Assignment (1)

Write an object-oriented program:

Expression Calculator

It would take an expression from the input, and produce the result of calculations.

Tasks:

1. 1st week:
   Design the class hierarchy representing syntax for expressions.
2. 1st or 2nd week:
   Develop functionality for parsing expressions & building AST.
3. 2nd or 3rd week:
   Implement expression calculation as tree traverser.

Assignment (2)

Expression syntax

- Only three relational operators
- Only three arithmetic operators

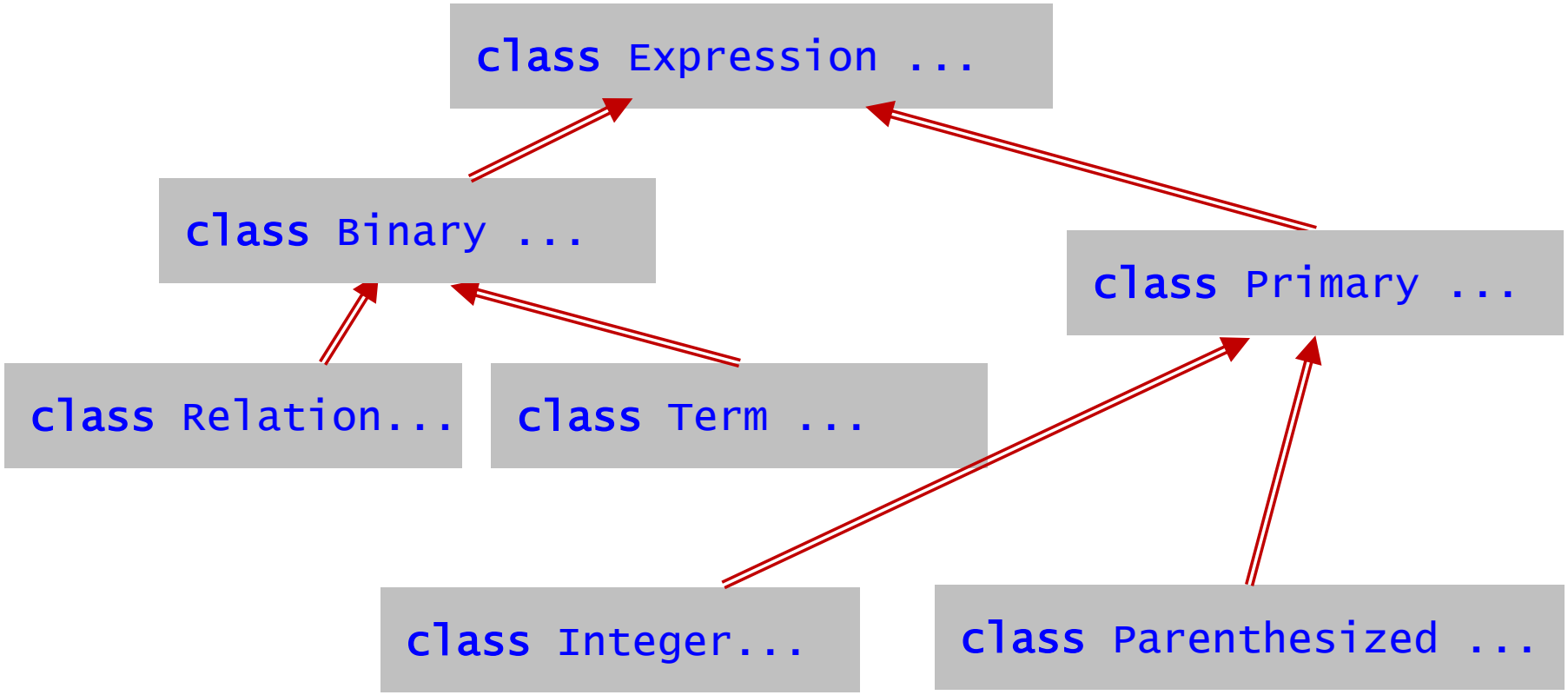
```
expression -> relation

relation -> term [ ( "<" | ">" | "=" ) term ]

term     -> factor { ( "+" | "-" ) factor }

factor   -> primary { "*" primary }

primary  -> integer | "(" expression ")"

integer  -> Any integer number (literal constant)
```

Metasymbols:

[ ]	group optional elements (repeated 0 or 1 time)
{ }	group elements that can be repeated 0 or more times
|	denotes alternatives
( )	simply group alternatives

Assignment (3)

AST Hierarchy



Assignment (4)

AST Implementation (proposed)

```
class Expression {
    // Empty for a while
}

class Binary extends Expression
{
    Expression left;
    Expression right;
}

class Relation extends Binary { ... }
    class Less extends Relation { ... }

...
```

Remark:

Let result of relational operators always be **integers**:

- Op1 < Op2 produces 1 if Op1 is less than Op2, and 0 otherwise.
- The same about other relational operators.

Assignment (5)

AST Implementation (proposed, contd)

```
...

class Primary extends Expression
{
}

class Integer extends Primary
{
    private long value;
    ...
}

class Parenthesized extends Primary
{
    private Expression expression;
    ...
}
```

Assignment (6)

Parser Implementation (proposed)

```
class Parser
{
    private String input;
    public Parser(String s) { input = s; }

    public Expression parse() { return parseRelation(); }

    private Expression parseRelation()
    {
        Expession result = parseTerm();
        while ( true ) {
            String op = getNext(); // takes the next token
            Expression right = parseTerm();
            if ( op == "<" )
                result = new Less(result,right);
            else if ( op == ">" )
                result = new Greater(result,right);
            else
                break;
        }
        return result;
    }
    ...
```

Assignment (7)

Parser Implementation (contd, proposed)

```
class Parser
{
    ...
    private Primary parsePrimary()
    {
        Primary result = null;
        if ( Char.IsDigit(nextChar()) )
            result = parseInteger();
        else if ( nextChar() == '(' ) {
            result = parse();
            skipNextChar(); // skip ‘)’
        }
        else
        { ... } // error
        return result;
    }
    ...
}
```

Assignment (8)

The Main Function (proposed)

```
class Program
{
    public static void main()
    {
        String input = ReadLine(); // 1+(26-98)/15+777<28
        Parser parser = new Parser(input);
        Expression expressionTree = parser.parse();
        long result = expressionTree.calculate();
    }
}
```