Interpréter des expressions mathématiques

François Biolet
Guillaume Delhumeau
Loïc Geffroy

Objectifs

- Interpréter des expressions mathématiques
  - A l'aide du design pattern « Interpréteur »
  - Implémentation en Java

Principe

- Analyse lexicale :
  - Il s'agit de reconnaître les différents « mots » lexicaux tels que les nombres, les variables, les opérateurs, ...
- Analyse grammaticale :
  - Tout comme les mots d'une langue s'enchaînent d'une façon logique pour donner du sens, les mots d'une expression mathématique s'enchaînent selon une logique que l'on nomme grammaire.

Principe (suite)

- Mise sous forme d'objets
  - L'expression est convertie en une arborescence d'objets.
- Interprétation
  - Il s'agit d'utiliser la structure que forme l'ensemble des objets pour exécuter un calcul.

Analyse Lexicale

- La chaîne de caractères fournie en entrée est découpée.
  - Les opérateurs délimitent les sous-parties de la chaîne.
  - Exemple : 8 + 55 * 21 devient :
    - [1] 8
    - [2] +
    - [3] 55
    - [4] *
    - [5] 21

Analyse lexicale

- Les différentes parties sont ensuite analysées pour déterminer leur « type »
  - Exemple :
    - « + » est un opérateur
    - « 42 » est un nombre
    - « maVar » est le nom d'une variable
    - « (» est une parenthèse ouvrante
    - etc...

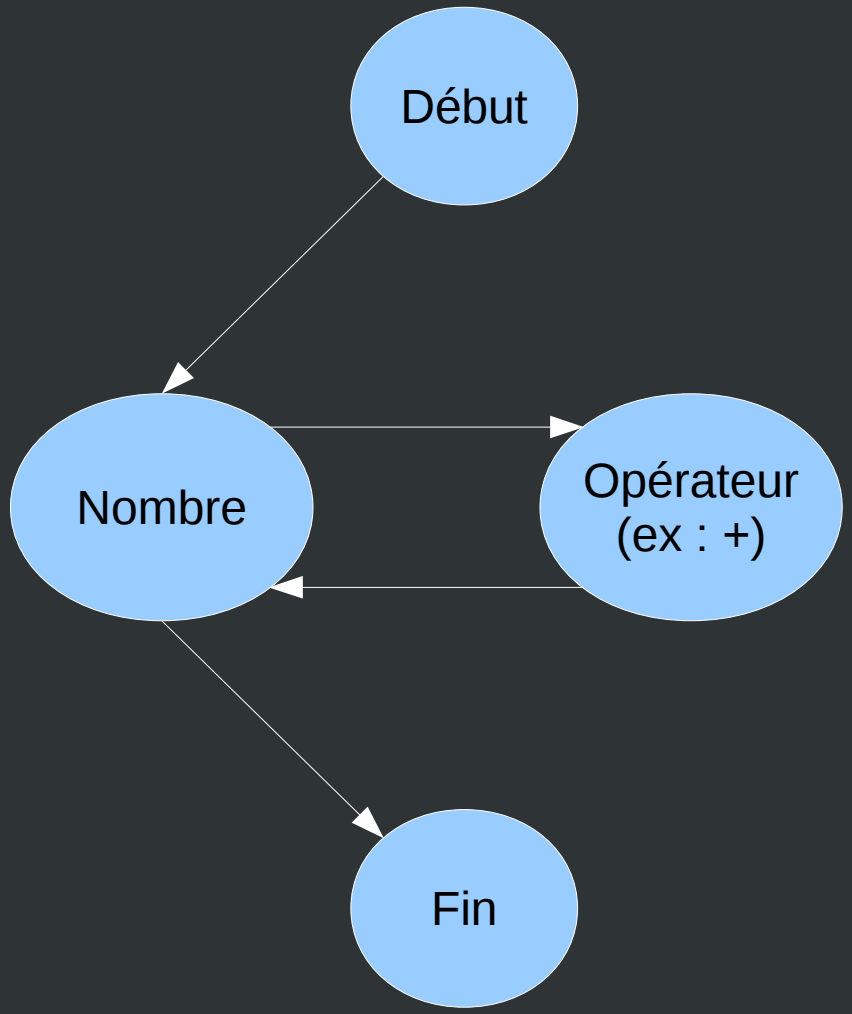
Analyse lexicale

- La classe `ParserLexical`
  - Permet de transformer une chaîne de caractères en une liste de **mots lexicaux.**
- La classe `MotLexical`
  - Représente les mots reconnus.
  - Possède un **type** (opérateur, variable, etc...).
  - Possède une **valeur** (”+”, ”42”, ”maVar”, etc...).

Analyse grammatical

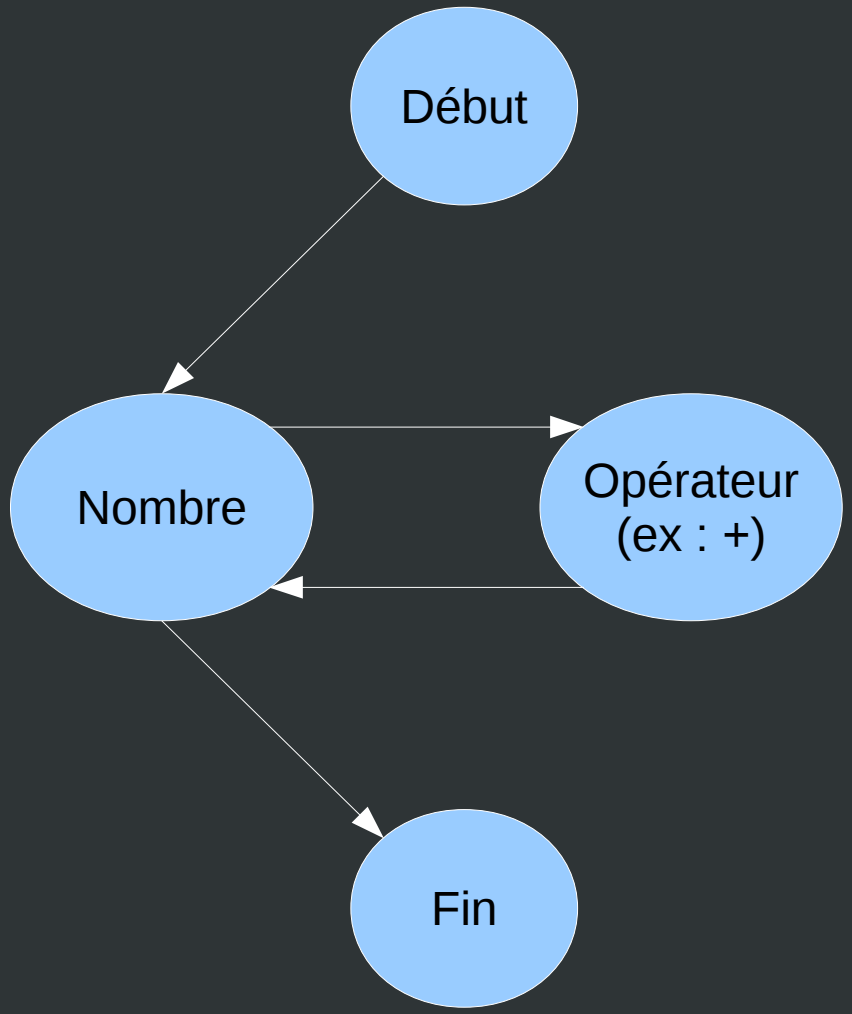
- Pour vérifier la conformité grammaticale d'une séquence de mots lexicaux, on utilise généralement un automate.
- Chaque état de l'automate représente un type de mot lexical.
- Les liaisons entre les états permettent de connaître les enchaînements de mots lexicaux autorisés.

Un exemple d'automate



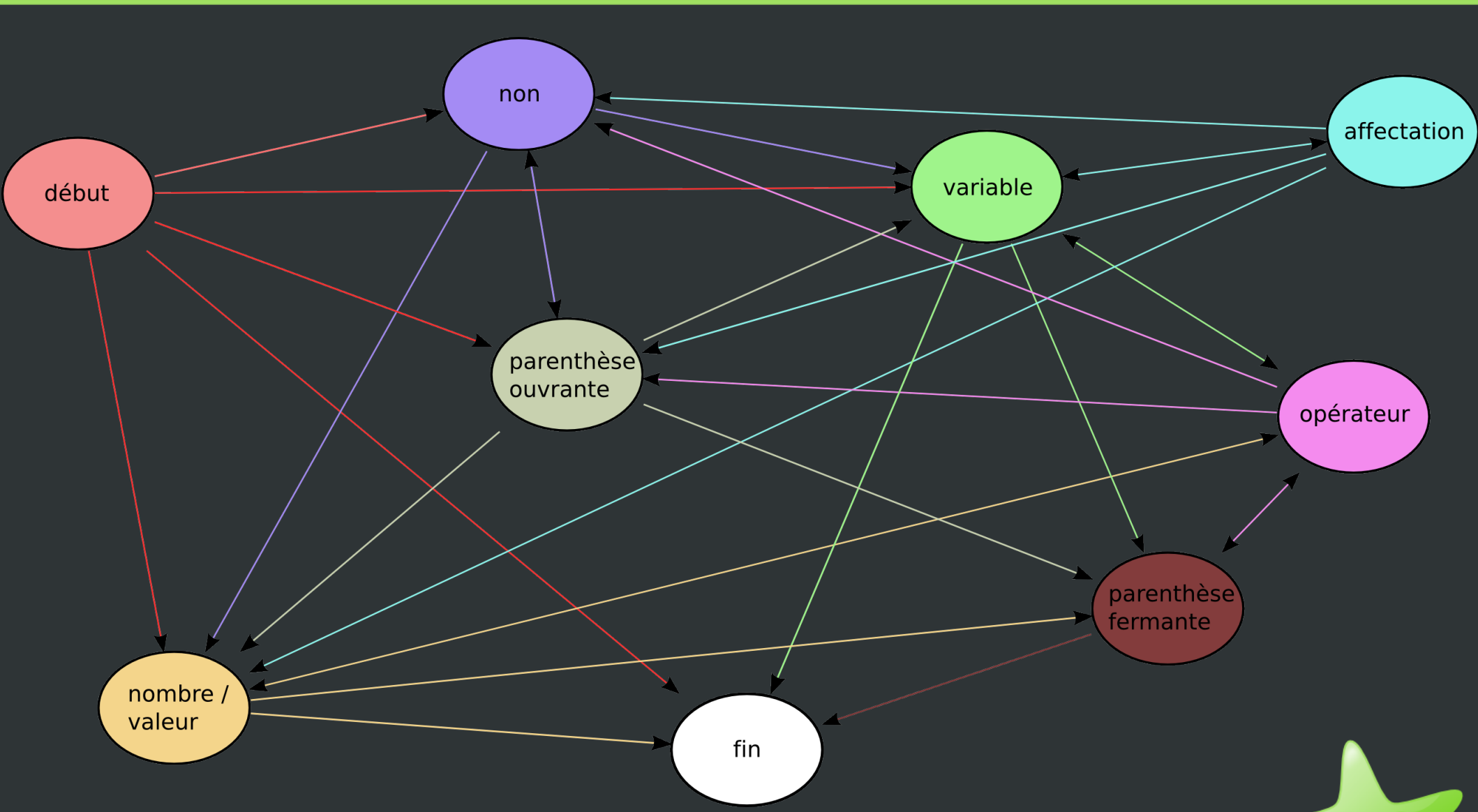
- Il est possible de passer de l'état «début» à l'état «nombre»
- Il est possible de passer de l'état «nombre» à l'état «opérateur» et à l'état «fin»
- Il est possible de passer de l'état opérateur à l'état «nombre»
- «Fin» est le seul état terminal : une expression doit finir sur un état terminal pour etre valide

Un exemple d'automate



- L'expression « 1 + 2 » est donc valide.
- L'expression « 1 + 2 + 4 + 5 » est valide.
- L'expression « 42 » est valide.
- L'expression « 1 + » n'est pas valide.

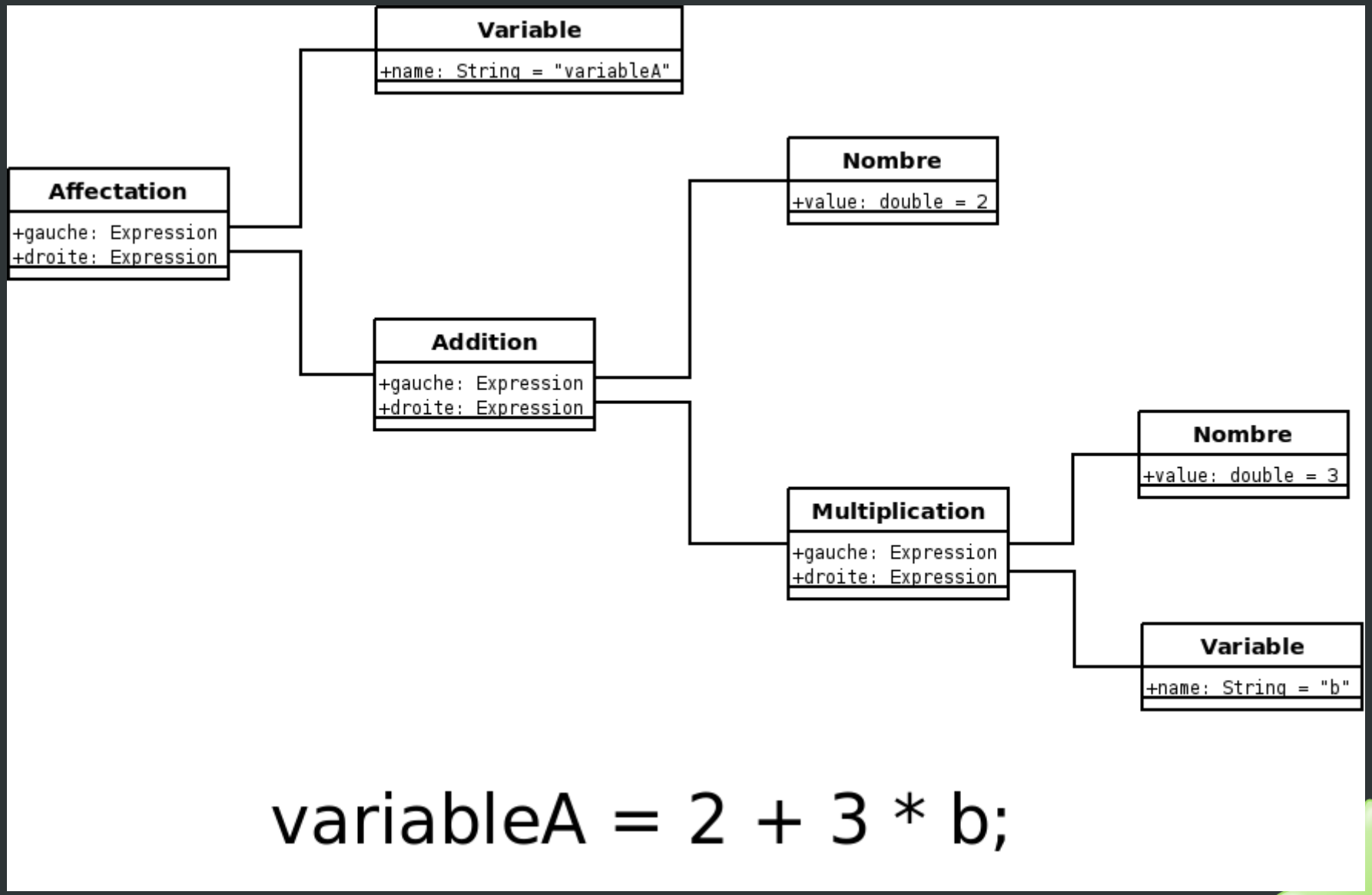
Automate de notre projet



Vue sous forme d'objets

- Les mots reconnus sont transformés sous forme d'objets respectant le pattern «observateur».
- Chaque objet peut contenir deux objets internes, un nommé «gauche» et un nommé «droite».
- Tous ces objets héritent de la classe «ExpressionAbstraite».
- Par construction, toute forme d'expression est donc possible.

Exemple


Variable
+name: String = "variableA"
Affectation
+gauche: Expression
+droite: Expression
Nombre
+value: double = 2
Addition
+gauche: Expression
+droite: Expression
Nombre
+value: double = 3
Multiplication
+gauche: Expression
+droite: Expression
Variable
+name: String = "b"
variableA = 2 + 3 * b;

Vue sous forme d'objets

- Dans notre programme, nous considérons qu'une expression est «complète à gauche» si son attribut «gauche» est défini (!= null).
- De même, une expression peut être «complète à droite» si son attribut «droite» est défini.
- Les expressions terminales comme les nombres sont complètes à gauche et à droite par défaut.
- Les expressions qui ne sont pas complètes devront être traitées par l'algorithme suivant.

Algorithme de transformation

- Les mots lexicaux sont stoqués dans un tableau que l'on nomme séquence.
- Chaque mot est transformé en objet correspondant.
- Si une expression n'est pas complète à gauche :
    - On défini son attribut «gauche» avec l'expression qui la précède dans la séquence.
    - L'expression précédente est remplacée par l'expression courante dans la séquence afin que celle qui la précède puisse prendre l'expression courante comme étant « à sa droite ».

Algorithme de transformation

- On applique le procédé correspondant à droite.

Exemple

- Considérons la séquence suivante:
  - [7] [+] [6]
  - Elle devient :
  - [Nombre : 7] [Addition] [Nombre 6]
- L'expression [Addition] est incomplète.
  - On la modifie donc :
  - [Addition (gauche = [Nombre 7], droite = [Nombre 6])]

Algorithme de transformation

- Les expressions incomplètes sont traitées dans l'ordre décroissant selon les règles de priorité mathématiques.
    - Exemple : la multiplication sera traitée avant l'addition.
- Les parenthèses () augmentent la priorité des expressions qu'elles contiennent, puis sont retirées de la séquence.

Exemple de déroulement

- **Début**
  - [0] 2
  - [1] (null + null)
  - [2] 3.0
  - [3] (null * null)
  - [4] 4.0
  - [5] (null) FIN
- **On traite : [3] (null * null)**
  - [0] 2.0
  - [1] (null + null)
  - [2] (3.0 * 4.0)
  - [3] (3.0 * 4.0)
  - [4] (3.0 * 4.0)
  - [5] (null) FIN
- **On traite : [1] (null + null)**
  - [0] (2.0 + (3.0 * 4.0))
  - [1] (2.0 + (3.0 * 4.0))
  - [2] (2.0 + (3.0 * 4.0))
  - [3] (2.0 + (3.0 * 4.0))
  - [4] (2.0 + (3.0 * 4.0))
  - [5] (null) FIN
- **On traite : [5] (null) FIN**
- **Fin de l'algorithme**, on choisit la dernière expression comme sommet de l'arbre d'expressions.
- On obtient : (2.0 + (3.0 * 4.0))

Interprétation

- Chaque objet de l'arbre d'expressions obtenu par l'algorithme de transformation possède une méthode «interprete()».
- Celle-ci retourne un résultat en exploitant à son tour la méthode «interprete()» de ses éléments gauche et droite.

Exemple

```
class Addition{

        public double interprete(){
                return gauche.interprete() + droite.interprete();
        }
        ...
}
```

Interprétation

- Notre implémentation est un peu plus complexe puisque le résultat retourné par la méthode «interprete» est une instance de la classe «Résultat» qui peut être un résultat booléen ou numérique.

Contexte

- Une classe «Context» stocke dans un dictionnaire «HashMap» les différentes variables définies par l'utilisateur à travers ses diverses affectations.
- Lors de l'appel à la méthode «interprete()», une instance de ce contexte est donnée en paramètre.
- Toute référence à une variable sera donc adressée à ce contexte qui retournera la variable concernée.

Utilisation de notre programme

- Il suffit de taper des expressions à interpréter dans le «prompt» pour que le résultat apparaisse.
  - Exemples :
    - >>> 1 + 2 * 6

      13.0
    - >>> a = 42
    - >>> b = 2 * a
    - >>> b > a

      vrai

Utilisation de notre programme

- La commande «context» affiche les variables présentes dans le contexte.
    - Exemple :

```
>>> context
b = 84.0
a = 42.0
```

- La commande «debug=on» active le mode «debug» et affichera le déroulement des algorithmes.
- Inversement, la commande «debug=off» désactive le mode «debug».

Points d'améliorations

- Utiliser des bibliothèques pour l'analyse lexicale et l'analyse grammaticale :
  - JFlex pour l'analyse lexicale
  - CUP pour l'analyse grammaticale
  - Ces deux bibliothèques ont fait leurs preuves
- Mieux séparer les expressions terminales des expressions non-terminales.
- Ajouter des fonctions mathématiques comme exp, ln, etc...

Conclusion

- Le design pattern interpréteur est facilement extensible.
- Il est efficace.
- Nous avons implémenté une petite calculatrice en mode texte qui s'avère aussi pratique que la calculatrice fournie par Windows.