Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Высшая школа программной инженерии

Курсовая работа  
По дисциплине «Функциональное программирование»

Выполнила студентка группы 3530904/80005 Соколова В.А.  
Руководитель Лукашин А.А.

22 декабря 2019 г.

Санкт-Петербург   
2019

Описание задачи  
Калькулятор на Scala.

1. Ход работы

|  |
| --- |
| Текст программы calculator.scala  package calculator |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| import util.parsing.combinator.JavaTokenParsers |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| abstract class Expr |
|  |

|  |
| --- |
| case class Val(value : Double) extends Expr |
|  |

|  |
| --- |
| case class UnOp(operator : String, operand : Expr) extends Expr |
|  |

|  |
| --- |
| case class BiOp(operator : String, lhs : Expr, rhs : Expr) extends Expr |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| object Calculator { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def parse(s : String) : Expr = { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| object ExpressionParser extends JavaTokenParsers |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| def expr : Parser[Expr] = |
|  |

|  |
| --- |
| (term ~ "+" ~ term) ^^ { case lhs~plus~rhs => BiOp("+", lhs, rhs) } | |
|  |

|  |
| --- |
| (term ~ "-" ~ term) ^^ { case lhs~minus~rhs => BiOp("-", lhs, rhs) } | |
|  |

|  |
| --- |
| term |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def term : Parser[Expr] = |
|  |

|  |
| --- |
| (factor ~ "\*" ~ factor) ^^ { case lhs~times~rhs => BiOp("\*", lhs, rhs) } | |
|  |

|  |
| --- |
| (factor ~ "/" ~ factor) ^^ { case lhs~div~rhs => BiOp("/", lhs, rhs) } | |
|  |

|  |
| --- |
| factor |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def factor : Parser[Expr] = |
|  |

|  |
| --- |
| "(" ~> expr <~ ")" | |
|  |

|  |
| --- |
| floatingPointNumber ^^ { x => Val(x.toDouble) } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def parse(s : String) = parseAll(expr, s) |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ExpressionParser.parse(s).get |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def simplify(e: Expr) : Expr = { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def combine(e : Expr) = e match { |
|  |

|  |
| --- |
| case UnOp("-", UnOp("-", x)) => x |
|  |

|  |
| --- |
| case UnOp("+", x) => x |
|  |

|  |
| --- |
| case BiOp("\*", x, Val(1)) => x |
|  |

|  |
| --- |
| case BiOp("\*", Val(1), x) => x |
|  |

|  |
| --- |
| case BiOp("\*", x, Val(0)) => Val(0) |
|  |

|  |
| --- |
| case BiOp("\*", Val(0), x) => Val(0) |
|  |

|  |
| --- |
| case BiOp("/", x, Val(1)) => x |
|  |

|  |
| --- |
| case BiOp("/", x1, x2) if x1 == x2 => Val(1) |
|  |

|  |
| --- |
| case BiOp("+", x, Val(0)) => x |
|  |

|  |
| --- |
| case BiOp("+", Val(0), x) => x |
|  |

|  |
| --- |
| case \_ => e |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| val subs = e match { |
|  |

|  |
| --- |
| case BiOp(op, lhs, rhs) => BiOp(op, simplify(lhs), simplify(rhs)) |
|  |

|  |
| --- |
| case UnOp(op, operand) => UnOp(op, simplify(operand)) |
|  |

|  |
| --- |
| case \_ => e |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| combine(subs) |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def evaluate(e : Expr) : Double = { |
|  |

|  |
| --- |
| e match { |
|  |

|  |
| --- |
| case Val(x) => x |
|  |

|  |
| --- |
| case UnOp("-", x) => -evaluate(x) |
|  |

|  |
| --- |
| case BiOp("+", l, r) => (evaluate(l) + evaluate(r)) |
|  |

|  |
| --- |
| case BiOp("-", l, r) => (evaluate(l) - evaluate(r)) |
|  |

|  |
| --- |
| case BiOp("\*", l, r) => (evaluate(l) \* evaluate(r)) |
|  |

|  |
| --- |
| case BiOp("/", l, r) => (evaluate(l) / evaluate(r)) |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
| } **client.scala**  |  | | --- | | package calculator | |  |  |  | | --- | |  | |  |  |  | | --- | | import Calculator.\_; | |  |  |  | | --- | |  | |  |  |  | | --- | | object Client { | |  |  |  | | --- | |  | |  |  |  | | --- | | val expressions = List( | |  |  |  | | --- | | "1", | |  |  |  | | --- | | "(2)", | |  |  |  | | --- | | "3 + 0", | |  |  |  | | --- | | "3 + 2", | |  |  |  | | --- | | "(0 + 6)", | |  |  |  | | --- | | "(7 + 8) + 9", | |  |  |  | | --- | | "(1 + 2) + (3 + 4)", | |  |  |  | | --- | | "(1 \* 6) / (7 \* 1)", | |  |  |  | | --- | | "9 - 1", | |  |  |  | | --- | | "(2 - 3) - 4", | |  |  |  | | --- | | "(5 / 6) / 7", | |  |  |  | | --- | | "(2 / 2) / (2 / 2)" | |  |  |  | | --- | | ) | |  |  |  | | --- | |  | |  |  |  | | --- | | def parsing() { | |  |  |  | | --- | | Console.println("\nPARSING") | |  |  |  | | --- | | for (text <- expressions) | |  |  |  | | --- | | Console.printf("%20s => %s\n", text, parse(text)) | |  |  |  | | --- | | } | |  |  |  | | --- | |  | |  |  |  | | --- | | def simplifying() { | |  |  |  | | --- | | Console.println("\nSIMPLIFYING") | |  |  |  | | --- | | for (text <- expressions) | |  |  |  | | --- | | Console.printf("%20s => %s\n", text, simplify(parse(text))) | |  |  |  | | --- | | } | |  |  |  | | --- | |  | |  |  |  | | --- | | def evaluating() { | |  |  |  | | --- | | Console.println("\nEVALUATING") | |  |  |  | | --- | | for (text <- expressions) | |  |  |  | | --- | | Console.printf("%20s == %s\n", text, evaluate(simplify(parse(text)))) | |  |  |  | | --- | | } | |  |  |  | | --- | |  | |  |  |  | | --- | | def main(args: Array[String]) { | |  |  |  | | --- | | parsing() | |  |  |  | | --- | | simplifying() | |  |  |  | | --- | | evaluating() | |  |  |  | | --- | | } | |  |   } |
|  |

1. Ссылка на репозиторий  
   <https://github.com/mycelium/hsse-fp-2019-2/tree/3530904/80005_sokolova-varvara>
2. Вывод  
   В результате выполнения работы я улучшила свои навыки программирования на языке Scala.