**Descripció de les Estructures de Dades i Algorismes usats en la Implementació**

**Cela (i fills CelaNum, CelaText, CelaData, CelaRef)**

Per a tal d’implementar la classe Cela no ha fet falta usar estructures de dades ni algorismes gaire complicats. Primer de tot, s’ha implementat la classe Cela com a abstracta, i les seves 4 filles CelaNum, CelaText, CelaData i CelaRef són les que han hagut d’implementar certs mètodes. La classe pare Cela inclou com a atributs una String per a l’input de l’usuari i una enumeració TipusCela per indicar de manera més fàcil amb quin tipus estem tractant. Les filles, a part de tenir aquests atributs, en tenen algun més per guardar el verdader valor de l’input de l’usuari: un double per a CelaNum, una altra String per a CelaText, un LocalDate per a CelaData i una Cela per CelaRef.

Pel que fa als mètodes, no hi ha molt a comentar ja que la gran majoria d’ells són getters i setters. Potser sí que fa falta mencionar, però, el mètode abstracte copy() i el compare(Cela):

* **copy()**: el mètode permet fer una Deep Copy de la Cela en qüestió (S’ha tingut en compte quins tipus són immutables)
* **compare(Cela):** fent ús dels mètodes compare() i compareTo() ja implementats pels diferents tipus de Java, aquest mètode permet comparar (i per tant ordenar) diferents obejctes Cela entre ells.

**MatriuCeles**

Per a d’implementar-la, hem fet ús de 3 atributs: dos *int* per guardar el número de files i columnes i un estructura de dades bastant curiosa per a implementar la matriu d’objectes Cela en sí. Aquesta estructura de dades és una ConcurrentSkipListMap de ConcurrentSkipListMaps d’objectes Cela (ambdós indexades per Integer). Al principi s’havia pensat d’usar TreeMap en comptes de ConcurrentSkipListMap, ja que els membres de l’equip era la que coneixíem més, però les següents raons van fer que la balança es decantés cap a l’altra banda:

* Degut a la seva implementació més complicada, TreeMap no permet l’addició (i amb penes i treballs l’eliminació) d’elements a l’estructura mentre s’itera sobre els elements d’aquesta (es llança ConcurrentModificationException). En canvi, usant ConcurrentSkipListMap no es té aquest problema, cosa que fa que la implementació dels mètodes sigui més fàcil i no s’hagi d’estar creant nous objectes sovint.
* Com el seu nom indica, les operacions principals de l’estructura es poden executar de forma concurrent per diversos threads, fet que podria ser útil en futures fases de desenvolupament de l’aplicació.
* L’estructura ConcurrentSkipListMap ofereix la mateixa funcionalitat que TreeMap (es podria dir que son dues implementacions diferents amb un únic objectiu o propòsit). Per aquest motiu, els elements que emmagatzemen estan ordenats segons la seva respectiva clau. Les operacions principals de l’estructura són, també, logarítmiques en mitjana en terme de complexitat. La complexitat de les mateixes operacions és moderadament millor en TreeMap (son sempre logarítmiques) però, en el nostre cas, la complexitat d’aquestes en ConcurrentSkipListMap ja és suficientment.

Pel que fa als mètodes, sobre la constructora i els getters i setters no hi ha molt a dir: com a molt s’ha de comentar que algunes fan ús de les operacions principals de ConcurrentSkipListMap, però adaptades al fet que n’estem usant una de dues dimensions (cal mencionar que la matriu que estem implementant està indexada primer per columnes i després per files, ja que per algunes operacions que s’havien de fer amb aquesta era molt més còmode).

Hi ha uns quants mètodes, però, que sí que paga la pena mencionar ja que són un pèl més complexos, però tots usen la mateixa idea. Els mètodes en qüestió són eliminaFila, eliminaColumna, getBloc, getEntrades i getEntradesColumna, i la idea que usen és la següent: mitjançant operacions pròpies de ConcurrentSkipListMap que permeten obtenir un subgrup d’entrades (tailMap, subMap…), s’itera (amb iteradors o ForEach) sobre aquests subgrups per tal d’obtenir totes les entrades que s’estan buscant i sense haver de mirar entrades innecessàries.

**Full**

Es tracta d’una classe que hereda de MatriuCeles. Com ja s’ha descrit MatriuCeles i Full no se’n diferencia gaire, no hi ha molt a comentar respecte l’anterior. Els mètodes principals de Full (buidaBloc, copiaBloc i mouBloc) usen la mateixa idea que les de MatriuCeles: s’itera sobre un grup reduït d’entrades gràcies a l’ús d’operacions pròpies de ConcurrentSkipListMap.

Potser la principal diferència amb la classe pare és l’atribut celaResultat, que es tracta d’una Cela especial que hi haurà per cada Full i, per referenciar-la, s’han d’usar els índexs (-1, -1). Per a implementar aquesta idea es podia haver afegit una entrada a la posició (-1, -1) de la matriu, però es va decidir que usant la primera manera era més coherent i llegible.

**EntradaMatriuCeles**

Res a mencionar d’aquesta classe, ja que només està composta per getters i setters (està pensada per ser com un “Struct” de C o C++).

**Controlador Full**

Es tracta d’una classe que actua de intermediari entre el Full i les Operacions que s’han de fer sobre el mateix i les Cel·les que conté. Rep l'operació ja parsejada del Controlador Domini i l’executa. En cas que l’operació sigui sobre una MatriuCeles la delega a Operador i en cas que sigui directament sobre full crida els mètodes de full adients.

Té com a atributs el Full que controla i l’Operador a que delegar les Operacions sobre Matrius Cela. Per saber a qui ha de delegar l’operació, analitza el ResultatParserFull que li passa el Controlador Domini al critar el metode executaOperacio().

Al tractar-se d’un operador, no té getters ni setters i més enllà del mètode executaOperació nomes conté algun mètode privat que es crida des d’executaOperació segregat d’aquest per a facilitar la lectura del codi.

**Document**

La classe Document actua com un vector d’elements de la classe Full amb un nom i una data de modificació. Les seves operacions permeten afegir fulls, obtenir un full en un índex concret i eliminar un full en un índex concret. Quan l’índex del full no sigui vàlid (menor a 0 o major igual que el nombre de fulls) els mètodes llencen excepcions d’índex de full.

**Operador**

Classe amb estructura de singleton que executa les operacions sobre MatriuCeles que li delega un Controlador Full. Conte un mètode per cada una de les Operacions implementades, que rep com a paràmetre una MatriuCeles sobre la qual aplicar la Operació i retorna com a resultat un altra fruit d’aplicar l’operació a la MatriuCeles d’entrada. Algun dels mètodes associats a operacions a executar reben també paràmetres extra d’entrada per complementar la informació necessaria per implementa la seva funcionalitat (Ex.: string a cercar, xifra a la qual truncar un número…).

Els mètodes, exceptuant alguns que serien cerca, ordena i algunes de les operacions estadístiques d’executa operació estadística, iteren sobre totes les cel·les de MatriuCeles i, segons si el tipus és el correcte per aplicar l’operació, s'apliquen i deixen enregistrat en el paràmetre InputUsuari de la Cel·la que aquesta operació ha estat executada a mode d’històric. Si el tipus no és l’adequat deixen la cel·la sense modificar i es retorna en el mateix estat en que havia arribat.

S’han implementat també alguns mètodes privats per ajudar a la lectura del codi dels mètodes públics. El seu codi podria trobar-se en els mètodes públics pero per facilitar la lectura i enteniment s’han separat en privats.

S’utilitza com a llibreria per facilitar càlculs matemàtics de operacions aritmètiques i estadístiques el paquet java.lang.Math. Per a al cerca s’utilitza també la llibreria json-lib per poder expressar tota la informació dels resultats de la cerca en format JSON.

El format de retorn de l’operació cerca és un JSON amb un atribut “ocurrencies” que conté un enter amb el nombre d’ocurrències de l’string cercat i, per a cada cel·la textual del bloc on s’opera, un atribut “fila:columna” que alhora és un objecte JSON amb dos atributs, “ocurrencies” amb el nombre d’ocurrències en aquella cel·la en particular i “indexs”, un vector d’enters que conté les posicions de les ocurrències a la cadena de text de la cel·la.

L’operació ordena de bloc aplica el següent algorisme: primer obté un vector d’EntradaMatriuCeles format pels elements de la columna del bloc respecte la que s’ordena. Acte seguit ordena (en ordre ascendent o descendent segons el criteri triat) els elements d’aquest vector fixant-se exclusivament en la cel·la dins de cada EntradaMatriuCeles. L’ordre en el que s’ordenen les cel·les de tipus diferents és el marcat per l’enumeració Cela.TipusCela, que és el següent: numèrica < textual < datada. Els tipus de cel·la igual s’ordenen segons el seu propi criteri (el marcat pel seu valor). Un cop tenim les dades ordenades, com pot ser que no totes les cel·les de la columna tinguéssin valor, el que fem és recórrer totes les files i marcar les que tenien valor (i per tant hem ordenat) en l’ordre corresponent, les files sense valor s’afegeixen en ordre ascendent a les ordenades. Finalment es crea la MatriuCeles resultat amb les mateixes cel·les que la inicial i mantenint la posició de les columnes de cadascuna però permutant les cel·les segons el nou ordre descrit.