МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторным работам №1-6

по дисциплине «Объектно-ориентированное программирование»

Студент гр. 8302 _____ Халитов Ю.Р.

Преподаватель _____ Красильников А.В.

Санкт-Петербург

2021

Лабораторная работа №1 «Матрицы».

Задание.

Необходимо реализовать класс для работы с матрицами.

Класс должен предоставлять возможности для

- инициализации матрицы
- просмотра значения элемента в позиции (i,j) и изменения этого элемента
- просмотра размерностей матрицы
- арифметические действия с матрицами +,-,*
- умножение матрицы на скаляр
- вычисления определителя матрицы
- сравнения двух матриц

В случае недопустимости какой-то операции необходимо сгенерировать исключение.

Исходный код.

```
class Matrix(val rows: Int, val cols: Int) {
    // . . .

    private fun cofactor(p : Int, q: Int) : Matrix {
        val result = Matrix(rows-1, cols-1)
        var i = 0
        var j = 0
        for (row in 0 until rows) {
            for (col in 0 until cols) {
                if (row != p && col != q) {
                    result[i, j++] = this[row, col]
                    if (j == cols - 1) {
                        j = 0
                        i++
                    }
                }
            }
        }
        return result
    }

    val det : Int
        get(): Int {
            if (rows != cols) {
                throw IllegalArgumentException("The determinant can be
calculated only for a square matrix")
            }
            if (rows == 1) return matrix[0][0]
            var d = 0
```

```
            var sign = 1
            for (i in 0 until cols) {
                val cofactor = this.cofactor(0, i)
                d += sign * matrix[0][i] * cofactor.det
                sign = -sign
            }
            return d
        }

    operator fun times(other: Matrix) : Matrix {
        if (cols != other.rows)
            throw IllegalArgumentException(
                "The product can be calculated only if left matrix columns same
as right matrix rows")
        val result = Matrix(rows, other.cols)
        for (i in 0 until rows) {
            for (j in 0 until other.cols) {
                for (k in 0 until cols) {
                    result[i, j] += this[i, k] * other[k, j]
                }
            }
        }
        return result
    }

    // . . .
}
```

Вывод.

В ходе выполнения лабораторной работы был реализован класс для работы с матрицами, а также программа, демонстрирующая возможности класса.

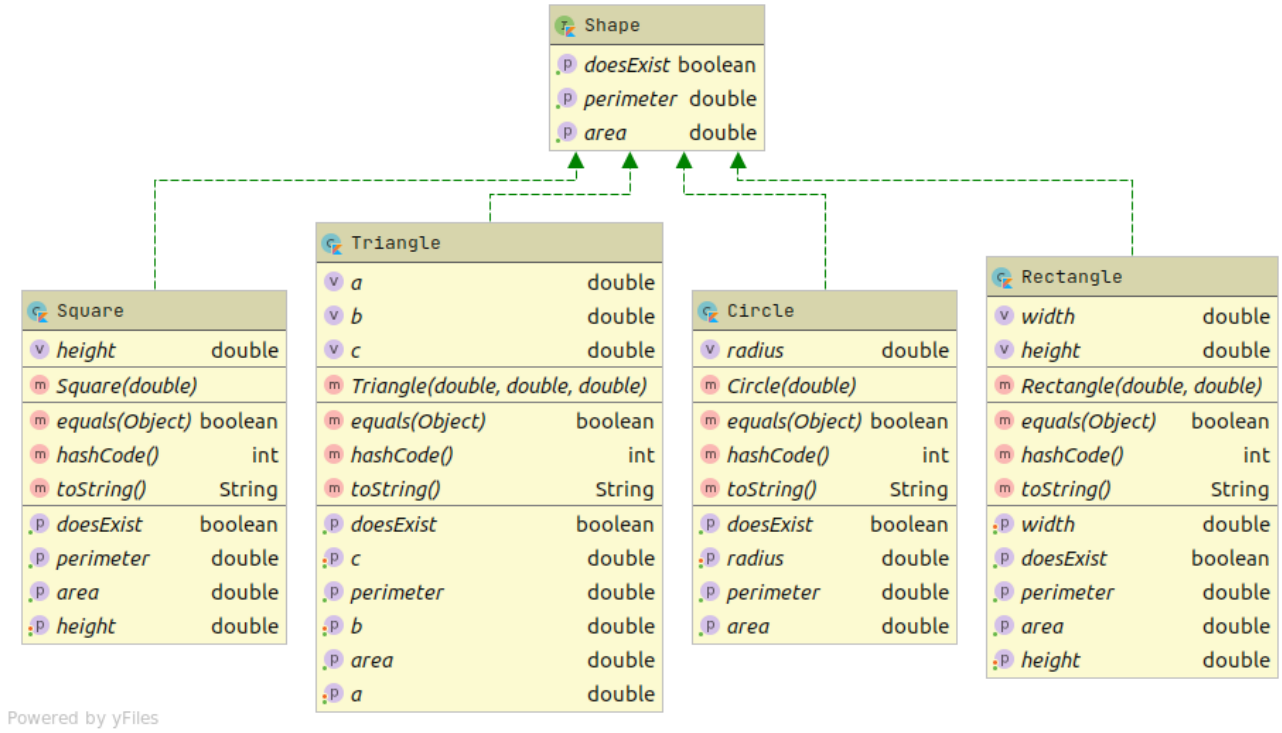
Лабораторная работа №2 «Геометрические фигуры».

Задание.

Реализовать классы для описания набора геометрических фигур (круг, квадрат, прямоугольник, треугольник). Для каждой фигуры предусмотреть возможность вычисления площади и периметра.

Для демонстрации работы программы создать в main несколько фигур каждого типа, сложить их все в список. В консоль вывести суммарную площадь всех фигур, фигуру с наибольшей и наименьшей площадями и периметрами.

UML-диаграмма.



Исходный код.

```
class Triangle(var a: Double, var b: Double, var c: Double) : Shape {
    override val area: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Triangle with a=$a, b=$b, c=$c
cannot exist")
            return sqrt(perimeter * (perimeter - a) * (perimeter - b) *
(perimeter - c))
        }

    override val perimeter: Double
```

```
        get() {
            if (!doesExist)
                throw IllegalStateException("Triangle with a=$a, b=$b, c=$c
cannot exist")
            return a + b + c
        }

    override val doesExist: Boolean
        get() = a > 0 && b > 0 && c > 0 && a + b > c && b + c > a && a + c > b

    // . . .
}

class Square(var height: Double) : Shape {

    override val area: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Square with height=$height cannot
exist")
            return height * height
        }

    override val perimeter: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Square with height=$height cannot
exist")
            return 4 * height
        }

    override val doesExist: Boolean
        get() = height > 0

    // . . .
}

class Rectangle(var width: Double, var height: Double) : Shape {

    override val area: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Rectangle with width=$width and
height=$height cannot exist")
            return width * height
        }

    override val perimeter: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Rectangle with width=$width and
height=$height cannot exist")
            return 2 * (width + height)
        }

    override val doesExist: Boolean
        get() = width > 0 && height > 0

    // . . .
}

class Circle(var radius: Double) : Shape {
```

```
    override val area: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Circle with radius=$radius cannot
exist")
            return PI * radius * radius
        }

    override val perimeter: Double
        get() {
            if (!doesExist)
                throw IllegalStateException("Circle with radius=$radius cannot
exist")
            return 2 * PI * radius
        }

    override val doesExist: Boolean
        get() = radius > 0

   // . . .
}
```

Вывод.

В ходе выполнения лабораторной работы был реализован классы для описания набора геометрических фигур, а также программа, демонстрирующая возможности класса.

Лабораторная работа №3 «Телефонная книга».

Задание.

Реализовать набор классов для моделирования приложения Телефонная книга. Требуемый функционал:

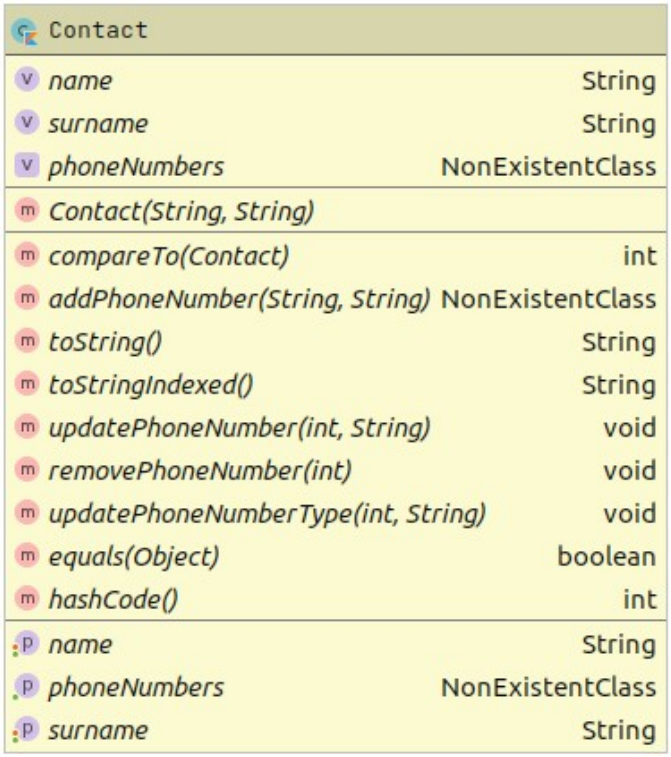
- Добавление, просмотр, редактирование и удаление контактов.
- Каждый контакт состоит из Фамилии, Имени и набора телефонных номеров, от нуля до ..бесконечности?!
- Каждый телефон имеет номер и тип (мобильный, рабочий, домашний)
- Метод для полнотекстового поиска контактов по подстроке, то есть, если в телефонной книге содержаться две записи:

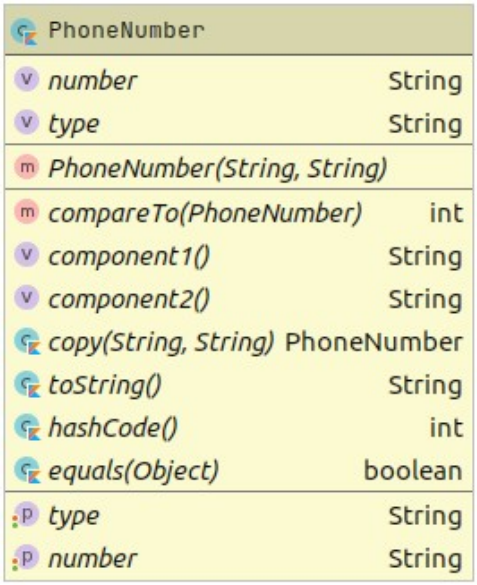
  Иван Иванов +79991234567, 88129923232
  Парикмахерская +9348732894723

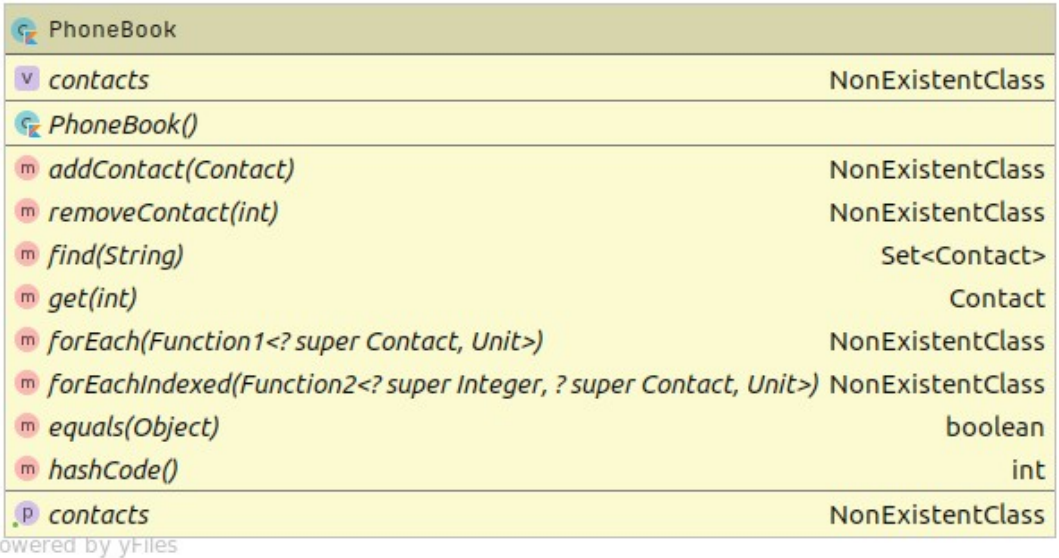
  phoneBook.find(“и”) // должен вернуть обе записи
  phoneBook.find(“123”) // только первую

Продемонстрировать работу приложения, например в функции main. Вывод данных в консоль.

UML-диаграмма.







Исходный код.

```
data class PhoneNumber (var number: String, var type: String) :
Comparable<PhoneNumber> {
    override fun compareTo(other: PhoneNumber): Int =
number.compareTo(other.number)
}

class Contact(var name: String,
              var surname: String) : Comparable<Contact> {
```

```
    val phoneNumbers = sortedSetOf<PhoneNumber>()

    override fun compareTo(other: Contact) : Int = "$name $surname".compareTo("$
{other.name} ${other.surname}")

    fun addPhoneNumber(number : String, type: String) =
phoneNumbers.add(PhoneNumber(number, type))

    override fun toString(): String {
        var result = "$name $surname "
        phoneNumbers.forEach { result += "${it.type}: ${it.number} " }
        return result
    }

    fun toStringIndexed() : String {
        var result = "$name $surname "
        phoneNumbers.forEachIndexed { i, phoneNumber -> result += "($i) $
{phoneNumber.type}: ${phoneNumber.number} " }
        return result
    }

    fun updatePhoneNumber(index: Int, number: String) {
        phoneNumbers.elementAt(index).number = number
    }
    fun removePhoneNumber(index: Int) {
        phoneNumbers.remove(phoneNumbers.elementAt(index))
    }

    fun updatePhoneNumberType(index: Int, type: String) {
        phoneNumbers.elementAt(index).type = type
    }

    override fun equals(other: Any?): Boolean {
        if (other == null || other !is Contact ||
                name != other.name || surname != other.surname || phoneNumbers !
= other.phoneNumbers) return false
        return true
    }

    override fun hashCode(): Int = listOf(name, surname,
phoneNumbers).hashCode()
}

class PhoneBook {
    val contacts = sortedSetOf<Contact>()

    fun addContact(contact: Contact) = contacts.add(contact)

    fun removeContact(index: Int) = contacts.remove(this[index])

    fun find(substr : String) : Set<Contact> {
        val result = mutableSetOf<Contact>()
        contacts.forEach { contact ->
            val stringFields = mutableListOf(contact.name, contact.surname)
            contact.phoneNumbers.forEach { phoneNumber ->
stringFields.add(phoneNumber.number) }
            for (str in stringFields) {
                if (str.contains(substr)) {
                    result.add(contact)
                    break
                }
            }
        }
```

```
        return result
    }

    operator fun get(i: Int) : Contact = contacts.elementAt(i)

    inline fun forEach(action: (Contact) -> Unit) = contacts.forEach
{ action(it) }

    inline fun forEachIndexed(action: (index: Int, Contact) -> Unit) =
contacts.forEachIndexed { index, contact ->  action(index, contact) }

    override fun equals(other: Any?): Boolean {
        if (other == null || other !is PhoneBook ||
                contacts != other.contacts) return false
        return true
    }

    override fun hashCode(): Int = contacts.hashCode()
}
```

Вывод.

В ходе выполнения лабораторной работы был реализован реализован набор классов для моделирования приложения Телефонная книга, а также программа, демонстрирующая возможности приложения.

Лабораторная работа №4 «Generics».

Задание.

Используя код из ЛБ 2, на основе Generics (обобщений) Реализовать класс ShapeAccumulator, у которого будут следующие методы:

- add - принимает фигуры любого типа
- addAll - принимает любую коллекцию любого типа фигур (например, список Shape или множество прямоугольников
- getMax[Min]AreaShape - возращает фигуру с макс/мин площадью
- getMax[Min]PerimeterShape - с макс/мин периметром
- getTotalArea - суммарную площадь фигур
- getTotalPerimeter - суммарный периметр фигур

Исходный код.

```
class ShapeAccumulator {
    private val shapes = mutableListOf<Shape>()

    fun add(shape: Shape) = shapes.add(shape)

    fun addAll(collection: Collection<Shape>) = shapes.addAll(collection)

    val maxAreaShape
        get() = shapes.maxByOrNull{ shape -> shape.area }!!

    val minAreaShape
        get() = shapes.minByOrNull{ shape -> shape.area }!!

    val maxPerimeterShape
        get() = shapes.maxByOrNull{ shape -> shape.perimeter }!!

    val minPerimeterShape
        get() = shapes.minByOrNull{ shape -> shape.perimeter }!!

    val totalArea
        get() = shapes.sumByDouble { it.area }

    val totalPerimeter
        get() = shapes.sumByDouble { it.perimeter }

    override fun equals(other: Any?): Boolean {
        if (other == null || other !is ShapeAccumulator ||
                shapes != other.shapes) return false
        return true
    }

    override fun hashCode() = shapes.hashCode()

    fun forEach(action: (Shape) -> Unit) = shapes.forEach { action(it) }
}
```

Вывод.

В ходе выполнения лабораторной работы был реализован класс ShapeAccumulator, а также программа, демонстрирующая возможности класса.

Лабораторная работа №5 «Работа с файлами».

Задание.

Используя код из ЛБ 2, научится сохранять список фигур (List<Shape>) в файл, а также читать фигуры из этого списка. Программа должна корректно обрабатывать ошибки файловой системы, например, если файла не оказалось или его не удалось открыть на чтение/запись

Исходный код.

```
fun jsonFormat() = Json {
    serializersModule = SerializersModule {
        polymorphic(Shape::class) {
            subclass(Circle::class)
            subclass(Rectangle::class)
            subclass(Square::class)
            subclass(Triangle::class)
        }
    }
}

fun deserializeShapes(format: Json, jsonData: String) : List<Shape> {
    return
format.decodeFromString(ListSerializer(PolymorphicSerializer(Shape::class)),
jsonData)
}

fun serializeShapes(format: Json, shapes: List<Shape>) : String{
    return
format.encodeToString(ListSerializer(PolymorphicSerializer(Shape::class)),
shapes)
}

fun main() {
    val format = jsonFormat()
    val shapes = mutableListOf<Shape>()
    val file = File("shapes.json")

    if (file.exists()) {
        val shapesJson: String
        try {
            shapesJson = file.readText()
        } catch (ex: Exception) {
            println("Error while reading data from file: ${ex.message}")
            return
        }
        val shapesFromFile = deserializeShapes(format, shapesJson)
        shapes.addAll(shapesFromFile)
        println("Read data: $shapes")
    } else if (!file.createNewFile()) {
        println("Cannot create shapes.json file!")
        return
    }

    val newShapes =  mutableListOf(
            Circle(Random.nextDouble(0.0, 10.0)),
```

```
            Triangle(3.0, 4.0, 5.0),
            Square(Random.nextDouble(0.0, 10.0)),
            Rectangle(Random.nextDouble(0.0, 10.0), Random.nextDouble(0.0,
10.0))
    )
    shapes.addAll(newShapes)
    val shapesJson = serializeShapes(format, shapes)
    try {
        file.writeText(shapesJson)
    } catch (ex: Exception) {
        println("Error while saving serialized data to file: ${ex.message}")
    }
    println("Saved data: $shapes")

}
```

Вывод.

В ходе выполнения лабораторной работы была реализована программа, умеющая сохранять список фигур в файл, а также читать фигуры из этого списка.

Лабораторная работа №6 «Графический интерфейс».

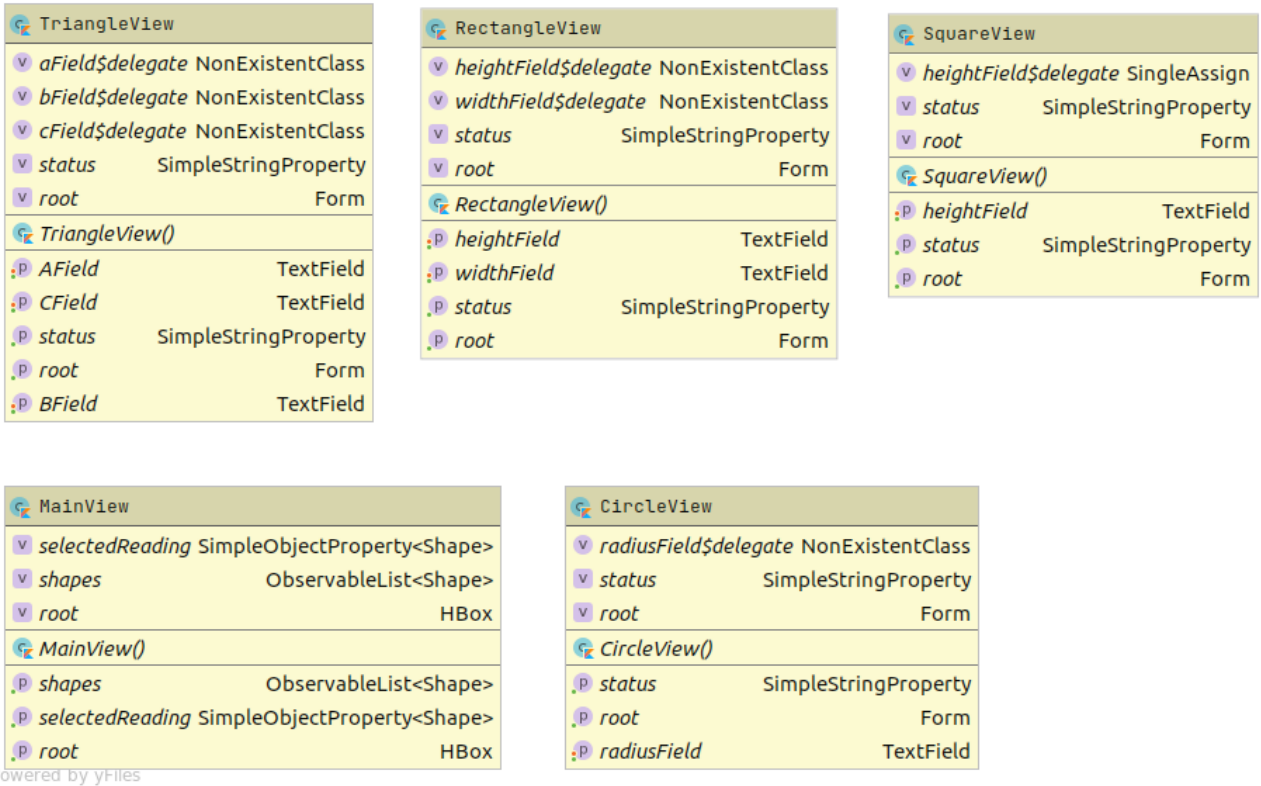
Задание.

Используя код ЛБ 2 и 5, реализовать графический интерфейс (на Java Swing или другом фреймворке на вашем ЯП), который будет предоставлять для пользователя следующие действия:

- Смотреть все фигуры (в виде списка, например)
- Удалять выбранную в списке фигуру
- Добавлять фигуры каждого типа
- Менять фигуры в списке местами

При запуске программа должна читать все фигуры из файла (ЛБ 5) а при закрытии гарантировать, что все фигуры из списка будут сохранены в файле и доступны на следующем запуске.

UML-диаграмма.



Исходный код.

```
fun jsonFormat() = Json {
    serializersModule = SerializersModule {
        polymorphic(Shape::class) {
            subclass(Circle::class)
            subclass(Rectangle::class)
            subclass(Square::class)
            subclass(Triangle::class)
        }
    }
}

fun deserializeShapes(format: Json, jsonData: String) : List<Shape> {
    return
format.decodeFromString(ListSerializer(PolymorphicSerializer(Shape::class)),
jsonData)
}

fun serializeShapes(format: Json, shapes: List<Shape>) : String{
    return
format.encodeToString(ListSerializer(PolymorphicSerializer(Shape::class)),
shapes)
}

fun main() {
    val format = jsonFormat()
    val shapes = mutableListOf<Shape>()
    val file = File("shapes.json")

    if (file.exists()) {
        val shapesJson: String
        try {
            shapesJson = file.readText()
        } catch (ex: Exception) {
            println("Error while reading data from file: ${ex.message}")
            return
        }
        val shapesFromFile = deserializeShapes(format, shapesJson)
        shapes.addAll(shapesFromFile)
        println("Read data: $shapes")
    } else if (!file.createNewFile()) {
        println("Cannot create shapes.json file!")
        return
    }

    val newShapes =  mutableListOf(
            Circle(Random.nextDouble(0.0, 10.0)),
            Triangle(3.0, 4.0, 5.0),
            Square(Random.nextDouble(0.0, 10.0)),
            Rectangle(Random.nextDouble(0.0, 10.0), Random.nextDouble(0.0,
10.0))
    )
    shapes.addAll(newShapes)
    val shapesJson = serializeShapes(format, shapes)
    try {
        file.writeText(shapesJson)
    } catch (ex: Exception) {
        println("Error while saving serialized data to file: ${ex.message}")
    }
    println("Saved data: $shapes")

}
```

Вывод.

В ходе выполнения лабораторной работы была реализована программа с графическим интерфейсом умеющая создавать, сохранять и читать фигуры из файла.