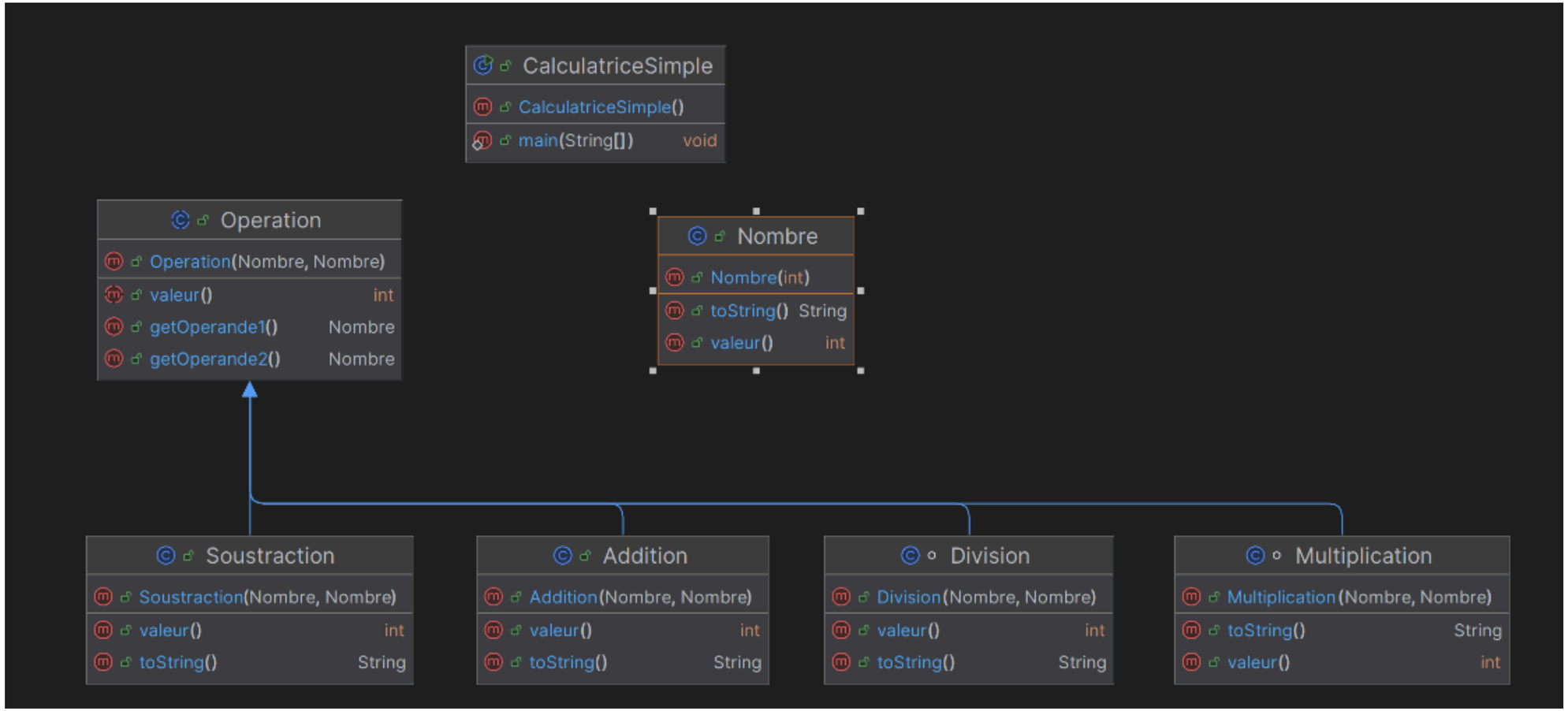
SAE R201
Ma Calculatrice


CalculatriceSimple
CalculatriceSimple()
main(String[]) void
Operation
Operation(Nombre, Nombre)
valeur() int
getOperande1() Nombre
getOperande2() Nombre
Nombre
Nombre(int)
toString() String
valeur() int
Soustraction
Soustraction(Nombre, Nombre)
valeur() int
toString() String
Addition
Addition(Nombre, Nombre)
valeur() int
toString() String
Division
Division(Nombre, Nombre)
valeur() int
toString() String
Multiplication
Multiplication(Nombre, Nombre)
toString() String
valeur() int

```java
public class Addition extends Operation {
    3 usages
    public Addition(Expression operande1, Expression operande2) {
        super(operande1, operande2);
    }

    17 usages
    public int valeur() { return getOperande1().valeur() + getOperande2().valeur(); }

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

```java
public class Soustraction extends Operation {
    3 usages
    public Soustraction(Expression operande1, Expression operande2) { super(operande1, operande2); }

    17 usages
    public int valeur() { return getOperande1().valeur() - getOperande2().valeur(); }

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

```java
public class Multiplication extends Operation {
    2 usages
    public Multiplication(Expression operande1, Expression operande2) { super(operande1, operande2); }

    17 usages
    public int valeur() { return getOperande1().valeur() * getOperande2().valeur(); }

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

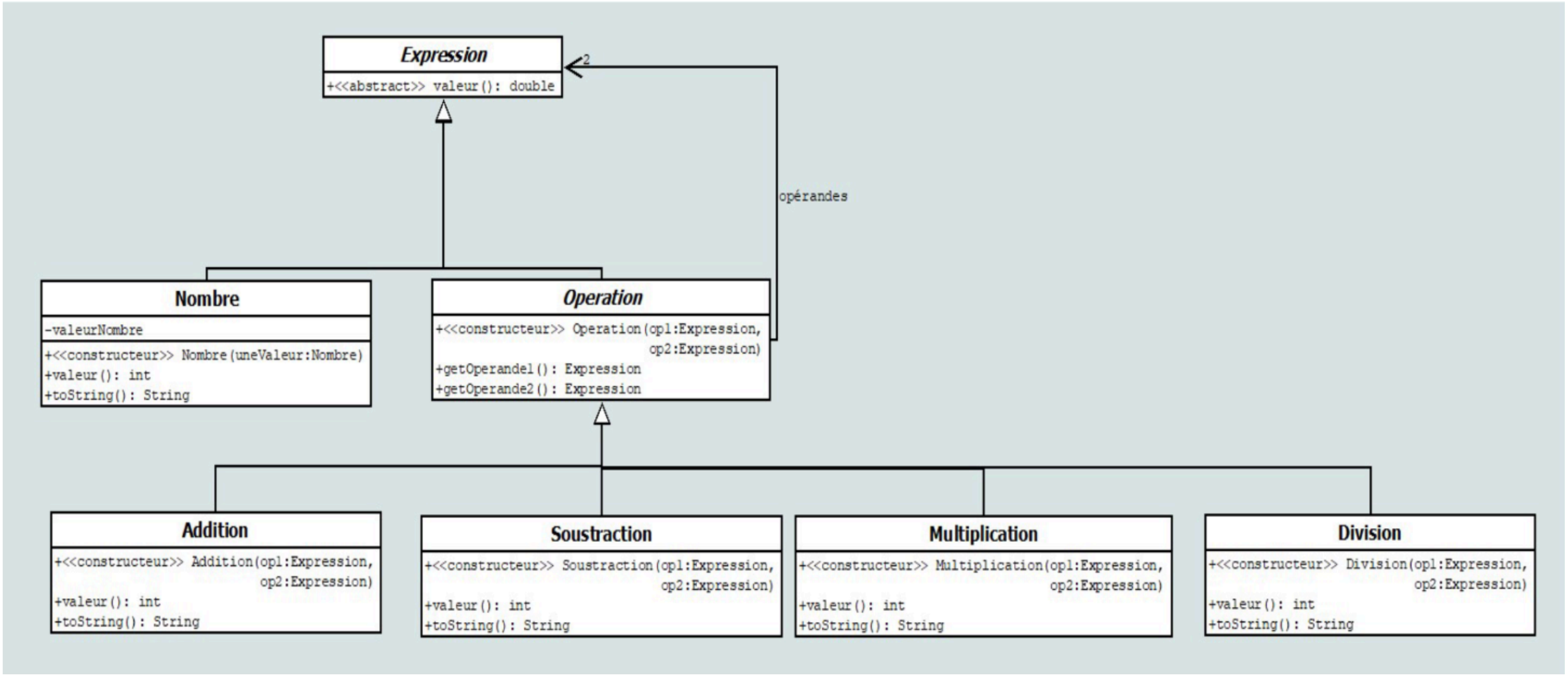
```java
public class Division extends Operation {
    3 usages
    public Division(Expression operande1, Expression operande2) {
        super(operande1, operande2);
    }

    17 usages
    public int valeur() throws ArithmeticException{
        if (getOperande2().valeur() == 0) {
            throw new ArithmeticException("Division by zero");
        }
        return getOperande1().valeur() / getOperande2().valeur();
    }

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

```java
public class CalculatriceSimple {
    public static void main(String[] args) {
        try {
            Nombre six = new Nombre( valeur: 6);
            Nombre dix = new Nombre( valeur: 10);
            Operation addition = new Addition(dix, six);
            Operation soustraction = new Soustraction(dix, six);
            Operation multiplication = new Multiplication(dix, six);
            Operation division = new Division(dix, six);

            System.out.println(addition + " = " + addition.valeur());
            System.out.println(soustraction + " = " + soustraction.valeur());
            System.out.println(multiplication + " = " + multiplication.valeur());
            System.out.println(division + " = " + division.valeur());
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```


Expression
+<<abstract>> valeur(): double
2
opérandes
Nombre
-valeurNombre
+<<constructeur>> Nombre(uneValeur:Nombre)
+valeur(): int
+toString(): String
Operation
+<<constructeur>> Operation(op1:Expression,
op2:Expression)
+getOperande1(): Expression
+getOperande2(): Expression
Addition
+<<constructeur>> Addition(op1:Expression,
op2:Expression)
+valeur(): int
+toString(): String
Soustraction
+<<constructeur>> Soustraction(op1:Expression,
op2:Expression)
+valeur(): int
+toString(): String
Multiplication
+<<constructeur>> Multiplication(op1:Expression,
op2:Expression)
+valeur(): int
+toString(): String
Division
+<<constructeur>> Division(op1:Expression,
op2:Expression)
+valeur(): int
+toString(): String

```java
public abstract class Expression {
    17 usages  6 implementations
    public abstract int valeur();
}
```

```java
public abstract class Operation extends Expression {
    2 usages
    private final Expression operande1;
    2 usages
    private final Expression operande2;

    4 usages
    public Operation(Expression operande1, Expression operande2) {
        this.operande1 = operande1;
        this.operande2 = operande2;
    }

    17 usages  4 implementations
    public abstract int valeur();

    8 usages
    public Expression getOperande1() { return operande1; }

    9 usages
    public Expression getOperande2() { return operande2; }
}
```

```java
public class Nombre extends Expression {
    3 usages
    private final int valeurNombre;

    6 usages
    public Nombre(int valeur) { this.valeurNombre = valeur; }

    17 usages
    public int valeur() {
        return valeurNombre;
    }

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

```java
public class Calculatrice {
    public static void main(String[] args) {
        Expression deux = new Nombre( valeur: 2);
        Expression trois = new Nombre( valeur: 3);
        Expression dixSept = new Nombre( valeur: 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 - 2) / (2 + 3)) = 3
    }
}
```

```java
public class Test {
    3 usages
    public static Expression fabriqueExpression(String e) {
        // Supprime les espaces blancs de la chaîne
        e = e.replaceAll( regex: "\\s",  replacement: "");
        // Analyse et retourne l'expression
        return parseExpression(e);
    }
    3 usages
    private static Expression parseExpression(String e) {
        // Vérifie si la chaîne est un nombre
        if (e.matches( regex: "\\d+")) {
            // Retourne une nouvelle instance de Nombre
            return new Nombre(Integer.parseInt(e));
        }
```

```java
    // Trouve l'opérateur principal
    int parenCount = 0;
    for (int i = 0; i < e.length(); i++) {
        char c = e.charAt(i);
        // Compte les parenthèses pour gérer les expressions imbriquées
        if (c == '(') parenCount++;
        else if (c == ')') parenCount--;
            // Vérifie si c'est un opérateur en dehors des parenthèses
        else if (parenCount == 0) {
            if (c == '+' || c == '-' || c == '*' || c == '/') {
                // Analyse les sous-expressions gauche et droite
                Expression left = parseExpression(e.substring(0, i));
                Expression right = parseExpression(e.substring( beginIndex: i + 1));
                // Retourne la nouvelle expression basée sur l'opérateur trouvé
                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);
                }
            }
        }
    }
    // Retourne null si l'expression est invalide
    return null;
}
```

```java
    // Méthode principale pour tester le programme
    public static void main(String[] args) {
        // Crée une expression à partir de la chaîne "3"
        Expression expr1 = fabriqueExpression( e: "3");
        // Affiche l'expression et sa valeur
        System.out.println(expr1 + " = " + expr1.valeur()); // affiche 3 = 3

        // Crée une expression à partir de la chaîne "17-2"
        Expression expr2 = fabriqueExpression( e: "17-2");
        // Affiche l'expression et sa valeur
        System.out.println(expr2 + " = " + expr2.valeur()); // affiche (17 - 2) = 15

        // Crée une expression à partir de la chaîne "(17-2)/3"
        Expression expr3 = fabriqueExpression( e: "(17-2)/3");
        // Affiche l'expression et sa valeur
        System.out.println(expr3 + " = " + expr3.valeur()); // affiche ((17 - 2) / 3) = 5
    }
}
```