УДК 004.415.3
Пех П.А., Корець Р.С.
Луцький національний технічний університет

РЕАЛІЗАЦІЯ ОСНОВНОЇ ФУНКЦІЇ КАЛЬКУЛЯТОРА - ОБЧИСЛЕННЯ ВИРАЗІВ - ЗА ДОПОМОГОЮ JAVA-КЛАСІВ

Пех П.А., Корець Р.С. Реалізація основної функції калькулятора – обчислення виразів – за допомогою Java-класів. В статті запропоновано один із можливих шляхів реалізації основної функції калькулятора – обчислення виразів – за допомогою Java-класів. Наведені програмні коди запропонованих класів.
Ключові слова: Java-клас, калькулятор, обчислення виразів, функція класу, типи даних

Пех Петр Антонович, Корец Роман Сергеевич. Реализация основной функции калькулятора – вычисление выражений – с помощью Java-классов. В статье предложен один из возможных путей реализации основной функции калькулятора – вычисление выражений – с помощью Java-классов. Приведены программные коды предложенных классов.
Ключевые слова: Java-класс, калькулятор, вычисление выражений, функция класса, типы данных

Pekh Petro, Korets Roman. Implementation of the calculator basic function - the calculation of expressions - by means of Java classes. In the article is considered one of the possible ways of realizing the calculator basic function - the calculation of expressions - by means of Java classes. The program codes of the proposed classes are given.
Keywords: Java class, calculator, calculation of expressions, function of class, data types

Постановка задачі. Загальновідомо, що сучасний калькулятор повинен:

— виконувати базові арифметичні операції, а саме – додавання, віднімання, множення та ділення, що містяться в обчислюваному виразі;
— обчислювати математичні функції;
— будувати графіки функцій, заданих арифметичними виразами та, можливо, мати інші функції.

У даній статті розглядається лише питання обчислення арифметичних виразів. Звичайно, калькулятор повинен обчислювати вираз у правильному порядку з урахуванням пріоритету операцій: у першу чергу обчислювати вирази в дужках, далі виконувати операції ділення та множення, а потім – операції додавання та віднімання.

Метою дослідження було розроблення засобами мови `Java` класів для реалізації основної функції калькулятора – обчислення виразів.

Новизна дослідження полягає у підході до вирішення проблеми з позицій об'єктно-орієнтованого програмування [1,2].

Основна частина. Оскільки калькулятор призначений для обчислення складних виразів, що передбачають використання дужок, то необхідно створити клас `QuoteAnalyser`, який реалізує функцію роботи з ними. Перш за все необхідно створити функцію, яка визначає і повертає позиції дужок, в межах яких потрібно проводити обчислення. Оскільки введений користувачем вираз буде мати стрічковий тип `String`, то для вирішення даної задачі зручно скористатись регулярними виразами.

Далі наведено реалізацію функції `getCoupleOfQoutes()` класу `QuoteAnalyser`, що повертає пару дужок, в межах яких будуть проводитись обчислення.

```
static int[] getCouplesOfQuotes(String Example){
   Pattern pattern = Pattern.compile("\\([0-9a-zA-Z\\-*/![^()]]*\\)");
   Matcher matcher = pattern.matcher(Example);
    int BeginOfExpression;
    int EndOfExpression;
    int[] Quotes = new int[2];
    if(matcher.find()){
        String Expression = matcher.group(0);
        BeginOfExpression = Example.indexOf(Expression);
        EndOfExpression = BeginOfExpression + Expression.length() - 1;
    } else return null;
    Quotes[0] = BeginOfExpression;
    Quotes[1] = EndOfExpression;
    return Quotes;
}
```

Дана функція приймає введений користувачем вираз, тобто об'єкт типу `String`. У самій функції створюється шаблон `\\([0-9a-zA-Z\\-+*/![^()]]*\\)`, що відповідає будь якому виразу, на початку якого стоїть відкриваюча дужка, а в кінці – закриваюча. У середині цього виразу мають бути відсутні інші дужки. Для прикладу, даний шаблон відповідає виразам виду `(21+34), (321+3*3.2/5-5),(-10 +50!),` але не відповідає виразам виду`((2+3)/5), (90*(9-5)/(60+17+13))`.

Далі створюються допоміжні змінні та масив типу `int`, який буде містити в собі значення індексу позиції дужки, що відкривається, та значення індексу позиції дужки, що закривається. Перший знайдений вираз, який відповідає описаному вище шаблону, поміщається в окрему змінну типу `String`. Далі за допомогою функції `indexOf`() знаходимо індекс початку виразу, що не містить внутрішніх дужок, у виразі, введеному користувачем. Індекс кінця виразу знаходимо, додавши довжину знайденого виразу до індексу початку, після чого поміщаємо значення індексів у масив. Якщо виразів, що відповідають шаблону, не знайдено, то функція повертає значення `Null`. Таким чином, дана функція повертає початкову та кінцеву позиції дужок у виразі, в межах яких на даному етапі необхідно провести обчислення.

Для того, щоб у подальшому спростити розробку, було створено ще декілька допоміжних функцій, а саме – функції, які повертають кількість лівих та правих дужок, загальну кількість дужок, та функцію, яка перевіряє, чи кількість лівих і правих дужок співпадає. Нижче представлена реалізація цих функцій:

```
static boolean CheckQuotes (String Example){
    if((QuoteAnalyser.getNumberOfQuotes(Example) & 1) != 0){
        return false;
    }
    int count_left_quotes = 0;
    int count_right_quotes = 0;
    for (int i = 0 ; i< Example.length(); i++){
        if (Example.charAt(i) == '('){
            count_left_quotes++;
        }
        if (Example.charAt(i) == ')'){
            count_right_quotes++;
        }
    }
    return count_left_quotes == count_right_quotes;
}
static int getNumberOfLeftQuotes (String Example){
    int count_left_quotes = 0;
    for (int i = 0 ; i< Example.length(); i++){
        if (Example.charAt(i) == '('){
            count_left_quotes++;
        }
    }
    return count_left_quotes;
}
static int getNumberOfRightQuotes (String Example){
    int count_right_quotes = 0;
    for (int i = 0 ; i< Example.length(); i++){
        if (Example.charAt(i) == ')'){
            count_right_quotes++;
        }
    }
    return count_right_quotes;
}
static int getNumberOfQuotes(String Example){
    int count_left_quotes = 0;
    int count_right_quotes = 0;
    for (int i = 0 ; i< Example.length(); i++){
        if (Example.charAt(i) == '('){
```

```
            count_left_quotes++;
        }
        if (Example.charAt(i) == ')'){
            count_right_quotes++;
        }
    }
    return count_left_quotes + count_right_quotes;
    // Return general count of quotes
}
```

Зауважимо, що всі функції даного класу є статичними, оскільки немає необхідності створювати екземпляри класу.

Наступним кроком реалізації основної функції калькулятора є написання класу `Reader`. Цей клас буде зчитувати операнди та арифметичні операції всередині дужок, які повертає функція `QuoteAnalyser.getCouplesOfQuotes()`, та поверетати їх список. Нижче наведена реалізація функції зчитування арифметичних знаків.

```
static List<Character>readSigns(String Example, int Begin, int End){
    List<Character> Signs = new ArrayList<>();
    String SubExample = Example.substring(Begin, End);
    Pattern pattern = Pattern.compile(".[+*\\-/]");
    Matcher matcher = pattern.matcher(SubExample);
    while (matcher.find()) {
        String Sign = matcher.group();
        if (Character.isDigit(Sign.charAt(0))) {
            Signs.add(Sign.charAt(1));
        }
    }
    return Signs;
}
```

Дана функція приймає введений користувачем вираз, і ще дві змінні типу `int`, які відповідають індексам початку і кінця дужок, та повертає об'єкт типу `List`. Створюється список типу `Character`, далі створюється допоміжна змінна `SubExample`, що буде зберігати вираз, який відповідає шаблону. Далі створено шаблон `.[+*\\-/]`, якому відповідають два символи: будь-який символ та знак арифметичної операції. У циклі перевіряється, чи перший символ є цифрою. Якщо так, то арифметичний знак додається у список. Нижче представлена функція зчитування операндів.

```
static List<Double> readNumbers(String Example, int Begin, int End){
    List <Double> Numbers = new ArrayList<>();
    String SubExample = Example.substring(Begin, End);
    Pattern pattern = Pattern.compile("(([(+\\-*/^]\\-)?)[0-9]+(\\.([0-
9])*)?(E[+\\-]?[0-9]+)?"); // Symbols at 0 and 1 indexes may be [+-*/]
    Matcher matcher = pattern.matcher(SubExample);
    while(matcher.find()){
        String Number = matcher.group();
        if (Number.length() == 1) {
            Numbers.add(Double.parseDouble(Number));
        }
        else if (Character.isDigit(Number.charAt(1)) || Number.charAt(1)
== '.') {
            Numbers.add(Double.parseDouble(Number));
        } else {
            char[] str = Number.toCharArray();
            str[0] = ' ';
            Number = new String(str);
            Numbers.add(Double.parseDouble(Number));
        }
    }
    return Numbers;
}
```

Дана функція приймає ті ж дані що й попередня функція, та повертає об'єкт типу `List` що містить у собі всі операнди в межах `Begin-End`. В середині функції створюється шаблон `(([(+\\-*/^]\\-)?)[0-9]+(\\.([0-9])*)?(E[+\\-]?[0-9]+)?`, що представляє собою будь-яке число, а також перші два символи перед ним, які можуть бути присутніми, або бути не присутніми. Якщо знайдена стрічка має довжину `1`, або символ під індексом `1` є цифрою чи крапкою, то стрічка конвертується у тип `double`, і додається у список. В іншому випадку із стрічки видаляється перший символ, конвертується в тип `double`, після чого додається у список.

Під час зчитування знаків та операндів з'являється проблема зі знаком `'-'`, оскільки потрібно визначити, чи потрібно додавати цей знак у список, чи присвоїти цей знак операнду. Тому було вирішено зчитувати у список знаків лише ті знаки `'-'`, які знаходяться безпосередньо після символу, що представляє собою цифру. Функції даного класу розроблені, виходячи з того, щоб калькулятор міг правильно зчитувати вирази з від'ємними числами, такі як: `(-5+10/-2)`. У даному випадку список арифметичних знаків буде `[ + ; / ]`, а список операндів `[-5 , 10 ; -2]`.

Наступним важливим кроком є розробка класу `BasicCalculator`. До складу цього класу входить допоміжня функція `String substituteString`, за допомогою якої можна замінювати будь-яку частину стрічки будь-якою іншою стрічкою:

```
static String substituteString(String Example, int Begin, int End, String
Result) {
    int LengthOfNewString = Example.length();
    int ChangeOfLenght = (End - Begin) + 1;
    LengthOfNewString -= ChangeOfLenght;
    int count = 0;
    String StringOfDouble = Result;
    ChangeOfLenght = StringOfDouble.length();
    LengthOfNewString += ChangeOfLenght;
    int j = 0;
    char[] NewExample = new char[LengthOfNewString];
    for (int i = 0; i < Example.length(); i++) {
        if (i == Begin) {
            while (count != StringOfDouble.length()) {
                NewExample[i] = StringOfDouble.charAt(count);
                count++;
                i++;
            }
            j = i;
            i = End;
        } else if (i > End) {
            NewExample[j] = Example.charAt(i);
            j++;
        } else NewExample[i] = Example.charAt(i);
    }
    return new String(NewExample);
}
```

Параметрами даної функції є два об'єкти типу `String` та дві змінні типу `int`. Перший об'єкт типу `String` є введений користувачем вираз, а другий об'єкт такого ж типу є стрічкою, яку необхідно вставити. Параметри типу `int` є початком і кінецем вставки. На початку функція обчислює довжину стрічки-результату шляхом віднімання різниці `End` та `Begin` від початкової довжини стрічки, та додавання довжини стрічки, яка вставляється. Далі в стрічку-результат вставляються всі символи старої стрічки з індексом, меншим від `Begin`, після чого вставляються символи стрічки `Result`. Далі вставляються ті символи зі старої стрічки, що мають індекс більший за `End`.

Для зручності було створено ще одну функцію, яка обгортає вираз користувача в дужки:

```
private static String prepareExample (String Example){
    char[] PreparedExample = new char[Example.length() + 2];
    PreparedExample[0] = '(';
    PreparedExample[PreparedExample.length - 1] = ')';
    for (int i = 1; i < PreparedExample.length - 1; i++){
```

```
            PreparedExample[i] = Example.charAt(i - 1);
        }
        Example = new String(PreparedExample);
        return Example;
    }
```

Далі було створено головну функцію `Calculate(String Example)`, яка буде використовувати всі вище описані функції для обчислення виразу. Нижче наведена її реалізація:

```
static String Calculate(String Example) {
    double ResultOfOperation = 0.0;
    Example = prepareExample(Example);
    int[] Quotes;
    List<Double> Numbers;
    List<Character> Signs;
    int i = 0;
    int NumberOfCycles = QuoteAnalyser.getNumberOfQuotes(Example) / 2;
    while(NumberOfCycles != 0) {
        Example = Example.replaceAll("\\-\\-", "+");
        Quotes = QuoteAnalyser.getCouplesOfQuotes(Example);
        Signs = Reader.readSigns(Example, Quotes[i], Quotes[i + 1]);
        Numbers = Reader.readNumbers(Example, Quotes[i], Quotes[i + 1]);
        if (!Signs.isEmpty()) {
            for (int j = 0; j < Signs.size(); j++) {
                if (Signs.get(j) == '*') {
                    ResultOfOperation = Numbers.get(j) * Numbers.get(j +
1);
                    Numbers.remove(j);
                    Numbers.remove(j);
                    Numbers.add(j, ResultOfOperation);
                    Signs.remove(j);
                    j--;
                } else if (Signs.get(j) == '/') {
                    ResultOfOperation = Numbers.get(j) / Numbers.get(j +
1);
                    Numbers.remove(j);
                    Numbers.remove(j);
                    Numbers.add(j, ResultOfOperation);
                    Signs.remove(j);
                    j--;
                }
            }
            for (int j = 0; j < Signs.size(); j++) {
                if (Signs.get(j) == '+') {
                    ResultOfOperation = Numbers.get(j) + Numbers.get(j +
1);
                    Numbers.remove(j);
                    Numbers.remove(j);
                    Numbers.add(j, ResultOfOperation);
                    Signs.remove(j);
                    j--;
                } else if (Signs.get(j) == '-') {
                    ResultOfOperation = Numbers.get(j) - Numbers.get(j +
1);
                    Numbers.remove(j);
                    Numbers.remove(j);
                    Numbers.add(j, ResultOfOperation);
                    Signs.remove(j);
                    j--;
                }
```

```
            }
            Example = substituteString(Example, Quotes[i], Quotes[i + 1],
String.valueOf(ResultOfOperation));
            NumberOfCycles--;
        }
        else {
            Example = substituteString(Example, Quotes[i], Quotes[i + 1],
String.valueOf(Numbers.get(0)));
            NumberOfCycles--;
        }
    }
    return Example;
}
```

Дана функція приймає лише один параметр, а саме - об'єкт типу `String`, що є виразом користувача. У цій функції спочатку створюється два порожніх об'єкти `Numbers` та `Signs` типу `List` з реалізацією `ArrayList`. Після чого знаходиться кількість ітерацій, необхідних для обчислення виразу. Вона дорівнює кількості пар дужок у виразі. Далі за допомогою функції `QuoteAnalyser.getCoupleOfQoutes()` у циклі знаходяться пара дужок, всередині яких відсутні інші пари дужок. Звідси і починаються обчислення. Наступним кроком є обчислення вмісту дужок. Кожна ітерація передбачає виконання таких кроків :

- Пошук чисел та додавання їх у список `Numbers` за допомогою методу `Reader.readNumbers()`.
- Пошук знаків та додавання їх у список Signs за допомогою методу `Reader.readSigns()`.
- Виконання множення та ділення у виразі шляхом знаходження індексу `j` відповідного знаку у списку(`*` або `/`), та виконання відповідної дії (множення чи ділення) над операндом, що має той самий індекс j, та операндом з індексом `j+1`. Далі зі списку `Signs` видаляється знак з індексом `j`, а зі списку `Numbers` видаляються операнди з індексом `j` та `j+1`.
- Виконання додавання та віднімання у виразі шляхом знаходження індексу `j` відповідного знаку у списку(`+` або `–`), та виконання відповідної дії (додавання чи віднімання) над операндом, що має той самий індекс `j` та операндом з індексом `j+1`. Далі зі списку `Signs` видаляється знак з індексом `j`, а зі списку `Numbers` видаляються операнди з індексом `j` та `j+1`.
- Перевірка списку `Signs`: якщо список порожній, то у списку `Numbers` залишилось одне число, яке і є результатом обчислень виразу в дужках.

Знайдений результат обчислення виразу, що знаходився в дужках, замінює весь вираз, що був в дужках, за допомогою допоміжної функції `substituteString()`.

Нижче наведено вигляд стрічки, списків арифметичних знаків та операндів для виразу `(-9+15/(16*-3)+(10+5*2))` після кожної ітерації:

Таблиця 1 – Покрокова робота функції `BasicCalculator.Calculate()`

Номер ітерації	Загальний вигляд виразу	Обчислювана частина виразу	Арифметичні знаки	Операнди
1	(-9+15/(16*-3)+(10+5*2))	(16*-3)	[*]	[16;-3]
2	(-9+15/-48+(10+5*2))	(10+5*2)	[+; *]	[10;5;2]
3	(-9+15/-48+20)	(-9+15/-48+20)	[+;/;+]	[-9;15;-48;20]
4	10.6875	-	-	[10.6875]

Висновок. В статті запропоновано один із можливих шляхів реалізації основної функції калькулятора – обчислення виразів – за допомогою `Java`-класів. Проблема вирішується з позицій об'єктно-орієнтованого програмування Наведені програмні коди запропонованих класів.

1. Герберт Шилдт. Java. Полное руководство, 10-е издание = Java. The Complete Reference, 10th Edition. — М.: «Диалектика», 2018. — 1488 с. — ISBN 978-5-6040043-6-4.
2. Кей С. Хорстманн. Java SE 9. Базовый курс = Core Java SE 9 for the Impatient. — М.: «Вильямс», 2018. — 576 с. — ISBN 978-5-6040043-0-2, 978-0-13-469472-6.