КИЇВСЬКИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

**Лабораторна робота №7**

З інженерії програмного забезпечення

**Тема**: «Шаблони поведінки. Шаблони Memento, State, Command та Interpreter»

залікова книжка № 3223

Виконав:

студент ІІ курсу ФІОТ

група ІО-32

Попенко Руслан

Перевірив:

Антонюк А. І.

Київ-2014

**Мета:** Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Memento, State, Command та Interpreter.

**1. Завдання.**

1. Повторити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.

2. Детально вивчити шаблони поведінки для проектування ПЗ - Memento, State, Command та Interpreter. Для кожного з них:

вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;

знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;

вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;

вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.

3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork7. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи. Приклад реалізації бізнес-методу:

void draw(int x, int y){

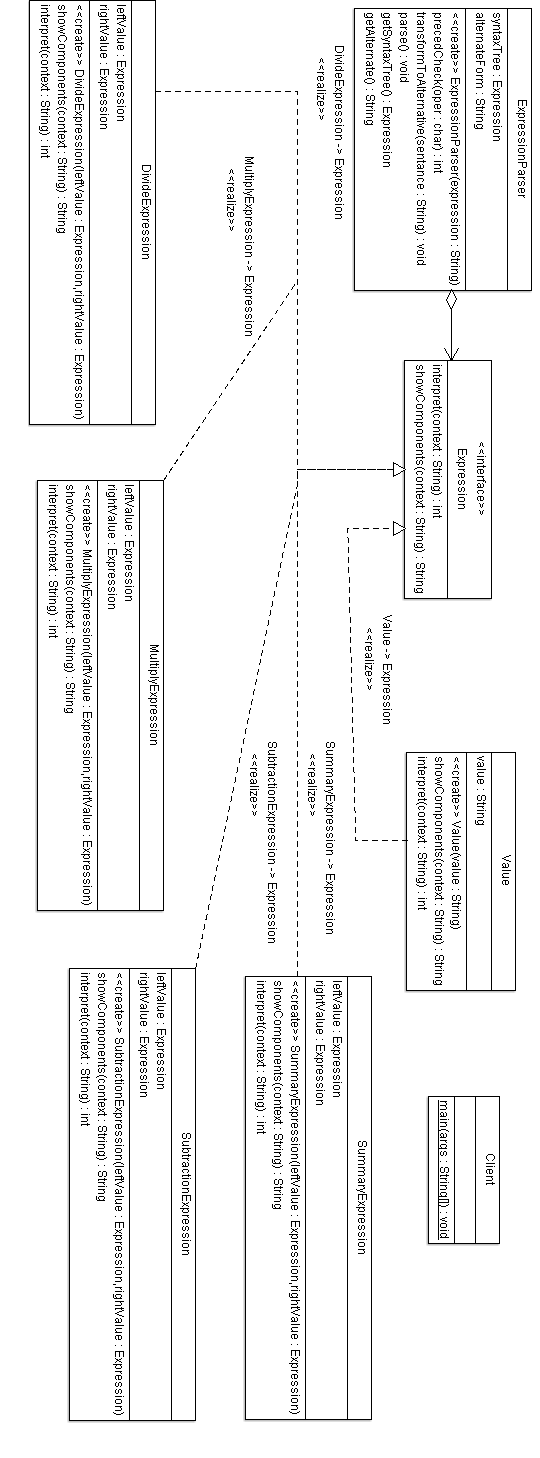
System.out.println(“Метод draw з параметрами x=”+x+” y=”+y);}

4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

**Мій варіант 7**

*Визначити специфікації класів для розбору алгебраїчних виразів з операціями +, -, /, \*.*

1. **Діаграма класів**



1. **JavaDoc**

## C

****[Client](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\Client.html" \o "class in ua.kpi.ipz.lab7)**** - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

A client-class that sets a mathematical expression in a traditional text-based form.

[**Client()**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Client.html#Client()) - Constructor for class ua.kpi.ipz.lab7.[Client](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Client.html)

## D

****[DivideExpression](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\DivideExpression.html" \o "class in ua.kpi.ipz.lab7)**** - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

Class describes the grammar rule DivideExpression::=Expression '/' Expression; The structure of the class is much alike the composite implementation.

[**DivideExpression(Expression, Expression)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\DivideExpression.html#DivideExpression(ua.kpi.ipz.lab7.Expression,%20ua.kpi.ipz.lab7.Expression)) - Constructor for class ua.kpi.ipz.lab7.[DivideExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\DivideExpression.html)

Constructor of the class, sets all the fields of it.

## E

****[Expression](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\Expression.html" \o "interface in ua.kpi.ipz.lab7)**** - Interface in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

An interface that describes operations that have to be implemented by other classes which represent grammar-rules.

[****ExpressionParser****](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\ExpressionParser.html) - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

A parser and analyser that gives a possibility to rewrite a given string in a post-fix form (instead of traditional infix-one).

[**ExpressionParser(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\ExpressionParser.html#ExpressionParser(java.lang.String)) - Constructor for class ua.kpi.ipz.lab7.[ExpressionParser](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\ExpressionParser.html)

Constructor of ExpressionParser.

## G

**[getAlternate()](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\ExpressionParser.html" \l "getAlternate())** - Method in class ua.kpi.ipz.lab7.[ExpressionParser](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\ExpressionParser.html)

Method returns an alternative form of a given in text-form expression

[**getSyntaxTree()**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\ExpressionParser.html#getSyntaxTree()) - Method in class ua.kpi.ipz.lab7.[ExpressionParser](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\ExpressionParser.html)

Method returns the syntax Tree that is kept as a field of class.

## I

**[interpret(String)](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\DivideExpression.html" \l "interpret(java.lang.String))** - Method in class ua.kpi.ipz.lab7.[DivideExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\DivideExpression.html)

The method interprets the given math-expression in a specific way.

[**interpret(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Expression.html#interpret(java.lang.String)) - Method in interface ua.kpi.ipz.lab7.[Expression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Expression.html)

A method that sets rules of interpreting the sentence.

[**interpret(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html#interpret(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[MultiplyExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html)

The method interprets the given math-expression in a specific way.

[**interpret(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html#interpret(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[SubtractionExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html)

The method interprets the given math-expression in a specific way.

[**interpret(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html#interpret(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[SummaryExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html)

The method interprets the given math-expression in a specific way.

[**interpret(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Value.html#interpret(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[Value](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Value.html)

An interpreter that returns an int-representation of a number given in a text-form.

## M

**[main(String[])](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\Client.html" \l "main(java.lang.String[]))** - Static method in class ua.kpi.ipz.lab7.[Client](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Client.html)

[****MultiplyExpression****](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html) - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

Class describes the grammar rule MultiplyExpression::=Expression '\*' Expression; The structure of the class is much alike the composite implementation.

[**MultiplyExpression(Expression, Expression)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html#MultiplyExpression(ua.kpi.ipz.lab7.Expression,%20ua.kpi.ipz.lab7.Expression)) - Constructor for class ua.kpi.ipz.lab7.[MultiplyExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html)

Constructor of the class, sets all the fields of it.

## S

**[showComponents(String)](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\DivideExpression.html" \l "showComponents(java.lang.String))** - Method in class ua.kpi.ipz.lab7.[DivideExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\DivideExpression.html)

Method prints nodes of the abstract syntax tree (in order from top to the bottom.

[**showComponents(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Expression.html#showComponents(java.lang.String)) - Method in interface ua.kpi.ipz.lab7.[Expression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Expression.html)

A method shows the components of sentence that has to be interpreted.

[**showComponents(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html#showComponents(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[MultiplyExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\MultiplyExpression.html)

Method prints nodes of the abstract syntax tree (in order from top to the bottom.

[**showComponents(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html#showComponents(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[SubtractionExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html)

Method prints nodes of the abstract syntax tree (in order from top to the bottom.

[**showComponents(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html#showComponents(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[SummaryExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html)

Method prints nodes of the abstract syntax tree (in order from top to the bottom.

[**showComponents(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Value.html#showComponents(java.lang.String)) - Method in class ua.kpi.ipz.lab7.[Value](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Value.html)

Method prints nodes of the abstract syntax tree (in order from top to the bottom.

[****SubtractionExpression****](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html) - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

Class describes the grammar rule SubtractionExpression::=Expression '-' Expression; The structure of the class is much alike the composite implementation.

[**SubtractionExpression(Expression, Expression)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html#SubtractionExpression(ua.kpi.ipz.lab7.Expression,%20ua.kpi.ipz.lab7.Expression)) - Constructor for class ua.kpi.ipz.lab7.[SubtractionExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SubtractionExpression.html)

Constructor of the class, sets all the fields of it.

[****SummaryExpression****](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html) - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

Class describes the grammar rule SummaryExpression::=Expression '+' Expression; The structure of the class is much alike the composite implementation.

[**SummaryExpression(Expression, Expression)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html#SummaryExpression(ua.kpi.ipz.lab7.Expression,%20ua.kpi.ipz.lab7.Expression)) - Constructor for class ua.kpi.ipz.lab7.[SummaryExpression](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\SummaryExpression.html)

Constructor of the class, sets all the fields of it.

## U

[ua.kpi.ipz.lab7](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\package-summary.html) - package ua.kpi.ipz.lab7

## V

****[Value](file:///C:\\Users\\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\\Desktop\\%D0%B8%D0%BF%D0%B7\\JavaDoc7.7\\ua\\kpi\\ipz\\lab7\\Value.html" \o "class in ua.kpi.ipz.lab7)**** - Class in [ua.kpi.ipz.lab7](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\package-summary.html)

Class describes the grammar rule Value::='1'|'2'|'3'|...; It's a simple leaf-class which only keeps and returns an integer-representation of numbers, given in a text-form as atoms of expressions.

[**Value(String)**](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Value.html#Value(java.lang.String)) - Constructor for class ua.kpi.ipz.lab7.[Value](file:///C:\Users\%D0%84%D0%B2%D0%B3%D0%B5%D0%BD%D1%96%D0%B9\Desktop\%D0%B8%D0%BF%D0%B7\JavaDoc7.7\ua\kpi\ipz\lab7\Value.html)

Constructor of the class, gets value of atom in expression.

1. **Роздруківка тексту**

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

/\*\*

\* A client-class that sets a mathematical expression in a traditional

\* text-based form. An expression has to be interpreted in the specific for

\* counting machine form and find the result of math. expression. Abstract

\* syntax-tree can be set either manually by client or automatically built by

\* parser.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*

\*/

public class Client {

public static void main(String[] args) {

String exp = new String("1+a\*(8-b)+4/2");

String exp1 = new String("a+b/3+7\*6+3+4\*2-1");

String context = new String("a=7, b=3");

ExpressionParser p = new ExpressionParser(exp);

ExpressionParser p1 = new ExpressionParser(exp1);

Expression syntaxTree = p.getSyntaxTree();

Expression syntaxTree1 = p1.getSyntaxTree();

int res = syntaxTree.interpret(context);

int res1 = syntaxTree1.interpret(context);

String res2 = syntaxTree.showComponents(context);

System.out.println("1+7\*(8-3)+4/2=" + res);

System.out.println("7+3/3+7\*6+3+4\*2-1=" + res1);

System.out.println();

System.out.println("Components of expression (with arguments):");

System.out.println(res2);

}

}

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

/\*\*

\* Class describes the grammar rule DivideExpression::=Expression '/'

\* Expression; The structure of the class is much alike the composite

\* implementation. But, unlike composite, this class mainly describes behaviour.

\* The aim is to divide expressions. Class represents an implementation of

\* Expression and contains some aggregated Expressions inside.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*

\*/

public class DivideExpression implements Expression {

private Expression leftValue;

private Expression rightValue;

/\*\*

\* Constructor of the class, sets all the fields of it.

\*

\* @param leftValue

\* the value of expression on the left side from operation.

\* @param rightValue

\* the value of expression on the right side from operation.

\*/

public DivideExpression(Expression leftValue, Expression rightValue) {

this.leftValue = leftValue;

this.rightValue = rightValue;

}

/\*\*

\* Method prints nodes of the abstract syntax tree (in order from top to the

\* bottom.

\*

\* @return info the components of the tree.

\*/

public String showComponents(String context) {

return new String("DivideExpression(" + leftValue.interpret(context)

+ "/" + rightValue.interpret(context) + "), "

+ leftValue.showComponents(context)

+ rightValue.showComponents(context));

}

/\*\*

\* The method interprets the given math-expression in a specific way.

\* Divides both expressions. But first, expressions are recursively

\* interpreted too.

\*/

public int interpret(String context) {

return leftValue.interpret(context) / rightValue.interpret(context);

}

}

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

/\*\*

\* An interface that describes operations that have to be implemented by other

\* classes which represent grammar-rules.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*

\*/

public interface Expression {

/\*\*

\* A method that sets rules of interpreting the sentence.

\*

\* @return int value of the counted expression

\*/

public int interpret(String context);

/\*\*

\* A method shows the components of sentence that has to be interpreted.

\*

\* @return String components

\*/

public String showComponents(String context);

}

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

import java.util.Stack;

/\*\*

\* A parser and analyser that gives a possibility to rewrite a given string in a

\* post-fix form (instead of traditional infix-one). This lets us easily handle

\* difficult operations with parenthesis and operations with different priority.

\* And finally, the class generates a syntax-tree. Syntax-tree is a

\* composite-element with a difficult structure.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*/

public class ExpressionParser {

private Expression syntaxTree;

private String alternateForm;

/\*\*

\* Constructor of ExpressionParser. Allows to hide details of implementation

\* from client. Creates

\*

\* @param expression

\*/

public ExpressionParser(String expression) {

this.transformToAlternative(expression);

this.parse();

}

/\*\*

\* The method checks the argument and returns the value that responds to

\* this argument. The higher the priority is, the higher is as a number.

\*

\* @param oper

\* character, symbol of operation.

\* @return int value of priority.

\*/

private int precedCheck(char oper) {

switch (oper) {

case ('\*'):

return 2;

case ('/'):

return 2;

case ('+'):

return 1;

case ('-'):

return 1;

}

return 0;

}

/\*\*

\* Method interprets the given string from the infix form to the postfix.

\* Uses the shunting-yard algorithm by Diykstra.

\*

\* @param sentance

\* given string with expr. in infix form

\*/

private void transformToAlternative(String sentance) {

Stack<String> stack = new Stack<String>();

StringBuilder build = new StringBuilder();

for (int i = 0; i < sentance.length(); i++) {

char temp = sentance.charAt(i);

boolean expr = (temp != '(') && (temp != ')') && (temp != '+')

&& (temp != '-') && (temp != '\*') && (temp != '/');

if (expr == true) {

build.append(sentance.charAt(i));

build.append(" ");

} else {

if (stack.isEmpty())

stack.push(String.valueOf(temp));

else {

String buf = stack.peek();

if ((temp != '(')

&& (temp != ')')

&& (this.precedCheck(buf.charAt(0)) > this

.precedCheck(temp))) {

build.append(stack.pop());

build.append(" ");

stack.push(String.valueOf(temp));

}

else if ((temp != '(')

&& (temp != ')')

&& (this.precedCheck(buf.charAt(0)) < this

.precedCheck(temp))) {

stack.push(String.valueOf(temp));

}

else if (temp == '(') {

stack.push(String.valueOf(temp));

}

else if (temp == ')') {

while (stack.peek().compareTo("(") != 0) {

build.append(stack.pop());

build.append(" ");

}

stack.pop();

}

else

stack.push(String.valueOf(temp));

}

}

}

if (!stack.isEmpty()) {

while (!stack.isEmpty()) {

build.append(stack.pop());

build.append(" ");

}

}

alternateForm = build.toString();

}

/\*\*

\* Parses an expression in postfix-form to the abstract syntax-tree.

\*/

private void parse() {

Stack<Expression> stack = new Stack<Expression>();

for (String token : alternateForm.split(" ")) {

if (token.equals("+")) {

Expression subExpr = new SummaryExpression(stack.pop(),

stack.pop());

stack.push(subExpr);

}

else if (token.equals("-")) {

Expression rightExpr = stack.pop();

Expression leftExpr = stack.pop();

Expression subExpr = new SubtractionExpression(leftExpr,

rightExpr);

stack.push(subExpr);

}

else if (token.equals("\*")) {

Expression subExpr = new MultiplyExpression(stack.pop(),

stack.pop());

stack.push(subExpr);

}

else if (token.equals("/")) {

Expression rightExpr = stack.pop();

Expression leftExpr = stack.pop();

Expression subExpr = new DivideExpression(leftExpr, rightExpr);

stack.push(subExpr);

} else {

Expression num = new Value(token);

stack.push(num);

}

}

syntaxTree = stack.pop();

}

/\*\*

\* Method returns the syntax Tree that is kept as a field of class.

\*

\* @return syntaxTree abstract syntax tree.

\*/

public Expression getSyntaxTree() {

return syntaxTree;

}

/\*\*

\* Method returns an alternative form of a given in text-form expression

\*

\* @return alternateForm alternative form of given text expression.

\*/

public String getAlternate() {

return alternateForm;

}

}

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

/\*\*

\* Class describes the grammar rule MultiplyExpression::=Expression '\*'

\* Expression; The structure of the class is much alike the composite

\* implementation. But, unlike composite, this class mainly describes behaviour.

\* The aim is to multiply expressions. Class represents an implementation of

\* Expression and contains some aggregated Expressions inside.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*

\*/

public class MultiplyExpression implements Expression {

private Expression leftValue;

private Expression rightValue;

/\*\*

\* Constructor of the class, sets all the fields of it.

\*

\* @param leftValue

\* the value of expression on the left side from operation.

\* @param rightValue

\* the value of expression on the right side from operation.

\*/

public MultiplyExpression(Expression leftValue, Expression rightValue) {

this.leftValue = leftValue;

this.rightValue = rightValue;

}

/\*\*

\* Method prints nodes of the abstract syntax tree (in order from top to the

\* bottom.

\*

\* @return info the components of the tree.

\*/

public String showComponents(String context) {

return new String("MultiplyExpression(" + leftValue.interpret(context)

+ "\*" + rightValue.interpret(context) + "), "

+ leftValue.showComponents(context)

+ rightValue.showComponents(context));

}

/\*\*

\* The method interprets the given math-expression in a specific way.

\* Multiplies both expressions. But first, expressions are recursively

\* interpreted too.

\*/

public int interpret(String context) {

return leftValue.interpret(context) \* rightValue.interpret(context);

}

}

/\*\*

\*

\*/

**package** ua.kpi.ipz.lab7;

/\*\*

\* Class describes the grammar rule SubtractionExpression::=Expression '-'

\* Expression; The structure of the class is much alike the composite

\* implementation. But, unlike composite, this class mainly describes behaviour.

\* The aim is to subtract expressions. Class represents an implementation of

\* Expression and contains some aggregated Expressions inside.

\*

\* **@author** Ruslan Popenko

\* **@version** 7.7

\*

\*/

**public** **class** SubtractionExpression **implements** Expression {

**private** Expression leftValue;

**private** Expression rightValue;

/\*\*

\* Constructor of the class, sets all the fields of it.

\*

\* **@param** leftValue

\* the value of expression on the left side from operation.

\* **@param** rightValue

\* the value of expression on the right side from operation.

\*/

**public** SubtractionExpression(Expression leftValue, Expression rightValue) {

**this**.leftValue = leftValue;

**this**.rightValue = rightValue;

}

/\*\*

\* Method prints nodes of the abstract syntax tree (in order from top to the

\* bottom.

\*

\* **@return** info the components of the tree.

\*/

**public** String showComponents(String context) {

**return** **new** String("SubtractionExpression("

+ leftValue.interpret(context) + "-"

+ rightValue.interpret(context) + "), "

+ leftValue.showComponents(context)

+ rightValue.showComponents(context));

}

/\*\*

\* The method interprets the given math-expression in a specific way.

\* Subtracts both expressions. But first, expressions are recursively

\* interpreted too.

\*/

**public** **int** interpret(String context) {

**return** leftValue.interpret(context) - rightValue.interpret(context);

}

}

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

/\*\*

\* Class describes the grammar rule SummaryExpression::=Expression '+'

\* Expression; The structure of the class is much alike the composite

\* implementation. But, unlike composite, this class mainly describes behaviour.

\* The aim is to summarize expressions. Class represents an implementation of

\* Expression and contains some aggregated Expressions inside.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*

\*/

public class SummaryExpression implements Expression {

private Expression leftValue;

private Expression rightValue;

/\*\*

\* Constructor of the class, sets all the fields of it.

\*

\* @param leftValue

\* the value of expression on the left side from operation.

\* @param rightValue

\* the value of expression on the right side from operation.

\*/

public SummaryExpression(Expression leftValue, Expression rightValue) {

this.leftValue = leftValue;

this.rightValue = rightValue;

}

/\*\*

\* Method prints nodes of the abstract syntax tree (in order from top to the

\* bottom.

\*

\* @return info the components of the tree.

\*/

public String showComponents(String context) {

return new String("SummaryExpression(" + leftValue.interpret(context)

+ "+" + rightValue.interpret(context) + "), "

+ leftValue.showComponents(context)

+ rightValue.showComponents(context));

}

/\*\*

\* The method interprets the given math-expression in a specific way.

\* Summarizes both expressions. But first, expressions are recursively

\* interpreted too.

\*/

public int interpret(String context) {

return leftValue.interpret(context) + rightValue.interpret(context);

}

}

/\*\*

\*

\*/

package ua.kpi.ipz.lab7;

/\*\*

\* Class describes the grammar rule Value::='1'|'2'|'3'|...; It's a simple

\* leaf-class which only keeps and returns an integer-representation of numbers,

\* given in a text-form as atoms of expressions. Class represents an

\* implementation of Expression.

\*

\* @author Ruslan Popenko

\* @version 7.7

\*

\*/

public class Value implements Expression {

private String value;

/\*\*

\* Constructor of the class, gets value of atom in expression.

\*

\* @param value

\* value of operand

\*/

public Value(String value) {

this.value = value;

}

/\*\*

\* Method prints nodes of the abstract syntax tree (in order from top to the

\* bottom.

\*

\* @return info the components of the tree.

\*/

public String showComponents(String context) {

if (value.equals("a")) {

int i = 0;

while (context.charAt(i) != 'a')

i++;

value = String.valueOf(context.charAt(i + 2));

}

if (value.equals("b")) {

int i = 0;

while (context.charAt(i) != 'b')

i++;

value = String.valueOf(context.charAt(i + 2));

}

return new String("number(" + value + "),");

}

/\*\*

\* An interpreter that returns an int-representation of a number given in a

\* text-form. So that we can find out the result of a math.expression.

\*/

public int interpret(String context) {

if (value.equals("a")) {

int i = 0;

while (context.charAt(i) != 'a')

i++;

value = String.valueOf(context.charAt(i + 2));

}

if (value.equals("b")) {

int i = 0;

while (context.charAt(i) != 'b')

i++;

value = String.valueOf(context.charAt(i + 2));

}

return (int) Integer.parseInt(value);

}

}+