

# Soutenance du Projet - Calculatrice Orientée Objet

Bienvenue à notre soutenance de projet. Nous sommes deux membres, Enrike et Mehdi. Aujourd'hui, nous allons vous présenter notre projet de calculatrice orientée objet. Cette présentation sera structurée de manière à répondre à vos attentes : démonstration du fonctionnement, organisation du travail, choix techniques, difficultés rencontrées et contribution de chacun.

 **2 contributeurs**



# Implémentation de la Hiérarchie d'Expressions

```
public abstract class Expression {  
    public abstract double valeur();  
}
```

Nous avons implémenté la hiérarchie des expressions mathématiques en Java, avec les classes **Expression** et **Operation**. La classe **Expression** est une classe abstraite définissant la méthode **valeur()** qui calcule la valeur de l'expression. La classe **Operation**, qui étend **Expression**, représente une opération mathématique binaire avec deux opérandes de type **Expression**.

```
public abstract class Operation extends Expression{  
    private Expression operande1;  
    private Expression operande2;  
  
    public Operation(Expression e1, Expression e2){  
        this.operande1=e1;  
        this.operande2=e2;  
    }  
  
    public Operation(Operation o){  
        this.operande1=o.getOperande1();  
        this.operande2=o.getOperande2();  
    }  
  
    public Expression getOperande1(){  
        return this.operande1;  
    }  
  
    public Expression getOperande2(){  
        return this.operande2;  
    }  
  
    public abstract double valeur();  
}
```

# Démonstration du Fonctionnement

## Classe Nombre

```
public class Nombre extends Expression{
    private double valeurNombre;

    public Nombre(){
        this.valeurNombre=0;
    }

    public Nombre(int v){
        this.valeurNombre=v;
    }

    public Nombre(Nombre n){
        this.valeurNombre=n.valeur();
    }

    public double valeur(){
        return this.valeurNombre;
    }

    public String toString() {
        return String.valueOf(this.valeurNombre);
    }
}
```

## Classe Addition

```
public class Addition extends Operation{
    public static final String operateur="+";

    public Addition(Expression e1, Expression e2){
        super(e1,e2);
    }

    public Addition(Operation o){
        super(o);
    }

    public double valeur(){
        return getOperande1().valeur() + getOperande2().valeur();
    }

    public String toString(){
        return getOperande1() + " " + operateur + " " + getOperande2();
    }
}
```

## Classe Soustraction

```
public class Soustraction extends Operation{
    public static final String operateur="-";

    public Soustraction(Expression e1, Expression e2){
        super(e1, e2);
    }

    public Soustraction(Operation o){
        super(o);
    }

    public double valeur(){
        return getOperande1().valeur() - getOperande2().valeur();
    }

    public String toString(){
        return getOperande1() + " " + operateur + " " + getOperande2();
    }
}
```

## Classe Multiplication

```
public class Multiplication extends Operation{
    public static final String operateur="*";

    public Multiplication(Expression e1, Expression e2){
        super(e1, e2);
    }

    public Multiplication(Operation o){
        super(o);
    }

    public double valeur(){
        return getOperande1().valeur() * getOperande2().valeur();
    }

    public String toString(){
        return getOperande1() + " " + operateur + " " + getOperande2();
    }
}
```

## Classe Division

```
public class Division extends Operation{
    public static final String operateur="/";

    public Division(Expression e1, Expression e2) {
        super(e1, e2);
    }

    public Division(Operation o){
        super(o);
    }

    public double valeur(){
        try {
            if (getOperande2().valeur() == 0) {
                throw new ArithmeticException("Division par zéro");
            }
            return getOperande1().valeur() / getOperande2().valeur();
        } catch (ArithmeticalException e) {
            System.out.println("Erreur: " + e.getMessage());
            return 0;
        }
    }

    public String toString(){
        return getOperande1() + " " + operateur + " " + getOperande2();
    }
}
```



# Démonstration du Fonctionnement (suite)

## Calculatrice

```
public class Calculatrice {  
    public static void main(String[] args) {  
        Expression deux = new Nombre(2);  
        Expression trois = new Nombre(3);  
        Expression dixSept = new Nombre(17);  
        Expression s = new Soustraction(dixSept, deux);  
        Expression a = new Addition(deux, trois);  
        Expression d = new Division(s, a);  
        System.out.println(d + " = " + d.valeur()); // Affiche ((17.0 - 2.0) / (2.0 + 3.0)) = 3.0  
    }  
}
```

## Calculatrice simple

```
public class CalculatriceSimple {  
    public static void main(String[] args) {  
        Nombre six = new Nombre(6);  
        Nombre dix = new Nombre(10);  
        Operation s = new Soustraction(dix, six);  
        System.out.println(s + " = " + s.valeur());  
    }  
}
```

## Bonus

```
public class Test {
    public static Expression fabriqueExpression(String e) {
        e = e.replaceAll("\\s", "");

        if (e.matches("\\d+")) {
            return new Nombre(Integer.parseInt(e));
        }

        int openBrackets = 0;
        for (int i = 0; i < e.length(); i++) {
            char c = e.charAt(i);
            if (c == '(') openBrackets++;
            else if (c == ')') openBrackets--;
            else if ((c == '+' || c == '-' || c == '*' || c == '/') && openBrackets == 0) {
                Expression left = fabriqueExpression(e.substring(0, i));
                Expression right = fabriqueExpression(e.substring(i + 1));
                switch (c) {
                    case '+':
                        return new Addition(left, right);
                    case '-':
                        return new Soustraction(left, right);
                    case '*':
                        return new Multiplication(left, right);
                    case '/':
                        return new Division(left, right);
                }
            }
        }

        if (e.startsWith("(") && e.endsWith(")")) {
            return fabriqueExpression(e.substring(1, e.length() - 1));
        }

        throw new IllegalArgumentException("Expression non valide: " + e);
    }

    public static void main(String[] args) {
        String expr1 = "3";
        String expr2 = "17-2";
        String expr3 = "(17-2)/3";

        System.out.println(fabriqueExpression(expr1));
        System.out.println(fabriqueExpression(expr2));
        System.out.println(fabriqueExpression(expr3));
    }
}
```

# Difficultés Rencontrées

1

## Gestion des Parenthèses et de la Précédence

La méthode fabriqueExpression devait gérer correctement les parenthèses et la précédence des opérateurs. Cela a nécessité une analyse minutieuse de l'expression.

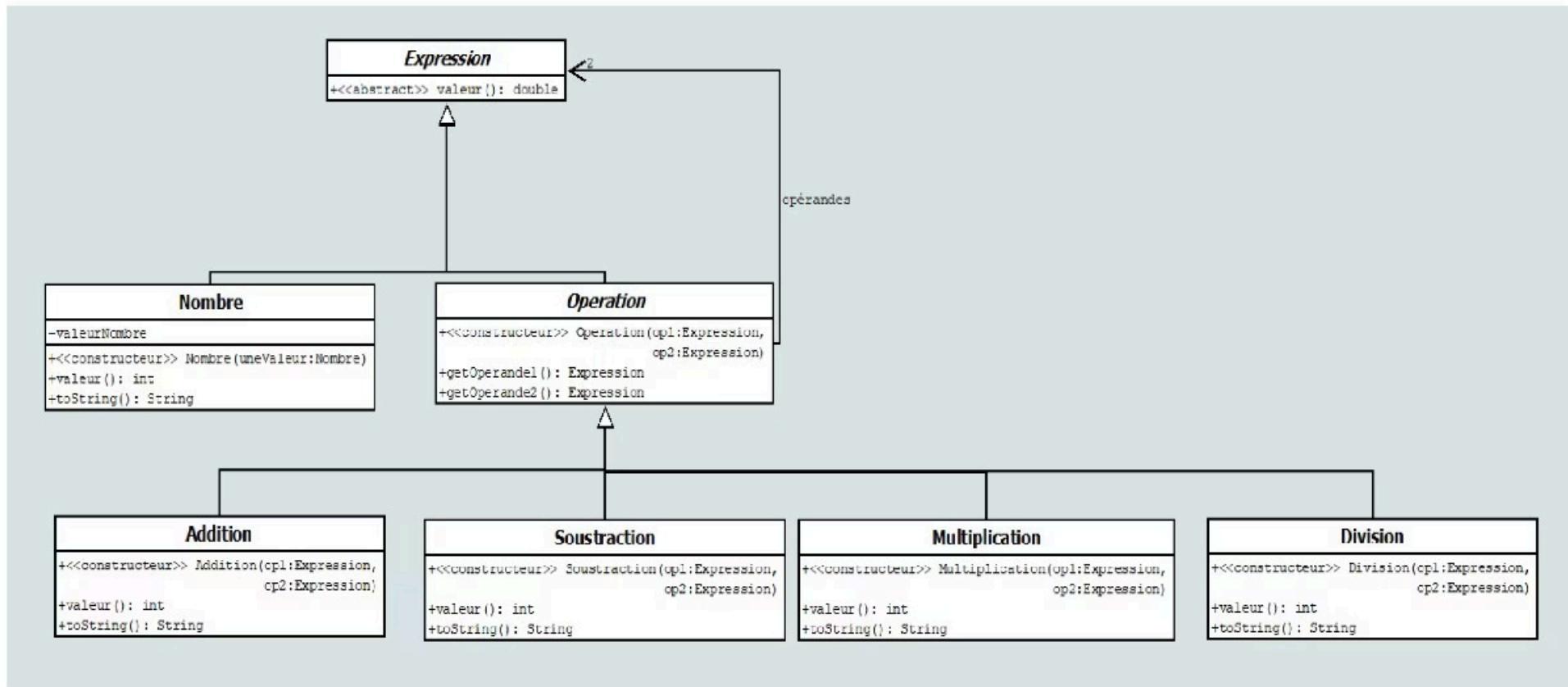
2

## Division par Zéro

Implémentation de la gestion des exceptions pour éviter les erreurs de division par zéro.

# Diagramme UML

Nous avons structuré notre projet en plusieurs classes et chaque membre du binôme a contribué aux différentes parties du code. Voici notre diagramme UML pour une vue d'ensemble :



# Conclusion

Pour conclure, nous espérons que cette présentation a démontré la fonctionnalité et l'architecture bien pensée de notre projet. Nous avons veillé à une répartition équilibrée des tâches et à une conception modulaire pour faciliter les extensions futures. Merci de votre attention. Nous sommes maintenant ouverts à vos questions.

