UNIVERSITÉ DE ROUEN

UFR SCIENCE ET TECHNIQUES

RAPPORT PROJET D'ARCHITECTURE LOGICIELLE EXPRESSIONS

RÉDIGÉ PAR

HADDAM YOUNES

2022 - 2023

Table des matières

1 Présentation du projet **2**

2 Architecture Gloable **3**

3 Package utilisés **3**

4 Patrons utilisés **3**
4.1 Composite 3
4.2 Builder 4
4.3 Facade 4
4.4 Chaîne de résponsabilité 5

5 Patrons ecartés **6**
5.1 Visiteur 6
5.2 Observateur 6

6 Annexe **6**

7 JavaDoc **8**

8 Exécution **9**
8.1 Arith 9
8.2 Rationnelle 9
8.3 Fonctions 10
8.4 Nullable 10

9 Conclusion **11**

1 Présentation du projet

Le but de ce projet est de développer une bibliothèque qui permet de manipuler des expressions de différents types. Les types pris en charge sont les expressions arithmétiques, les fonctions et les expressions rationnelles. D'autres types d'expressions pourront être ajoutés à la bibliothèque à l'avenir.

Les expressions arithmétiques sont des calculs arithmétiques sans variables avec des opérations unaires et binaires. Les constantes utilisées sont des nombres réels.

Les fonctions permettent de faire des calculs arithmétiques avec une variable unique. Elles utilisent les mêmes opérations unaires et binaires que les expressions arithmétiques.

Les expressions rationnelles permettent de représenter des langages (ensembles de mots) à l'aide d'opérateurs rationnels tels que l'étoile de Kleene (répétition zéro ou plus) et l'union. Les constantes sont des mots composés de lettres de l'alphabet [a-z], de longueur comprise entre 1 et 20, ainsi que le mot vide représenté par le symbole "1".

— Le projet est composés de 3 programmes de manipulation des expressions. Expedid (Éditeur d'éxpressions) permet de saisir des expressions dans une notation polonaise inverse, de les charger ou de les sauvegarder dans un fichier au format XML.

— Le programme "calc" permet d'évaluer des expressions arithmétiques ou des fonctions, en fournissant éventuellement une valeur pour la variable "x" dans le cas des fonctions.

— Le programme "nullable" permet de déterminer si une expression rationnelle reconnaît le mot vide, en indiquant si le langage de l'expression contient le mot vide ou non.

2 Architecture Gloable

Cette architecture a été écrite en utilisant Graphviz qui est un outil opensource fait par dreampuf

Vous trouverez une image sur le ZIP.

3 Package utilisés

Pour l'intégration de la partie d'xml j'ai utilisé ce package qui se trouve dans le fichier pom.xml comme vous pouvez le voir dans la photo ci-dessous

Lien Git :

— https ://github.com/FasterXML/jackson-databind
— https ://github.com/FasterXML/jackson-dataformat-xml
— https ://github.com/FasterXML/jackson-annotations

```xml
<dependencies>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.13.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-xml</artifactId>
        <version>2.13.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.13.0</version>
    </dependency>

</dependencies>
```

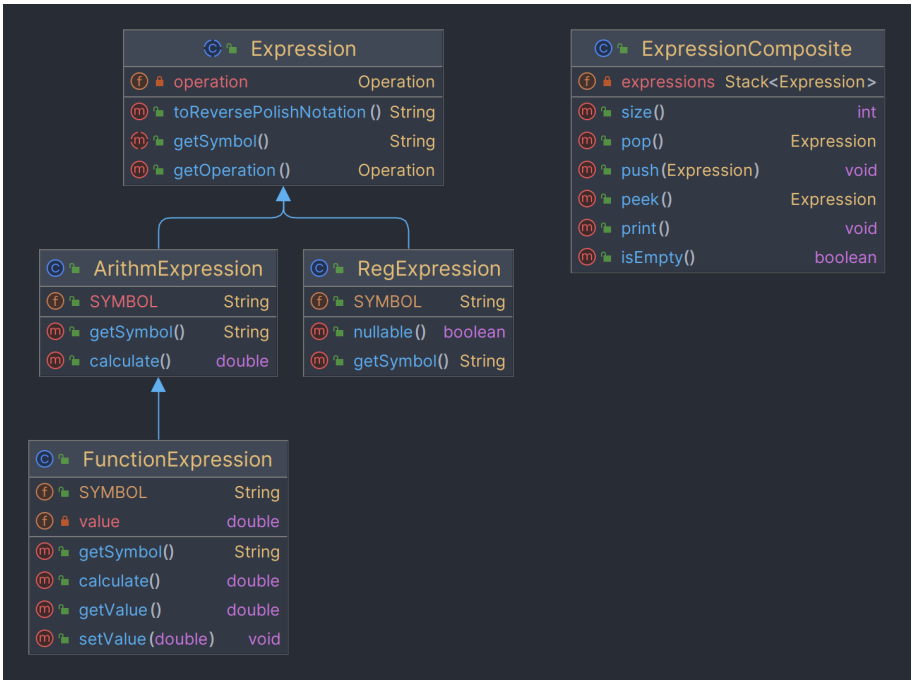
4 Patrons utilisés

Ces diagrammes ont été générés en utilisant le générateur de diagramme de Intellij

4.1 Composite

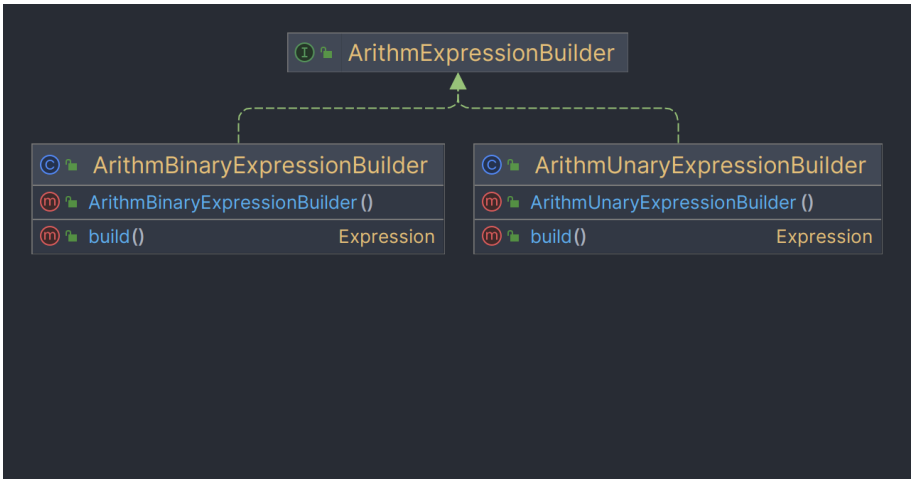
On a des expressions qui peuvent être composées d'autres expressions (les opérations unaires et binaires impliquent d'autres expressions). Le patron Composite est parfait pour cette situation car il nous permet de traiter de manière uniforme les objets individuels (opérations simples ou

variables) et les compositions d'objets (expressions). Ce patron peut simplifier le code car il nous permet de traiter à la fois les objets simples et les objets composites de la même manière.



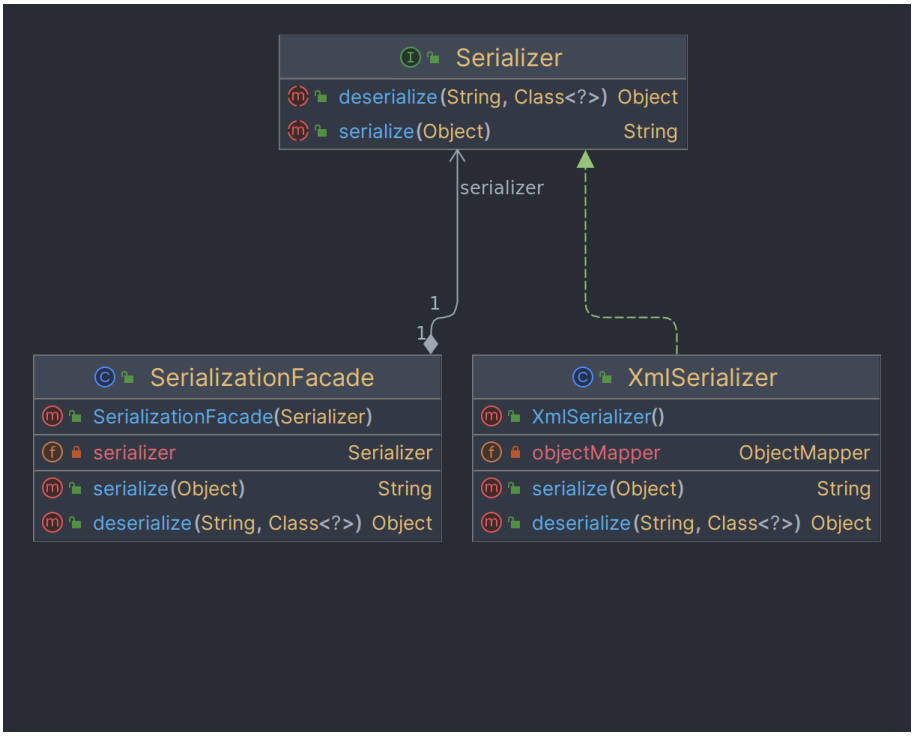
4.2 Builder

Dans le contexte de la création d'expressions complexes, le patron Builder serait utile. Les expressions peuvent avoir de multiples configurations et représentations. Le patron Builder permet une construction étape par étape d'expressions complexes, offrant ainsi une manière claire et flexible de créer ces objets.



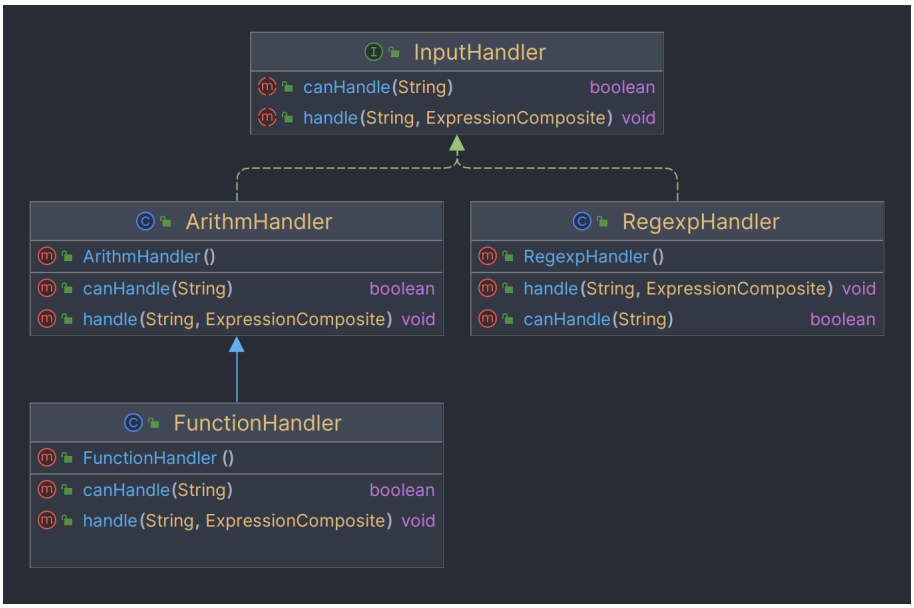
4.3 Facade

Dans notre conception, le SerializationFacade semble jouer le rôle d'une façade. Le patron Façade offre une interface simplifiée à un sous-système complexe. Ici, il nous fournit une manière simple de sérialiser et désérialiser les objets Expression complexes, en masquant les subtilités de la façon dont cette opération est effectuée.



4.4 Chaîne de résponsabilité

Nous avons différents handlers (ArithmHandler, FunctionHandler, RegexpHandler) pour traiter différents types d'entrées. Le patron chaine de responsabilité permet à ces gestionnaires d'être liés et de faire passer l'entrée jusqu'à ce qu'il trouve un gestionnaire qui peut la traiter. Cela permet de découpler les objets émetteurs et récepteurs, améliorant ainsi la modularité de notre code.



5 Patrons ecartés

Voici les patrons ecartés dans ce mini-projet :

5.1 Visiteur

Ce patron est une peu compliqué pour moi pour l'utiliser dans mon code mais j'ai utilisé d'autre patrons pour gérer la structure et les opérations sur les expressions.

5.2 Observateur

Il n'y avait pas de besoin spécifique d'établir une dépendance et une notification entre les objets.

6 Annexe

Cette annexe contient la liste complète des paquetages
Package arith : Ils contient des entités et des sous package :
— **Sous-package operation** : Il contient les entités suivantes :
  — Interface **ArithmOperation**
  — Classe **ArithmBinaryExpressionBuilder**
  — Classe **ArithmBinaryOperation**
  — Classe **ArithmConst**
  — Classe **ArithmOperationFactory**
  — Classe **ArithmUnaryExpressionBuilder**
  — Classe **ArithmUnaryOperation**
— **Sous-package operator** : Il contient les entités suivantes :
  — Enumération **ArithmBinaryOperator** : Cette classe représente des opérateurs arithmétiques binaires tels que l'addition, la soustraction, la multiplication et la division.
  — Enumération **ArithmUnaryOperator** : Cette classe représente un opérateur unaire, en particulier l'opérateur de négation.
— Classe **ArithmExpression**
— Classe **ArithmExpressionBuilder**
— Classe **ArithmHandler**
Package calc : Ils contient les entités suivantes :
— Classe **Calc**
— Classe **Calculator**
Package data : Il contient les entités suivantes :
— Classe **SerializationFacade**
— Classe **Serializer**
— Classe **XmlSerializer**
Package editor
— **Sous-package command**
  — Interface **CommandHandler**
  — Classe **LoadCommandHandler**
  — Classe **QuitCommandHandler**

    - Classe **SaveCommandHandler**
    - Classe **TypeCommandHandler**
- Classe **Editor**
- Classe **EditorContext**
- Classe **ExpressionComposite**
- Classe **ExpressionEditor**

Package expression

- **sous-package operation**
    - Classe abstraite **BinaryOperation**
    - Interface **Operation**
    - Classe abstraite **UnaryOperation**
- **sous-package operator**
    - Interface **BinaryOperation**
    - Interface **Operation**
    - Interface **UnaryOperation**
- Classe abstraite **BinaryExpressionBuilder**
- Classe abstraite **Expression**
- Interface **ExpressionBuilder**
- Enumération **ExpressionType**
- Interface **IExpression**
- Interface **InputHandler**
- Classe abstraite **UnaryExpressionBuilder**

Package function

- **sous-package operation**
    - Classe **FunctionBinaryExpressionBuilder**
    - Classe **FunctionBinaryOperation**
    - Classe **FunctionConst**
    - Interface **FunctionOperation**
    - Classe **FunctionOperationFactory**
    - Classe **FunctionUnaryExpressionBuilder**
    - Classe **FunctionUnaryOperation**
    - Classe **FunctionVariable**
- **sous-package operator**
    - Classe enumération**FunctionBinaryOperator**
    - Classe enumération **FunctionUnaryOperator**
- Classe **FunctionExpression**
- Interface **FunctionExpressionBuilder**
- Classe **FunctionHandler**

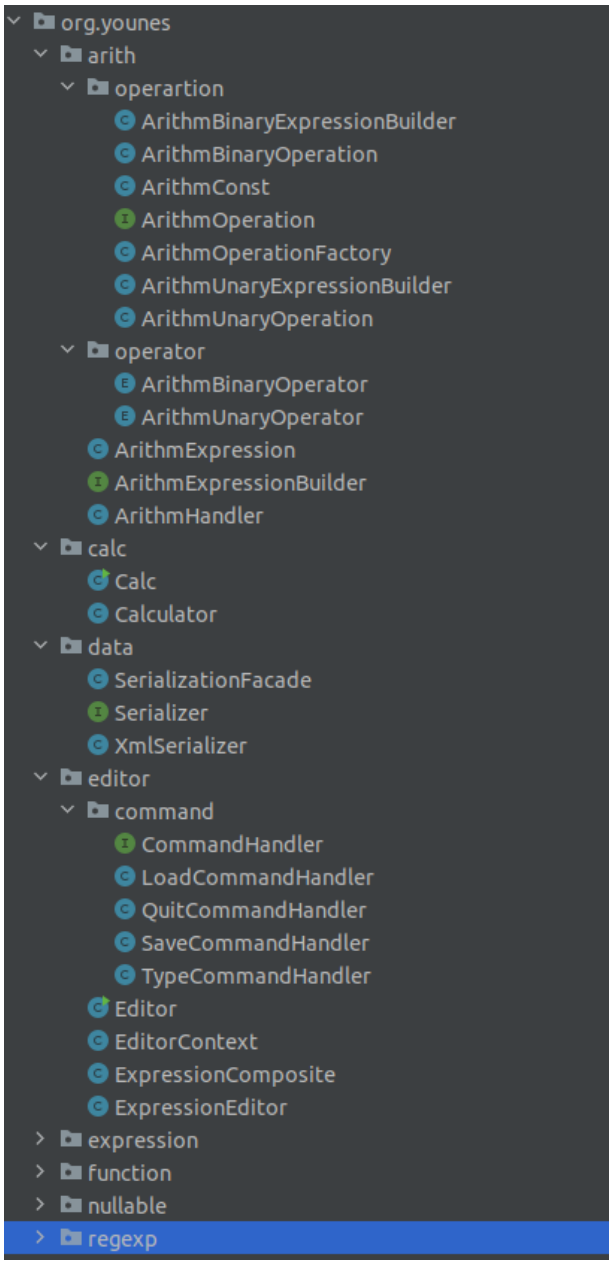
Package nullable

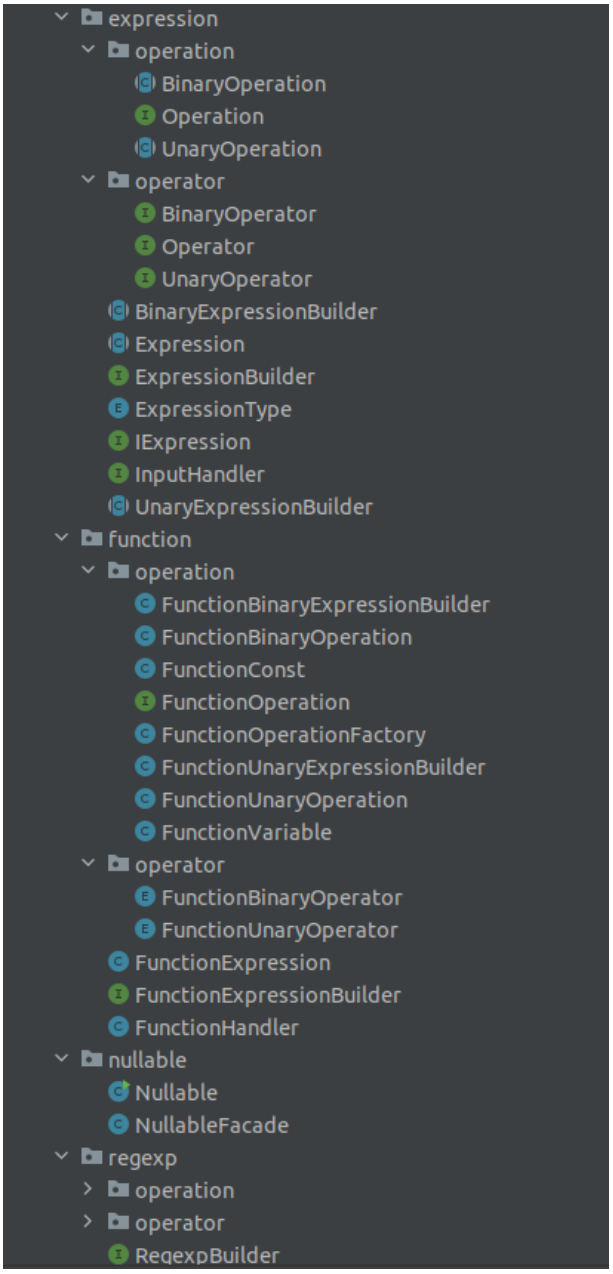
- Classe **Nullable**
- Classe **NullableFacade**

Package regexp

- **sous-package operation**
    - Classe **RegexpBinaryExpressionBuilder**
    - Classe **RegexpBinaryOperation**
    - Classe **RegexpConst**
    - Interface **RegexpOperation**

    - Classe **RegexpOperationFactory**
    - Classe **RegexpUnaryExpressionBuilder**
    - Classe **RegexpUnaryOperation**
- **sous-package operator**
    - Classe enumérationRegexpBinaryOperator**
    - Classe enumération **RegexpUnaryOperator**
- Classe **RegexpHandler**
- Interface **RegexpBuilder**
- Classe **RegExpression**



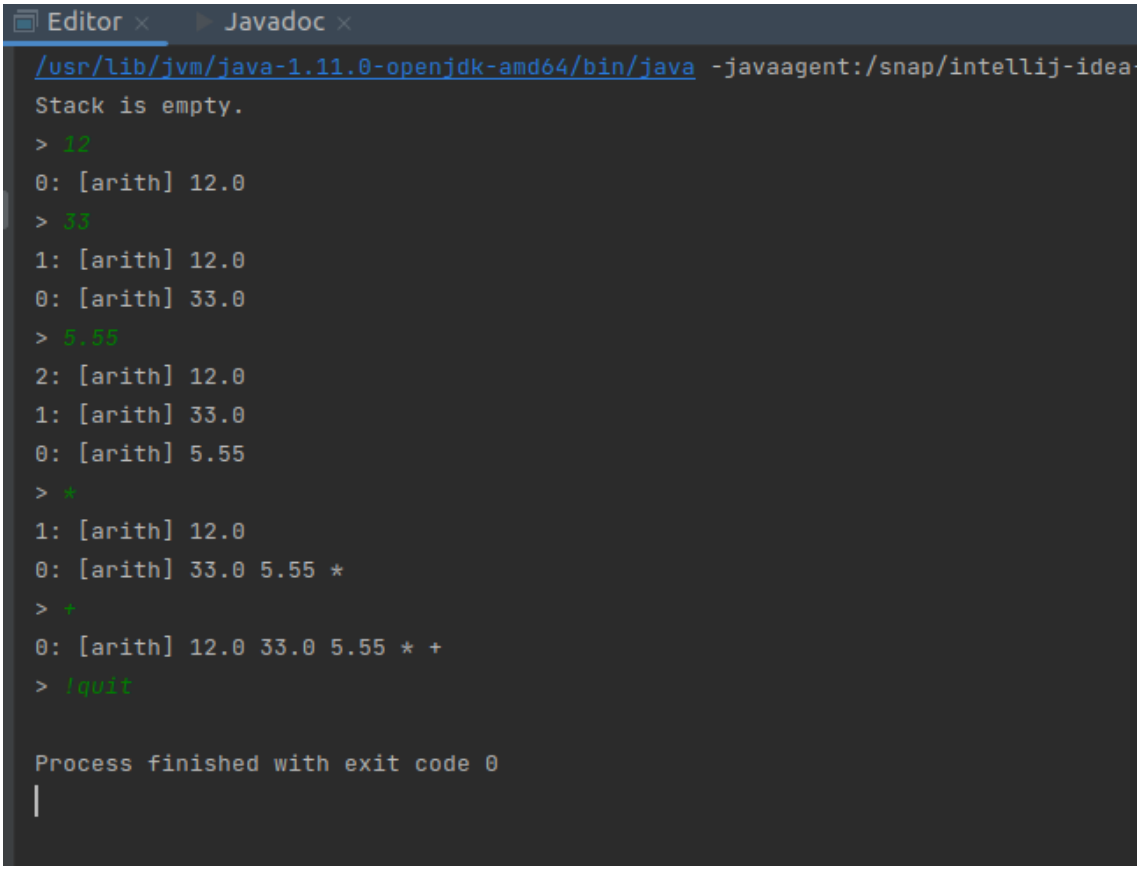


7 JavaDoc

Le fichier zip contient le javadoc générée par Intellij. Vous pouvez y accéder à partir de là.

8 Exécution

8.1 Arith



8.2 Rationnelle

8.3 Fonctions

```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:/snap/intellij-
Stack is empty.
> !type function
Changing type to: function
Stack is empty.
> 33
0: [function] 33.0
> 6.66
1: [function] 33.0
0: [function] 6.66
> *
0: [function] 33.0 6.66 *
> !save functions.xml
0: [function] 33.0 6.66 *
> quit
Invalid input: quit
0: [function] 33.0 6.66 *
> !quit

Process finished with exit code 0
```

functions.xml

```xml
<FunctionExpression>
    <operation operation="FunctionBinaryOperation">
        <left operation="FunctionConst">
            <value>33.0</value>
        </left>
        <right operation="FunctionConst">
            <value>6.66</value>
        </right>
        <operator>MULTIPLY</operator>
    </operation>
</FunctionExpression>
```

Load XML

```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:
Stack is empty.
> !type function
Changing type to: function
Stack is empty.
> 3
0: [function] 3.0
> !load functions.xml
1: [function] 3.0
0: [function] 33.0 6.66 *
>
```

8.4 Nullable

En raison d'un problème lors de la compilation, le filename a été hardcodé.

```java
public class Nullable {

    public static void main(String[] args) {

        // Hard coded :))
        args = new String[] { "src/main/resources/regex.xml" };
        if (args.length != 1) {
            System.out.println("Usage: java nullable <filename>");
            System.exit( status: 1);
        }

        String fileName = args[0];
```

Nullable

```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:/snap/intellij
TRUE

Process finished with exit code 0
```

9 Conclusion

Ce projet m'a permis de mettre en pratique les connaissances et compétences acquises lors du module enseigné en cours et en Tp. Toutefois, j'ai réalisé que la réalisation d'un logiciel efficace nécessite une planification de l'architecture du logiciel, en suivant les différents modèles établis. la solution que j'ai présentée soit une idée générale de notre concept, il est probable que des modifications et des améliorations doivent être apportées pour atteindre une réalisation optimale.

P.S : Il est possible que quelques photos ne sont pas claires, vous les trouverez tous dans le fichier ZIP.