContrasenyaIncorrecta*
- *EliminarDocument*

  Es crida a la funció Eliminar document de el controlador de documents.

  ExceptionDocumentNoExisteix, *ExceptionNoSessioActiva*, *ExceptionAutorNoExisteix.*
- *ModificarDocument:*

  Primer es comprova que el document existeix, si es aixi, l'usuari introdueix les noves dades. Llavors es crida a la funció Modificar document de el controlador de documents.

ExceptionDocumentNoExisteix, *ExceptionNoSessioActiva*, *ExceptionAutorNoExisteix*, *ExceptionFormatNoValid*, *ExceptionAutorJaExisteix*, *ExceptionDocumentJaExisteix*.

- CarregarDocument:

  Al implementar la capa de persistencia aquest mètode permet afegir un document a l'aplicació.

  ExceptionNoSessioActiva.

- ObrirDocumentAutorTitol:

  Es crida a la funció *getDocument* del controlador de documents. Si l'operació no ha llençat cap excepció es crea una consulta amb els paràmetres pertinents i amb tipus *docAutorTitol* i s'afegeix al historial.

  ExceptionNoSessioActiva, *ExceptionDocumentNoExisteix*, *ExceptionConsultaLimitParametres*.

- RecuperarDocument:

  Guarda un document de l'aplicació en el directori local a escollir.

  ExceptionNoSessioActiva, ExceptionDocumentNoExisteix.

- LlistarAutorsPrefix:

  Es crida a la funció *llistarAutorsPrefix* de el controlador de documents. Aquesta funció retorna un *ArrayList* amb els noms dels autors. Si l'operació no ha llençat cap excepció es crea una consulta amb els paràmetres pertinents i amb tipus autors Prefix i s'afegeix al historial.

  ExceptionNoSessioActiva, *ExceptionConsultaLimitParametres*, *ExceptionAutorNoExisteix*.

- LlistarDocumentsExpressio:

  Es crida a la funció *LlistarExpressioBooleana* del controlador de documents. Aquesta funció retorna un *ArrayList* de *Document*. Si l'operació no ha llençat cap excepció es crea una consulta amb els paràmetres pertinents i amb tipus *docsExpressio* i s'afegeix al historial.

  ExceptionNoSessioActiva, *ExceptionExpressioBooleanaIncorrecta*, *ExceptionConsultaLimitParametres*.

- LlistarDocumentsKSemblants:

  Es crida a la funció *LlistarDocumentsSemblants* del controlador de documents. Aquesta funció retorna un *ArrayList* de *Document* amb els Documents que són

semblants al Document amb l'autor i títol passat com a paràmetre. Si l'operació no ha llançat cap excepció es crea una consulta amb els paràmetres pertinents i amb tipus *docsKSemblants* i s'afegeix a l'historial.

ExceptionDocumentNoExisteix, *ExceptionNoSessioActiva*, *ExceptionNoEnterValid*, *ExceptionConsultaLimitParametres.*

- *LlistarDocumentsAutor*

  Es crida a la funció *LlistarDocumentsAutor* de el controlador de documents. Aquesta funció retorna un ArrayList de Document amb els Documents que tenen l'autor passat com a paràmetre. Si l'operació no ha llançat cap excepció es crea una consulta amb els paràmetres pertinents i amb tipus *docsAutor* i s'afegeix a l'historial.

  ExceptionNoSessioActiva, *ExceptionAutorNoExisteix*, *ExceptionConsultaLimitParametres.*

- *LlistarDocumentsPC*:

  Es crida a la funció *Llistar_query* de el controlador de documents. Aquesta funció retorna un ArrayList de Document amb els Documents que contenen les paraules clau passades com a paràmetre. Si l'operació no ha llançat cap excepció es crea una consulta amb els paràmetres pertinents i amb tipus *docsPC* i s'afegeix a l'historial.

  ExceptionNoSessioActiva, *ExceptionConsultaLimitParametres*, *ExceptionNoEnterValid*, *ExceptionDocumentNoExisteix*, *ExceptionFormatNoValid*, *ExceptionDocumentJaExisteix*, *ExceptionAutorNoExisteix*, *ExceptionAutorJaExisteix*

- *CrearUsuari*

  Es crida a la funció *crearUsuari* de el controlador d'usuaris. Es passen coma paràmetres el nom i contrasenya del nou usuari.

  ExceptionNoSessioActiva.

- *EliminarUsuari*

  Es crida a la funció *eliminarUsuari* de el controlador d'usuaris. Es passen coma parametres el nom i contrasenya de l'usuari.

  ExceptionUsuariNoExisteix, *ExceptionContrasenyaIncorrecta.*

- *TancarSessio*

  Es crida a la funció tancar de el controlador d'usuaris.

ExceptionNoSessioActiva.

- *EliminarConsulta*

  Es crida a la funció *esborrarConsulta* del controlador de consultes. Es passa com a paràmetre l'índex de la consulta a eliminar.

  ExceptionIndexConsultaNoValid, *ExceptionNoSessioActiva*.

- *LlistarConsultes*

  Es crida a la funció *DisplayHistorial* de e controlador de consultes. Retorna un ArrayList amb totes les consultes guardades al historial, ordenades de més antiga a menys.

  ExceptionNoSessioActiva.

- *ExisteixDocument*

  Retorna true si el Document amb el nom de autor i títol proporcionats existeixen al conjunt de *CtrlDocuments*. Aquesta funció és necessària per comprovar que un document existeix abans de demanar l'entrada de la operació *modificarDocument*.

- *SessioActiva*

  Retorna *true* si hi ha un usuari en sessió a l'aplicació.

RELACIONS AMB ALTRES CLASSES:

Aquesta classe està formada per la agregació dels tres controladors: *CtrlConsulta*, *CtrlDocument i CtrlUsuari*. Així, per cada mètode de aquesta classe només s'ha de cridar a les operacions pertinents.

CtrlDocument

ESTRUCTURES DE DADES I ATRIBUTS:

El Controlador de Documents, és una classe pare de Documents, que emmagatzema tots els documents existents així com altre informació important d'aquests.

Les estructures de dades i els atributs d'aquesta classe són:

- *HashMap<KeyP, Document> Cjt_documents*

  Es tracta d'un HashMap on es guarden tots els documents existents. S'utilitza un HashMap, amb clau una clau doble formada per l'autor i el títol, ja que el cost de cerca és constant i a més, ens permet que les claus primàries no es repeteixin.

- *HashMap<String, PairP<Integer, Boolean>> ParaulaDoc*

  Es tracta d'un HashMap amb clau un string que fa referència a una paraula que pertany al contingut d'algun o més d'un document. El valor d'aquest objecte del HashMap, és el nombre de documents diferents on apareix la clau (la paraula).

- *IndexAutor index*

  Aquest atribut ens permet relacionar (agregar) la classe IndexAutor a la classe CtrlDocument. Amb aquesta agregació, es pot mantenir de manera sincronitzada els autors amb els documents.

- *IndexDocumentsAutor indexDA*

  Aquest atribut ens permet relacionar (agregar) la classe IndexDocumentsAutor a la classe CtrlDocument. Amb aquesta agregació, es pot mantenir de manera sincronitzada la informació que fa referència als documents dels autors.

- *IndexExpressionsBooleanes indexEB*

  Aquest atribut ens permet relacionar (agregar) la classe IndexExpressionsBooleanes a la classe CtrlDocument. Amb aquesta agregació, es pot mantenir de manera sincronitzada les expressions booleanes amb els documents.

- *StopWords*

  Finalment trobem tres *HashSet<String> eng_stop_words*, *ca_stop_words* i *sp_stop_words*, on estan emmagatzemades les stopwords dels diferents idiomes.

EXCEPCIONS:

Per tal de respectar i conservar la integritat del nostre gestor, tenim certes excepcios que calen ser comprovades quan tractem amb documents.

- *ExceptionNotPrimaryKeys*

  Aquesta excepció salta quan s'ha fet un tractament amb un document i, les noves primary keys d'aquest document, no són correctes (no compleixen les propietats de claus primàries i tenen valor null). Bàsicament s'aplica quan es crea un document nou o quan es modifica un d'existent.
- *ExceptionDocumentNoExisteix* Aquesta excepció salta quan es vol fer un tractament amb un document que no existeix. És a dir, les claus primàries no estan intanciades en el conjunt de claus.
- *ExceptionDocumentNoExisteix* Aquesta excepció salta quan l'autor no existeix (no hi ha cap document amb atribut autor igual a l'autor passat com a paràmetre).
- *ExceptionDocumentJaExisteix* aquesta excepció salta quan es troben dos documents iguals en el Cjt_Documents.
- *ExceptionNoEnterValid* Aquesta excepció salta quan es vol llistar un número enter negatiu de documents.
- *ExceptionFormatNoValid* aquesta excepció salta quan el format d'un document no pertany a cap tipus dels que existeixen; és a dir, el document no té cap format del enumeration de formats existent.
- *ExceptionExpressioBooleanaIncorrecta* Aquesta excepció salta quan la expressió no compleix les propietats necessàries i per tant, és sintàcticament incorrecte.
- *ExceptionAutorNoExisteix*

  Salta quan l'usuari intenta fer una operació d'eliminar o cercar un element al Trie, que no forma part d'aquest.

MÈTODES I ALGORISMES EMPRATS:

- *CtrlDocument*

  aquesta funció ens permet crear una instància de l'objecte CtrlDocument amb els seus atributs buits. El seu cost és O(1).

- *EliminarDocument*

  Aquesta funció ens elimina un document del Cjt_Documents de manera irreversible. Es comprova que el document amb claus primàries els valors passats com a paràmetre existeix (sinó salta l'excepció ExceptionDocumentNoExisteix) i crida a les funcions de les classes agregades per actualitzar la informació dels autors. El cost de la funció és la suma de costos de la crida a getDocument (constant O(1)), l'eliminació del document a Cjt_Documents (constant O(1)), la crida a actualitzar_docs (O(n) on n és el nombre de paraules del conjunt) i la crida a eliminarTitolAutor (constant O(1)).

- *NouDocument*

  Crea un nou document amb el valor dels atributs el valor dels paràmetres passats, el *HashSet<String>stopwords*. Finalment afageix aquest Document en el Cjt_Documents comprovant prèviament que no existeix cap document amb aquestes claus i que el format és vàlid. Finalment, crida a les funcions de les classes agregades per actualitzar la informació dels autors.

  El cost d'aquesta funció és la suma de costos de la creadora de Document (O(n) on n és el nombre de paraules del contingut), la crida a add_document (O(n) on n és el nombre de paraules del contingut), la crida a afegirAutor_sempre (constant O(1)) i la crida a afegeixTitolAutor (constant O(1)).

  Per tant el cost és O(n).

- *ModificarInformacioDocument*

  Aquesta funció modifica el contingut del document amb claus primàries l'autor i el títol passat com a paràmetre. Comprova que el document amb claus primàries els atributs passats com a paràmetres d'entrada existeix. Obté el document especificat, l'elimina, crida la funció modificar per tal de canviar-ne els atributs i el torna a afegir en el *Cjt_Documents*.

  En la crida a la funció de la classe Document, es passa un HashSet<String> de les stopwords que s'han de tenir en compte. La tria d'aquest HashSet depèn de l'idioma passat com a paràmetre d'entrada.

  Finalment, crida a les funcions de les classes agregades per actualitzar la informació dels autors.

El cost és la suma de costos de la crida a *getDocument*, *EliminarDocument*, *IndexEB.esborratDocument*,*Document.ModificarInformacioDocument* *IndexEB.evaluarTotesExpressions*, *NouDocument*.

$cost = O(1) + O(n) + O(e^2) + O(n) + O(e) + O(n)$ on n és el nombre de paraules del contingut i e el nombre d'expressions del index.
Per tant, el cost és quadràtic.

- *ObrirDocumentAutorTitol*

  Aquesta funció ens permet obtenir el contingut del document que té com a claus primàries l'autor i el títol passats com a paràmetre d'entrada. Prèviament, comprova que el document existeix.

  El cost és la suma de costos de la crida a *getDocument* (constant O(1)) i la crida a *Informacio_Document* (O(n) on n és el nombre de paraules del contingut).
- *add_document*

  Ens afegeix el document passat com a paràmetre en el *Cjt_Document* i actualitza el *HashMap ParaulaDoc*. El cost és la suma de costos de la crida a *getDocument* (constant O(1)) i la crida a *actualitzar_docs* (O(n) on n és el nombre de paraules del contingut).
- *actualitzar_docs*

  Ens actualitza el valor dels HashMap dels nombres de documents per paraula. Es fa un recorregut de totes les paraules de ParaulaDoc i es posa l'atriu booleà a false (aquest ens indica que aquesta paraula encara no a aparegut en el document passat com a paràmetre).

  Per cada document, si l'acció és "Afegir", suma, només una vegada per paraula, el nombre de documents on apareix aquesta. En el cas de "Eliminar" es resta aquest nombre. És a dir, es fa un recorregut de les paraules del contingut del document passat com a paràmetre i si aquestes apareixen almenys una vegada, s'actualitza el valor del HashMap.

  El cost és O(n) on n és el nombre de paraules del contingut.
- *boolean existeix_document*

Permet saber si un document amb claus primàries els atributs passats com a paràmetre d'entrada, pertany al *CjtDocuments* o no. Es retorna true si existeix, false en cas contrari. El cost és constant O(1).

- *getDocument*

  Amb aquesta funció es pot obtenir el document que té com a clau primària l'autor i el títol passats com a paràmetre. Es llença excepció si no existeix, es retorna el document en cas contrari. El cost és constant O(1).

En aquestes tres funcions, s'inicialitzen els HashSet de les stopwords de cadascun dels tres idiomes.

HashSet<String> cat_stop_words

HashSet<String> eng_stop_words

HashSet<String> sp_stop_words

El cost és constant O(1).

- *LlistarDocumentsSemblants*

  En aquesta funció retorna un *ArrayList<Document>* dels k documents més semblants (pel que fa al contingut) al document que té com a claus primàries l'autor i el títol passats com a paràmetre. Es comprova que l'enter k, passat com a paràmetre, sigui vàlid i que el document existeix.

  Es fa un recorregut per tots els documents del Cjt_Documents i es calcula els cosinus de l'angle entre el vector que forma el contingut del document passat com a paràmetre d'entrada i el document de la iteració (di).

  Per fer aquests càlculs s'utilitzen el tf (els pesos o les freqüències unitàries de les paraules del contingut del document d en el document di), l'idf (logaritme en base 10 del quocient entre els documents totals i els documents on apareix la paraula) així com les normes dels dos documents. Amb aquests operadors podem calcular la semblança aplicant el model de espai vectorial i afegir el document di, amb clau el títol i l'autor i valor el resultat de la semblança, en un HashMap de documents.

Finalment apliquem l'ordenació SortedbyValue de la classe SortedValue per obtindre els documents ordenats decreixentment segons la semblança respecte el document d.

Si k és més petit que el nombre de documents de *Cjt_Documents* (k'), es llisten els k' documents més semblants.

El cost de la funció és el cost de la suma de la crida a *getDocument* (constant O(1)), la crida a *existeix_document* (constant O(1)), recorregut del *Cjt_Documents* O(d*n) on d és el nombre de documents i n el nombre de paraules del contingut, i O(k) on k és el nombre de documents a llistar.

El cost per tant serà quadràtic si d i n són semblants, lineal en cas contrari.

- *LlistarDocumentsAutor*

  En aquesta funció retorna un *ArrayList<Document>* de tots els documents que pertanyen a l'autor passat com a paràmetre d'entrada. Es comprova que aquest existeix i si no en té, es retorna una llista buida.

  El cost és la suma de la crida a *index.cercarAutors* (O(p+l) on *p* és la longitud del string prefix i *l* és el nombre d'autors que podem trobar a l'índex), la crida a indexDA.buscarTitolsAutor (constant O(1)) i recorregut O(t) on t és el nombre de títols de l'autor.

- *Llistar_query*

  Aquesta funció retorna un *ArrayList<Document>* dels k documents més rellevants (en quant a contingut) per aquesta query. Crea un document amb claus "query" i contingut el *ArrayList<String>* passat com a paràmetre, crida a la funció *LlistarDocumentsSemblants* i l'aplica sobre aquest document creat. Finalment, l'elimina i retorna el *ArrayList<Document>* dels k documents més semblants. El cost és el mateix que el de *LlistarDocumentsSemblants*.

- *LlistarAutorPrefix*

  Aquesta funció retorna un *ArrayList<Document>* tots els autors que tenen el prefix passat com a paràmetre d'entrada. El que fa és cridar la funció *cercarAutors(prefix)* de la classe *IndexAutor* i retornar el *ArrayList<String>* que retorna aquesta crida.

  El cost de la funció és de O(p+l) on *p* és la longitud del string prefix i *l* és el nombre d'autors que podem trobar a l'índex.

- *LlistarExpressioBooleana*

Aquesta funció retorna un *ArrayList<Document>* de tots els documents que satisfan la expressió booleana passada com a paràmetre d'entrada. El que es fa és afegir una expressió booleana, la passada com a paràmetre, al conjunt d'expressions booleanes. Seguidament, s'obtenen els documents que satisfan aquesta expressió i es retornen en un *ArrayList<Document>*.

El cost és la suma de la crida a *indexEB.afegirExpressio* (lineal O(e) on e es el nombre d'expressions), recorregut i crida a indexEB.evaluarTotesExpressions (O(d*e) on d és nombre documents i e nombre expressions),crida a i*ndexEB.getDocumentsExpressio* (lineal O(n) on *n* és el nombre de documents) i recorregut (O(d') on d' és el nombre de documents que satisan les expressions). Per tant el cost és quadràtic respecte d i e.

RELACIONS AMB ALTRES CLASSES:

La classe *CtrlDocument*, es relaciona amb *CtrlDomini*, *Document*, *IndexAutor*, *IndexDocumentsAutor*, *IndexExpressionsBooleanes*. Aquesta primer relació és deguda a que *CtrlDomini* engloba totes les classes controladors i per tant, també a *CtrlDocument*. Les relació *CtrlDocument-Document* apareix per a tal de poder tractar des de *CtrlDocument* amb els documents, mentre que les relacions *CtrlDocument-IndexAutor*, *CtrlDocument - IndexDocumentsAutor* i *CtrlDocument - IndexExpressionsBooleanes* apareixen per optimitzar les cerques.

CtrlConsulta

ESTRUCTURES DE DADES I ATRIBUTS:

ArrayList<Consulta> TotalConsultes

int MAXSIZE = 15;

Per aquesta classe es fa servir un conjunt de consultes, ordenades per antiguitat.
La mida maxima son 15 consultes.

EXCEPCIONS:

- ExceptionIndexConsultaNoValid: Al accedir a una consulta per índex, es llença aquesta excepció si l'índex és incorrecte. Pot ser llançada per els mètodes *getConsulta(int)* i *esborrarConsulta(int)*.

MÈTODES I ALGORISMES EMPRATS:

- *NovaConsulta*

  Aquesta funció afegeix una nova consulta al conjunt de consultes. Aquest conjunt ha de estar ordenat per antiguitat, per el que el primer que es fa es eliminar aquesta consulta del conjunt si ja hi era.

  Un cop fet això, es comprova que el conjunt no excedeix la capacitat màxima, si es el cas, s'elimina la consulta més antiga(index 0).

  Per últim segrega la consulta al conjunt.

- *DisplayHistorial*

  Retorna un ArrayList<Consulta> *amb totes les consultes del conjunt.*

- *EsborrarConsultes*

  Esborra totes les consultes del conjunt.

- *GetConsulta*

  Retorna la consulta segons índex que escull l'usuari. Si l'índex és incorrecte salta ExceptionIndexConsultaNoValid.

- *EsborrarConsulta*

  Esborra la consulta segons index que escull l'usuari. Si l'índex és incorrecta salta ExceptionIndexConsultaNoValid.

Totes les funcions d'aquesta classe tenen cost constant O(1).

RELACIONS AMB ALTRES CLASSES: *CtrlDomini*, *Consulta*

CtrlConsulta conté un conjunt de instàncies de la classe *Consulta*, ordenades per antiguitat. Aquest conjunt té una mida maxima de 15 consultes.

CtrlUsuari

ESTRUCTURES DE DADES I ATRIBUTS:

ArrrayList<Usuari> usuaris

int usuari actiu

Aquesta classe conte un conjunt d'usuaris i l'índex de l'usuari actiu si n'hi ha, sino val -1

EXCEPCIONS:

- ExceptionUsuariJaExisteix: No s'ha pogut afegir un usuari al conjunt perquè ja existeix dins d'aquest.
- ExceptionUsuariNoExisteix: No 'ha trobat l'usuari dins de el conjunt.
- ExceptionContrasenyaIncorrecta: Al intentar fer alguna operació que requeris de autentificació s'ha subministrat una contrasenya incorrecta.
- ExceptionSessioJaActiva: No es pot iniciar sessio perque ja hi ha un usuari en sessió.
- ExceptionNoSessioActiva: No es pot tancar la sessió perquè no hi ha cap activa.

MÈTODES I ALGORISMES EMPRATS:

- *NumeroUsuaris*

  Retorna el número de usuaris registrats a l'aplicació.
- *CrearUsuari*

  Crea un nou usuari amb el nom i la contrasenya proporcionats per l'usuari i l'inserta al conjunt d'usuaris.

  Si l'usuari ja existeix salta ExceptionUsuariJaExisteix.
- *EliminarUsuari*

  Elimina el usuari amb el nom proporcionat per l'usuari del conjunt de usuaris.

  És necessari que l'usuari introdueixi la contrasenya.

  Si l'usuari no existeix salta ExceptionUsuariNoExisteix.

  Si la contrasenya es incorrecta salta ExceptionContrasenyaIncorrecta.
- *IniciarSessio*

  Inicia sessió amb l'usuari amb el nom i contrasenya aportats per l'usuari.

  Si ja hi ha un usuari actiu salta ExceptionSessioJaActiva.

  Si l'usuari no existeix salta ExceptionUsuariNoExisteix.

  Si la contrasenya es incorrecta salta ExceptionContrasenyaIncorrecta.
- *TancarSessio*

Tanca la sessió activa. Si no hi ha cap sessió activa salta ExceptionNoSessioActiva.

- *GetUsuaris*

  Retorna el conjunt dels usuaris.

- *GetUsuariActiu*

  Retorna l'usuari que te sessió activa.

  Si no hi ha cap sessió activa salta ExceptionNoSessioActiva.

Totes les funcions explicades fins ara d'aquesta classe tenen cost constant O(1).

- *IndexUsuari*

  Retorna l'índex de el conjunt en el que es troba l'usuari.

  Aquesta funció té cost lineal O(u) on u és el número d'usuaris creats al gestor de documents.

RELACIONS AMB ALTRES CLASSES: *CtrlDomini*, *Usuari*

CtrlUsuari conté un conjunt de instàncies de la classe Usuari. Aquest conjunt conte tots els usuaris de l'aplicació.

Utils

KeyP

ESTRUCTURES DE DADES I ATRIBUTS:

Aquesta classe representa una clau com a clau doble. Té una alta semblança amb la classe PairP, però a diferència d'aquesta, s'utilitza només en Maps i permet comparar instàncies.

És formada per dos strings K1 i K2, que en aquest gestor de documents, seran l'autor i el títol d'un document.

EXCEPCIONS:

Aquesta classe no fa saltar cap excepció.

MÈTODES I ALGORISMES EMPRATS:

- *KeyP*

  Aquesta funció ens permet crear una instància de tipus KeyP amb el valor dels atributs K1 i K2 inicialitzats respecte dos paràmetres d'entrada. El cost és constant O(1).
- *getK i getK2*

  Aquestes funcions ens permeten obtenir el valor dels atributs K1 i K2 del KeyP especificat. El cost és constant O(1).
- *Equals*

  Aquesta funció ens permet comparar un KeyP amb un altre KeyP passat com a paràmetre d'entrada.
- *hashCode*

  Aquesta funció ens permet calcular el valor de hash d'una instància KeyP. Aquest valor és la suma de valors hash dels atributs K1 i K2. El cost és constant O(1).

RELACIONS AMB ALTRES CLASSES:

Aquesta classe té com a objectiu facilitar la programació i claredat del codi en Maps on la clau és una clau doble. És per això, que aquesta s'utilitza en classes on apareix aquesta estructura de dades amb aquesta propietat; per exemple a CtrlDocument.

PairP

ESTRUCTURES DE DADES I ATRIBUTS:

Aquesta classe representa una clau formada per dos objectes.

Els dos objectes que formen un PairP poden ser qualsevol tipus de variables. Per exemple, a la classe CtrlDocuments es fa servir PairP<Integer, Boolean>, PairP<Double, Integer> i PairP<String, String>.

EXCEPCIONS:

Aquesta classe no fa saltar cap excepció.

MÈTODES I ALGORISMES EMPRATS:

- *PairP(F, S): void*

  És la creadora d'un PairP amb els valors dels paràmetres que li passem a la funció.
- *getFirst(): F*

  Retorna el valor del primer paràmetre d'un PairP.
- *getSecond(): S*

  Retorna el valor del segon paràmetre d'un PairP.

RELACIONS AMB ALTRES CLASSES:

L'objectiu d'aquesta classe és poder guardar valors relacionats en forma de parella, independentment del tipus de variable que siguin.

Podrem veure com s'utilitza aquesta classe a *CtrlDocuments*.

SortedValue

ESTRUCTURES DE DADES I ATRIBUTS:

Aquesta classe no té cap atribut

EXCEPCIONS:

Aquesta classe no fa saltar cap excepció.

MÈTODES I ALGORISMES EMPRATS:

- *Map<PairP<String, String>, Double> sortByValue*

  Aquesta funció ens ordena en ordre creixent de valors, el *Map* passat com a paràmetre d'entrada.

  Per fer-ho, fa un stream de totes les entrades del *Map* passat com a paràmetre ordenant-les de manera decreixent segons el valor. És per això que apareix *comparingByValue().reversed()*.

  Finalment aplica *collectors*() per tal d'agafar el *PairP* i el double de cada instància, unificar-les i col·leccionar-les en un *LinkedMap*. FInalment, s'acaba recollint aquest *LinkedMap* i es retorna.

Node

ESTRUCTURES DE DADES I ATRIBUTS:

Aquesta estructura de dades representa node d'un arbre binari. És a dir, un arbre on totes les seves fulles tenen com a màxim arietat 2. Conté tres atributs: un atribut de tipus *T* amb les dades del node en qüestió i dos nodes *l* i *r*, que corresponen a cadascun dels fills del node en qüestió.

EXCEPCIONS: Aquesta classe no té excepcions.

MÈTODES I ALGORISMES EMPRATS: Els únics mètodes implementats són *getters* i *setters*.

RELACIONS AMB ALTRES CLASSES: La classe és subclasse de *Tree*.

Tree

ESTRUCTURES DE DADES I ATRIBUTS:

Aquesta estructura de dades representa un arbre binari. És a dir, un arbre on totes les seves fulles tenen com a màxim arietat 2. L'únic atribut que conté és *root*. *root* és una instància de la subclasse *Node*.

EXCEPCIONS: Aquesta classe no té excepcions.

MÈTODES I ALGORISMES EMPRATS: Els mètodes de la classe són les creadores i un *getter* de l'arrel de l'arbre.

RELACIONS AMB ALTRES CLASSES: *Tree* es relaciona amb la classe *ExpressioBooleana* que fa ús de l'estructura per construir els arbres amb expressions.

Trie

ESTRUCTURES DE DADES I ATRIBUTS:

L'estructura de dades del trie és un arbre on cada node conté un HashMap i un booleà. El HashMap *children* té com a clau un character, que és una lletra d'un string, i té com a valor un altre node TrieNode al qual apunta i on s'hi guarda el següent character de la paraula. El booleà *endOfWord* indica el final d'una paraula. Aquest booleà només és true en el node al qual apunta la última lletra de cada string que afegim al trie. En tots els altres nodes aquest valor es false.

Aquesta classe conté la classe TrieNode. Aquesta classe no fa saltar cap excepció i només té una funció implementada, la creadora d'un TrieNode format per un nou HashMap i el booleà *endOfWord* amb el valor inicialitzat a false.

EXCEPCIONS:

Aquesta classe no fa saltar cap excepció.

MÈTODES I ALGORISMES EMPRATS:

- *Trie(): void*

  És la creadora d'un Trie. La creadora d'un node inicial del trie té cost constant.

- *insert(String): void*

  Afegeix un string al trie.

  Inicialment guardem en una variable *current* el node arrel del trie. Tenim un bucle que itera per cada lletra del string que volem afegir. Comprova si el node al qual estem apuntant conté la lletra de la paraula que estem iterant. En cas negatiu, creem un nou node del tipus TrieNode i afegim el character al seu HashMap. A continuació, guardem a *current* el node al qual apunta el character en qüestió. Un cop acabada l'execució del bucle, posem el valor del booleà de l'últim node que ens hem guardat a true.

  El cost d'aquesta funció és lineal, O(m) on m és la mida de la paraula que volem inserir al trie. El bucle que itera totes les lletres de la paraula que volem afegir té aquest cost constant, i la resta d'accions que fem tenen cost constant.

- *removeautor(TrieNode, String, int): boolean*

  Elimina el String de mida int que té la primera lletra al TrieNode.

Aquesta funció és cridada quan volem eliminar un autor del trie. Li passem com a paràmetres el TrieNode a partir del qual apareix la paraula que volem eliminar, el string amb la paraula que volem treure del trie i l'identificador int que indica la lletra que estem eliminant actualment.

En primer lloc, comprovem si la lletra que estem intentant eliminar en aquesta crida és l'última lletra de la paraula. En cas afirmatiu, mirem quin valor té el booleà *endOfWord*. Si es false, no podrem eliminar el string ja que no serà un autor vàlid del gestor. Si es true, posem el valor a false i retornem un booleà que indica si el HashMap del node al qual apuntava el character eliminat és buit.

Si la lletra que estem intentant eliminar no és l'última de la paraula, comprovem si el node al qual apunta la lletra actual és buit. En cas afirmatiu, retornem false i no continuem intentant eliminar el string. En cas negatiu, comprovem si hem d'eliminar el node on es troba la lletra actual. Per fer-ho, comprovem què ens retorna la crida recursiva a la mateixa funció *removeautor(TrieNode, String, int+1)* intentant eliminar la següent lletra del string i quin és el valor del booleà *endOfWord* d'aquest node. Si el node que volem eliminar té el HashMap buit i el seu booleà *endOfWord* és false, l'eliminem.

Finalment la funció retorna true si hem pogut eliminar el string correctament del trie, i false en cas contrari.

Aquesta funció té cost lineal O(m) on m és el nombre de lletres que ens falten recorrer del string que volem eliminar, ja que totes les comprovacions que fem tenen cost constant, però la funció fa una crida recursiva per recórrer totes les lletres de la paraula. És a dir, m és la longitud de la paraula menys el valor de l'enter.

- *remove(String): boolean*

  Elimina el string del trie.

  Crida a la funció *removeautor(TrieNode, String, int)* i li passa com a paràmetres el root de l'arbre, el string de la paraula que volem eliminar i l'enter 0 per començar a eliminar la paraula desde la primera lletra.

  El cost d'aquesta funció és lineal O(m) on m és la longitud del string que volem eliminar.

- *search(String): boolean* i *search_prefix(String): boolean*

  Comproven si el string es troba al trie.

Hi ha una petita diferència entre aquestes dues funcions. La funció *search* comprova si el string és una paraula que es troba al trie, és a dir, que l'última lletra d'aquesta apunta a un node que el seu *endOfWord* és true. En canvi, la funció *search_prefix* comprova si el string es troba al trie independentment de si és una paraula o no. És a dir, és indiferent el valor del booleà *endOfWord* del node al qual apunta l'última lletra, la funció retorna true si existeix com a prefix d'una altra paraula.

Les dues funcions comparteixen gran part del codi. En els dos casos comencem assignant a una variable TrieNode *current* l'arrel de l'arbre. A continuació, recorrem tot el string que volem cercar, i per cada lletra comprovem si es troba al node *current*. En cas negatiu, la funció retorna false i deixa de cercar el string. En cas afirmatiu, continua la busqueda.

El codi és diferent a les dues funcions un cop hem sortit del bucle que recorre totes les lletres de la paraula. La funció *search_prefix* retorna sempre true, ja que haurem comprovat que el string es troba al trie independentment del valor del booleà. En canvi, la funció *search* retorna el mateix valor que tingui el booleà *endOfWord* de l'últim node.

Les dues funcions tenen cost lineal O(m) on m és la longitud del string que volem cercar al trie.

- *rec(String, TrieNode): ArrayList<String>*

  Retorna una llista de tots els autors existents al trie a partir del TrieNode.

  El string *actual* que passem com a paràmetre a la funció

  En primer lloc, creem el ArrayList<String> que retornarà la funció i comprovem si el node actual té el booleà *endOfWord* a true. En cas afirmatiu, afegim la paraula actual que hem passat com a paràmetre de la funció al ArrayList. A continuació, mirem per cada lletra de l'abecedari si es troba al HashMap del TrieNode. En cas afirmatiu, cridem de manera recursiva a la funció *rec(actual + c, next)*. El string actual+c és el mateix string amb el qual havíem cridat a la funció, però li hem afegit la lletra de l'abecedari que estavem comprovant. El TrieNode next és el node en el qual es troba la lletra c. Un cop hem acabat de comprovar totes les lletres de l'abecedari, retornem el ArrayList amb tots els autors trobats.

  El cost d'aquesta funció és lineal O(l) on l és el nombre d'autors que podem trobar desde el node indicat.

- *retornautors(String): ArrayList<String>*

  Retorna una llista de tots els autors existents al trie que contenen el string prefix que li passem com a paràmetre a la funció.

  Guardem a una variable TrieNode *current* un apuntador al node arrel del trie. Fem una cerca lineal lletra per lletra del string prefix amb el qual hem cridat a la funció. Per cada lletra comprovem que es trobi al node *current*. En cas afirmatiu, guardem a la variable *current* l'apuntador al node al qual apunta aquesta lletra. En cas negatiu, retornem una llista buida.

  Un cop hem trobat el node TrieNode al qual apunta l'última lletra del string prefix, cridem a la funció *rec(String, TrieNode)* i li passem com a paràmetres el string prefix i el node *current*. Finalment retornem el ArrayList que ens hagi retornat aquesta funció.

  El cost d'aquesta funció és O(p+l) on p és la longitud del string prefix i l és el nombre d'autors que podem trobar a partir del node indicat.

RELACIONS AMB ALTRES CLASSES:

La funció d'aquesta classe és implementar el correcte funcionament d'un trie. És a dir, que es puguin afegir, eliminar i cercar paraules i prefixos.

Està connectada a la classe ÍndexAutor, ja que l'estructura de dades d'aquesta és un trie.

6. Llibreries

Per realitzar els tests hem fet servir les llibreries JUnit-4.12 i Hamcrest-core-1.3.

JUnit és una biblioteca per realitzar proves unitàries en aplicacions amb llenguatge de programació Java.

Hamcrest serveix per escriure proves de programari en el llenguatge de programació Java.