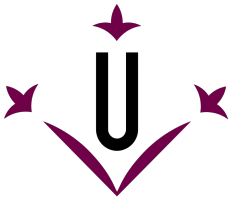

Universitat de Lleida

Pràctica de Patrons de Disseny

Full de càlcul

Fet per:
Oriol Alàs Cercós, Àiax Faura Vilalta, Joan Martí Olivart

Data de lliurament:
24 de maig de 2020

Universitat de Lleida
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Pràctica de Patrons de Disseny

Professorat:
Gimeno Illa, Juan Manuel

Índex

1 Introducció

En el present document es detallarà la implementació i les principals dificultats trobades d'aquesta. A més a més, es parlarà dels diferents casos de prova realitzats i com han estat implementats. Es va agafar com a disseny inicial el proposat en l'enunciat de la pràctica. L'actual disseny del projecte és el mostrat en la *Figura 1*.

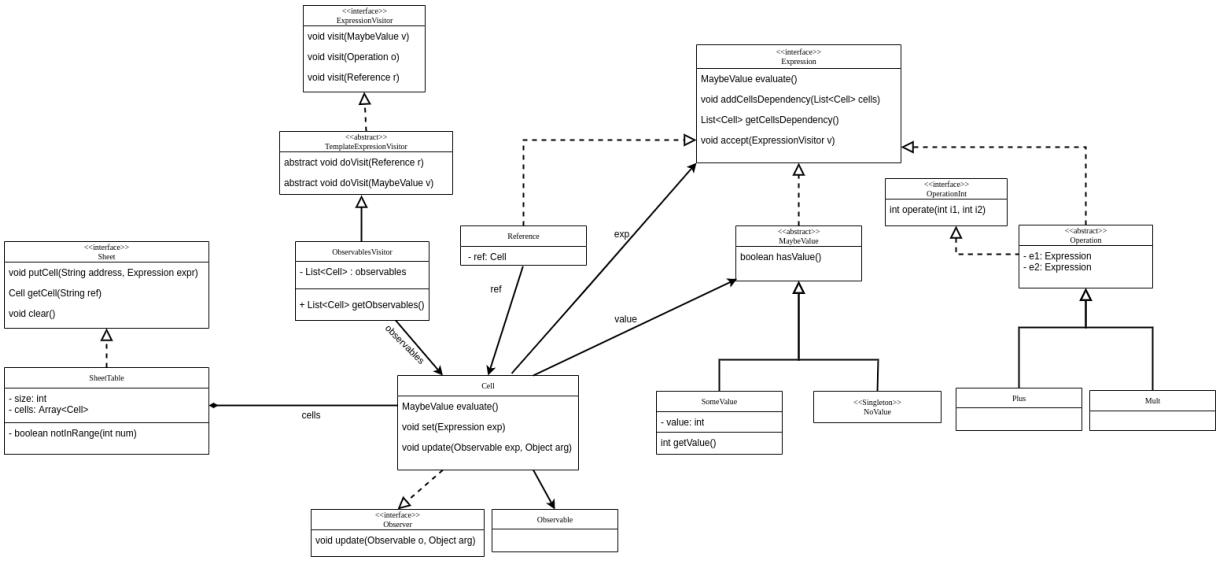

Figura 1: Disseny UML del projecte.

Per tal de realitzar el treball, s'ha utilitzat el control de versions *git*, a més a més que es va dividir les tasques del projecte en:

- Implementació del *package* expression.
- Realització de tests de expression.
- Implementació senzilla[1] del *package* cell.
- Implementació del *packages* sheet i spreadsheet.
- Realització de tests dels respectius paquets.
- Implementació del recàlcul de cel·les.
- Realització dels tests de l'última tasca.

El document seguirà cronològicament la implementació fins arribar a la final. No obstant això, encara que el recàlcul de cel·les hagi estat l'última tasca a implementar, s'ha tingut en compte el disseny en tot moment per tal què l'extensió del projecte es pugui realitzar.

[1]Sense tenir en compte el requeriment del recàlcul de cel·les ni utilitzant el patró observador

2 Implementació d'expression

Per tal d'implementar les expressions, s'ha utilitzat el patró Composite, on les fulles de l'arbre són les referències i els possibles valors i els nodes són les operacions.

Com ja es va esmentar a l'enunciat, per simplificar el treball, s'ha decidit que les expressions són immutables, és a dir, no es pot modificar el comportament un cop instanciat l'objecte. Per aquest motiu, tots els membres de les classes que implementen la *interface Expression* són finals, doncs els valors no poden canviar.

Aquesta decisió ajuda a que el valor d'una cel·la només pugui canviar si aquesta canvia d'expressió, per tant, la responsabilitat del recàlcul d'una cel·la només estarà en la classe Cel·la i no en classes que implementin Expression.

Es va realitzar la proposta d'utilizar la classe `Optional<E>` però es va decidir utilitzar el patró ObjectNull per estar obert a noves funcionalitats, seguint el principi d'Obert-Tancat.

Es va crear la interfície OperationInt al *package* operation, ja que, aquesta, a més d'ajudar-nos després per la realització de tests[2], també segrega interfícies. A més a més, al crear Operation es va pensar en el patró plantilla, creant-la com una classe abstracta, per tal de tenir el comportament compartit entre les seves diferents subclasses.

3 Implementació de Cell

Una cel·la ha d'encapsular el valor d'ella i com ha arribat a tenir aquest valor, és a dir, el seu càlcul, que ve predeterminat per Expression. Dintre del constructor, veiem que la cel·la ja calcula el seu valor per primer cop. Un cop evaluat, si es crida al mètode `evaluate()`, aquest sempre retornarà la mateixa instància.

El valor d'una cel·la només és recalcularà si canvia el càlcul de la cel·la o, si alguna de les cel·les referenciades que la cel·la depèn, canvia. Per aquest motiu, s'ha implementat el mètode `set(Expression exp)`, que haurà de canviar la referència d'Expression i, també, guardar el nou càlcul.

4 Implementació de Sheet i SpreadSheet

En lloc de crear una classe anomenada Sheet, es va crear una interfície que té els mètodes d'afegir un càlcul a una cel·la, agafar la cel·la o buidar el full

[2] Al ser una interfície funcional, podem crear l'operació binària esperada de Operation per tal de saber el comportament desitjat en els tests.

de càlcul. La seva implementació, anomenada SheetTable, realitza aquests mètodes utilitzant un vector bidimensional.

Per tal de passar d'adreça (String) a la localització de dos nombres, s'ha creat una classe annidada anomenada Address. Aquesta, llança una excepció creada anomenada *NotValidAddressException* si no és una adreça vàlida.

Una altra decisió important de disseny a esmentar, és que si es vol agafar una cel·la la qual no hi havia cap valor, aquesta es crearà amb la expressió `NoValue.INSTANCE` i es retornarà.

És important esmentar que, un cop afegida una cel·la en una adreça, no hi podrà haver cap més cel·la en aquella adreça, sinó que si es volgués realitzar una altre càlcul (el que s'entèn per sobre-escriure), només es cridarà al mètode `set(Expression expr)`. Aqueta decisió ens ajudarà a l'hora de no haver de re-mirar si les referències a les cel·les continuen sent vàlides.

La implementació de SpreadSheet s'ha creat tal i com es proposava a l'enunciat de la pràctica i ha estat una gran ajuda per tal de realitzar els tests d'una forma més còmode.

5 Recàlcul de les cel·les

Un cop implementada i fets els tests de la primera part de la lògica del full de càlcul, només faltava implementar el recàlcul de les cel·les en cas que el valor de les referències del càlcul canvïi.

Sabent que les expressions són immutables, el valor d'una cel·la només pot canviar si aquesta canvia el seu càlcul. Per tant, una cel·la haurà de notificar a les altres en cas que ho faci, és a dir, la classe *Cell* serà observable d'altres instàncies[3] de la mateix classe. Per agilitzar la implementació s'ha aprofitat la classe *Observable* i *Observer* de `java.util`. En concret, el tipus de patró utilitzat és el push.

Per tal de saber quines cel·les són observadores de l'actual, només cal mirar la expressió i mirar quantes instàncies de References hi ha. En primer lloc, es va declarar un mètode a la classe Expression per tal que retornès un llistat[4] de les seves referències. D'una manera semblant, s'iteraria per tot l'arbre que forma l'expressió fins afegir les referències.

Encara que el mètode funcionava per tal de trobar les diferents dependències,

[3]Es suposa que no hi ha cicle de dependències.

[4]Es va decidir utilitzar una llista en lloc d'un conjunt per la seva facilitat d'agafar una cel·la.

era poc elegant. Per tant, es va decidir utilitzar el patró *Visitor* per tal de visitar la classe Expression i rebre les cel·les a les quals s'hi feia referència. El tipus del visitor és extern, ja que es voldrà utilitzar des de la classe Cell. Per tal de mantenir el principi d'obert-tancat, s'ha utilitzat el mètode plantilla per si es volgués afegir classes visitants. S'ha utilitzat el Double Dispatch a l'hora que una expressió accepti un visitant. Gràcies a aquest patró, només cal crear el visitor a dintre de la classe Cell quan es necessiti, i aquest ens retornarà el llistat de cel·les que observen la cel·la en qüestió.

6 Tests

Per tal de realitzar les proves més àgils, com que les diferents operacions[5] a implementar tenen el mateix comportament, s'han creat classes abstractes per definir-lo, i s'ha deixat la feina a les seves subclasses de definir quin tipus d'operació s'està realitzant.

A més a més, per tal de provar si funcionava bé el visitor, s'ha creat una façana molt senzilla per no haver de repetir codi.

[5]Les operacions a realitzar són la suma i la multiplicació, implementades a les classes Plus i Mult respectivament.