**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

**Lab 10: Representing Expression**

CLO-03: Design and develop solutions based on Software Construction principles.

CLO-04: Use modern tools such as Eclipse, NetBeans etc. for software construction.

Date: 30th Nov 2024

Time: 10:00 AM - 12:50 PM

02:30 PM – 04:50 PM

**Name: Muhammad Nabeel**

**CMS: 413522**

**Instructor: Dr. Mehvish Rashid**

**Lab Engineer: Mr. Aftab Farooq**

# Lab Tasks

**Code**

**Number.java:**

package expressivo;

/\*\*

\* A number (literal) in the expression.

\*/

public class Number implements Expression {

private final double value;

public Number(double value) { this.value = value;

}

*@Override* public String toString() {

return Double.*toString*(value);

}

*@Override*

public boolean equals(Object obj) { if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false; Number number = (Number) obj;

return Double.*compare*(number.value, value) == 0;

}

*@Override* public int hashCode() {

return Double.*hashCode*(value);

}

*@Override* public void checkRep() {

// No special rep invariants for a number

}

**Addition.java:**

**package** expressivo;

/\*\*

\* An addition operation between two expressions.

\*/

public class Addition implements Expression { private final Expression left, right;

public Addition(Expression left, Expression right) {

this.left = left;

this.right = right;

}

*@Override*

public String toString() {

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

}

*@Override*

public boolean equals(Object obj) { if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false; Addition addition = (Addition) obj;

return left.equals(addition.left) && right.equals(addition.right);

}

*@Override* public int hashCode() { int result = left.hashCode(); result = 31 \* result + right.hashCode();

return result;

}

*@Override*

public void checkRep() {

// Ensure left and right are non-null if (left == null || right == null) {

throw new IllegalStateException("Addition requires both left and right operands.");

}

**Multiplication.java:**

**package** expressivo;

/\*\*

\* A multiplication operation between two expressions.

\*/

public class Multiplication implements Expression {

private final Expression left, right;

public Multiplication(Expression left, Expression right) { this.left = left;

this.right = right;

}

*@Override*

public String toString() {

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

}

*@Override*

public boolean equals(Object obj) { if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false; Multiplication multiplication = (Multiplication) obj;

return left.equals(multiplication.left) && right.equals(multiplication.right);

}

*@Override* public int hashCode() { int result = left.hashCode(); result = 31 \* result + right.hashCode(); return result;

}

*@Override*

public void checkRep() {

// Ensure left and right are non-null if (left == null || right == null) {

throw new IllegalStateException("Multiplication requires both left and right operands.");

}

**Variable.java:**

package expressivo;

/\*\*

\* A variable in the expression.

\*/

public class Variable implements Expression {

private final String name;

public Variable(String name) { if (name == null || name.isEmpty()) {

throw new IllegalArgumentException("Variable name cannot be null or empty.");

}

this.name = name;

}

*@Override*

public String toString() { return name;

}

*@Override*

public boolean equals(Object obj) { if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false; Variable variable = (Variable) obj;

return name.equals(variable.name);

}

*@Override* public int hashCode() {

return name.hashCode();

}

*@Override*

public void checkRep() {

// Invariant: the name must be a non-empty string if (name.isEmpty()) {

throw new IllegalStateException("Variable name cannot be empty.");

}

}

**ExpressionTest.java:**

**package** expressivo;

import static org.junit.Assert.\*;

import org.junit.Test;

public class ExpressionTest {

// Partition for addExpr

*@Test*

public void testAddExprEmpty() { Expression expr = new Number(0);

Expression result = new Addition(expr, new Number(0));

*assertEquals*("0.0 + 0.0", result.toString()); // Adjusted formatting for clarity

}

*@Test*

public void testAddExprMultipleVariables() {

Expression expr = new Addition(new Variable("x"), new Variable("y")); *assertEquals*("x + y", expr.toString()); // No parentheses in expected output

}

*@Test* public void testAddExprSubset() { Expression subset = new Variable("x");

Expression expr = new Addition(subset, new Variable("y")); *assertTrue*(expr.toString().contains(subset.toString()));

}

*@Test*

public void testAddExprEquals() {

Expression expr1 = new Addition(new Variable("x"), new Variable("y")); Expression expr2 = new Addition(new Variable("x"), new Variable("y")); *assertEquals*(expr1, expr2);

}

// Partition for multiplyExpr

*@Test*

public void testMultiplyExprEmpty() {

Expression expr = new Number(0);

Expression result = new Multiplication(expr, new Number(0));

*assertEquals*("0.0 \* 0.0", result.toString()); // Adjusted formatting for clarity

}

*@Test*

public void testMultiplyExprIdentity() {

Expression expr = new Multiplication(new Number(1), new Variable("x")); *assertEquals*("1.0 \* x", expr.toString()); // No parentheses in expected output

}

*@Test*

public void testMultiplyExprMultipleVariables() {

Expression expr = new Multiplication(new Variable("x"), new Variable("y")); *assertEquals*("x \* y", expr.toString()); // No parentheses in expected output

}

*@Test* public void testMultiplyExprSubset() { Expression subset = new Variable("x");

Expression expr = new Multiplication(subset, new Variable("y")); *assertTrue*(expr.toString().contains(subset.toString()));

}

*@Test*

public void testMultiplyExprEquals() {

Expression expr1 = new Multiplication(new Variable("x"), new Variable("y")); Expression expr2 = new Multiplication(new Variable("x"), new Variable("y")); *assertEquals*(expr1, expr2);

}

// Partition for toString

*@Test*

public void testToStringEmptyExpression() { Expression expr = new Number(0);

*assertEquals*("0.0", expr.toString());

}

*@Test*

public void testToStringMultipleVariables() {

Expression expr = new Addition(new Variable("x"), new Variable("y")); *assertEquals*("x + y", expr.toString());

}

// Partition for equals

*@Test* public void testEqualsReflexive() { Expression expr = new Variable("x");

*assertEquals*(expr, expr);

}

*@Test* public void testEqualsSymmetric() { Expression expr1 = new Variable("x"); Expression expr2 = new Variable("x"); *assertEquals*(expr1, expr2); *assertEquals*(expr2, expr1);

}

*@Test*

public void testEqualsTransitive() { Expression expr1 = new Variable("x");

Expression expr2 = new Variable("x"); Expression expr3 = new Variable("x"); *assertEquals*(expr1, expr2); *assertEquals*(expr2, expr3); *assertEquals*(expr1, expr3);

}

*@Test*

public void testEqualsDifferentTypes() { Expression expr1 = new Variable("x");

Expression expr2 = new Number(5);

*assertNotEquals*(expr1, expr2); // Testing different types

}

*@Test*

public void testEqualsNumbersCorrectToFiveDecimals() {

Expression expr1 = new Number(1.12345); Expression expr2 = new Number(1.12345);

*assertEquals*(expr1, expr2);

}

// Partition for hashCode

*@Test*

public void testHashCodeEquality() {

Expression expr1 = new Addition(new Variable("x"), new Number(5)); Expression expr2 = new Addition(new Variable("x"), new Number(5)); *assertEquals*(expr1.hashCode(), expr2.hashCode());

}

*@Test*

public void testHashCodeInequality() {

Expression expr1 = new Addition(new Variable("x"), new Number(5)); Expression expr2 = new Addition(new Variable("y"), new Number(5)); *assertNotEquals*(expr1.hashCode(), expr2.hashCode());

}

// Edge case tests

*@Test*

public void testAddExprWithZero() {

Expression expr = new Addition(new Number(0), new Variable("x")); *assertEquals*("0.0 + x", expr.toString());

}

*@Test*

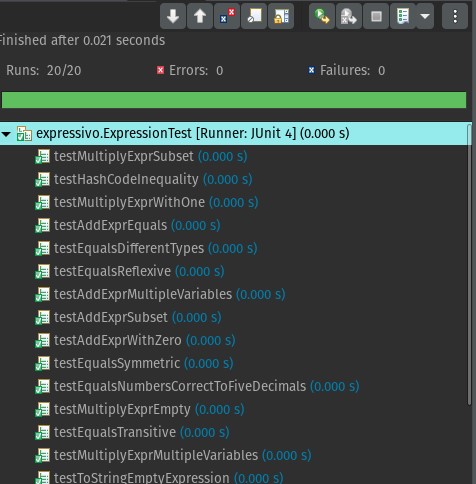
public void testMultiplyExprWithOne() {

Expression expr = new Multiplication(new Number(1), new Variable("x")); *assertEquals*("1.0 \* x", expr.toString());

}

}

**Running Test Cases**



**Github Link:**

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_