КУРСОВая Работа

Создание приложения на языке Scala

по дисциплине «Функциональное программирование»

Выполнил

студент гр.3530904/80004 < > Корнилов Д.Ф.

Руководитель <> Лукашин А.А.

«17» декабря 2019 г.

Санкт-Петербург

2019

**Оглавление**

[Оглавление 2](#_Toc27520817)

[Задания 2](#_Toc27520818)

[Скриншоты 3](#_Toc27520819)

[Код программ 4](#_Toc27520820)

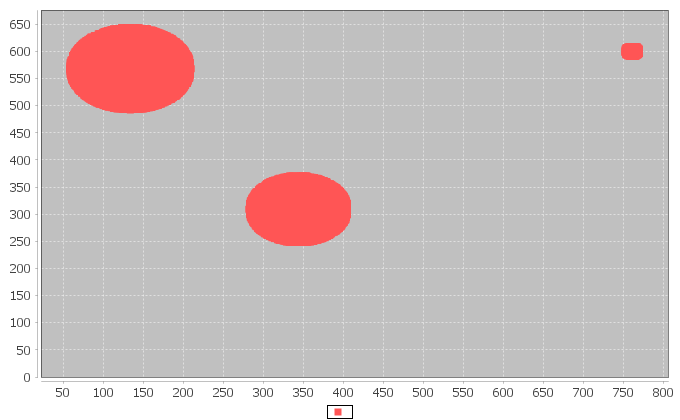
[Заключение 14](#_Toc27520821)

# **Задания**

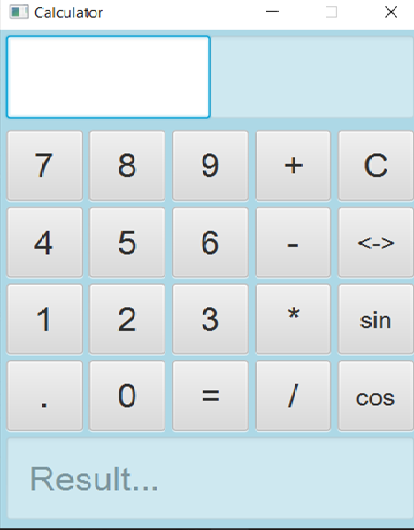
1. Кластеризация точек - реализовать считывание точек из файла и отображение на графике (для построения графика выбрать любую библиотеку). Количество точек от 1 миллиона в двумерном пространстве
2. Калькулятор

# **Скриншоты**

1. **Кластеризация**



1. **Калькулятор**



# **Код программ**

1. **Кластеризация**

object Clusters {  
  
 object Cluster1 {  
 val *x*: Int = scala.util.Random.nextInt(900)  
 val *y*: Int = scala.util.Random.nextInt(900)  
 val *rad*: Int = scala.util.Random.nextInt(100)  
 }  
  
 object Cluster2 {  
 val *x*: Int = scala.util.Random.nextInt(900)  
 val *y*: Int = scala.util.Random.nextInt(900)  
 val *rad*: Int = scala.util.Random.nextInt(100)  
 }  
  
 object Cluster3 {  
 val *x*: Int = scala.util.Random.nextInt(900)  
 val *y*: Int = scala.util.Random.nextInt(900)  
 val *rad*: Int = scala.util.Random.nextInt(100)  
 }  
  
 def checkDistance(x: Int, y: Int): Boolean = {  
 val dis1 = Math.*pow*(x - Cluster1.*x*, 2) + Math.*pow*(y - Cluster1.*y*, 2)  
 val dis2 = Math.*pow*(x - Cluster2.*x*, 2) + Math.*pow*(y - Cluster2.*y*, 2)  
 val dis3 = Math.*pow*(x - Cluster3.*x*, 2) + Math.*pow*(y - Cluster3.*y*, 2)  
 if (dis1 <= Math.*pow*(Cluster1.*rad*, 2)) return true  
 if (dis2 <= Math.*pow*(Cluster2.*rad*, 2)) return true  
 if (dis3 <= Math.*pow*(Cluster3.*rad*, 2)) return true  
 false  
 }  
  
}

import java.io.{File, PrintWriter}  
  
object Input {  
 val *filePath* = "data.txt"  
  
 def writeValuesToFile() {  
 val pw = new PrintWriter(new File(*filePath*))  
 for (\_ <- 0 to 1000000) {  
 val x = scala.util.Random.nextInt(1000)  
 val y = scala.util.Random.nextInt(1000)  
 pw.write("(" + x.toString + ", " + y.toString + ")\n")  
 }  
 pw.close()  
 }  
}

import Input.\_  
import Output.\_  
import Clusters.\_  
import scala.collection.mutable.ArrayBuffer  
  
object MyChartApp extends App with scalax.chart.module.Charting {  
 *writeValuesToFile*()  
 *readValuesFromFile*(filePath = *filePath*)  
 val *arrX*: ArrayBuffer[Int] = scala.collection.mutable.ArrayBuffer.empty[Int]  
 val *arrY*: ArrayBuffer[Int] = scala.collection.mutable.ArrayBuffer.empty[Int]  
 for (i <- *bufX*.indices) {  
 if (*checkDistance*(*bufX*(i), *bufY*(i))) {  
 *arrX* += *bufX*(i)  
 *arrY* += *bufY*(i)  
 }  
 }  
 val *data* = for (i <- *arrX*.indices) yield (*arrX*(i), *arrY*(i))  
 val *chart* = *XYLineChart*(*data*)  
 *chart*.plot.setRenderer(new org.jfree.chart.renderer.xy.XYLineAndShapeRenderer(false, true))  
 *chart*.saveAsPNG("clusters.png")  
 *chart*.show("Chart", (800, 600), scrollable = false)  
}

import scala.collection.mutable.ArrayBuffer  
import scala.io.Source  
  
object Output {  
 val *bufX*: ArrayBuffer[Int] = scala.collection.mutable.ArrayBuffer.empty[Int]  
 val *bufY*: ArrayBuffer[Int] = scala.collection.mutable.ArrayBuffer.empty[Int]  
  
 def readValuesFromFile(filePath: String) {  
 val fSource = Source.*fromFile*(filePath)  
 for (line <- fSource.getLines) {  
 val opBr = line.indexOf("(")  
 val cm = line.indexOf(",")  
 val clBr = line.indexOf(")")  
 val first = line.substring(opBr + 1, cm).toInt  
 val second = line.substring(cm + 2, clBr).toInt  
 *bufX* += first  
 *bufY* += second  
 }  
 fSource.close  
 }  
}

1. **Калькулятор**

import javafx.event.ActionEvent  
import scalafx.application.JFXApp  
import scalafx.scene.Scene  
import scalafx.scene.control.\_  
import scalafx.scene.layout.GridPane  
import scalafx.scene.paint.Color.\_  
import scalafx.scene.text.Font  
  
object CalculatorUI extends JFXApp {  
 *stage* = new JFXApp.PrimaryStage {  
 title.value = "Calculator"  
 width = 345  
 height = 460  
 resizable = false  
 scene = new Scene {  
 fill = *LightBlue* val *pane* = new GridPane  
 *pane*.setHgap(5)  
 *pane*.setVgap(5)  
  
 var *font*: Font = new Font("Arial", 28)  
  
 var *isFirstTestFieldEnabled*: Boolean = true  
  
 val *text1* = new TextField  
 val *maxLength* = 7  
 *text1*.setEditable(false)  
 *text1*.setLayoutX(5)  
 *text1*.setLayoutY(5)  
 *text1*.prefWidth = 160  
 *text1*.prefHeight = 70  
 *text1*.setFont(*font*)  
  
 val *text2* = new TextField  
 *text2*.setEditable(false)  
 *text2*.setDisable(true)  
 *text2*.setLayoutX(165)  
 *text2*.setLayoutY(5)  
 *text2*.prefWidth = 160  
 *text2*.prefHeight = 70  
 *text2*.setFont(*font*)  
  
 def checkTextField(tf: TextField): Unit = {  
 if (tf.text.value == "sin" || tf.text.value == "cos") {  
 tf.text.value = ""  
 }  
 if (tf.getText().length() > *maxLength*) {  
 val s: String = tf.getText.substring(0, *maxLength*)  
 tf.setText(s)  
 }  
 }  
  
 val *result* = new TextField  
 *result*.setEditable(false)  
 *result*.setDisable(true)  
 *result*.setFont(*font*)  
 *result*.text = "Result..."  
 *result*.setLayoutX(5)  
 *result*.setLayoutY(345)  
 *result*.prefWidth = 320  
 *result*.prefHeight = 70  
  
  
 val *buttonWidth* = 60  
 val *buttonHeight* = 60  
  
 val *button0*: Button = new Button {  
 text = "0"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button0*.setFont(*font*)  
  
 val *button1*: Button = new Button {  
 text = "1"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button1*.setFont(*font*)  
  
 val *button2*: Button = new Button {  
 text = "2"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button2*.setFont(*font*)  
  
 val *button3*: Button = new Button {  
 text = "3"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button3*.setFont(*font*)  
  
 val *button4*: Button = new Button {  
 text = "4"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button4*.setFont(*font*)  
  
 val *button5*: Button = new Button {  
 text = "5"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button5*.setFont(*font*)  
  
 val *button6*: Button = new Button {  
 text = "6"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button6*.setFont(*font*)  
  
 val *button7*: Button = new Button {  
 text = "7"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button7*.setFont(*font*)  
  
 val *button8*: Button = new Button {  
 text = "8"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button8*.setFont(*font*)  
  
 val *button9*: Button = new Button {  
 text = "9"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *button9*.setFont(*font*)  
  
 val *divide*: Button = new Button {  
 text = "/"  
 tooltip = "Divide"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *divide*.setFont(*font*)  
  
 val *plus*: Button = new Button {  
 text = "+"  
 tooltip = "Add"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *plus*.setFont(*font*)  
  
 val *minus*: Button = new Button {  
 text = "-"  
 tooltip = "Subtract"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *minus*.setFont(*font*)  
  
 val *floatPoint*: Button = new Button {  
 text = "."  
 tooltip = "Float point"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *floatPoint*.setFont(*font*)  
  
 val *equals*: Button = new Button {  
 text = "="  
 tooltip = "Calculate"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *equals*.setFont(*font*)  
  
 val *multiply*: Button = new Button {  
 text = "\*"  
 tooltip = "Multiply"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *multiply*.setFont(*font*)  
  
 val *clearDisplay*: Button = new Button {  
 text = "C"  
 tooltip = "Clear Screen"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *clearDisplay*.setFont(*font*)  
  
 *font* = new Font("Arial", 20)  
  
 val *changeTextField*: Button = new Button {  
 text = "<->"  
 tooltip = "Change text field"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *changeTextField*.setFont(*font*)  
  
 val *sin*: Button = new Button {  
 text = "sin"  
 tooltip = "Sinus"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *sin*.setFont(*font*)  
  
 val *cos*: Button = new Button {  
 text = "cos"  
 tooltip = "Cosine"  
 prefWidth = *buttonWidth* prefHeight = *buttonHeight* }  
 *cos*.setFont(*font*)  
  
 *button0*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "0"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "0"  
 }  
 }  
  
 *button1*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "1"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "1"  
 }  
 }  
  
 *button2*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "2"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "2"  
 }  
 }  
  
 *button3*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "3"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "3"  
 }  
 }  
  
 *button4*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "4"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "4"  
 }  
 }  
  
 *button5*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "5"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "5"  
 }  
 }  
  
 *button6*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "6"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "6"  
 }  
 }  
  
 *button7*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "7"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "7"  
 }  
 }  
  
 *button8*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "8"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "8"  
 }  
 }  
  
 *button9*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 *text1*.text = *text1*.text.value + "9"  
 } else {  
 checkTextField(*text2*)  
 *text2*.text = *text2*.text.value + "9"  
 }  
 }  
  
 *floatPoint*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 checkTextField(*text1*)  
 if ((*text1*.text.value != "") && (!*text1*.text.value.contains("."))) *text1*.text = *text1*.text.value + "."  
 } else {  
 checkTextField(*text2*)  
 if ((*text2*.text.value != "") && (!*text2*.text.value.contains("."))) *text2*.text = *text2*.text.value + "."  
 }  
 }  
 *clearDisplay*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) *text1*.text = ""  
 else *text2*.text = ""  
 }  
  
 *sin*.onAction = (\_: ActionEvent) => {  
 *text1*.text = "sin"  
 }  
  
 *cos*.onAction = (\_: ActionEvent) => {  
 *text1*.text = "cos"  
 }  
  
 *changeTextField*.onAction = (\_: ActionEvent) => {  
 if (*isFirstTestFieldEnabled*) {  
 *text1*.setDisable(true)  
 *text2*.setDisable(false)  
 *isFirstTestFieldEnabled* = false  
 } else {  
 *text1*.setDisable(false)  
 *text2*.setDisable(true)  
 *isFirstTestFieldEnabled* = true  
 }  
 }  
  
 *multiply*.onAction = (\_: ActionEvent) => {  
 if ((*text1*.text.value != "") && (*text2*.text.value != "")) {  
 if (*text1*.text.value != "cos" || *text1*.text.value != "sin") {  
 val op1 = *text1*.text.value.toDouble  
 val op2 = *text2*.text.value.toDouble  
 val res = op1 \* op2  
 *result*.text.value = res.toString  
 }  
 }  
 }  
  
 *divide*.onAction = (\_: ActionEvent) => {  
 if ((*text1*.text.value != "") && (*text2*.text.value != "")) {  
 if (*text1*.text.value != "cos" || *text1*.text.value != "sin") {  
 val op1 = *text1*.text.value.toDouble  
 val op2 = *text2*.text.value.toDouble  
 val res = op1 / op2  
 *result*.text.value = res.toString  
 }  
 }  
 }  
  
 *plus*.onAction = (\_: ActionEvent) => {  
 if ((*text1*.text.value != "") && (*text2*.text.value != "")) {  
 if (*text1*.text.value != "cos" || *text1*.text.value != "sin") {  
 val op1 = *text1*.text.value.toDouble  
 val op2 = *text2*.text.value.toDouble  
 val res = op1 + op2  
 *result*.text.value = res.toString  
 }  
 }  
 }  
  
 *minus*.onAction = (\_: ActionEvent) => {  
 if ((*text1*.text.value != "") && (*text2*.text.value != "")) {  
 if (*text1*.text.value != "cos" || *text1*.text.value != "sin") {  
 val op1 = *text1*.text.value.toDouble  
 val op2 = *text2*.text.value.toDouble  
 val res = op1 - op2  
 *result*.text.value = res.toString  
 }  
 }  
 }  
  
 *equals*.onAction = (\_: ActionEvent) => {  
 if (*text1*.text.value == "sin" && *text2*.text.value != "") {  
 val res = Math.*sin*(*text2*.text.value.toDouble)  
 *result*.text.value = res.toString  
 }  
 if (*text1*.text.value == "cos" && *text2*.text.value != "") {  
 val res = Math.*cos*(*text2*.text.value.toDouble)  
 *result*.text.value = res.toString  
 }  
 }  
  
 *pane*.add(*button7*, 1, 1)  
 *pane*.add(*button4*, 1, 2)  
 *pane*.add(*button1*, 1, 3)  
 *pane*.add(*floatPoint*, 1, 4)  
  
 *pane*.add(*button8*, 2, 1)  
 *pane*.add(*button5*, 2, 2)  
 *pane*.add(*button2*, 2, 3)  
 *pane*.add(*button0*, 2, 4)  
  
 *pane*.add(*button9*, 3, 1)  
 *pane*.add(*button6*, 3, 2)  
 *pane*.add(*button3*, 3, 3)  
 *pane*.add(*equals*, 3, 4)  
  
 *pane*.add(*plus*, 4, 1)  
 *pane*.add(*minus*, 4, 2)  
 *pane*.add(*multiply*, 4, 3)  
 *pane*.add(*divide*, 4, 4)  
  
 *pane*.add(*clearDisplay*, 5, 1)  
 *pane*.add(*changeTextField*, 5, 2)  
 *pane*.add(*sin*, 5, 3)  
 *pane*.add(*cos*, 5, 4)  
  
 *pane*.setLayoutY(80)  
  
 content = *List*(*text1*, *text2*, *result*, *pane*)  
 }  
 }  
}

# **Заключение**

В данной курсовой работе был использован язык функционального программирования Scala для изображения кластеров точек на графике и реализации калькулятора. Работа продемонстрировала приемущества функицонального программирования: краткость, лаконичность и быстродействие.