|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальный анализ больших**

**данных в системах поддержки принятия решений.**

**Отчет**

**по лабораторной работе № 3**

**Вариант № 12**

**Название:** Классы, наследование и полиморфизм

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Т.М. Курохтин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

**Цель:** освоить принципы ООП на языке программирования Java.

**Задание 1:** определить класс Вектор размерности n. Реализовать методы сложения, вычитания, умножения, инкремента, декремента, индексирования. Определить массив из m объектов. Каждую из пар векторов передать в методы, возвращающие их скалярное произведение и длины. Вычислить и вывести углы между векторами.

Код класса Vector:

class Vector(private val elements: MutableList<Double>) {

// Размерность вектора

private val size: Int

get() = elements.size

// Индексация

operator fun get(index: Int): Double = elements[index]

// Индексация для изменения

operator fun set(index: Int, value: Double) {

elements[index] = value

}

// Сложение векторов

operator fun plus(other: Vector): Vector {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return Vector((elements.indices).map { i -> elements[i] + other[i] }.toMutableList())

}

// Вычитание векторов

operator fun minus(other: Vector): Vector {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return Vector((elements.indices).map { i -> elements[i] - other[i] }.toMutableList())

}

// Умножение вектора на скаляр

operator fun times(scalar: Double): Vector {

return Vector(elements.map { it \* scalar }.toMutableList())

}

// Инкремент

operator fun inc(): Vector {

return this + Vector(MutableList(size) { 1.0 })

}

// Декремент

operator fun dec(): Vector {

return this - Vector(MutableList(size) { 1.0 })

}

// Скалярное произведение

fun dot(other: Vector): Double {

require(size == other.size) { "Vectors must be of the same size." }

return elements.indices.sumOf { i -> elements[i] \* other[i] }

}

// Длина вектора

private fun magnitude(): Double {

return sqrt(elements.sumOf { it \* it })

}

// Угол между векторами в градусах

fun angleWith(other: Vector): Double {

val dotProduct = this.dot(other)

val magnitudes = this.magnitude() \* other.magnitude()

return acos(dotProduct / magnitudes) \* (180 / PI)

}

}

Код функции Main:

fun main() {

val vectors = arrayOf(

Vector(mutableListOf(1.0, 2.0, 3.0)),

Vector(mutableListOf(4.0, 5.0, 6.0)),

Vector(mutableListOf(7.0, 8.0, 9.0))

)

for (i in vectors.indices) {

for (j in i + 1..<vectors.size) {

val v1 = vectors[i]

val v2 = vectors[j]

println("Угол между вектором ${i+1} и вектором ${j+1} - ${v1.angleWith(v2)} градусов")

}

}

}

Результат работы программы представлен на рисунке 1.

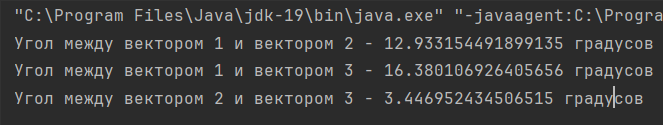


Рисунок 1 – Результат работы программы

**Задание 2:** определить класс Вектор размерности n. Определить несколько конструкторов. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Объявить массив объектов. Написать метод, который для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными.

Код класса Vector2:

class Vector2(private val elements: MutableList<Double>) {

// Размерность вектора

private val size: Int

get() = elements.size

// Индексация

operator fun get(index: Int): Double = elements[index]

// Индексация для изменения

operator fun set(index: Int, value: Double) {

elements[index] = value

}

// Конструктор, принимающий размерность вектора и инициализирующий нулями

constructor(size: Int) : this(MutableList(size) { 0.0 })

// Конструктор, принимающий массив значений

constructor(elements: Array<Double>) : this(elements.toMutableList())

// Сложение векторов

operator fun plus(other: Vector2): Vector2 {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return Vector2((elements.indices).map { i -> elements[i] + other[i] }.toMutableList())

}

// Вычитание векторов

operator fun minus(other: Vector2): Vector2 {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return Vector2((elements.indices).map { i -> elements[i] - other[i] }.toMutableList())

}

// Умножение вектора на скаляр

operator fun times(scalar: Double): Vector2 {

return Vector2(elements.map { it \* scalar }.toMutableList())

}

// Скалярное произведение

fun dot(other: Vector2): Double {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return elements.indices.sumOf { i -> elements[i] \* other[i] }

}

// Модуль вектора (длина)

fun magnitude(): Double {

return sqrt(elements.sumOf { it \* it })

}

// Проверка отношений между векторами

fun checkRelationship(other: Vector2): String {

if (this.dot(other) == 0.0) {

return "Ортогональные"

}

else {

var ratio: Double? = null

for (i in elements.indices) {

if (elements[i] != 0.0) {

if (other.elements[i] == 0.0) {

return "Не ортогональны и не коллинеарны"

}

val currentRatio = other.elements[i] / elements[i]

if (ratio == null) {

ratio = currentRatio

} else if (ratio != currentRatio) {

return "Не ортогональны и не коллинеарны"

}

} else if (other.elements[i] != 0.0) {

return "Не ортогональны и не коллинеарны"

}

}

return "Коллинеарны"

}

}

}

Код функции Main:

fun main() {

val vectors = arrayOf(

Vector2(arrayOf(1.0, 2.0, 3.0)),

Vector2(arrayOf(2.0, 4.0, 6.0)), // Коллинеарный с Vector 1

Vector2(arrayOf(0.0, 3.0, -1.0)),

Vector2(arrayOf(1.0, 1.0, -1.0)) // Ортогонален к v1

)

// Проверка пар векторов

for (i in vectors.indices) {

for (j in i + 1..<vectors.size) {

println("Отношение между вектором ${i+1} и вектором ${j+1}: ${vectors[i].checkRelationship(vectors[j])}")

}

}

}

Результат работы программы показан на рисунке 2.

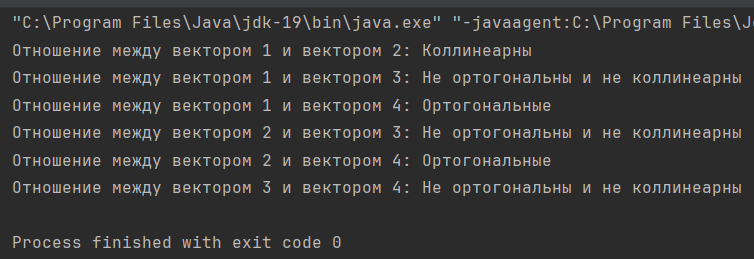


Рисунок 2 – Результат работы программы

**Задание 3:** Student: id, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Факультет, Курс, Группа. Создать массив объектов. Вывести: a) список студентов заданного факультета; b) списки студентов для каждого факультета и курса; c) список студентов, родившихся после заданного года; d) список учебной группы.

Код программы:

import java.time.LocalDate

import java.time.format.DateTimeFormatter

class Student(

private val id: Int,

private var lastName: String,

private var firstName: String,

private var middleName: String,

private var birthDate: LocalDate,

private var address: String,

private var phone: String,

private var faculty: String,

private var course: Int,

private var group: String

) {

fun setLastName(name: String) {

lastName = name

}

fun getLastName() = lastName

fun setFirstName(name: String) {

firstName = name

}

fun getFirstName() = firstName

fun setMiddleName(name: String) {

middleName = name

}

fun getMiddleName() = middleName

fun setBirthDate(date: LocalDate) {

birthDate = date

}

fun getBirthDate() = birthDate

fun setAddress(add: String) {

address = add

}

fun getAddress() = address

fun setPhone(num: String) {

phone = num

}

fun getPhone() = phone

fun setFaculty(fac: String) {

faculty = fac

}

fun getFaculty() = faculty

fun setCourse(crs: Int) {

course = crs

}

fun getCourse() = course

fun setGroup(grp: String) {

group = grp

}

fun getGroup() = group

override fun toString() = "{id=$id, name=$lastName $firstName $middleName, " +

"birthDate=${birthDate.format(DateTimeFormatter.ISO\_LOCAL\_DATE)}, " +

"address=$address, phone=$phone, faculty=$faculty, course=$course, group=$group}"

}

class Students(private val students: Array<Student>) {

fun getStudentsByFaculty(faculty: String): List<Student>{

return students.filter { it.getFaculty() == faculty }

}

fun getGroupedStudents(): Map<Pair<String, Int>, List<Student>>{

return students.groupBy { it.getFaculty() to it.getCourse() }

}

fun getStudentsBornAfter(year: Int): List<Student> {

return students.filter { it.getBirthDate().year > year }

}

fun getStudentsByGroup(group: String): List<Student> {

return students.filter { it.getGroup() == group }

}

}

fun main() {

val studentsArray = arrayOf(

Student(1, "Smith", "John", "Edward", LocalDate.of(1998, 6, 24), "123 Elm St", "555-1234", "Engineering", 3, "ENGR202"),

Student(2, "Doe", "Jane", "Marie", LocalDate.of(2000, 3, 14), "456 Maple St", "555-5678", "Arts", 2, "ARTS105"),

Student(3, "Brown", "Jim", "Thomas", LocalDate.of(1999, 8, 16), "789 Oak St", "555-8765", "Engineering", 3, "ENGR202"),

)

val students = Students(studentsArray)

// a) Список студентов заданного факультета

val faculty = "Engineering"

println("Students from the $faculty faculty:")

students.getStudentsByFaculty(faculty).forEach { println(it) }

// b) Списки студентов для каждого факультета и курса

val groupedByFacultyAndCourse = students.getGroupedStudents()

groupedByFacultyAndCourse.forEach { (key, list) ->

println("Faculty: ${key.first}, Course: ${key.second}")

list.forEach { println(it) }

}

// c) Список студентов, родившихся после заданного года

val year = 1999

println("Students born after $year:")

students.getStudentsBornAfter(year).forEach { println(it) }

// d) Список учебной группы

val group = "ENGR202"

println("Students in the group $group:")

students.getStudentsByGroup(group).forEach { println(it) }

}

Результат работы программы представлен на рисунке 3.

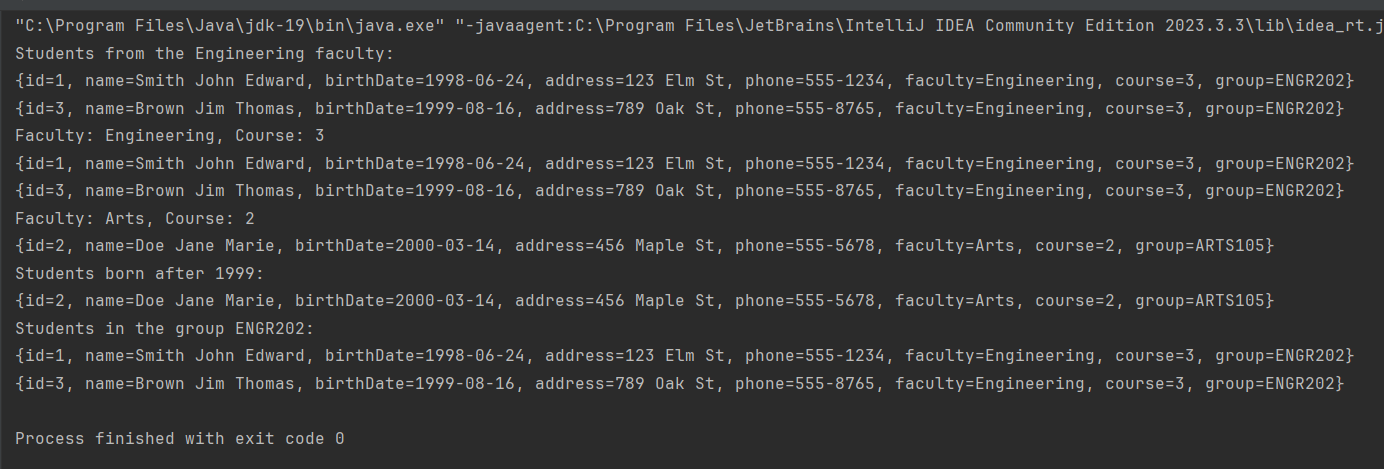
****

Рисунок 3 – Результат работы программы

**Задание 4:** Customer: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Номер банковского счета. Создать массив объектов. Вывести: a) список покупателей в алфавитном порядке; b) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

Код программы:

class Customer(

private val id: Int,

private var lastName: String,

private var firstName: String,

private var middleName: String,

private var address: String,

private var creditCardNumber: Long,

private var bankAccountNumber: Long

) {

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

fun getLastName() = lastName

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

fun getFirstName() = firstName

fun setMiddleName(middleName: String) { this.middleName = middleName }

fun getMiddleName() = middleName

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

fun getAddress() = address

fun setCreditCardNumber(creditCardNumber: Long) { this.creditCardNumber = creditCardNumber }

fun getCreditCardNumber() = creditCardNumber

fun setBankAccountNumber(bankAccountNumber: Long) { this.bankAccountNumber = bankAccountNumber }

fun getBankAccountNumber() = bankAccountNumber

override fun toString(): String {

return "{id=$id, lastName='$lastName', firstName='$firstName', middleName='$middleName', " +

"address='$address', creditCardNumber='$creditCardNumber', bankAccountNumber='$bankAccountNumber'}"

}

}

class Customers(private val customers: Array<Customer>){

fun sortCustomers(): List<Customer> {

return customers.sortedBy { it.getLastName() + it.getFirstName() + it.getMiddleName() }

}

fun getCustomersByCreditCardNumberInInterval(start: Long, end: Long): List<Customer>{

return customers.filter {

it.getCreditCardNumber() in (start + 1)..<end

}

}

}

fun main() {

val customersArray = arrayOf(

Customer(1, "Smith", "John", "Edward", "123 Elm St", 1234567890123456, 9876543210),

Customer(2, "Doe", "Jane", "Marie", "456 Maple St", 9876543210987654, 1234567890),

Customer(3, "Brown", "Jim", "Thomas", "789 Oak St", 5555666677778888, 1122334455)

)

val customers = Customers(customersArray)

// a) список покупателей в алфавитном порядке

val sortedCustomers = customers.sortCustomers()

println("Customers sorted by last name:")

sortedCustomers.forEach { println(it) }

// b) список покупателей с номерами кредитных карт в заданном интервале

val startCardNumber = 1234567890123450

val endCardNumber = 6234567890123456

val customersInRange = customers.getCustomersByCreditCardNumberInInterval(startCardNumber, endCardNumber)

println("\nCustomers with credit card numbers in the range $startCardNumber-$endCardNumber:")

customersInRange.forEach { println(it) }

}

Результат работы программы показан на рисунке 4.

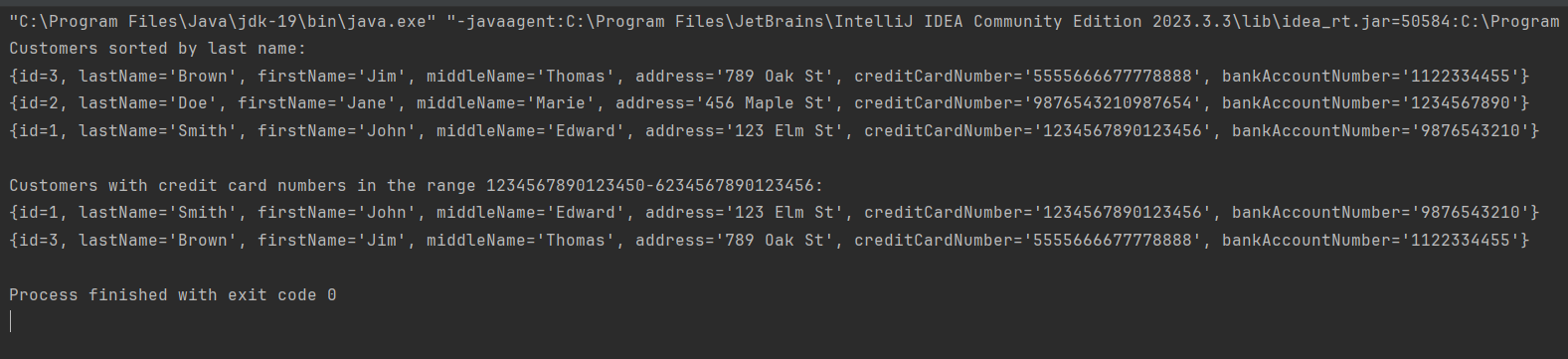


Рисунок 4 – Результат работы программы

**Вывод:** были освоены принципы ООП на языке программирования Java.