Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

О Т Ч Е Т

по лабораторной работе №3

Название: Классы. Наследование. Полиморфизм

Дисциплина: Языки программирования для работы с большими данными

Студент	ИУ6-22М		И.Л. Баришпол
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			П.В. Степанов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2023

Задания

3. Определить класс Вектор в R3. Реализовать методы для проверки векторов на ортогональность, проверки пересечения не ортогональных векторов, сравнения векторов. Создать массив из m объектов. Определить, какие из векторов компланарны.

```
package lab3

/**
 * Определить класс Вектор в R3. Реализовать методы для проверки векторов
 * - на ортогональность,
 * - проверки пересечения не ортогональных векторов,
 * - сравнения векторов.
 * Создать массив из m объектов. Определить, какие из векторов компланарны.
 */
class Vector(private val _x: Double = 0.0, private val _y: Double = 0.0, private val _z: Double = 0.0) {
    /** Возвращает скалярное произведение этого вектора на другой */
    fun dot(other: Vector): Double {
        return _x * other._x + _y * other._y + _z * other._z
    }

    /** Возвращает векторное произведение этого вектора на другой */
    fun cross(other: Vector): Vector {
        val cx = _y * other._z - _z * other._y
        val cy = _z * other._x - _x * other._z
        val cz = _x * other._y - _y * other._x
        return Vector(cx, cy, cz)
    }

    /** Возвращает, является ли этот вектор ортогональным с другим */
    fun isOrthogonal(other: Vector): Boolean {
        return dot(other) == 0.0
```

```
        }

        /** Возвращает, пересекаются ли этот вектор и другой */
        fun intersects(other: Vector): Boolean {
            return !cross(other).isEqual(Vector(0.0, 0.0, 0.0))
        }

        /** Возвращает, равны ли этот вектор и другой */
        fun isEqual(other: Vector): Boolean {
            return _x == other._x && _y == other._y && _z ==
other._z
        }

        /** Возвращает, компланарны ли 3 вектора */
        fun isCoplanarWith(v1: Vector, v2: Vector): Boolean {
            val n = v1.cross(v2)
            return n.dot(this) == 0.0
        }

        override fun toString(): String {
            return "(${_x}, ${_y}, ${_z})"
        }

        companion object {
            /** Возвращает, компланарные вектора */
            fun      findCoplanar(vectors:      Array<Vector>):
ArrayList<Set<Vector>> {
                val arr = ArrayList<Set<Vector>>()
                for (i in 0 until vectors.size - 2) {
                    for (j in i + 1 until vectors.size - 1) {
                        val            coplanarSet            =
mutableSetOf<Vector>(vectors[i], vectors[j])
                        for (k in j + 1 until vectors.size) {
                            if
(vectors[i].isCoplanarWith(vectors[j], vectors[k]))
                                coplanarSet.add(vectors[k])
```

```
                        }
                        arr.add(coplanarSet)
                    }
                }
                return removeRepeated(arr)
            }

            private fun removeRepeated(arr:
ArrayList<Set<Vector>>): ArrayList<Set<Vector>> {
                for (i in 0 until arr.size - 1) {
                    for (j in i + 1 until arr.size) {
                        while (j < arr.size &&
arr[i].containsAll(arr[j])) {
                            arr.removeAt(j)
                        }
                    }
                }
                return arr
            }
        }
    }

    fun main() {
        val vectors = arrayOf(
            Vector(1.0, 1.0, 1.0),
            Vector(1.0, 2.0, 0.0),
            Vector(0.0, -1.0, 1.0),
            Vector(3.0, 3.0, 3.0)
        )
        val coplanarSet = Vector.findCoplanar(vectors)
        coplanarSet.forEach { set -> set.forEach { println(it) }
}
    }
```

4. Определить класс Матрица размерности (n x n). Класс должен содержать несколько конструкторов. Реализовать методы для сложения, вычитания, умножения

матриц. Объявить массив объектов. Создать методы, вычисляющие первую и вторую нормы матрицы

```
package lab3

import kotlin.math.sqrt

class Matrix(private val n: Int) {
    private val matrix: Array<DoubleArray> = Array(n) { DoubleArray(n) }

    override fun toString(): String {
        var s = ""
        for (i in 0 until n) {
            for (j in 0 until n) {
                s += "${matrix[i][j].toString()} "
            }
            s += "\n"
        }
        return s
    }

    constructor(n: Int, values: Array<DoubleArray>) : this(n) {
        for (i in 0 until n) {
            for (j in 0 until n) {
                matrix[i][j] = values[i][j]
            }
        }
    }

    constructor(n: Int, vararg values: Double) : this(n) {
        for (i in 0 until n) {
            for (j in 0 until n) {
                matrix[i][j] = values[i * n + j]
            }
        }
```

```
        }

        fun add(other: Matrix): Matrix {
            val result = Matrix(n)
            for (i in 0 until n) {
                for (j in 0 until n) {
                    result.matrix[i][j]  =  this.matrix[i][j]  +
other.matrix[i][j]
                }
            }
            return result
        }

        fun subtract(other: Matrix): Matrix {
            val result = Matrix(n)
            for (i in 0 until n) {
                for (j in 0 until n) {
                    result.matrix[i][j]  =  this.matrix[i][j]  -
other.matrix[i][j]
                }
            }
            return result
        }

        fun multiply(other: Matrix): Matrix {
            val result = Matrix(n)
            for (i in 0 until n) {
                for (j in 0 until n) {
                    for (k in 0 until n) {
                        result.matrix[i][j] += this.matrix[i][k]
* other.matrix[k][j]
                    }
                }
            }
            return result
        }
```

```
        fun firstNorm(): Double {
            var norm = 0.0
            for (j in 0 until n) {
                var sum = 0.0
                for (i in 0 until n) {
                    sum += Math.abs(matrix[i][j])
                }
                norm = Math.max(norm, sum)
            }
            return norm
        }

        fun secondNorm(): Double {
            var sum = 0.0
            for (i in 0 until n) {
                for (j in 0 until n) {
                    sum += matrix[i][j] * matrix[i][j]
                }
            }
            return sqrt(sum)
        }

        companion object {
            fun                withSmallestFirstNorm(matrixArray:
Array<Matrix>): Matrix {
                var smallestNorm = Double.MAX_VALUE
                var matrixWithSmallestNorm = Matrix(0)

                for (matrix in matrixArray) {
                    val norm = matrix.firstNorm()

                    if (norm < smallestNorm) {
                        smallestNorm = norm
                        matrixWithSmallestNorm = matrix
                    }
```

```
                }

                return matrixWithSmallestNorm
            }

            fun             withSmallestSecondNorm(matrixArray:
Array<Matrix>): Matrix {
                var smallestNorm = Double.MAX_VALUE
                var matrixWithSmallestNorm = Matrix(0)

                for (matrix in matrixArray) {
                    val norm = matrix.secondNorm()

                    if (norm < smallestNorm) {
                        smallestNorm = norm
                        matrixWithSmallestNorm = matrix
                    }
                }

                return matrixWithSmallestNorm
            }
        }
    }

    fun main() {
        val matrixArray = arrayOf(
            Matrix(3,   arrayOf(doubleArrayOf(1.0,   2.0,   0.0),
doubleArrayOf(4.0, 5.0, 6.0), doubleArrayOf(7.0, 8.0, 9.0))),
            Matrix(3, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0,
9.0),
            Matrix(3, 1.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0,
1.0),
        )

        println(Matrix.withSmallestFirstNorm(matrixArray))
        println(Matrix.withSmallestSecondNorm(matrixArray))
```

```
}
```

Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы setТип(), getТип(), toString(). Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль.

3. Patient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз. Создать массив объектов. Вывести: a) список пациентов, имеющих данный диагноз; b) список пациентов, номер медицинской карты у которых находится в заданном интервале.

```
package lab3

/**
 * Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы setТип(), getТип(), toString().
 * Определить дополнительно методы в классе, создающем массив объектов.
 * Задать критерий выбора данных и вывести эти данные на консоль.
 * 3. Patient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз.
 * Создать массив объектов. Вывести:
 * a) список пациентов, имеющих данный диагноз;
 * b) список пациентов, номер медицинской карты у которых находится в заданном интервале.
 */
class Patient(
    private var id: Int,
    private var firstName: String,
    private var lastName: String,
    private var patronymic: String,
    private var address: String,
    private var phoneNumber: String,
    private var medicalCardNumber: Int,
    private var diagnosis: String
) {
```

```
        override fun toString(): String {
            return    "Patient(id=$id,    lastName='$lastName',
firstName='$firstName',                patronymic='$patronymic',
address='$address',                  phoneNumber='$phoneNumber',
medicalCardNumber=$medicalCardNumber, diagnosis='$diagnosis')"
        }

        fun getId(): Int {
            return id
        }

        fun setId(v: Int) {
            this.id = v
        }

        fun getFirstName(): String {
            return firstName
        }

        fun setFirstName(v: String) {
            this.firstName = v
        }

        fun getLastName(): String {
            return lastName
        }

        fun setLastName(v: String) {
            this.lastName = v
        }

        fun getPatronymic(): String {
            return patronymic
        }
```

```
    fun setPatronymic(v: String) {
        this.patronymic = v
    }

    fun getAddress(): String {
        return address
    }

    fun setAddress(v: String) {
        this.address = v
    }

    fun getPhoneNumber(): String {
        return phoneNumber
    }

    fun setPhoneNumber(v: String) {
        this.phoneNumber = v
    }

    fun getMedicalCardNumber(): Int {
        return medicalCardNumber
    }

    fun setMedicalCardNumber(v: Int) {
        this.medicalCardNumber = v
    }

    fun getDiagnosis(): String {
        return diagnosis
    }

    fun setDiagnosis(v: String) {
        this.diagnosis = v
    }
}
```

```
    fun Array<Patient>.filterByDiagnosis(diagnosis: String): Array<Patient> {
        return this.filter { it.getDiagnosis() == diagnosis }.toTypedArray()
    }

    fun Array<Patient>.filterByMedicalCardNumber(start: Int, end: Int): Array<Patient> {
        return this.filter { it.getMedicalCardNumber() in start..end }.toTypedArray()
    }

    fun main() {
        val p = arrayOf(
            Patient(1, "Ivan", "Ivanov", "Ivanovich", "Moscow", "+79999999999", 2343, "cancer"),
            Patient(1, "Alex", "Petrov", "Igorevich", "Saint-Petersburg", "+79889999999", 1324, "hiv"),
            Patient(1, "Kirill", "Sidorov", "Petrivich", "Voronezh", "+79779999999", 5234, "lupus"),
            Patient(1, "Roman", "Lebedev", "Stepanovich", "Rostov", "+79669999999", 4576, "undefined"),
        )
        p.filterByDiagnosis("hiv").forEach { println(it.toString()) }
        println("\n")
        p.filterByMedicalCardNumber(4000, 6000).forEach { println(it.toString()) }

    }
```

4. Abiturient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки. Создать массив объектов. Вывести: a) список абитуриентов, имеющих неудовлетворительные оценки; b) список абитуриентов, средний балл у которых выше заданного; c) выбрать заданное число

n абитуриентов, имеющих самый высокий средний балл (вывести также полный список абитуриентов, имеющих полупроходной балл).

```
package lab3

/**
 * Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы setТип(), getТип(), toString().
 * Определить дополнительно методы в классе, создающем массив объектов.
 * Задать критерий выбора данных и вывести эти данные на консоль.
 * 4. Abiturient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки. Создать массив объектов. Вывести:
 * a) список абитуриентов, имеющих неудовлетворительные оценки;
 * b) список абитуриентов, средний балл у которых выше заданного;
 * c) выбрать заданное число n абитуриентов, имеющих самый высокий средний балл
 * (вывести также полный список абитуриентов, имеющих полупроходной балл).
 */
class Abiturient(
    private var id: Int,
    private var surname: String,
    private var firstName: String,
    private var patronymic: String,
    private var address: String,
    private var phone: String,
    private var grades: List<Int>
) {
    fun setId(id: Int) {
        this.id = id
    }

    fun setSurname(surname: String) {
```

```
    this.surname = surname
}

fun setFirstName(firstName: String) {
    this.firstName = firstName
}

fun setPatronymic(patronymic: String) {
    this.patronymic = patronymic
}

fun setAddress(address: String) {
    this.address = address
}

fun setPhone(phone: String) {
    this.phone = phone
}

fun setGrades(grades: List<Int>) {
    this.grades = grades
}

fun getId(): Int {
    return id
}

fun getSurname(): String {
    return surname
}

fun getFirstName(): String {
    return firstName
}

fun getPatronymic(): String {
```

```
            return patronymic
        }

        fun getAddress(): String {
            return address
        }

        fun getPhone(): String {
            return phone
        }

        fun getGrades(): List<Int> {
            return grades
        }

        fun getAverageGrade(): Double {
            return grades.average()
        }

        override fun toString(): String {
            return    "Abiturient(id=$id,    surname='$surname',
firstName='$firstName',                 patronymic='$patronymic',
address='$address', phone='$phone', grades=$grades)"
        }

        companion object {

        }
    }

    class AbiturArray(var abiturients: Array<Abiturient>) {
        fun filterByUnsatisfactoryGrades(): List<Abiturient> {
            return  abiturients.filter  {  it.getGrades().any  {
grade -> grade < 3 } }
        }
```

```
        fun filterByAverageGradeGreaterThan(average: Double):
List<Abiturient> {
            return abiturients.filter { it.getAverageGrade() >
average }
        }

        fun getTopNAbiturients(n: Int): List<Abiturient> {
            val semiPassingScore = 3.5
            val sortedAbiturients =
abiturients.sortedByDescending { it.getAverageGrade() }
            val topNAbiturients = sortedAbiturients.take(n)
            val semiPassingAbiturients =
                sortedAbiturients.filter { it.getAverageGrade()
>= semiPassingScore && it !in topNAbiturients }
            println("Semi-passing abiturients:
$semiPassingAbiturients")
            return topNAbiturients
        }
    }


    fun main() {
        val abit = arrayOf(
            Abiturient(1, "Ivan", "Ivanov", "Ivanovich",
"Moscow", "+79999999999", listOf(2, 3, 2, 4, 5)),
            Abiturient(1, "Alex", "Petrov", "Igorevich", "Saint-
Petersburg", "+79889999999", listOf(5, 4, 5, 5, 5)),
            Abiturient(1, "Kirill", "Sidorov", "Petrivich",
"Voronezh", "+79779999999", listOf(5, 3, 5, 4, 4)),
            Abiturient(1, "Roman", "Lebedev", "Stepanovich",
"Rostov", "+79669999999", listOf(4, 5, 2, 4, 5)),
        )

        val abiturs = AbiturArray(abit)
        abiturs.filterByUnsatisfactoryGrades().forEach {
println(it.toString()) }
```

```
        println("\n")
        abiturs.filterByAverageGradeGreaterThan(3.0).forEach    {
println(it.toString()) }
        println("\n")
        abiturs.getTopNAbiturients(3).forEach                   {
println(it.toString()) }
    }
```

Вывод: В данной лабораторной работе мы рассмотрели множество тем, связанных с объектно-ориентированным программированием на Java, включая классы, наследование и полиморфизм. Эта лабораторная работа обеспечила всестороннее понимание концепций объектно-ориентированного программирования на Java и практический опыт их реализации.