El full de càlcul (v1)

La pràctica consisteix en implementar la infraestructura d'un full de càlcul. En concret, es voldrà:

- Disposar d'un full format per cel·les adreçables *estil excel*: "a1" primera fila i primera columna; "b3" segona fila i tercera columna.
  - De cara a fer proves no cal que la mida sigui massa gran (amb 5 files i columnes n'hi ha suficient).
- Les cel·les poden estar buides o contenir una expressió (fórmula de càlcul).
  - Una fórmula de càlcul pot contenir valors constants, referències a cel·les del full o operacions binàries aplicades sobre subfórmules.
  - Per tant, tractarem els valors constants com a fórmules de càlcul i només admetrem un tipus de números (int o double, com vulgueu).
    - Una expressió que representa una constant té com a valor a sí mateix.
  - Com operacions binàries només cal que considereu sumes y productes.
- Les fórmules serveixen per calcular els valors que tenen les cel·les:
  - Si la fórmula de càlcul, una vegada avaluada, genera un valor, una cel·la té aquest valor.
  - Si no, té el mateix valor que una cel·la buida.
  - Per exemple, una cel·la que depèn del valor d'una altra que encara no té valor, no té valor.
- **El valor d'una cel·la només es recalcula quan alguna de les cel·les de les que depèn ha canviat**.
- **Repeteixo**: **el valor d'una cel·la NO es recalcula cada vegada que hi accedim.**
- **Aquest és el punt central de la pràctica i no respectar-lo vol dir no fer una solució vàlida de la mateixa.**

Simplificacions:

- Totes les classes que representen expressions són **immutables**.
- En canvi les cel·les poden canviar de valor:
  - perquè se'ls ha assignat una nova expressió
  - perquè alguna de les cel·les de les quals depèn la seva fórmula de càlcul ha canviat.

Possibles pistes

Aplicació del NullObject

De cara a no tenir que estar pendents tota l'estona de si una cel·la té un valor o està buida (perquè o bé no té res; o bé la fórmula que conté depèn de cel·les que estan buides), és convenient usar el patró null object (comentat al problema de les interseccions de figures geomètriques).
Per això, el valor que té una cel·la serà d'una classe que anomenarem `MaybeValue`, que té dues subclasses:

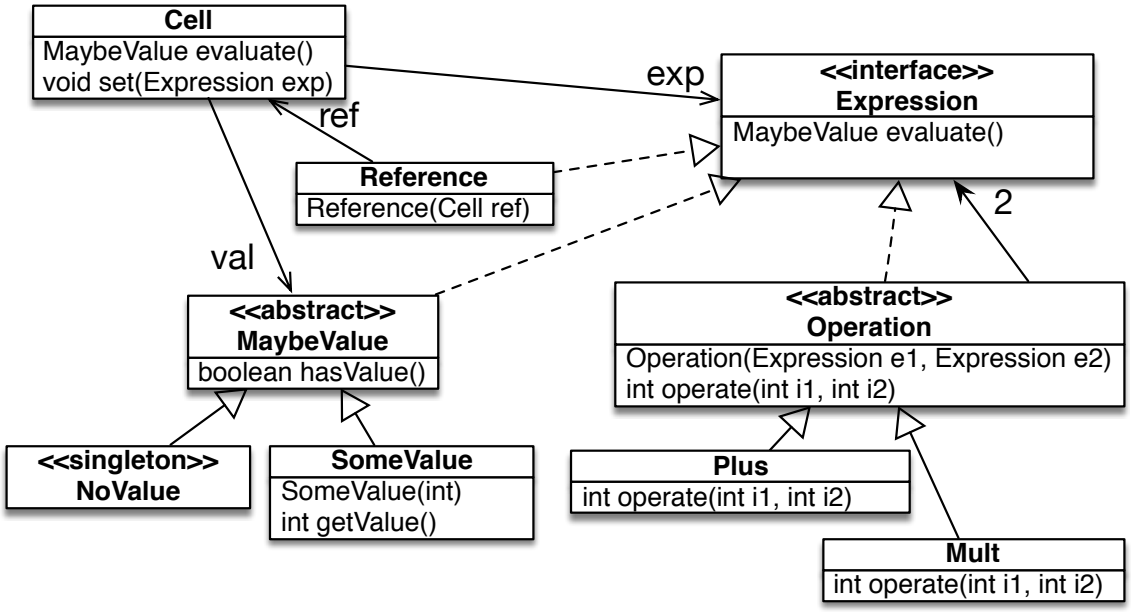
- `NoValue`, que és el null object que representa l'absència de valor. Només en cal un per a tota l'aplicació.
- `SomeValue`, que encapsula un `int` (o `double`, depenent del que hagueu triat).

Ambdues classes també són immutables.
A més, si considereu els `MaybeValues` com expressions, podreu simplificar una mica el problema.

Tant si ho considereu com una expressió més, com si no, **les instàncies d'aquestes dues classes són també immutables**[1].

Disseny preliminar

Amb les pistes donades fins ara, podem fer un petit disseny preliminar, que encara no té en compte el recàlcul de valors quan hi ha modificacions.
En aquest disseny tampoc apareix la classe `Sheet` (faig servir el nom que uso posteriorment), que representa el full de càlcul.
El seu constructor rep un enter, que és la dimensió del full de càlcul (suposarem que és quadrat). Entre d'altres coses, s'encarrega de veure si una referència a una cel·la és vàlida o no (per exemple, si la mida del full de càlcul es de 5x5, les cel·les vàlides serien des de "a1" fins a "e5").
Per simplificar el vostre codi, només cal que funcioni amb dimensions en les que una cel·la pot referenciar-se amb una única lletra i un número (dic número, no dígit) i, en cas de referenciar una cel·la no vàlida, podeu simplement llençar una excepció (que no cal que sigui comprovada).



Per tal d'incloure tota la part de notificacions, és convenient afegir a les expressions el mètode

```
Set<Cell> references();
```

que retorni el conjunt de cel·les referenciades, directa o indirectament, per una expressió (que podria ser perfectament buit).

Façana SpreadSheet

De cara a poder usar totes les classes de les que constarà la sol·lució de forma còmode, es pot aplicar un patró anomenat Façade, per tal de definir una classe que amagui la complexitat en l'ús i la combinació de les mateixes.
Com aquest patró no l'hem vist, us comento l'esquelet que tindria aquesta classe façana, a més de mostrar-vos uns quants tests que exemplifiquen com s'utilitza.

[1] En Java 8, com veurem al darrer tema de l'assignatura, hi ha la clase genérica Optional<E> que permetria modelar això d'una altra manera.

Això fa que, a cadascuna de les classes internes, no us hagueu de preocupar d'afegir mètodes que simplifiquin el seu ús des de fora.
La única finalitat de la façana és simplificar, en aquest cas, la creació i manipulació d'instàncies de les classes implicades; la feina d'avaluació i propagacions, etc., etc., no és tasca de la façana. Tot hauria de funcionar exactament igual si no usem la façana i treballem amb la resta de classes del nostre disseny.

Aplicació del patró façade

```
1  public class SpreadSheet {
2
3      private static int SIZE = 5;
4      private static final Sheet SHEET = new Sheet(SIZE);
5
6      public static Expression plus(Expression expr1, Expression expr2) {
7        // Crea i retorna una expressió corresponent a la
8        // suma de les dues subexpressions
9        ¿?
10     }
11
12     public static Expression plus(Expression expr1, int value2) {
13        // Crea i retorna una expressió corresponent a la
14        // suma de expr1 i de l'expressió que representa
15        // la constant value2
16        ¿?
17     }
18
19     public static Expression plus(Expression expr1, String ref2) {
20        // Crea i retorna una expressió corresponent a la
21        // suma de expr1 i de l'expressió que representa
22        // una referència a la cel·la amb nom ref2
23        ¿?
24     }
25
26     public static Expression plus(int value1, Expression expr2) {
27        ¿?
28     }
29
30     public static Expression plus(int value1, int value2) {
31        ¿?
32     }
33
34     public static Expression plus(int value1, String ref2) {
35        ¿?
36     }
37
38     public static Expression plus(String ref1, Expression expr2) {
39        ¿?
40     }
41
42     public static Expression plus(String ref1, int value2) {
43        ¿?
44     }
45
46     public static Expression plus(String ref1, String ref2) {
47        ¿?
48     }
49
50     // El mateix per a totes les combinacions de mult
```

```
51
52      ...
53
54      public static MaybeValue get(String name) {
55        // Retorna el valor que potser hi ha a la cel·la
56        // amb nom name.
57        // Si hi ha un valor, es retorna una instància de
58        // SomeValue; si no hi ha, NoValue.
59        ¿?
60      }
61
62      public static void put(String name, Expression expr) {
63        // Assigna a la cel·la amb nom name l'expressió
64        // expr.
65        // Això provocarà l'avaluació de la cel·la (la
66        // qual cosa pot propagar la avaluació a d'altres
67        // cel·les)
68
69        ¿?
70      }
71
72      public static void put(String name, int value) {
73        // Assigna a la cel·la amb nom name l'expressió
74        // el valor value (Òbviament caldrà construir la
75        // representació d'aquest int com Expression).
76        // Això pot provocar avaluacions d'aquesta o
77        // d'altres cel·les
78
79        ¿?
80      }
81
82      public static void put(String name, String refName) {
83        // Assigna a la cel·la amb nom name la referència
84        // a la cel·la amb nom refName (Òbviament caldrà
85        // construir la representació d'aquesta
86        // referència com Expression).
87        // Això pot provocar avaluacions d'aquesta o
88        // d'altres cel·les
89
90        ¿?
91      }
92
93      public static void clear() {
94        // Esborra totes les cel·les del full de càlcul.
95        ¿?
96      }
97  }
98
```

Tests

Aquesta façana permetrà definir tests com els següents, en els que la manipulació del full de càlcul i les complexitats de crear expressions i accedir a cel·les queden amagades darrera de la façade.

```
1  import static spreadsheet.SpreadSheet.*;
2  // més imports ...
3
4  public class SpreadSheetTest {
5
```

```
 6     @BeforeEach
 7     public void setUpSheet() {
 8         put("a3", mult("a1", "a2"));
 9     }
10
11     @Test
12     public void
13     cell_has_no_value_if_ depends_on_empty_cells() {
14         assertFalse(get("a3").hasValue());
15     }
16
17     @Test
18     public void recalculation_works() {
19         put("a1", 42);
20         put("a2", 20);
21         assertEquals(new SomeValue(840), get("a3"));
22     }
23
24     @AfterEach
25     public void clearSheet() {
26         clear();
27     }
28 }
```

Comentaris:

- Línia 1: importem de forma estàtica els mètodes de la façade. D'aquesta manera els podrem usar directament.
- Línia 8: el full de càlcul només té una fórmula a la cel·la "a3" que és el producte de les cel·les "a1" i "a2".
- Línia 14: com la fórmula depèn de cel·les que estàn buides, la cel·la "a3" no té valor.
- Línies 19-21": Si assignem valors a "a1" i "a2", la cel·la "a3" té valor i és el producte dels valors de les dues cel·les.
- Línia 26: Hem d'esborrar el full de càlcul després d'executar cada test, ja que aquests han de ser independents.

O també:

```
29 public class ComplexSpreadSheetTest {
30
31     @BeforeEach
32     public void setUp() {
33         put("c1", mult("a1", "b1"));
34         put("c2", mult("a2", "b2"));
35         put("c3", plus("c1", "c2"));
36
37         put("a1", 10); put("b1", 20);
38         put("a2", 30); put("b2", 40);
39     }
40
41     @Test
42     public void chained_expressions() {
43         assertEquals(new SomeValue(1400), get("c3"));
44     }
45
46     @Test
47     public void chained_propagations() {
48         put("a1", "b1");
49         assertEquals(new SomeValue(1600), get("c3"));
```

```
50      }
51
52      @AfterEach
53      public void tearDown() {
54          clear();
55      }
56  }
```

Aspectes de metodologia

- És important que feu tests de cara a anar provant les diferents classes que feu.
  - No cal fer servir objectes o classes substitutes als tests
  - Proveu lo suficient per a tirar endavant amb seguretat i per a comprovar que al reorganitzar el codi les coses continuen funcionant.
  - No és necessari fer tests de la façana, però si algun test que usa la façana falla, repasseu que no ho faci perquè la façana està malament implementada
- Feu servir control de versions i aneu fent commits cada vegada que assoliu una fita, fer proves en branques diferents, etc, etc.
- Nivell principiant de https://github.com/carlesm/doing-it-right

Què cal entregar?

Entregueu un fitzer **ZIP** que contingui:

- Projecte IntelliJ amb tot el codi del projecte.
  - no és obligatori que classes i mètodes públics tinguin javadoc
- Un **informe en PDF** que:
  - descrigui el disseny de la vostra solució (podeu fer servir diagrames de classes per mostrar diferents parts del disseny)
  - podeu fer els diagrames a mà i després escanejar-los per afegir-los al document (o, per exemple, utilitzar https://draw.io/)
  - els principals problemes que us heu trobat i com els heu solucionat