RAPPORT ANALYSE ET CONCEPTION

ANDJEMBE Maksoudath, TANIEL Rémi

Le but du projet est de réaliser un tableur “basique” mais facilement extensible, l'application sera divisée en 2 parties : * le noyau * la partie graphique

Le noyau s'occupera de toutes les opérations de notre grille, les cases ne pourront contenir que des réels ou des formules(opération binaire ou des fonctions acceptant des plages de cases).

SCHEMA UML

Voici le schéma UML de notre application, les classes et méthodes abstraites sont en italique :

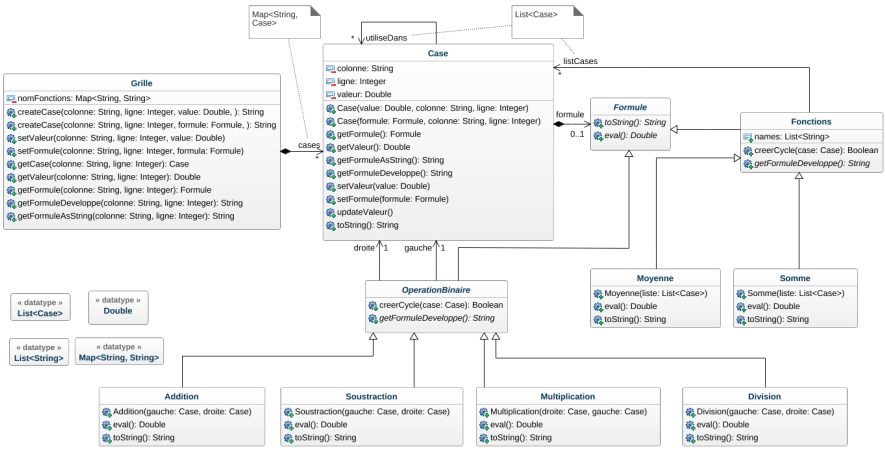

Figure 1: UML

PSEUDO-JAVA CREATION GRILLE,CASES

Voici un exemple de création d'une grille et de l'ajout / modification / affichage de plusieurs types de case :

```
class Application {

    public static void main(String[] args) {
        Grille g = new Grille();

        g.createCase("b",1); //Ajout case vide
```

```
        g.createCase("a",1,100.0); //Ajout case avec valeur
        g.createCase("a",2,50.0); //Ajout case avec valeur
        g.createCase("a",3,new Addition(g.getCase("a",2),g.getCase("a",1))); //Ajout case av

        List<Case> plageCase1 = new ArrayList<Case>(); // Crée une liste de case
        plageCase1.add(g.getCase("a",1));
        plageCase1.add(g.getCase("a",2));
        plageCase1.add(g.getCase("a",3));

        g.createCase("a",4,new Somme(plageCase1)); //Ajout case avec fonctions

        g.setValeur("b",1,100); //Met la valeur de b1 à 100

        List<Case> plageCase2 = new ArrayList<Case>(); // Crée une liste de case
        plageCase1.add(g.getCase("a",4));
        plageCase1.add(g.getCase("a",2));
        plageCase1.add(g.getCase("a",3));

        g.setFormule("b",2,new Moyenne(plageCase2)); //Met la formule dans b2

        g.getValeur("a",1); //Affichera 100.0
        g.getValeur("a",4); //Affichera (100+50+150)=100
        g.getFormuleAsString("b",2); //Affichera MOYENNE(a4,a2,a3)
        g.getFormuleDeveloppe("b",2); //Affichera MOYENNE(Somme(a1,a2,a3),a2,(a1+a2))
    }
}
```

CHOIX STRUCTURE DE DONNÉES

Nous devons choisir une structure de donnée pour stocker les cases dans notre grille, nous savons déjà que nous allons utiliser ne collection pour les stocker,voici celles que nous connaissons: - des tableaux - des listes - des maps - des sets

D'après le schéma UML ci-dessus, nous allons donc utiliser une `HashMap` pour stocker les cases de notre grille : * Pour rechercher une case et, effectuer des opérations dessus ce sera plus facile, la clé de la Map sera une chaine de caractère (String) qui représente la coordonnée de cette case (c'est-à-dire la concaténation du nom de ligne et de l'indice de la colonne, exemple "A1", "B9", etc...)

Une case peut etre utilisée dans plusieurs autres cases, on ne sait pas le nombre d'autres cases où elle sera utilisée, on stockera donc cette donée dans une `ArrayList` de `Case`.

Certaines fonctions (`Moyenne`, `Somme`) utilise également une plage de case, pour stocker ces cases, nous allons également une `ArrayList` de `Case`.

METHODES ESSENTIELLES EN PSEUDO-JAVA

1. Methode getValeur

```
class Grille {
  Map<String, Case> cases  = new HashMap<>();

  double getValeur(String colonne, int ligne){
    String code=colonne+ligne;
    return cases.get(code).getValeur();
  }
}


class Case {
  String colonne;
  int ligne;
  double valeur;

  double getValeur() {
    return valeur;
  }
}
```

2. Methode getFormuleAsString

```
class Grille {
  Map<String, Case> cases = new HashMap<>();

  String getFormule(String colonne, int ligne) {
    String code=colonne+ligne;
    return cases.get(code).getFormuleAsString();
  }
}


class Case {
  String colonne;
  int ligne;
  double valeur;
  Formule formule;

  String getFormuleAsString() {
      if (formule != null)
```

```
            return formule.toString();
        else
            return toString();
    }
}

// Exemple pour Addition
class Addition {
    Case gauche;
    Case droite;

    String toString() {
        return gauche.toString() + "+" + droite.toString();
    }
}
```

3. Methode getFormuleDeveloppe

```
class Grille{
  Map<String, Case> cases  = new HashMap<>();

  String getFormuleDeveloppe(String colonne, int ligne) {
    String code=colonne+ligne;
    return cases.get(code).getFormuleDeveloppe();
  }
}

class Case{
  String colonne;
  int ligne;
  double valeur;
  Formule formule;
  ArrayList<Case> utiliseDans = new ArrayList<Case>();

  String getFormuleDeveloppe() {
      if (formule != null)
          return formule.getFormuleDeveoppe();
      else
          return toString();
  }
}

// Exemple pour Moyenne
class Moyenne {
    List<Case> listCases = new ArrayList<Case>();
```

```
    String getFormuleDeveloppe() {
        return "Moyenne(" + listCases.stream().map(c -> c.getFormuleDeveloppe()).collect((Co
    }
}
```

4. Methode eval()

- Dans Addition :

```
class Addition {
  Case gauche;
  Case droite;

  double eval() {
      return gauche.getValeur() + droite.getValeur();
  }
}
```

- Dans Multiplication :

```
class Multiplication {
  Case gauche;
  Case droite;

  double eval() {
      return gauche.getValeur() * droite.getValeur();
  }
}
```

- Dans Soustraction :

```
class Soustraction {
  Case gauche;
  Case droite;

  double eval() {
      return gauche.getValeur() - droite.getValeur();
  }
}
```

- Dans Division :

```
class Division {
  Case gauche;
  Case droite;

  double eval() {
    if (droite.getValeur() != 0)
```

```
        return gauche.getValeur() / droite.getValeur();
    else
        lève une exception
  }
}
```

- Dans Moyenne :

```
class Moyenne {
  List<Case> listCases = new ArrayList<Case>();

  double eval() {
    double val=0;

    if (listCases.size() != 0)
      for(int i=0; i<listCases.size(); i++)
            val += listCases.get(i).getValeur();
      return val / listCases.size();
    else
      lève une exception
  }
}
```

- Dans Somme :

```
class Somme {
  List<Case> listCases = new ArrayList<Case>();

  double eval() {
    double val=0;
    if (listCases.size() != 0)
      for(int i=0; i<listCases.size(); i++)
            val += listCases.get(i).getValeur();
      return val;
    else
        lève une exception
  }
}
```

5. Methode setValeur

```
class Grille {
  Map<String, Case> cases  = new HashMap<>();

  void setValeur(String colonne, int ligne, double value) {
    String code = colonne + ligne;
    return cases.get(code).setValeur(value);
```

```
  }
}

class Case {
  String colonne;
  int ligne;
  double valeur;
  Formule formule;
  List<Case> utiliseDans = new ArrayList<Case>();

  void setValeur(double value) {
    valeur = value;
    for(int i=0; i<utiliseDans.size(); i++)
       utiliseDans.get(i).updateValeur();
  }
}
```

5. Methode setFormule

```
class Grille {
  Map<String, Case> cases  = new HashMap<>();

  void setFormule(String colonne, int ligne, Formule formule) {
    String code = colonne + ligne;
    return cases.get(code).setFormule(formula);
  }
}


class Case {
  String colonne;
  int ligne;
  double valeur;
  Formule formule;
  List<Case> utiliseDans = new ArrayList<Case>();

  void updateValeur() {
      valeur = formule.eval();
  }

  void setFormule(Formule formula) {
    if (!formula.creerCycle(this))
      formule = formula;
      updateValeur();
      for(int i=0; i<utiliseDans.size(); i++)
```

```
        utiliseDans.get(i).updateValeur();
    else
      lève une exception
  }
}

// Exemple pour OperationBinaire
class OperationBinaire {
    Case gauche;
    Case droite;

    boolean creerCycle(Case case) {
        if (gauche != case && droite != case) {
            if (gauche.isFormula() && droite.isFormula())
                return gauche.getFormule().creerCycle(case) && droite.getFormule().creerCycl
            else
                return false;
        }

        return true;
    }
}
```

LISTE DE TESTS

Afin de s'assurer de la maintenabilité de notre code et de la qualité de celui-ci, nous allons réaliser plusieurs tests sur les différentes méthodes que nous allons programmé dans notre application, voici quelques exemples :

- Création d'une case avec une valeur fixe
- Création d'une case avec une formule d' `Opération binaire` et une fonction comme `Moyenne`
- Modification d'une case avec une valeur sans qu'elle soit utilisée dans une autre case
- Modification d'une case avec une valeur utilisée dans une autre case
- Vérification qu'une erreur se lève lors de la création des 2 types de cycles (direct et indirect)
- Renvoie de la formule dévéloppée d'une case avec une formule assez compliqué