 Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики

Кафедра автоматики и процессов управления

**ОТЧЕТ**

по лабораторной работе №2

по дисциплине «СМиСПИС»

Вариант №3

|  |  |  |
| --- | --- | --- |
| Студент гр. 5371 |  | Мартынов М. |
| Студентка гр. 5371 |  | Козлова С. |
| Студент гр. 5371 |  | Аверкиев В. |
| Преподаватель |  | Кораблев Ю.А. |

Санкт-Петербург

2020

1. **Цель работы**

Изучение объектной модели Java, классы, интерфейсы, наследование, полиморфизм.

1. **Задание на лабораторную работу №1. Вариант №3.**

Разработать упрощенную иерархическую модель классов предметной области согласно варианту задания в таблице 1. Реорганизовать код в соответствии с разработанной диаграммой классов.

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| Вариант | Предметная область | Программная реализация |
| 3 | Стохастические фракталы | Рандомизированные снежинки Коха |

1. **Выполнение лабораторной работы**

**Диаграмма классов предметной области**

На риc. 1 изображена UML диаграмма классов переработанной программы в стиле ООП.

A screenshot of a cell phone

Description automatically generated

Рисунок 1 – UML диаграмма классов разработанной программы

**Описание классов предметной области**

В табл. 2 представлены описания основных классов переработанной программы.

Таблица 2 – Описание классов предметной области

|  |  |
| --- | --- |
| Имя класса | Описание |
| DrawableElement | Основной интерфейс для элемента, который может быть нарисован, содержащий единственный метод draw() |
| ColorElement | Абстрактный класс, хранящий цвет элемента |
| RandomKochCurve | Класс, описывающий кривую Коха с рандомизацией |
| RandomKochSnowflake | Класс, описывающий снежинку Коха с рандомизацией |
| RandomKochSnowflakesFractal | Класс, представляющий собой фрактал с рандомизированными снежинками Коха |
| MegaRandomKochSnowflake | Класс, описывающий комбинацию снежинок Коха с рандомизацией |
| OopStyleKochFractal | Файл, содержащий функцию main(), с которой начинается исполнение программы |
| ColorResolver | Класс, определяющий цвет для раскрашивания области на рисунке |
| Randomization | Класс – обертка для генерации случайны чисел |

**Тестирование переписанной программы**

На рис.2 показана отрисовка фрактала в исходной программе, написанной в парадигме функционального стиля.

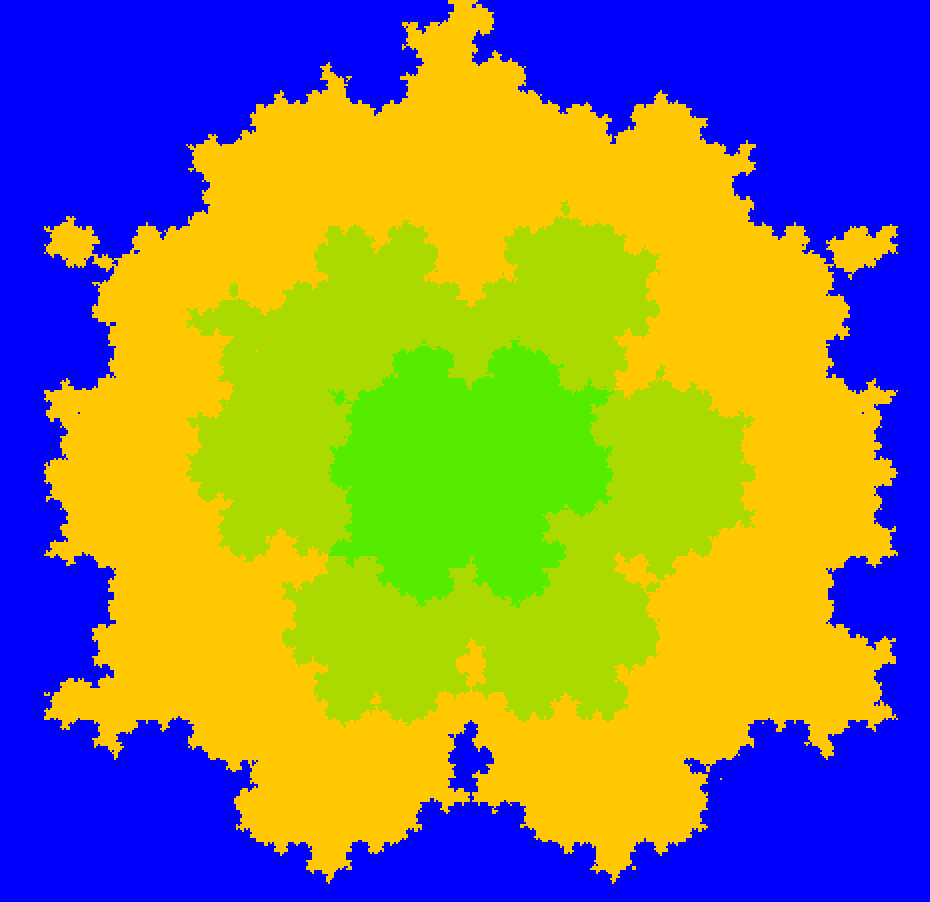


Рисунок – 2. Результат работы исходной программы до рефакторинга.

На рис.3 показана отрисовка фрактала в переработанной программе, переписанной в парадигме объектно-ориентированного подхода.

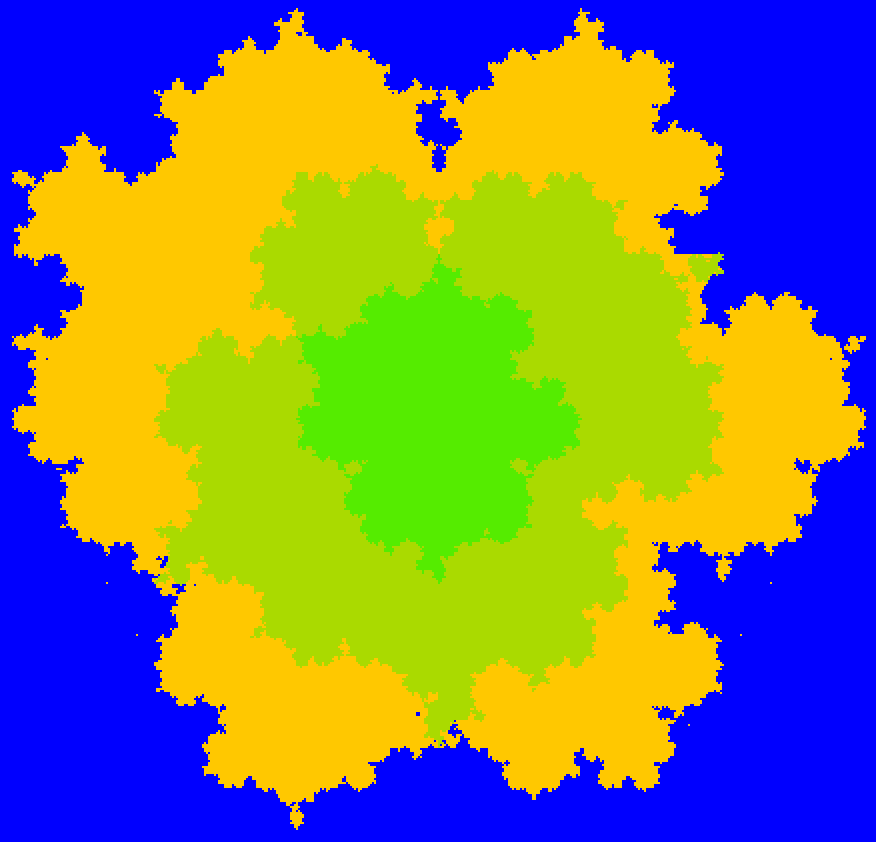
****

Рисунок – 3. Результат работы переписанной программы.

Визуально сравнивая результаты работы программы до рефакторинга и после можно сделать вывод о том, что переписанная программа работает корректно.

1. **Зачетное задание**

**Описание задания**

Введите в структуру Ваших классов элементы для подсчета, хранения и чтения числа минимальных элементов, формирующих фрактал. Что Вы будете понимать под минимальным элементом определите сами.

**Решение**

Под минимальными элементами, формирующими фрактал будем понимать количество треугольников, из которых фрактал состоит.

Таким образом, нужно добавить подсчет количества вызовов функции draw() у объектов типа RandomKochCurve, так как именно объекты данного класса отвечают за отрисовку 3-х линий, составляющих треугольник – минимальную единицу фрактала.

Модифицированная UML-диаграмма изображена ниже на рисунке 4.

A screenshot of a social media post

Description automatically generated

Рисунок 4 – UML диаграмма иерархии классов для выполнения зачетного задания

Введем интерфейc CountableElement с 3-мя методами:

* count() – для подсчета использований
* getCounter(): Long – для получения текущего значения счетчика
* resetCounter() – для обнуления счетчика

А также создадим абстрактный класс - реализацию данного интерфейса CountableAndColoredRandomKochCurve, которую будет наследовать класс RandomKochCurve. Класс RandomKochCurve не может напрямую наследовать два класса CountableRandomKochCurve и ColoredRandomKochCurve, так как в Kotlin, как и в Java запрещено множественное наследование.

CountableAndColoredRandomKochCurve содержит внутри себя Котлиновский объект-компаньон CounterStorage, это аналог статическим методам и полям в Java. Внутри CounterStorage содержится только одно поле *counter* типа *Long*, которая и является счетчиком минимальных элементов фрактала. Это поле является статическим (static), если проводить аналогию с языком Java.

Все методы интерфейса CountableElement обращаются к статическому полю *counter* объекта компаньона CounterStorage.

Исходный код добавленного интерфеса CountableElement и CountableAndColoredRandomKochCurve можно найти ниже в разделе «Исходный код переписанной программы».

**Исходный код переписанной программы**

Исходный код класса MegaRandomKochSnowflake.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements.impl  
  
**import** java.awt.Color  
**import** java.awt.Graphics2D  
**import** java.awt.geom.Point2D  
  
**class** MegaRandomKochSnowflake(  
 inColor: Color,  
 **private val endColor**: Color,  
 outColor: Color,  
 graph: Graphics2D,  
 c: Point2D,  
 d: Double,  
 m: Int,  
 **private val k**: Int,  
 n: Int  
) : RandomKochSnowflake(inColor, outColor, graph, c, d, m, n) {  
  
 **override fun** draw() {  
 RandomKochSnowflake(  
 **endColor**,  
 **outColor**,  
 **graph**,  
 **c**,  
 **d**,  
 **m**,  
 **n** ).draw()  
 **for** (i **in** 1 *until* **k**) {  
 RandomKochSnowflake(  
 getBetweenColor(**endColor**, **inColor**, i.toDouble() / **k**),  
 getBetweenColor(  
 **endColor**,  
 **inColor**,  
 (i - 1).toDouble() / **k** ),  
 **graph**,  
 **c**,  
 **d** \* (**k** - i) / **k**,  
 **m**,  
 **n** ).draw()  
 }  
 }  
  
 **companion object** ColorResolver {  
  
 **fun** getBetweenColor(  
 startColor: Color, endColor: Color, p: Double  
 ): Color {  
 **return** Color(  
 (startColor.*red* +  
 (endColor.*red* - startColor.*red*) \* p).toInt(),  
 (startColor.*green* +  
 (endColor.*green* - startColor.*green*) \* p).toInt(),  
 (startColor.*blue* +  
 (endColor.*blue* - startColor.*blue*) \* p).toInt()  
 )  
 }  
  
 }  
}

Исходный код класса RandomKochCurve.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements.impl  
  
**import** com.github.sindicat.lab2.oop.fratctal.elements.ColorElement  
**import** java.awt.Color  
**import** java.awt.Graphics2D  
**import** java.awt.geom.Path2D  
**import** java.awt.geom.Point2D  
**import** java.util.\*  
**import** kotlin.math.sqrt  
  
**class** RandomKochCurve(  
 graph: Graphics2D,  
 inColor: Color,  
 outColor: Color,  
 **private val pInit**: Point2D,  
 **private val qInit**: Point2D,  
 **private val nInit**: Int  
): ColorElement(inColor, outColor, graph) {  
  
 **override fun** draw() {  
 drawRandomKochCurve(**pInit**, **qInit**, **nInit**)  
 }  
  
 **fun** drawRandomKochCurve(p: Point2D, q: Point2D, n: Int) {  
 **val** w = randomSgn()  
 **val** r: Point2D = Point2D.Double(  
 (2 \* p.*x* + q.*x*) / 3,  
 (2 \* p.*y* + q.*y*) / 3  
 )  
 **val** s: Point2D = Point2D.Double(  
 (p.*x* + q.*x*) / 2 -  
 w \* (p.*y* - q.*y*) \* *sqrt*(3.0) / 6,  
 (p.*y* + q.*y*) / 2 +  
 w \* (p.*x* - q.*x*) \* *sqrt*(3.0) / 6  
 )  
 **val** t: Point2D = Point2D.Double(  
 (p.*x* + 2 \* q.*x*) / 3,  
 (p.*y* + 2 \* q.*y*) / 3  
 )  
 **val** path: Path2D = Path2D.Double()  
 path.moveTo(r.*x*, r.*y*)  
 path.lineTo(s.*x*, s.*y*)  
 path.lineTo(t.*x*, t.*y*)  
 path.lineTo(r.*x*, r.*y*)  
 path.closePath()  
 **if** (w == 1) {  
 **graph**.*color* = **inColor** } **else** */\* if w == -1 \*/* {  
 **graph**.*color* = **outColor** }  
 **graph**.fill(path)  
 **if** (n == 0) {  
 **return** }  
 drawRandomKochCurve(p, r, n - 1)  
 drawRandomKochCurve(r, s, n - 1)  
 drawRandomKochCurve(s, t, n - 1)  
 drawRandomKochCurve(t, q, n - 1)  
 }  
  
 **companion object** Randomization {  
 **val random** = Random()  
  
 **fun** randomSgn(): Int {  
 **return random**.nextInt(2) \* 2 - 1}  
}

Исходный код класса RandomKochSnowflake.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements.impl  
  
**import** com.github.sindicat.lab2.oop.fratctal.elements.ColorElement  
**import** java.awt.Color  
**import** java.awt.Graphics2D  
**import** java.awt.geom.Path2D  
**import** java.awt.geom.Point2D  
**import** kotlin.math.cos  
**import** kotlin.math.sin  
  
**open class** RandomKochSnowflake(  
 inColor: Color,  
 outColor: Color,  
 graph: Graphics2D,  
 **protected val c**: Point2D,  
 **protected val d**: Double,  
 **protected val m**: Int,  
 **protected val n**: Int  
  
) : ColorElement(inColor, outColor, graph) {  
  
 **override fun** draw() {  
 **val** vs = *arrayOfNulls*<Point2D>(**m**)  
 **for** (i **in** 0 *until* **m**) {  
 vs[i] = Point2D.Double(  
 **c**.*x* + **d** \* *cos*(2 \* Math.*PI* / **m** \* i),  
 **c**.*y* - **d** \* *sin*(2 \* Math.*PI* / **m** \* i)  
 )  
 }  
 **val** path: Path2D = Path2D.Double()  
 path.moveTo(vs[0]!!.*x*, vs[0]!!.*y*)  
 **for** (i **in** 0 *until* **m**) {  
 path.lineTo(vs[(i + 1) % **m**]!!.*x*, vs[(i + 1) % **m**]!!.*y*)  
 }  
 path.closePath()  
 **graph**.*color* = **inColor  
 graph**.fill(path)  
 **for** (i **in** 0 *until* **m**) {  
 RandomKochCurve(  
 **graph**,  
 **inColor**,  
 **outColor**,  
 vs[(i + 1) % **m**]!!,  
 vs[i]!!,  
 **n** ).draw()  
 }  
 }  
  
}

Исходный код класса ColorElement.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements  
  
**import** java.awt.Color  
**import** java.awt.Graphics2D  
  
**abstract class** ColorElement(  
 **protected val inColor**: Color,  
 **protected val outColor**: Color,  
 **protected val graph**: Graphics2D  
) : DrawableElement

Исходный код класса DrawableElement.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements  
  
**interface** DrawableElement {  
  
 **fun** draw()  
  
}

Исходный код класса RandomKochSnowflakesFractal.kt

**package** com.github.sindicat.lab2.oop.fratctal  
  
**import** com.github.sindicat.lab2.oop.fratctal.elements.DrawableElement  
**import** com.github.sindicat.lab2.oop.fratctal.elements.impl.MegaRandomKochSnowflake  
**import** java.awt.Color  
**import** java.awt.Graphics  
**import** java.awt.Graphics2D  
**import** java.awt.geom.Point2D  
**import** java.awt.geom.Rectangle2D  
**import** java.awt.image.BufferedImage  
**import** javax.swing.JFrame  
**import** javax.swing.JPanel  
  
  
**class** RandomKochSnowflakesFractal(  
 **private val imageWidth**: Int,  
 **private val imageHeight**: Int,  
 **private val m**: Int,  
 **private val k**: Int,  
 **private val n**: Int  
) : DrawableElement {  
  
 **private val image** = BufferedImage(**imageWidth**, **imageHeight**, BufferedImage.*TYPE\_INT\_RGB*)  
  
 **private val graph**: Graphics2D = **image**.createGraphics().*apply* **{** *color* = Color.*BLUE* fill(Rectangle2D.Double(0.0, 0.0, **imageWidth**.toDouble(), **imageHeight**.toDouble()))  
 **}  
  
 override fun** draw() {  
 MegaRandomKochSnowflake(  
 Color.*GREEN*, Color.*ORANGE*, Color.*BLUE*, **graph**,  
 Point2D.Double((**imageWidth** / 2).toDouble(), (**imageHeight** / 2).toDouble()),  
 **imageWidth** / 3.toDouble(), **m**, **k**, **n** ).draw()  
 showFractal()  
 }  
  
 **private fun** showFractal() = JFrame().*apply* **{** addNotify()  
 setSize(  
 *insets*.**left** + *insets*.**right** + **imageWidth**,  
 *insets*.**top** + *insets*.**bottom** + **imageHeight** )  
 *defaultCloseOperation* = JFrame.*EXIT\_ON\_CLOSE* add(**object** : JPanel() {  
 **public override fun** paintComponent(g: Graphics) {  
 g.drawImage(**image**, 0, 0, **null**)  
 }  
 })  
 *isVisible* = **true  
 }**}

Исходный код класса CountableElement.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements  
  
**interface** CountableElement {  
  
 **fun** count()  
  
 **fun** getCounter(): Long  
  
 **fun** resetCounter()  
  
}

Исходный код класса CountableElement.kt

**package** com.github.sindicat.lab2.oop.fratctal.elements.curve  
  
**import** com.github.sindicat.lab2.oop.fratctal.elements.ColorElement  
**import** com.github.sindicat.lab2.oop.fratctal.elements.CountableElement  
**import** java.awt.Color  
**import** java.awt.Graphics2D  
  
**abstract class** CountableAndColoredRandomKochCurve(  
 graph: Graphics2D,  
 inColor: Color,  
 outColor: Color) : ColorElement(inColor, outColor, graph),  
 CountableElement {  
  
 **override fun** count() {  
 **counter**++  
 }  
  
 **override fun** getCounter(): Long =  
 **counter  
  
 override fun** resetCounter() {  
 **counter** = 0  
 }  
  
 **companion object** CounterStorage {  
 **private var counter**: Long = 0L  
 }  
  
}

1. **Вывод**

В ходе выполнения лабораторной работы была изучена объектная модель Java, а также классы, интерфейсы, наследование и полиморфизм, полученные знания были применены на практике и использованы при переписывании программы, написанной в функциональном стиле, в ООП стиль.