**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 SAAD UMER |
| **CLASS** | BESE 13-A |
| **CMS** | 408485 |

**Instructor: Dr. Mehvish Rashid  
Lab Engineer: Mr. Aftab Farooq**

**Introduction:**

# Lab 10: Representing Expressions

Students will have hands-on experience of representing expressions.

Material:

https://ocw.mit.edu/ans7870/6/6.005/s16/psets/ps3/

**Lab Tasks**

Solve problem 1 of problem set 3 listed on the link.

Define an immutable, recursive abstract data type to represent expressions as abstract syntax trees.

Your AST should be defined in the provided Expression interface (in Expression.java ) and implemented by several concrete variants, one for each kind of expression. Each variant should be defined in its own appropriately-

named .java file.Concrete syntax in the input, such as parentheses and whitespace, should not be represented at all in your AST.

## Expression

To repeat, your data type must be **immutable** and **recursive** . Follow the recipe for creating an ADT:

* + - **Spec** . Choose and specify operations. For this part of the problem set, the only operations Expression needs are creators and producers for building up an expression, plus the standard observers toString() , equals() , and hashCode() . We are strengthening the specs for these standard methods; see below.
    - **Test** . Partition and test your operations in ExpressionTest.java , including tests for toString() , equals() , and hashCode() . Note that we will not run your test cases on other implementations, just on yours.
    - **Code** . Write the rep for your Expression as a data type definition in a comment inside Expression . Implement the variant classes of your data type.

Remember to include a Javadoc comment above every class and every method you write; define abstraction functions and rep invariants, and write checkRep; and document safety from rep exposure.

## toString()

Define the toString() operation on Expression so it can output itself as a string. This string must be a valid expression as defined above. You have the freedom to decide how to format the output with whitespace and parentheses for readability, but the expression must have the same mathematical meaning.

Your toString()implementation must be recursive, and must not use instanceof . Use the @Override annotation to ensure you are overriding the toString() inherited from Object .

Remember that your tests must obey the spec. If your toString() tests expect a certain formatting of whitespace and parentheses, you must specify this formatting in your spec.

## equals() and hashCode()

Define the equals() and hashCode() operations on your AST to implement

*structural equality* .

**Structural equality** defines two expressions to be equal if:

1. the expressions contain the same variables, numbers, and operators;
2. those variables, numbers, and operators are in the same order, read left-to-right; c and they are grouped in the same way.

For example, the AST for 1 + x is *not* equal to the AST for x + 1 , but it is equal to the ASTs for 1+x , (1+x) , and (1)+(x) .

For *n* -ary groupings where *n* is greater than 2:

* Such expressions must be equal to themselves. For example, the ASTs for 3 + 4 + 5 and (3 + 4 + 5) must be equal.
* However, whether they are equal or not to different groupings with the same mathematical meaning is *not specified* , and you should choose an appropriate specification and implementation for your AST. For example, you must determine whether the ASTs for (3 + 4) + 5 and 3 + (4 + 5) are equal.

For equality of numbers, you have the freedom to choose a reasonable definition. Integers that can be represented exactly as a double should be considered equal. For example, the ASTs for x + 1 and x + 1.000 must be equal.

Remember: concrete syntax, including parentheses, should not be represented in your AST. Grouping, for example, should be reflected in the AST’s structure.

Be sure that AST instances which are considered equal according to this definition and according to equals() also satisfy observational equality .

Your equals() and hashCode() implementations must be recursive. Only

equals() can use instanceof , and hashCode() must not. Remember to use the @Override annotation.

|  |
| --- |
| SPECS AND CODE |
| package expressivo;import java.util.Objects;/\*\*\* Represents a numeric constant in an expression.\*/public class Number implements Expression {private final double value;public Number(double value) {this.value = value;}*@Override*public String toString() {return String.*valueOf*(value);}*@Override*public boolean equals(Object obj) {if (this == obj) return true;if (!(obj instanceof Number)) return false;Number that = (Number) obj;return Double.*compare*(this.value, that.value) == 0;}*@Override*public int hashCode() {return Objects.*hash*(value);}}package expressivo;import java.util.Objects;/\*\*\* Represents the product of two expressions.\*/public class Product implements Expression {private final Expression left;private final Expression right;public Product(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 instanceof Product)) return false;Product that = (Product) obj;return this.left.equals(that.left) && this.right.equals(that.right);}*@Override*public int hashCode() {return Objects.*hash*(left, right);}}package expressivo;import java.util.Objects;/\*\*\* Represents the sum of two expressions.\*/public class Sum implements Expression {private final Expression left;private final Expression right;public Sum(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 instanceof Sum)) return false;Sum that = (Sum) obj;return this.left.equals(that.left) && this.right.equals(that.right);}*@Override*public int hashCode() {return Objects.*hash*(left, right);}}package expressivo;import java.util.Objects;/\*\*\* Represents a variable in an expression.\*/public class Variable implements Expression {private final String name;public Variable(String name) {this.name = name;}*@Override*public String toString() {return name;}*@Override*public boolean equals(Object obj) {if (this == obj) return true;if (!(obj instanceof Variable)) return false;Variable that = (Variable) obj;return this.name.equals(that.name);}*@Override*public int hashCode() {return Objects.*hash*(name);}} |
| TESTING |
| package expressivo;import static org.junit.Assert.\*;import org.junit.Test;public class ExpressionTest {@Testpublic void testNumber() {Expression num = new Number(5.0);assertEquals("5.0", num.toString());assertEquals(new Number(5.0), num);assertEquals(new Number(5), num);}@Testpublic void testVariable() {Expression var = new Variable("x");assertEquals("x", var.toString());assertEquals(new Variable("x"), var);}@Testpublic void testSum() {Expression sum = new Sum(new Number(1), new Variable("x"));assertEquals("(1.0 + x)", sum.toString());assertEquals(new Sum(new Number(1), new Variable("x")), sum);}@Testpublic void testProduct() {Expression product = new Product(new Number(3), new Variable("y"));assertEquals("(3.0 \* y)", product.toString());assertEquals(new Product(new Number(3), new Variable("y")), product);}@Testpublic void testComplexExpression() {Expression expr = new Sum(new Product(new Number(2), new Variable("x")),new Number(3));assertEquals("((2.0 \* x) + 3.0)", expr.toString());}} |

## Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to [aftab.farooq@seecs.edu.pk.](mailto:aftab.farooq@seecs.edu.pk.)