Министерство образования и науки РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа программной инженерии

Курсовой проект

по дисциплине «Языки и средства функционального программирования»

**Графическая программа «Калькулятор»**

Выполнил студент гр. 3530904/80003 Богданов Д.А.

Руководитель Лукашин А.А.

Оглавление

[Оглавление 2](#_Toc28047999)

[1. Цель 2](#_Toc28048000)

[2. Решение 2](#_Toc28048001)

[2.1. Язык программирования 2](#_Toc28048002)

[2.2. Графический интерфейс 2](#_Toc28048003)

[2.3. Функциональная реализация 3](#_Toc28048004)

[3. Алгоритм программы 3](#_Toc28048005)

[4. Демонстрация работы программы 3](#_Toc28048006)

[5. Выводы 4](#_Toc28048007)

[6. Приложение 4](#_Toc28048008)

[6.1. Исходный код 4](#_Toc28048009)

[6.1.1. Main.scala 4](#_Toc28048010)

[6.1.2. Calculator.scala 5](#_Toc28048011)

[6.1.3 CalculatorShell.scala 7](#_Toc28048012)

# 1. Цель

Реализовать графическое приложение «калькулятор», активно используя подходы функционального программирования.

# 2. Решение

## Язык программирования

В качестве функционального языка программирования используется Scala.

## Графический интерфейс

Графический интерфейс решено было создать с помощью библиотеки Swing. Она позволяет не только упростить расположение графических элементов, но и назначить действия, которые будут произведены по нажатию клавиши клавиатуры или кнопки в программе.

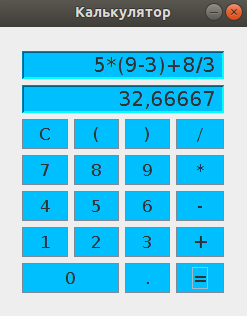
## Функциональная реализация

Есть некий абстрактный класс Expression, от которого наследуются классы Value (дробное число), UnaryOperation (оператор как строка и операнд как Expression) и BinaryOperation (оператор как строка и два операнда как Expression). Тогда объект Calculator, используя синтаксический анализатор (JavaTokenParsers - для преобразования строки, введённой пользователем, в выражение Expression), правил упрощения выражений и простого рекурсивного метода вычисления, позволит описать всю логику калькулятора в функциональном языке программирования Scala.

# Алгоритм программы

1. Пользователь с помощью графического интерфейса или с помощью клавиатуры вводит выражение, которое требуется вычислить.
   1. По нажатию на кнопку « = » запускается синтаксический анализ выражения.   
      Если пользователь ввёл некорректное выражение, то выводится строка “Error”.
2. В случае корректного ввода выражение упрощается очевидными случаями и затем рекурсивно вычисляется.
3. Вывод результата вычисления

# Демонстрация работы программы



# Выводы

Я написал графическое приложение «калькулятор», активно используя подходы функционального программирования. В ходе разработки узнал про синтаксические анализаторы и познакомился с библиотекой Swing в Scala.

# Приложение

## Исходный код

### Main.scala

package coursework.calculator.src.main.scala

import scala.swing.\_

import javax.swing.{ActionMap, InputMap, JComponent, KeyStroke}

import scala.swing.event.Key

object Main extends App with scalax.chart.module.Charting {

  val calculator = new CalculatorShell

  val frame: Frame = new Frame {

    title = "Калькулятор"

    val panel: GridBagPanel = calculator.CalcPanel()

    contents = panel

    val inputMap: InputMap = panel.peer.getInputMap(JComponent.WHEN\_IN\_FOCUSED\_WINDOW)

    val actionMap: ActionMap = panel.peer.getActionMap

    val symbols: List[(Char, List[Int])] = List(

      ('0', List(Key.Numpad0.id, Key.Key0.id)),

      ('1', List(Key.Numpad1.id, Key.Key1.id)),

      ('2', List(Key.Numpad2.id, Key.Key2.id)),

      ('3', List(Key.Numpad3.id, Key.Key3.id)),

      ('4', List(Key.Numpad4.id, Key.Key4.id)),

      ('5', List(Key.Numpad5.id, Key.Key5.id)),

      ('6', List(Key.Numpad6.id, Key.Key6.id)),

      ('7', List(Key.Numpad7.id, Key.Key7.id)),

      ('8', List(Key.Numpad8.id, Key.Key8.id)),

      ('9', List(Key.Numpad9.id, Key.Key9.id)),

      ('+', List(Key.Plus.id)),

      ('-', List(Key.Minus.id)),

      ('\*', List(Key.Multiply.id)),

      ('/', List(Key.Slash.id)),

      ('(', List(Key.OpenBracket.id)),

      (')', List(Key.CloseBracket.id)),

      ('.', List(Key.Comma.id))

    )

    def actionsGen(list: List[(Char, List[Int])]): List[(Action, List[Int])] = {

      if (list.isEmpty) {

        List()

      } else {

        (Action("key" + list.head.\_1) { calculator.inputField.text += list.head.\_1 }, list.head.\_2) :: actionsGen(list.tail)

      }

    }

    val actions: List[(Action, List[Int])] = actionsGen(symbols)

    actions.foreach(action => {

      actionMap.put(action.\_1.title, action.\_1.peer)

      action.\_2.foreach(key => inputMap.put(KeyStroke.getKeyStroke(key, 0), action.\_1.title))

    })

    val actBck: Action = Action("keyBck") {

      if (calculator.inputField.text.length > 0) {

        calculator.inputField.text = calculator.inputField.text.substring(0, calculator.inputField.text.length - 1)

      }

    }

    inputMap.put(KeyStroke.getKeyStroke(Key.BackSpace.id, 0), "keyBck")

    actionMap.put("keyBck", actBck.peer)

    size = new Dimension(250, 320)

    centerOnScreen()

    resizable = false

  }

  frame.open()

}

### Calculator.scala

package coursework.calculator.src.main.scala

import util.parsing.combinator.JavaTokenParsers

abstract class Expression

case class Value(value: Double) extends Expression

case class UnaryOperation(operator: String, operand: Expression) extends Expression

case class BinaryOperation(operator: String, firstArg: Expression, secondArg: Expression) extends Expression

object Calculator {

  // синтаксический анализатор выражения

  def parse(s: String): Expression = {

    object ExpressionParser extends JavaTokenParsers {

      def expr: Parser[Expression] =

        (term ~ "+" ~ term) ^^ { case firstArg ~ \_ ~ secondArg => BinaryOperation("+", firstArg, secondArg) } |

          (term ~ "-" ~ term) ^^ { case firstArg ~ \_ ~ secondArg => BinaryOperation("-", firstArg, secondArg) } |

          term

      def term: Parser[Expression] =

        (factor ~ "\*" ~ factor) ^^ { case firstArg ~ \_ ~ secondArg => BinaryOperation("\*", firstArg, secondArg) } |

          (factor ~ "/" ~ factor) ^^ { case firstArg ~ \_ ~ secondArg => BinaryOperation("/", firstArg, secondArg) } |

          factor

      def factor: Parser[Expression] =

        "(" ~> expr <~ ")" |

          floatingPointNumber ^^ { x => Value(x.toDouble) }

      def parse(s: String): ExpressionParser.ParseResult[Expression] = parseAll(expr, s)

    }

    ExpressionParser.parse(s).get

  }

  def simplify(e: Expression): Expression = {

    // упрощяем выражения с помощью очевидных арифметических случаев

    def combine(e: Expression) = e match {

      case UnaryOperation("-", UnaryOperation("-", x)) => x

      case UnaryOperation("+", x) => x

      case BinaryOperation("\*", x, Value(1)) => x

      case BinaryOperation("\*", Value(1), x) => x

      case BinaryOperation("\*", \_, Value(0)) => Value(0)

      case BinaryOperation("\*", Value(0), \_) => Value(0)

      case BinaryOperation("/", x, Value(1)) => x

      case BinaryOperation("/", x1, x2) if x1 == x2 => Value(1)

      case BinaryOperation("+", x, Value(0)) => x

      case BinaryOperation("+", Value(0), x) => x

      case \_ => e

    }

    // упрощяем подвыражения при помощи модели абстрактного синтаксического дерева

    val subs = e match {

      case BinaryOperation(op, firstArg, secondArg) => BinaryOperation(op, simplify(firstArg), simplify(secondArg))

      case UnaryOperation(op, operand) => UnaryOperation(op, simplify(operand))

      case \_ => e

    }

    combine(subs)

  }

  def calculate(e: Expression): Double = {

    e match {

      case Value(x) => x

      case UnaryOperation("-", x) => -calculate(x)

      case BinaryOperation("+", l, r) => calculate(l) + calculate(r)

      case BinaryOperation("-", l, r) => calculate(l) - calculate(r)

      case BinaryOperation("\*", l, r) => calculate(l) \* calculate(r)

      case BinaryOperation("/", l, r) => calculate(l) / calculate(r)

    }

  }

}

### 6.1.3 CalculatorShell.scala

package coursework.calculator.src.main.scala

import Main.{Color, Font}

import javax.swing.BorderFactory

import scala.swing.{Alignment, Button, Dimension, GridBagPanel, Insets, TextField}

import scala.swing.event.ButtonClicked

class CalculatorShell() {

  private val functionalCalculator: Calculator.type = Calculator

  val btnFont = new Font("Arial", 0, 17)

  val fieldFont = new Font("Arial", 0, 20)

  val btnColor = new Color(0, 191, 255)

  val btnSize = new Dimension(50, 45)

  val doubleBtnSize = new Dimension(50, 90)

  var inputField: TextField = new TextField(" ") {

    font = fieldFont

    background = btnColor

    opaque = true

    border = BorderFactory.createCompoundBorder(

      BorderFactory.createLoweredBevelBorder(),

      BorderFactory.createEmptyBorder(0, 5, 0, 5))

    editable = false

    horizontalAlignment = Alignment.Right

  }

  var outputField: TextField = new TextField(" ") {

    font = fieldFont

    background = btnColor

    opaque = true

    border = BorderFactory.createCompoundBorder(

      BorderFactory.createLoweredBevelBorder(),

      BorderFactory.createEmptyBorder(0, 5, 0, 5))

    editable = false

    horizontalAlignment = Alignment.Right

  }

  val btn1: Button = new Button("1") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn2: Button = new Button("2") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn3: Button = new Button("3") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn4: Button = new Button("4") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn5: Button = new Button("5") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn6: Button = new Button("6") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn7: Button = new Button("7") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn8: Button = new Button("8") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn9: Button = new Button("9") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btn0: Button = new Button("0") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnPoint: Button = new Button(".") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnCalculate: Button = new Button("=") {

    font = btnFont

    background = btnColor

    preferredSize = doubleBtnSize

  }

  val btnMinus: Button = new Button("-") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnPlus: Button = new Button("+") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnMultiply: Button = new Button("\*") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnDivide: Button = new Button("/") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnOpenBracket: Button = new Button("(") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnCloseBracket: Button = new Button(")") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  val btnClean: Button = new Button("C") {

    font = btnFont

    background = btnColor

    preferredSize = btnSize

  }

  def CalcPanel(): GridBagPanel = {

    val contents: GridBagPanel = new GridBagPanel() {

      var c = new Constraints()

      c.gridx = 0

      c.gridy = 0

      c.gridwidth = 4

      c.insets = new Insets(3, 3, 3, 3)

      c.fill = GridBagPanel.Fill.Both

      add(inputField, c)

      c.gridwidth = 4

      c.gridx = 0

      c.gridy = 1

      add(outputField, c)

      c.gridwidth = 1

      c.gridx = 0

      c.gridy = 2

      add(btnClean, c)

      c.gridx = 0

      c.gridy = 3

      add(btn7, c)

      c.gridx = 0

      c.gridy = 4

      add(btn4, c)

      c.gridx = 0

      c.gridy = 5

      add(btn1, c)

      c.gridx = 1

      c.gridy = 2

      add(btnOpenBracket, c)

      c.gridx = 2

      c.gridy = 2

      add(btnCloseBracket, c)

      c.gridx = 3

      c.gridy = 2

      add(btnDivide, c)

      c.gridx = 1

      c.gridy = 3

      add(btn8, c)

      c.gridx = 2

      c.gridy = 3

      add(btn9, c)

      c.gridx = 3

      c.gridy = 3

      add(btnMultiply, c)

      c.gridx = 1

      c.gridy = 4

      add(btn5, c)

      c.gridx = 2

      c.gridy = 4

      add(btn6, c)

      c.gridx = 3

      c.gridy = 4

      add(btnMinus, c)

      c.gridx = 1

      c.gridy = 5

      add(btn2, c)

      c.gridx = 2

      c.gridy = 5

      add(btn3, c)

      c.gridx = 3

      c.gridy = 5

      add(btnPlus, c)

      c.gridwidth = 2

      c.gridx = 0

      c.gridy = 6

      add(btn0, c)

      c.gridwidth = 1

      c.gridx = 2

      c.gridy = 6

      add(btnPoint, c)

      c.gridx = 3

      c.gridy = 6

      add(btnCalculate, c)

      listenTo(btn0)

      listenTo(btn1)

      listenTo(btn2)

      listenTo(btn3)

      listenTo(btn4)

      listenTo(btn5)

      listenTo(btn6)

      listenTo(btn7)

      listenTo(btn8)

      listenTo(btn9)

      listenTo(btnPlus)

      listenTo(btnMinus)

      listenTo(btnPoint)

      listenTo(btnCalculate)

      listenTo(btnClean)

      listenTo(btnMultiply)

      listenTo(btnDivide)

      listenTo(btnCloseBracket)

      listenTo(btnOpenBracket)

      reactions += {

        case ButtonClicked(`btn0`) => inputField.text += btn0.text

        case ButtonClicked(`btn1`) => inputField.text += btn1.text

        case ButtonClicked(`btn2`) => inputField.text += btn2.text

        case ButtonClicked(`btn3`) => inputField.text += btn3.text

        case ButtonClicked(`btn4`) => inputField.text += btn4.text

        case ButtonClicked(`btn5`) => inputField.text += btn5.text

        case ButtonClicked(`btn6`) => inputField.text += btn6.text

        case ButtonClicked(`btn7`) => inputField.text += btn7.text

        case ButtonClicked(`btn8`) => inputField.text += btn8.text

        case ButtonClicked(`btn9`) => inputField.text += btn9.text

        case ButtonClicked(`btnPlus`) => inputField.text += btnPlus.text

        case ButtonClicked(`btnMinus`) => inputField.text += btnMinus.text

        case ButtonClicked(`btnPoint`) => inputField.text += btnPoint.text

        case ButtonClicked(`btnCloseBracket`) => inputField.text += btnCloseBracket.text

        case ButtonClicked(`btnOpenBracket`) => inputField.text += btnOpenBracket.text

        case ButtonClicked(`btnMultiply`) => inputField.text += btnMultiply.text

        case ButtonClicked(`btnDivide`) => inputField.text += btnDivide.text

        case ButtonClicked(`btnClean`) => inputField.text = ""

        case ButtonClicked(`btnCalculate`) => outputField.text = {

          try {

            s"%1.5f".format(functionalCalculator.calculate(functionalCalculator.parse(inputField.text)))

          } catch {

            case \_: Exception => "Error"

          }

        }

      }

    }

    contents

  }

}