**Description Document**

Code:  
 Test File: (ExpressionTest.java):

package expressivo;

import static org.junit.Assert.\*;

import org.junit.Test;

/\*\*

 \* Tests for the Expression abstract data type.

 \*/

public class ExpressionTest {

    // Testing strategy:

    //   - Test constants (numbers) and their toString(), equals(), and hashCode().

    //   - Test variables and their toString(), equals(), and hashCode().

    //   - Test addition and multiplication expressions.

    //   - Test equality of expressions with the same value but different structures.

    @Test

    public void testConstantToString() {

        Expression e = new Constant(5.0);

        assertEquals("5.0", e.toString());

    }

    @Test

    public void testVariableToString() {

        Expression e = new Variable("x");

        assertEquals("x", e.toString());

    }

    @Test

    public void testAdditionToString() {

        Expression e = new Addition(new Variable("x"), new Constant(2.0));

        assertEquals("x + 2.0", e.toString());

    }

    @Test

    public void testMultiplicationToString() {

        Expression e = new Multiplication(new Variable("x"), new Constant(3.0));

        assertEquals("x \* 3.0", e.toString());

    }

    @Test

    public void testConstantEquals() {

        Expression e1 = new Constant(5.0);

        Expression e2 = new Constant(5.0);

        assertTrue(e1.equals(e2));

    }

    @Test

    public void testDifferentConstantEquals() {

        Expression e1 = new Constant(5.0);

        Expression e2 = new Constant(10.0);

        assertFalse(e1.equals(e2));

    }

    @Test

    public void testVariableEquals() {

        Expression e1 = new Variable("x");

        Expression e2 = new Variable("x");

        assertTrue(e1.equals(e2));

    }

    @Test

    public void testDifferentVariableEquals() {

        Expression e1 = new Variable("x");

        Expression e2 = new Variable("y");

        assertFalse(e1.equals(e2));

    }

    @Test

    public void testAdditionEquals() {

        Expression e1 = new Addition(new Variable("x"), new Constant(2.0));

        Expression e2 = new Addition(new Variable("x"), new Constant(2.0));

        assertTrue(e1.equals(e2));

    }

    @Test

    public void testDifferentAdditionEquals() {

        Expression e1 = new Addition(new Variable("x"), new Constant(2.0));

        Expression e2 = new Addition(new Variable("y"), new Constant(2.0));

        assertFalse(e1.equals(e2));

    }

    @Test

    public void testMultiplicationEquals() {

        Expression e1 = new Multiplication(new Variable("x"), new Constant(3.0));

        Expression e2 = new Multiplication(new Variable("x"), new Constant(3.0));

        assertTrue(e1.equals(e2));

    }

    @Test

    public void testDifferentMultiplicationEquals() {

        Expression e1 = new Multiplication(new Variable("x"), new Constant(3.0));

        Expression e2 = new Multiplication(new Variable("x"), new Constant(4.0));

        assertFalse(e1.equals(e2));

    }

    @Test

    public void testHashCode() {

        Expression e1 = new Constant(5.0);

        Expression e2 = new Constant(5.0);

        assertTrue(e1.hashCode() == e2.hashCode());

    }

    @Test

    public void testDifferentHashCode() {

        Expression e1 = new Constant(5.0);

        Expression e2 = new Constant(10.0);

        assertFalse(e1.hashCode() == e2.hashCode());

    }

    @Test

    public void testHashCodeEquality() {

        Expression e1 = new Addition(new Variable("x"), new Constant(2.0));

        Expression e2 = new Addition(new Variable("x"), new Constant(2.0));

        assertTrue(e1.hashCode() == e2.hashCode());

    }

    @Test

    public void testHashCodeInequality() {

        Expression e1 = new Addition(new Variable("x"), new Constant(2.0));

        Expression e2 = new Addition(new Variable("y"), new Constant(2.0));

        assertFalse(e1.hashCode() == e2.hashCode());

    }

    // Edge case: testing equality between expressions with different structures

    @Test

    public void testAdditionMultiplicationInequality() {

        Expression e1 = new Addition(new Variable("x"), new Constant(2.0));

        Expression e2 = new Multiplication(new Variable("x"), new Constant(2.0));

        assertFalse(e1.equals(e2));

    }

    // Check for null or different class types

    @Test

    public void testEqualsNull() {

        Expression e = new Constant(5.0);

        assertFalse(e.equals(null));

    }

    @Test

    public void testEqualsDifferentClass() {

        Expression e = new Constant(5.0);

        String str = "5.0";

        assertFalse(e.equals(str));

    }

}

Constant.java:   
package expressivo;

public class Constant implements Expression {

    private final double value;

    /\*\*

     \* Constructs a constant expression with the specified numeric value.

     \*

     \* @param value the numeric value of this constant

     \*/

    public Constant(double value) {

        this.value = value;

    }

    /\*\*

     \* Returns a string representation of this constant, showing its numeric value.

     \*

     \* @return the numeric value as a string

     \*/

    @Override

    public String toString() {

        return Double.toString(value);

    }

    /\*\*

     \* Compares this constant to the specified object for equality.

     \*

     \* @param obj the object to compare to

     \* @return true if the specified object is a Constant with the same value, false otherwise

     \*/

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true; // Same instance

        if (obj == null || getClass() != obj.getClass()) return false; // Null or different class

        Constant constant = (Constant) obj;

        return Double.compare(constant.value, value) == 0; // Compare values

    }

    /\*\*

     \* Returns a hash code for this constant based on its numeric value.

     \*

     \* @return the hash code for this constant

     \*/

    @Override

    public int hashCode() {

        return Double.hashCode(value); // Use Double's hashCode method

    }

}

Addition.java:

package expressivo;

public class Addition implements Expression {

    private final Expression left;

    private final Expression right;

    /\*\*

     \* Constructs an Addition expression with the specified left and right sub-expressions.

     \*

     \* @param left  the left sub-expression

     \* @param right the right sub-expression

     \*/

    public Addition(Expression left, Expression right) {

        this.left = left;

        this.right = right;

    }

    /\*\*

     \* Returns a string representation of this addition in the format "left + right".

     \*

     \* @return a string representation of this addition

     \*/

    @Override

    public String toString() {

        return left.toString() + " + " + right.toString();

    }

    /\*\*

     \* Compares this addition to the specified object for equality.

     \*

     \* @param obj the object to compare to

     \* @return true if the specified object is an Addition with equal sub-expressions, false otherwise

     \*/

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true; // Same instance

        if (obj == null || getClass() != obj.getClass()) return false; // Null or different class

        Addition addition = (Addition) obj;

        return left.equals(addition.left) && right.equals(addition.right); // Compare sub-expressions

    }

    /\*\*

     \* Returns a hash code for this addition based on its sub-expressions.

     \*

     \* @return the hash code for this addition

     \*/

    @Override

    public int hashCode() {

        return 31 \* left.hashCode() + right.hashCode(); // Combine hash codes of sub-expressions

    }

}

Multiplication.java:  
package expressivo;

public class Multiplication implements Expression {

    private final Expression left;

    private final Expression right;

    /\*\*

     \* Constructs a multiplication expression with the given left and right subexpressions.

     \*

     \* @param left  the left operand of the multiplication

     \* @param right the right operand of the multiplication

     \*/

    public Multiplication(Expression left, Expression right) {

        this.left = left;

        this.right = right;

    }

    /\*\*

     \* Returns a string representation of the multiplication in the format "left \* right".

     \*

     \* @return the string representation of the multiplication

     \*/

    @Override

    public String toString() {

        return left.toString() + " \* " + right.toString();

    }

    /\*\*

     \* Compares this multiplication to the specified object for equality.

     \*

     \* @param obj the object to compare to

     \* @return true if the specified object is a Multiplication with equal left and right expressions, false otherwise

     \*/

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true; // Same instance

        if (obj == null || getClass() != obj.getClass()) return false; // Null or different class

        Multiplication multiplication = (Multiplication) obj;

        return left.equals(multiplication.left) && right.equals(multiplication.right); // Compare expressions

    }

    /\*\*

     \* Returns a hash code for this multiplication based on its left and right subexpressions.

     \*

     \* @return the hash code for this multiplication

     \*/

    @Override

    public int hashCode() {

        return 31 \* left.hashCode() + right.hashCode(); // Combine hash codes

    }

}

Variable.java:   
package expressivo;

public class Variable implements Expression {

    private final String name;

    /\*\*

     \* Constructs a variable with the given name.

     \*

     \* @param name the name of the variable

     \*/

    public Variable(String name) {

        this.name = name;

    }

    /\*\*

     \* Returns a string representation of the variable (its name).

     \*

     \* @return the name of the variable as a string

     \*/

    @Override

    public String toString() {

        return name;

    }

    /\*\*

     \* Compares this variable to the specified object for equality.

     \*

     \* @param obj the object to compare to

     \* @return true if the specified object is a Variable with the same name, false otherwise

     \*/

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true; // Same instance

        if (obj == null || getClass() != obj.getClass()) return false; // Null or different class

        Variable variable = (Variable) obj;

        return name.equals(variable.name); // Compare names

    }

    /\*\*

     \* Returns a hash code for this variable based on its name.

     \*

     \* @return the hash code for this variable

     \*/

    @Override

    public int hashCode() {

        return name.hashCode(); // Use the hash code of the name

    }

}

Github Link:

<https://github.com/RyanSikandar/SCLAB>