COURS PATTERN INTERPRETER

Saliou Touré

Amadou Moctar Thiam

Bineta Sow

Pape Ibrahima Diop

PLAN

PRESENTATION DU PATTERN

- INTRODUCTION
- CONTEXTE ET PROBLEMATIQUE

SPECIFICATIONS

- STRUCTURE
- PARTICIPANTS
- EXEMPLES
- ARBRE DE SYNTAXE ABSTRAIT
- APPLICABILTE
- AVANTAGES ET INCONVENIENTS

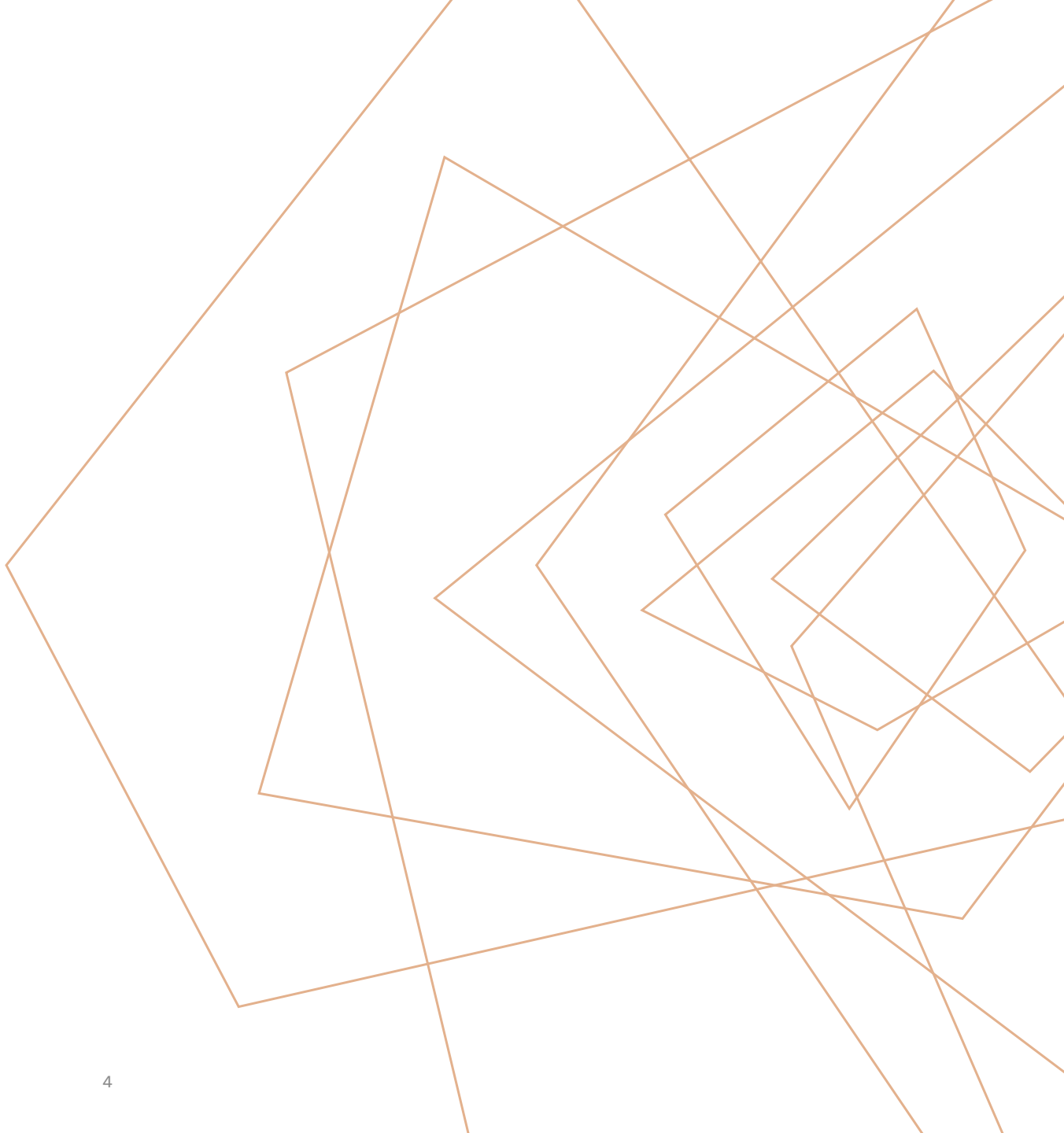
IMPLEMENTATION

- STRUCTURE DE LA DEMO DU PATTERN
- IMPLEMENTATION ET EXPLICATION DU CODE

PRESENTATION

INTRODUCTION

Le modèle d'interprète fournit un moyen d'évaluer la grammaire ou l'expression du langage. Ce type de modèle relève du modèle comportemental. Ce modèle implique la mise en œuvre d'une interface d'expression qui indique d'interpréter un contexte particulier. Ce modèle est utilisé dans l'analyse SQL, le moteur de traitement de symboles, etc.



CONTEXTE ET PROBLEMATIQUE

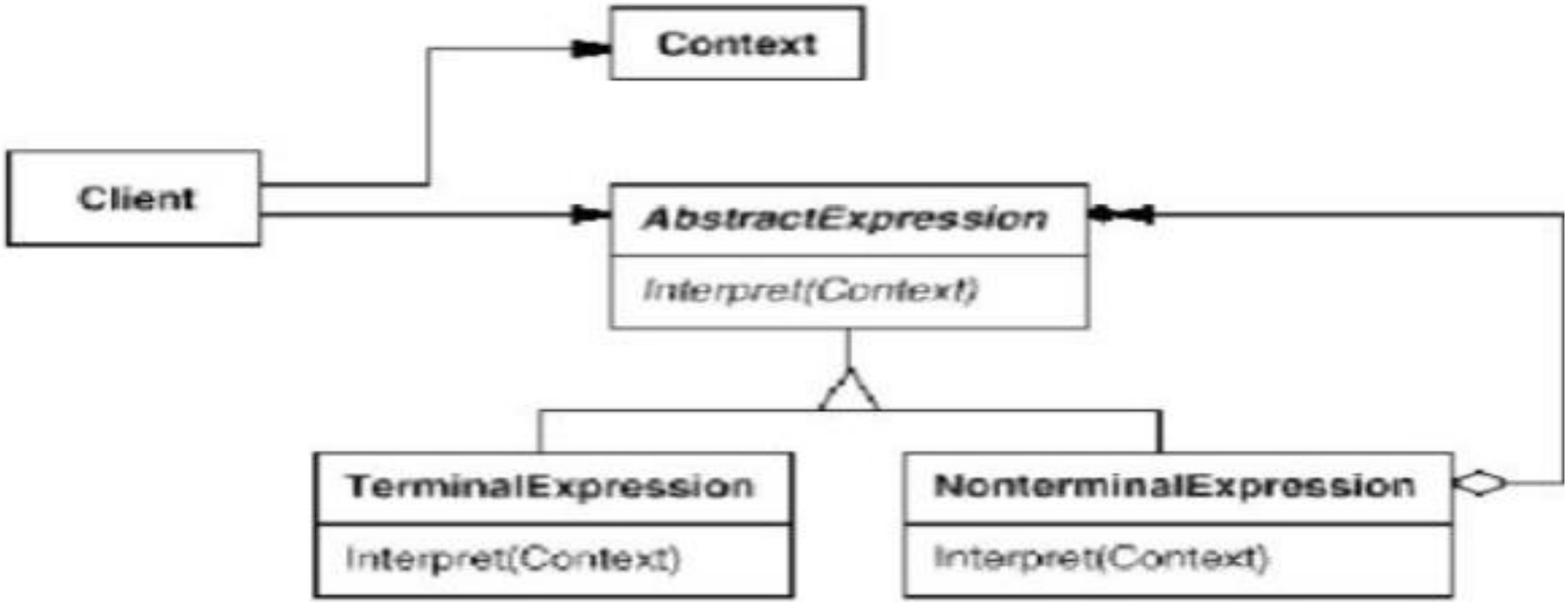
Étant donné une langue, définissez une représentation de sa grammaire ainsi qu'un interprète qui utilise la représentation pour interpréter des phrases dans la langue.

Si un type particulier de problème se produit assez souvent, il peut être intéressant d'exprimer les instances du problème sous forme de phrases dans un langage simple. Ensuite, vous pouvez créer un interprète qui résout le problème en interprétant ces phrases.

Par exemple, la recherche de chaînes qui correspondent à un modèle est un problème courant. Les expressions régulières sont un langage standard pour spécifier des modèles de chaînes. Plutôt que de créer des algorithmes personnalisés pour faire correspondre chaque modèle à des chaînes, les algorithmes de recherche pourraient interpréter une expression régulière qui spécifie un ensemble de chaînes à faire correspondre.

SPECIFICATION

STRUCTURE


Context
Client
AbstractExpression
Interpret(Context)
TerminalExpression
Interpret(Context)
NonterminalExpression
Interpret(Context)

PARTICIPANTS

- AbstractExpression (Expression régulière)

Déclare une opération d'interprétation abstraite qui est commune à tous les nœuds de l'arbre de syntaxe abstraite

- TerminalExpression (Expression littérale)

Implémente une opération Interpreter associée aux symboles terminaux dans la grammaire.

- NonterminalExpression (Expression d'alternance, Expression de répétition, Expressions de séquence)

Une telle classe est requise pour chaque règle R ::= R1 R2 ... Rn dans la grammaire

- Contexte

Contient des informations globales à l'interprète

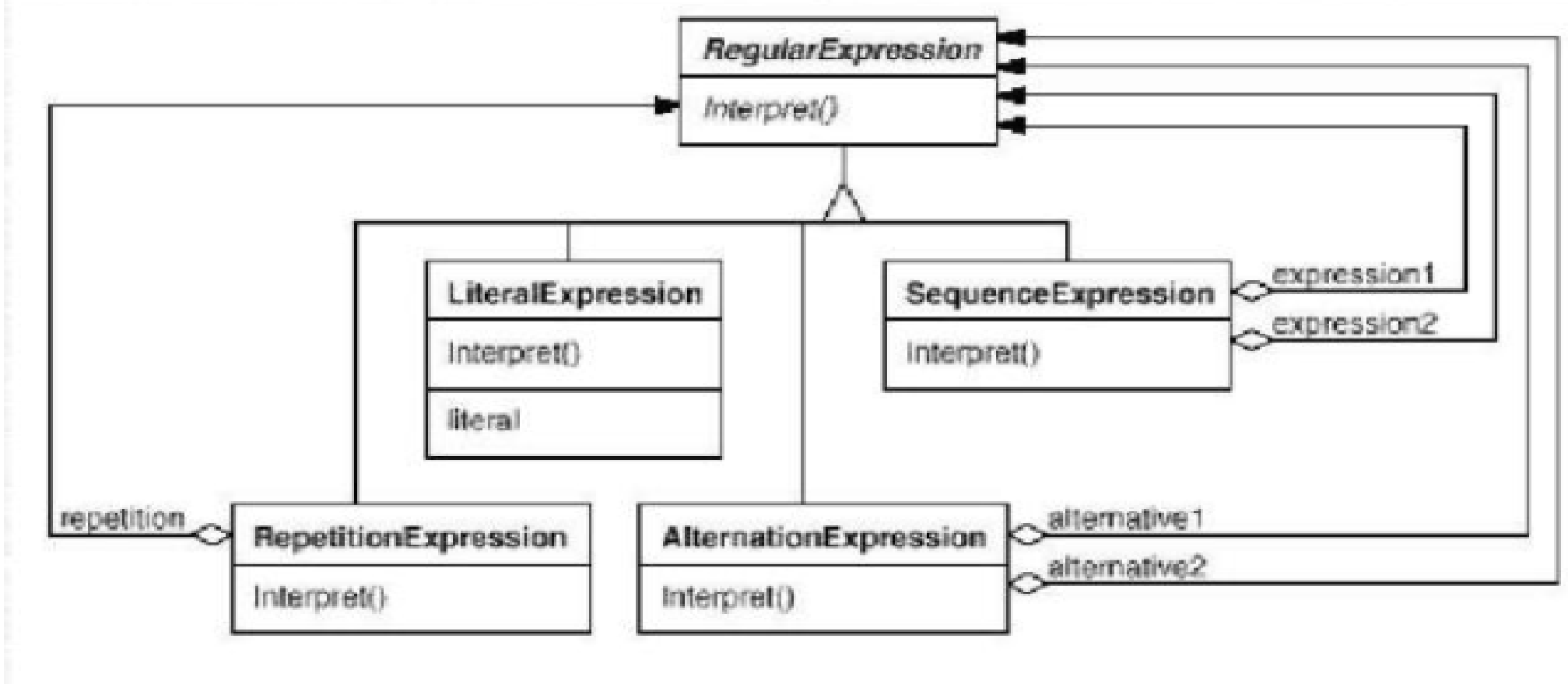
- Client

Construit (ou est donné) un arbre syntaxique abstrait représentant une phrase particulière dans la langue que la grammaire définit invoque l'opération Interpréter

EXEMPLES

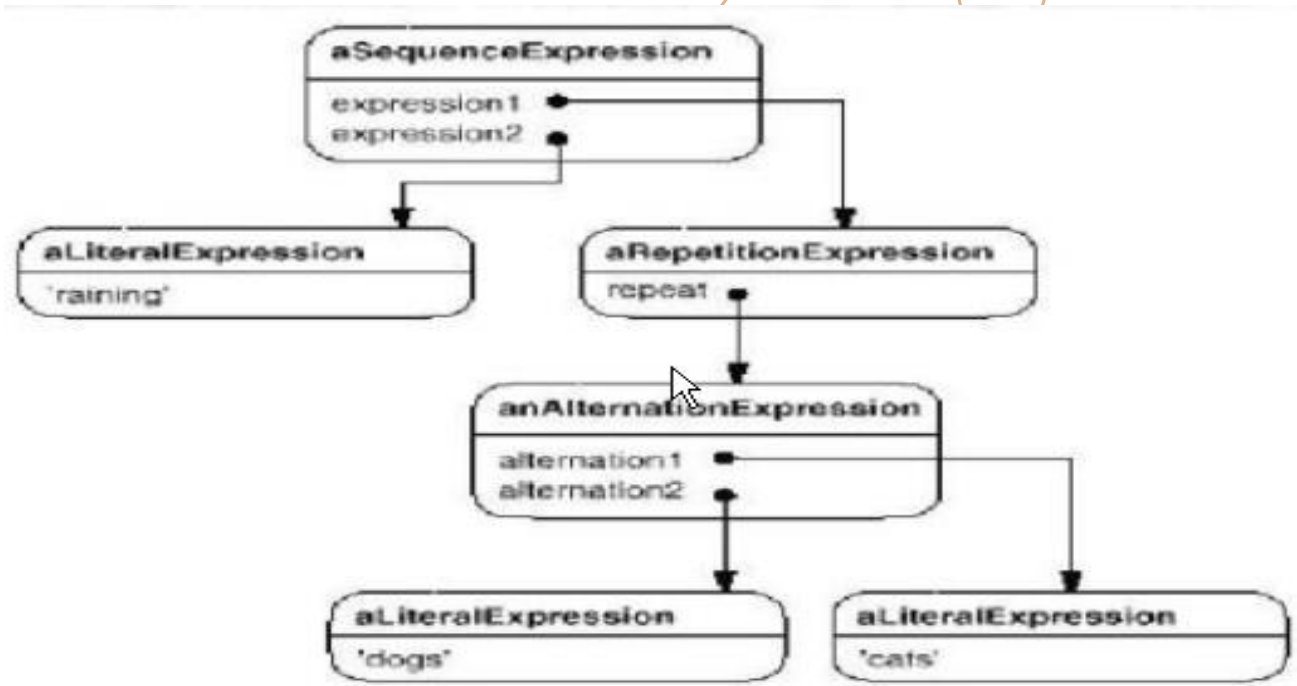
- expression ::= littéral | alternance | séquence | répétition | '(' expression ‘)’
- alternation ::= expression '|' expression
- sequence ::= expression '&' expression
- repetition ::= expression ‘*’
- literal ::= 'a' | 'b' | 'c' | ... { 'a' | 'b' | 'c' | ... }*

ARBRE DE SYNTAXE ABSTRAIT



ARBRE DE SYNTAXE ABSTRAIT

Expression régulière : il pleut & (chiens | chats)*



APPLICABILITE

Utilisez le modèle Interpreter lorsqu'il y a une langue à interpréter, et vous pouvez représenter les instructions du langage sous forme d'arbres de syntaxe abstraite. Le modèle Interpreter fonctionne mieux quand :

- La grammaire est simple. Pour les grammaires complexes, la hiérarchie des classes de la grammaire devient grande et ingérable. Des outils tels que les générateurs d'analyseurs syntaxiques sont une meilleure alternative dans de tels cas. Ils peuvent interpréter des expressions sans construire d'arbres de syntaxe abstraite, ce qui peut économiser de l'espace et éventuellement du temps.
- L'efficacité n'est pas une préoccupation majeure. Les interpréteurs les plus efficaces ne sont généralement pas implémentés en interprétant directement les arbres d'analyse mais en les traduisant d'abord sous une autre forme. Par exemple, les expressions régulières sont souvent transformées en machines à états. Mais même dans ce cas, le traducteur peut être implémenté par le modèle Interpreter, de sorte que le modèle est toujours applicable.

AVANTAGES ET INCONVENIENTS

AVANTAGES

Il est facile de modifier et d'étendre la grammaire.

La mise en œuvre de la grammaire est également facile.

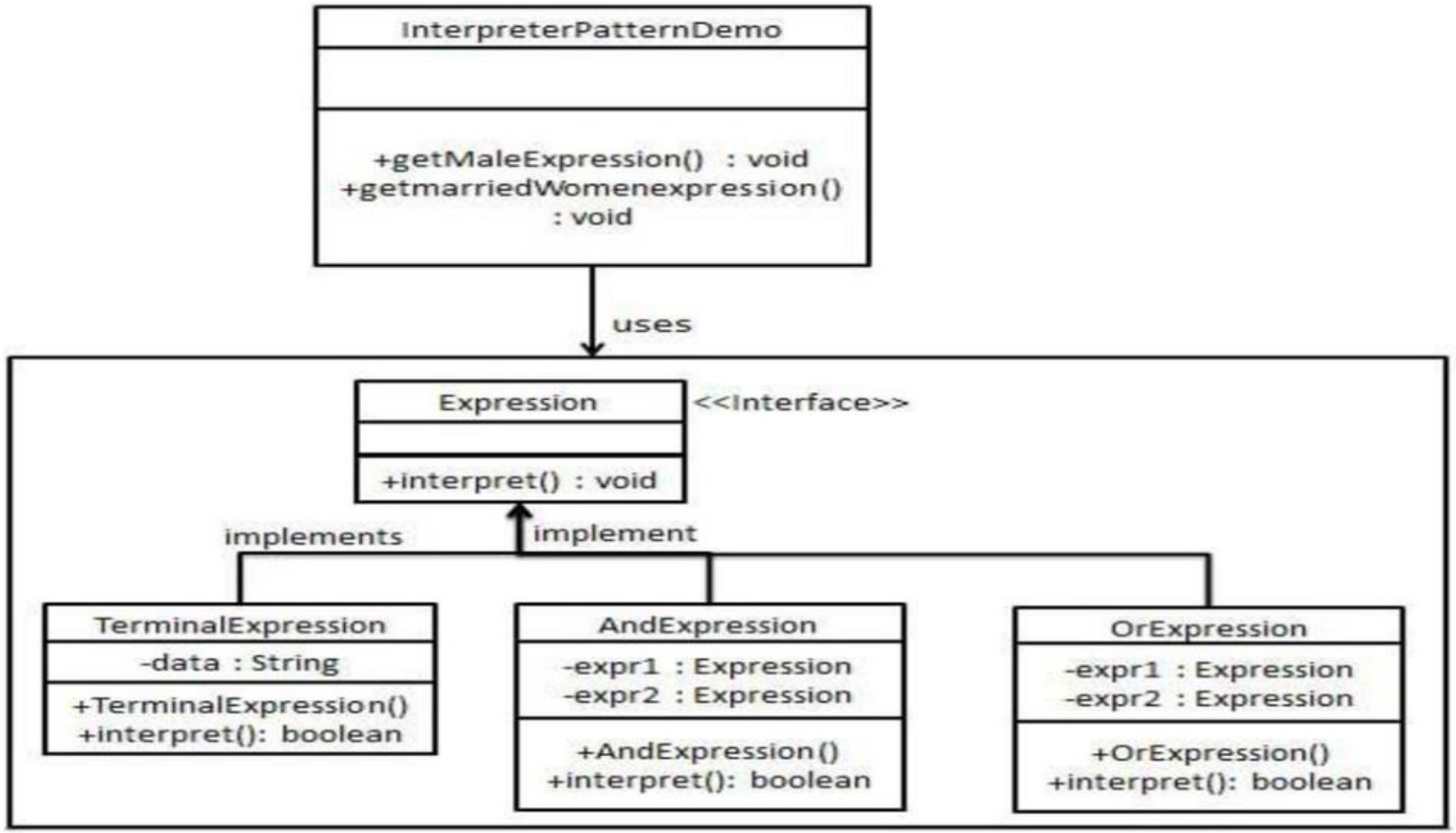
INCONVENIENTS

Les grammaires complexes sont difficiles à maintenir.

Ajout de nouvelles façons d'interpréter les expressions.

IMPLEMENTATION

STRUCTURE DE LA DEMO DU PATTERN



ÉTAPE 1 CRÉER UNE INTERFACE D'EXPRESSION

Expression.java

```java
public interface Expression
{
public boolean interpret(String context);
}
```

ÉTAPE 2 CRÉER DES CLASSES CONCRÈTES IMPLÉMENTANT L'INTERFACE CI-DESSUS

TerminalExpression.java > TerminalExpression

```java
public class TerminalExpression implements Expression {

    private String data;

    public TerminalExpression(String data) {
        this.data = data;
    }

    public boolean interpret(String context) {
        if (context.contains(data)) {
            return true;
        }
        return false;
    }
}
```

ÉTAPE 2 CREATION DE LA CLASSE OREXPRESSION

OrExpression.java > OrExpression > interpret(String)

```java
public class OrExpression implements Expression {

    private Expression expr1 = null;
    private Expression expr2 = null;

    public OrExpression(Expression expr1, Expression expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }

    public boolean interpret(String context) {
        return expr1.interpret(context) || expr2.interpret(context);
    }

}
```

ÉTAPE 2 CREATION DE LA CLASSE ANDEXPRESSION

AndExpression.java > AndExpression

```java
public class AndExpression implements Expression {
    private Expression expr1 = null;
    private Expression expr2 = null;

    public AndExpression(Expression expr1, Expression expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }

    public boolean interpret(String context) {
        return expr1.interpret(context) && expr2.interpret(context);
    }
}
```

ÉTAPE 3 INTERPRETERPATTERNDEMO UTILISE LA CLASSE EXPRESSION POUR CRÉER DES RÈGLES, PUIS LES ANALYSER.

nterpreterPatternDemo.java > InterpreterPatternDemo

```java
public class InterpreterPatternDemo {
    // Rule: Robert and John are male
    public static Expression getMaleExpression() {
        Expression robert = new TerminalExpression("Robert");
        Expression john = new TerminalExpression("John");
        return new OrExpression(robert, john);
    }

    // Rule: Julie is a married women
    public static Expression getMarriedWomanExpression() {
        Expression julie = new TerminalExpression("Julie");
        Expression married = new TerminalExpression("Married");
        return new AndExpression(julie, married);
    }

    Run | Debug
    public static void main(String[] args) {
        Expression isMale = getMaleExpression();
        Expression isMarriedWoman = getMarriedWomanExpression();
        System.out.println("John is male? " + isMale.interpret("John"));
        System.out.println("Julie is a married woman? " + isMarriedWoman.interpret("Married Julie"));
    }
}
```

ÉTAPE 4 VÉRIFIEZ LA SORTIE

```
PS P:\LICENCE\Design Pattern> JAVAC *.java
PS P:\LICENCE\Design Pattern> java InterpreterPatternDemo
John is male? true
Julie is a married woman? true
PS P:\LICENCE\Design Pattern>
```

MERCI