КУРСОВАЯ РАБОТА

расчетно-графическая работа

По дисциплине «Языки и средства функционального программирования»

Выполнил студент группы 3530904/80003

Родченков С. Н.

Руководитель

Лукашин А. А.

# Содержание

[Содержание 2](#_Toc27524976)

[Введение 3](#_Toc27524977)

[Формулировка задачи 3](#_Toc27524978)

[Решение задачи 4](#_Toc27524979)

[Класс «Bitmap» 4](#_Toc27524980)

[Класс «Complex» 4](#_Toc27524981)

[Объект «Mandelbrot» 4](#_Toc27524982)

[Объект «Main» 4](#_Toc27524983)

[Результат работы 4](#_Toc27524984)

[Вывод 6](#_Toc27524985)

[Приложение А 7](#_Toc27524986)

[Листинг «Main.scala» 7](#_Toc27524987)

[Листинг «Bitmap.scala» 8](#_Toc27524988)

[Листинг «Complex.scala» 8](#_Toc27524989)

[Листинг «Mandelbrot.scala» 8](#_Toc27524990)

# Введение

Функциональное программирование — раздел дискретной математики и парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Противопоставляется парадигме императивного программирования, которая описывает процесс вычислений как последовательное изменение состояний (в значении, подобном таковому в теории автоматов). При необходимости, в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например, как список.

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от императивного, где одной из базовых концепций является переменная, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма).

## Формулировка задачи

Реализовать построение множества Мандельброта и представления его в виде картинки, активно используя подходы функционального программирования.

# Решение задачи

С целью решения поставленной задачи, был реализован ряд классов, отвечающий за те или иные этапы построения требуемого изображения. Классы, за исключением тех, которые явно подразумевали императивный подход программирования, были созданы в преимущественно функциональном стиле.

## Класс «Bitmap»

Класс является контейнером для растрового изображения и обладает методами установки и считывания цветового кода в формате RGB каждого отдельного пикселя изображения.

## Класс «Complex»

Класс является представлением комплексного числа и позволяет производить с ним базовую арифметику.

## Объект «Mandelbrot»

Производит подсчет функции, формирующей конечное изображение, а также, исходя из результатов подсчета, определяет цветовой код каждого пикселя. В результате своей работы создает корректное растровое изображение, представляющее собой отображение множества Мандельброта в заданных границах координат

## Объект «Main»

Отвечает за отображение полученного изображения на полотне, перемещение и масштабирование полотна. Реализован с помощью библиотеки Swing.

## Результат работы

После сборки программы, состоящей из описанных классов, было создано окно, представленное на рисунке 1. Построенное в процессе работы программы изображение полностью отвечало обозначенным требованиям. Исходный код работы представлен в приложении «А».

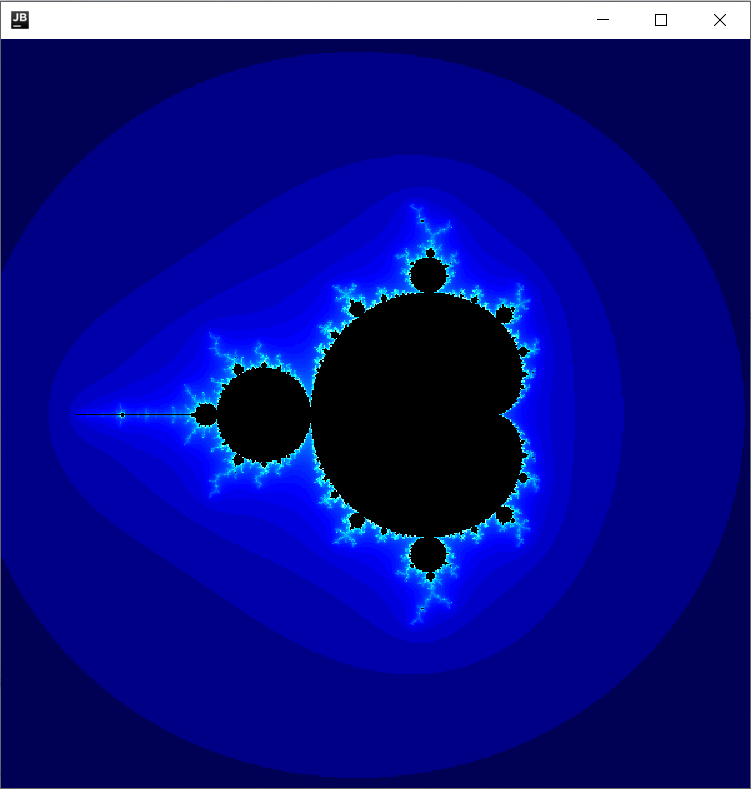


Рисунок 1 - Результат работы приложения

# Вывод

В результате проделанной работы на практике были закреплены знания, полученные за курс введения в функциональное программирование. Было создано графическое приложение, позволяющее наглядно представлять множество Мандельброта.

# Приложение А

## Листинг «Main.scala»

import javax.swing.ImageIcon

import mandel.Mandelbrot

import scala.swing.\_

import scala.swing.event.\_

object Main extends SimpleSwingApplication {

def top: MainFrame = new MainFrame {

private val width = 600

private val height = 600

private val maxIter = 500

private var minX: Double = -2

private var maxX: Double = 2

private var minY: Double = -2

private var maxY: Double = 2

private val label = new Label {

icon = new ImageIcon(Mandelbrot.compute(width, height, minX, maxX, minY, maxY, maxIter).img)

}

private def dx() = (maxX - minX) / 10.0

private def dy() = (maxY - minY) / 10.0

private def move(dx: Double, dy: Double) = {

minX += dx

maxX += dx

minY -= dy

maxY -= dy

new ImageIcon(Mandelbrot.compute(width, height, minX, maxX, minY, maxY, maxIter).img)

}

def scale(dx: Double, dy: Double): ImageIcon = {

maxX -= dx

minX += dx

maxY -= dy

minY += dy

new ImageIcon(Mandelbrot.compute(width, height, minX, maxX, minY, maxY, maxIter).img)

}

contents = new BoxPanel(Orientation.Vertical) {

contents += label

listenTo(keys)

reactions += {

case KeyPressed(\_, Key.Right, \_, \_) => label.icon = move(dx(), 0)

case KeyPressed(\_, Key.Left , \_, \_) => label.icon = move(-dx(), 0)

case KeyPressed(\_, Key.Up , \_, \_) => label.icon = move(0, dy())

case KeyPressed(\_, Key.Down , \_, \_) => label.icon = move(0, -dy())

case KeyPressed(\_, Key.Enter, \_, \_) => label.icon = scale(dx(), dy())

case KeyPressed(\_, Key.Space, \_, \_) => label.icon = scale(-dx(), -dy())

}

focusable = true

requestFocus

}

}

}

## Листинг «Bitmap.scala»

package mandel.bitmap

import java.awt.image.BufferedImage

import java.awt.Color

class Bitmap(val width: Int, val height: Int) {

val img = new BufferedImage(width, height, BufferedImage.TYPE\_3BYTE\_BGR)

def getColor(x: Int, y: Int): Color = new Color(img.getRGB(x, y))

def setColor(x: Int, y: Int, c : Color): Unit = img.setRGB(x, y, c.getRGB)

}

## Листинг «Complex.scala»

package mandel.complex

case class Complex(real: Double, imag: Double) {

def + (rhs: Complex) = Complex(real + rhs.real, imag + rhs.imag)

def - (rhs: Complex) = Complex(real - rhs.real, imag - rhs.imag)

def \* (rhs: Complex) = Complex(real \* rhs.real - imag \* rhs.imag, real \* rhs.imag + imag \* rhs.real)

lazy val abs: Double = math.hypot(real, imag)

}

## Листинг «Mandelbrot.scala»

package mandel

import java.awt.Color

import scala.concurrent.\_

import scala.concurrent.duration.\_

import scala.concurrent.ExecutionContext.Implicits.global

import scala.concurrent.Future

import complex.\_

import bitmap.\_

package object Mandelbrot {

def compute(width: Int, height: Int, xMin: Double, xMax: Double, yMin: Double, yMax: Double, maxIter: Int): Bitmap = {

val bm = new Bitmap(width, height)

val cx = (xMax - xMin) / width

val cy = (yMax - yMin) / height

val futures = {

for (y <- 0 until bm.height) yield Future {

for (x <- 0 until bm.width) {

val c = Complex(xMin + x \* cx, yMin + y \* cy)

val iter = itMandel(c, maxIter, 4)

bm.setColor(x, y, getColor(iter, maxIter))

}

}

}.map(Await.result(\_, 10.second))

bm

}

private def itMandel(c: Complex, iMax: Int, bailout: Int): Int = {

@scala.annotation.tailrec

def inner(z: Complex, i: Int): Int = i match {

case x if x == iMax => iMax

case x if z.abs > bailout => x

case x => inner(z \* z + c, x + 1)

}

inner(Complex(0, 0), 0)

}

private def getColor(iter: Int, max: Int): Color = (iter, 3 \* math.log(iter) / math.log(max - 1.0)) match {

case (itr, \_) if itr == max => Color.BLACK

case (\_, col) if col < 1 => new Color(0, 0, (255 \* col).toInt)

case (\_, col) if col < 2 => new Color(0, (255 \* (col - 1)).toInt, 255)

case (\_, col) => new Color((255 \* (col - 2)).toInt, 255, 255)

}

}