|  |  |
| --- | --- |
| 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 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных**

**Отчет**

**по лабораторной работе № 3**

**Вариант № 6**

**Название:** Классы, наследование и полиморфизм

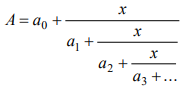
**Дисциплина:** Языки программирования для работы с большими данными

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

Москва, 2023

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

**Задание 1:** определить класс Цепная дробь



Определить методы сложения, вычитания, умножения, деления. Вычислить значение для заданного n, x, a[n].

Код класса ContinuedFraction:

class ContinuedFraction(private val coefficients: List<Double>, private var x: Double) {

fun evaluate( n: Int): Double {

var value = coefficients.getOrElse(n) { 0.0 }

for (i in n - 1 downTo 0) {

value = coefficients.getOrElse(i) { 0.0 } + x / value

}

return value

}

operator fun plus(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) + other.evaluate(other.coefficients.size - 1)

}

operator fun minus(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) - other.evaluate(other.coefficients.size - 1)

}

operator fun times(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) \* other.evaluate(other.coefficients.size - 1)

}

operator fun div(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) / other.evaluate(other.coefficients.size - 1)

}

}

Код функции Main:

fun main(){

println("Введите количество констант первой дроби")

val n1 = IOHelper.readIntMy(4)

println("Введите константы первой дроби")

val coefficientsFirst = IOHelper.arrayFloatInput(n1)

println("Введите количество констант второй дроби")

val n2 = IOHelper.readIntMy(4)

println("Введите константы второй дроби")

val coefficientsSecond = IOHelper.arrayFloatInput(n2)

val fractionFirst = ContinuedFraction(coefficientsFirst, 5.0)

val fractionSecond = ContinuedFraction(coefficientsSecond, 5.0)

println("Введите n")

val n = IOHelper.readIntMy(3)

println("Введите x")

val x = IOHelper.readIntMy(3)

println("Значение первой дроби при x = $x и n = $n - ${fractionFirst.evaluate(n)}")

println("Значение второй дроби при x = $x и n = $n - ${fractionSecond.evaluate(n)}")

println("Сложение дробей: ${fractionFirst + fractionSecond}")

println("Вычитание дробей: ${fractionFirst + fractionSecond}")

println("Умножение дробей: ${fractionFirst + fractionSecond}")

println("Деление дробей: ${fractionFirst + fractionSecond}")

}

Результат работы программы представлен на рисунке 1.

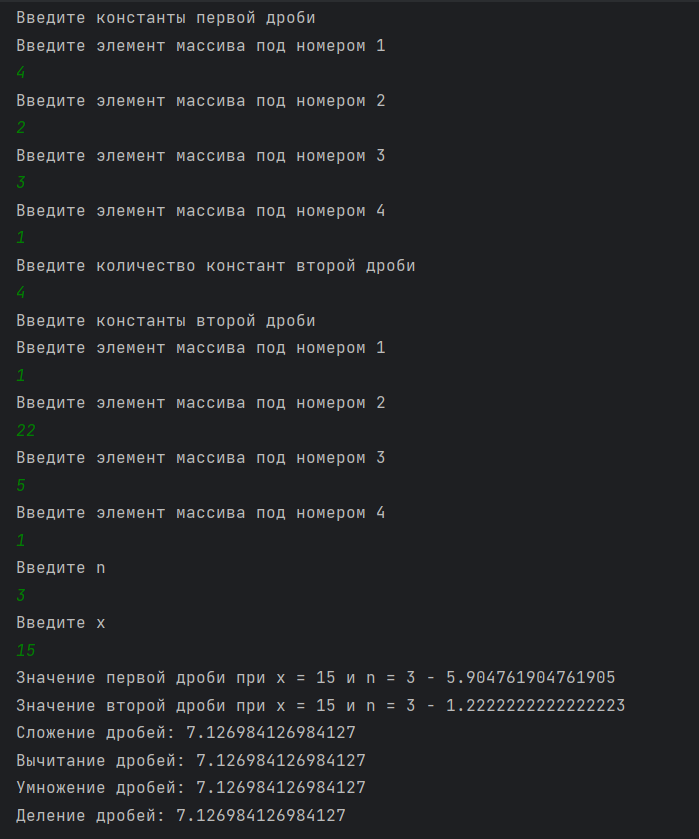


Рисунок 1 – Результат работы программы

**Задание 2:**  определить класс Дробь в виде пары (m,n). Класс должен содержать несколько конструкторов. Реализовать методы для сложения, вычитания, умножения и деления дробей. Объявить массив из k дробей, ввести/вывести значения для массива дробей. Создать массив объектов и передать его в метод, который изменяет каждый элемент массива с четным индексом путем добавления следующего за ним элемента массива.

Код класса Fraction:

data class Fraction(val numerator: Int, val denominator: Int) {

init {

require(denominator != 0) { "Знаменатель не может быть равен 0" }

}

operator fun plus(other: Fraction): Fraction {

if (this.denominator == other.denominator) {

return Fraction(this.numerator + other.numerator, this.denominator).simplify()

} else {

val commonDenominator = this.denominator \* other.denominator

val newNumerator = this.numerator \* other.denominator + other.numerator \* this.denominator

return Fraction(newNumerator, commonDenominator).simplify()

}

}

operator fun minus(other: Fraction): Fraction {

if (this.denominator == other.denominator) {

return Fraction(this.numerator - other.numerator, this.denominator).simplify()

} else {

val commonDenominator = this.denominator \* other.denominator

val newNumerator = this.numerator \* other.denominator - other.numerator \* this.denominator

return Fraction(newNumerator, commonDenominator).simplify()

}

}

operator fun times(other: Fraction): Fraction {

return Fraction(this.numerator \* other.numerator, this.denominator \* other.denominator).simplify()

}

operator fun div(other: Fraction): Fraction {

return Fraction(this.numerator \* other.denominator, this.denominator \* other.numerator).simplify()

}

fun simplify(): Fraction {

val gcd = gcd(this.numerator, this.denominator)

return Fraction(this.numerator / gcd, this.denominator / gcd)

}

private fun gcd(a: Int, b: Int): Int {

return if (b == 0) a else gcd(b, a % b)

}

override fun toString(): String = "$numerator/$denominator"

}

// Функция для изменения каждой дроби с четным индексом

fun modifyEvenIndexedFractions(fractions: Array<Fraction>) {

for (i in fractions.indices step 2) {

if (i + 1 < fractions.size) {

fractions[i] = fractions[i] + fractions[i + 1]

}

}

}

Код класса Main:

fun main() {

val fractions = arrayOf(Fraction(1, 2), Fraction(2, 3), Fraction(3, 4), Fraction(4, 5))

println("Исходный массив дробей: ${fractions.contentToString()}")

modifyEvenIndexedFractions(fractions)

println("Модифицированный массив дробей: ${fractions.contentToString()}")

println("Сложение первых двух дробей модифицированного массива: ${fractions[0] + fractions[1]}")

println("Вычитания первых двух дробей модифицированного массива: ${fractions[0] - fractions[1]}")

println("Умножение первых двух дробей модифицированного массива: ${fractions[0] \* fractions[1]}")

println("Деление первых двух дробей модифицированного массива: ${fractions[0] / fractions[1]}")

}

Результат работы программы показан на рисунке 2.

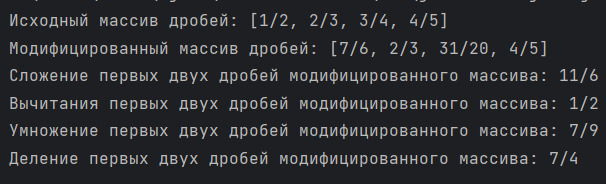


Рисунок 2 – Результат работы программы

**Задание 3:** House: id, Номер квартиры, Площадь, Этаж, Количество комнат, Улица, Тип здания, Срок эксплуатации. Создать массив объектов. Вывести: a) список квартир, имеющих заданное число комнат; b) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке; c) список квартир, имеющих площадь, превосходящую заданную.

Код классов Home и Homes:

class House(

private var id: Int,

private var apartmentNumber: Int,

private var area: Double,

private var floor: Int,

private var numberOfRooms: Int,

private var street: String,

private var buildingType: String,

private var operationTerm: Int

) {

fun getId() = id

fun setId(value: Int) {

id = value

}

fun getApartmentNumber() = apartmentNumber

fun setApartmentNumber(value: Int) {

apartmentNumber = value

}

fun getArea() = area

fun setArea(value: Double) {

area = value

}

fun getFloor() = floor

fun setFloor(value: Int) {

floor = value

}

fun getNumberOfRooms() = numberOfRooms

fun setNumberOfRooms(value: Int) {

numberOfRooms = value

}

fun getStreet() = street

fun setStreet(value: String) {

street = value

}

fun getBuildingType() = buildingType

fun setBuildingType(value: String) {

buildingType = value

}

fun getOperationTerm() = operationTerm

fun setOperationTerm(value: Int) {

operationTerm = value

}

override fun toString(): String {

return "House(id=$id, apartmentNumber=$apartmentNumber, area=$area, floor=$floor, " +

"numberOfRooms=$numberOfRooms, street='$street', buildingType='$buildingType', " +

"operationTerm=$operationTerm)"

}

}

class Houses(val houses: Array<House>){

fun listApartmentsWithRooms(rooms: Int) {

println("Список квартир с заданным числом комнат")

houses.filter { it.getNumberOfRooms() == rooms }

.forEach { println(it) }

}

fun listApartmentsWithRoomsAndFloor(rooms: Int, floorRange: IntRange) {

println("Список квартир с заданным числом комнат и этажом в промежутке")

houses.filter { it.getNumberOfRooms() == rooms && it.getFloor() in floorRange }

.forEach { println(it) }

}

fun listApartmentsWithAreaGreaterThan(area: Double) {

println("Список квартир с площадью превышающей заданную ")

houses.filter { it.getArea() > area }

.forEach { println(it) }

}

}

Код функции Main:

fun main() {

val houses = Houses(arrayOf(

House(1, 12, 55.0, 3, 2, "Green Street", "Apartment", 5),

House(2, 45, 68.0, 6, 3, "Sunset Boulevard", "Townhouse", 10),

House(3, 78, 42.0, 1, 1, "Oak Street", "Apartment", 2),

House(4, 10, 120.0, 9, 4, "Elm Street", "Penthouse", 15),

House(5, 2, 85.0, 5, 3, "Maple Avenue", "Condominium", 8)

))

houses.houses.forEach {

println(it)

}

println("Введите число комнат")

val roomsNum = IOHelper.readIntMy(4)

println("Введите промежуток этажей")

val floorRangeFrom = IOHelper.readIntMy(4)

val floorRangeTo = IOHelper.readIntMy(4)

val floorRange = IntRange(floorRangeFrom, floorRangeTo)

println("Введите площадь")

val area = IOHelper.readDoubleMy(10.0)

houses.listApartmentsWithRooms(roomsNum)

houses.listApartmentsWithRoomsAndFloor(roomsNum, floorRange)

houses.listApartmentsWithAreaGreaterThan(10.0)

}

Результат работы программы представлен на рисунке 3.

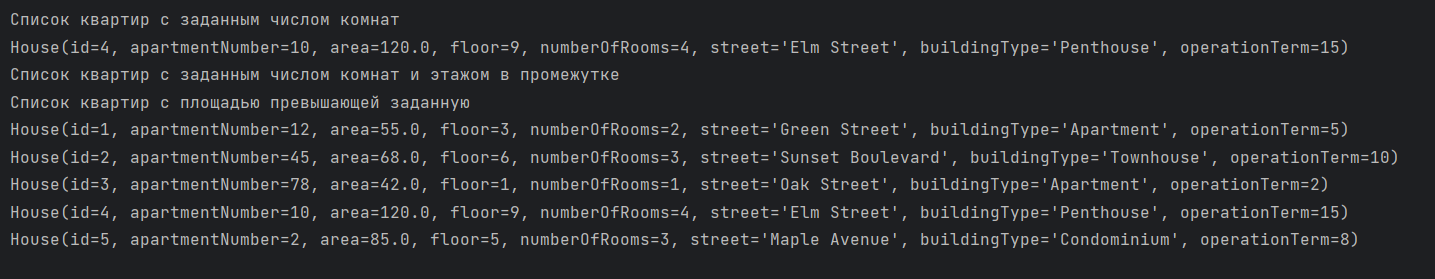
****

Рисунок 3 – Результат работы программы

**Задание 4:** Phone: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Дебет, Кредит, Время городских и междугородных разговоров. Создать массив объектов. Вывести: a) сведения об абонентах, у которых время внутригородских разговоров превышает заданное; b) сведения об абонентах, которые пользовались междугородной связью; c) сведения об абонентах в алфавитном порядке.

Код классов Phone и Phones:

class Phone(

private var id: Int,

private var lastName: String,

private var firstName: String,

private var middleName: String,

private var address: String,

private var creditCardNumber: String,

private var debit: Double,

private var credit: Double,

private var cityCallTime: Int,

private var intercityCallTime: Int

) {

fun getId() = id

fun setId(value: Int) {

id = value

}

fun getLastName() = lastName

fun setLastName(value: String) {

lastName = value

}

fun getFirstName() = firstName

fun setFirstName(value: String) {

firstName = value

}

fun getMiddleName() = middleName

fun setMiddleName(value: String) {

middleName = value

}

fun getAddress() = address

fun setAddress(value: String) {

address = value

}

fun getCreditCardNumber() = creditCardNumber

fun setCreditCardNumber(value: String) {

creditCardNumber = value

}

fun getDebit() = debit

fun setDebit(value: Double) {

debit = value

}

fun getCredit() = credit

fun setCredit(value: Double) {

credit = value

}

fun getCityCallTime() = cityCallTime

fun setCityCallTime(value: Int) {

cityCallTime = value

}

fun getIntercityCallTime() = intercityCallTime

fun setIntercityCallTime(value: Int) {

intercityCallTime = value

}

override fun toString(): String {

return "Phone(id=$id, lastName='$lastName', firstName='$firstName', middleName='$middleName', " +

"address='$address', creditCardNumber='$creditCardNumber', debit=$debit, " +

"credit=$credit, cityCallTime=$cityCallTime, intercityCallTime=$intercityCallTime)"

}

}

class Phones(val phones: Array<Phone>) {

fun listSubscribersWithHighCityCallTime(minCityCallTime: Int) {

println("Сведения об абонентах с временем внутригородских разговоров выше заданного")

phones.filter { it.getCityCallTime() > minCityCallTime }

.forEach { println(it) }

}

fun listSubscribersWithIntercityCalls() {

println("Сведения об абонентах, которые пользовались междугородной связью")

phones.filter { it.getIntercityCallTime() > 0 }

.forEach { println(it) }

}

fun listSubscribersAlphabetically() {

println("Сведения об абонентах в алфавитном порядке")

phones.sortedWith(compareBy { it.getLastName() + it.getFirstName() + it.getMiddleName() })

.forEach { println(it) }

}

}

Код класса Main:

fun main() {

val phonesArray = arrayOf(

Phone(1, "Ivanov", "Ivan", "Ivanovich", "Some Address 1", "1111222233334444", 100.0, 50.0, 60, 30),

Phone(2, "Petrov", "Petr", "Petrovich", "Some Address 2", "5555666677778888", 200.0, 40.0, 40, 50),

Phone(3, "Sidorov", "Sidr", "Sidrovich", "Some Address 3", "2222333344445555", 300.0, 60.0, 30, 10),

Phone(4, "Smirnov", "Smir", "Smirnovich", "Some Address 4", "6666777788889999", 400.0, 80.0, 20, 20),

Phone(5, "Kuznetsov", "Kuzma", "Kuzmich", "Some Address 5", "7777888899990000", 500.0, 100.0, 70, 40)

)

val phones = Phones(phonesArray)

println("Введите минимальное время внутригородских разговоров для фильтрации")

val minCityCallTime = IOHelper.readIntMy(40)

phones.listSubscribersWithHighCityCallTime(minCityCallTime)

println("Абоненты, использовавшие междугородную связь:")

phones.listSubscribersWithIntercityCalls()

println("Абоненты в алфавитном порядке:")

phones.listSubscribersAlphabetically()

}

Результат работы программы показан на рисунке 4.

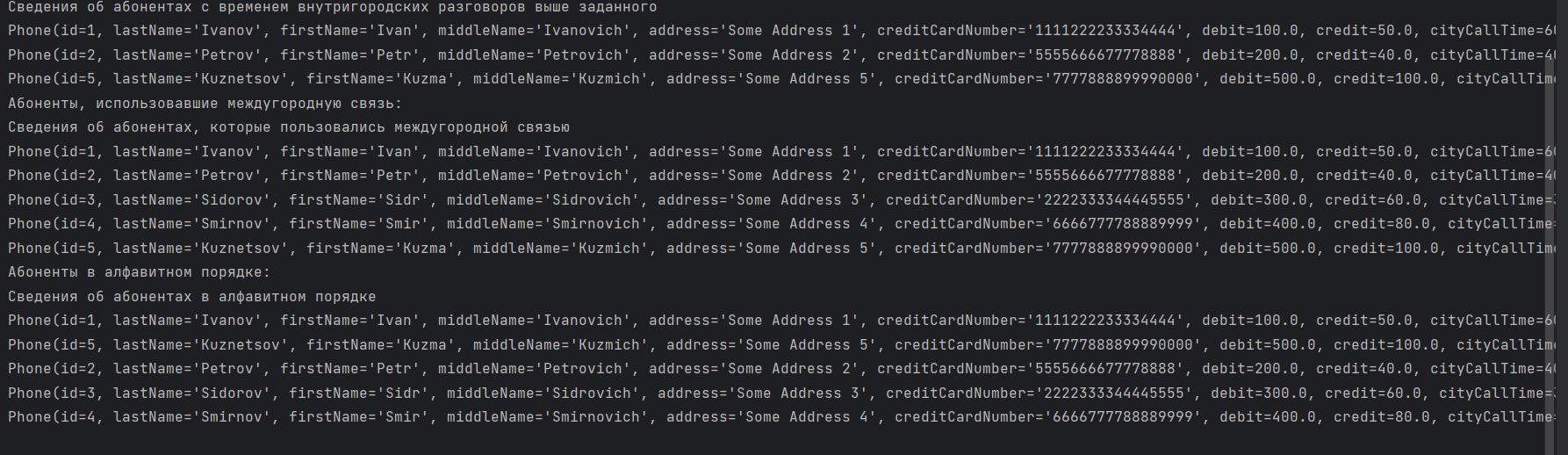


Рисунок 4 – Результат работы программы

**Задание 5:** создать объект класса Роза, используя классы Лепесток, Бутон. Методы: расцвести, завять, вывести на консоль цвет бутона.

Код Petal:

class Petal {

override fun equals(other: Any?): Boolean {

if (this === other) return true

if (javaClass != other?.javaClass) return false

return true

}

override fun hashCode(): Int {

return javaClass.hashCode()

}

override fun toString(): String {

return "Petal()"

}

}

Код Bud:

class Bud(private var color: String = "green", private var isOpen: Boolean = false) {

fun bloom() {

isOpen = true

color = "red"

}

fun wilt() {

isOpen = false

color = "brown"

}

fun printColor() {

println("Цвет бутона: $color")

}

override fun equals(other: Any?): Boolean {

if (this === other) return true

if (other == null || this::class != other::class) return false

other as Bud

if (color != other.color) return false

if (isOpen != other.isOpen) return false

return true

}

override fun hashCode(): Int {

var result = color.hashCode()

result = 31 \* result + isOpen.hashCode()

return result

}

override fun toString(): String {

return "Bud(color='$color', isOpen=$isOpen)"

}

}

Код Rose:

class Rose(private val bud: Bud, private val petals: List<Petal>) {

fun bloom() {

bud.bloom()

}

fun wilt() {

bud.wilt()

}

fun printBudColor() {

bud.printColor()

}

override fun equals(other: Any?): Boolean {

if (this === other) return true

if (other == null || this::class != other::class) return false

other as Rose

if (bud != other.bud) return false

if (petals != other.petals) return false

return true

}

override fun hashCode(): Int {

var result = bud.hashCode()

result = 31 \* result + petals.hashCode()

return result

}

override fun toString(): String {

return "Rose(bud=$bud, petals=$petals)"

}

}

Код Main:

fun main() {

val rose1 = Rose(Bud("green", false), listOf(Petal(), Petal()))

val rose2 = Rose(Bud("green", false), listOf(Petal(), Petal()))

val rose3 = Rose(Bud("red", true), listOf(Petal()))

println(rose1.toString())

println(rose2.toString())

println(rose3.toString())

println("HashCode rose1: ${rose1.hashCode()}")

println("HashCode rose2: ${rose2.hashCode()}")

println("HashCode rose3: ${rose3.hashCode()}")

println("rose1 == rose2: ${rose1 == rose2}")

println("rose1 == rose3: ${rose1 == rose3}")

rose1.bloom()

println("После расцвета rose1: $rose1")

println("rose1 == rose2 после изменения состояния rose1: ${rose1 == rose2}")

rose3.wilt()

println("После заведания rose3: $rose3")

rose1.printBudColor()

rose2.printBudColor()

rose3.printBudColor()

}

Результаты работы программы представлены на рисунке 5:

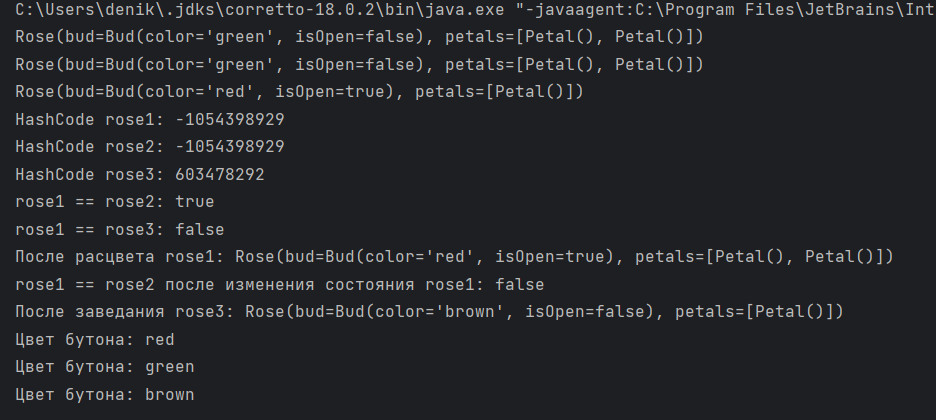


Рисунок 5 – Результаты работы программы

**Задание 6:** создать объект класса Дерево, используя классы Лист. Методы: зацвести, опасть листьям, покрыться инеем, пожелтеть листьям.

Код класса Leaf:

class Leaf(private var color: String = "green") {

fun bloom() {

color = "pink"

}

fun fall() {

color = "brown"

}

fun frost() {

color = "white"

}

fun yellow() {

color = "yellow"

}

override fun equals(other: Any?): Boolean {

if (this === other) return true

if (other == null || javaClass != other.javaClass) return false

other as Leaf

return color == other.color

}

override fun hashCode(): Int {

return color.hashCode()

}

override fun toString(): String {

return "Leaf(color='$color')"

}

}

Код класса Tree:

class Tree(private val leaves: MutableList<Leaf> = mutableListOf()) {

fun bloom() {

leaves.forEach { it.bloom() }

}

fun fallLeaves() {

leaves.clear()

}

fun frost() {

leaves.forEach { it.frost() }

}

fun yellow() {

leaves.forEach { it.yellow() }

}

override fun equals(other: Any?): Boolean {

if (this === other) return true

if (other == null || javaClass != other.javaClass) return false

other as Tree

return leaves == other.leaves

}

override fun hashCode(): Int {

return leaves.hashCode()

}

override fun toString(): String {

return "Tree(leaves=$leaves)"

}

}

Код функции Main:

fun main() {

val tree = Tree(mutableListOf(Leaf(), Leaf(), Leaf()))

println("Исходное состояние дерева: $tree")

tree.bloom()

println("После зацветания: $tree")

tree.yellow()

println("Листья пожелтели: $tree")

tree.frost()

println("Листья покрылись инеем: $tree")

tree.fallLeaves()

println("Листья опали: $tree")

}

Результат работы программы представлен на рисунке 6:

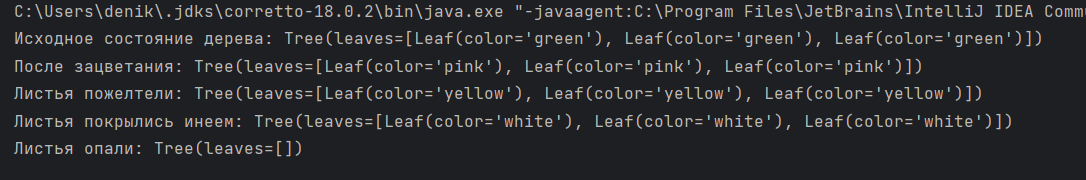


Рисунок 6 – Результат работы программы

**Задание 7:** система Конструкторское бюро. Заказчик представляет Техническое Задание (ТЗ) на проектирование многоэтажного Дома. Конструктор регистрирует ТЗ, определяет стоимость проектирования и строительства, выставляет Заказчику Счет за проектирование и создает Бригаду Конструкторов для выполнения Проекта.

Код класса TechnicalSpecification:

data class TechnicalSpecification(val requirements: String)

Код класса Client:

data class Client(val name: String)

Код класса Invoice:

data class Invoice(val amount: Double, val client: Client)

Код класса DesignTeam:

data class DesignTeam(val members: List<String>)

Код класса Project:

data class Project(val specification: TechnicalSpecification, val cost: Double)

Код класса Designer:

class Designer {

fun registerSpecification(specification: TechnicalSpecification): Project {

val projectCost = calculateCost(specification)

return Project(specification, projectCost)

}

fun calculateCost(specification: TechnicalSpecification): Double {

val baseCost = 50000.0

val costPerWord = 100.0

val wordCount = specification.requirements.split(" ").size

return baseCost + (wordCount \* costPerWord)

}

fun issueInvoice(client: Client, project: Project): Invoice {

return Invoice(project.cost, client)

}

fun createDesignTeam(): DesignTeam {

return DesignTeam(listOf("Designer 1", "Designer 2"))

}

}

Код функции Main:

fun main() {

val client = Client("Заказчик 1")

val designer = Designer()

val specification = TechnicalSpecification("Требования к дому")

val project = designer.registerSpecification(specification)

val invoice = designer.issueInvoice(client, project)

val team = designer.createDesignTeam()

println("Создан проект: $project")

println("Выставлен счет: $invoice")

println("Создана бригада конструкторов: $team")

}

Результат работы программы представлен на рисунке 7:

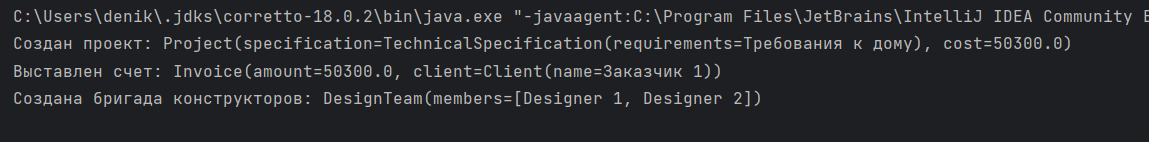


Рисунок 7 – Результаты работы программы

**Задание 8:** система Телефонная станция. Абонент оплачивает Счет за разговоры и Услуги, может попросить Администратора сменить номер и отказаться от услуг. Администратор изменяет номер, Услуги и временно отключает Абонента за неуплату.

Код класса Subscriber:

data class Subscriber(var name: String, var phoneNumber: String, var services: MutableList<Service> = mutableListOf()) {

fun payBill(bill: Bill) {

println("$name оплатил счет на сумму ${bill.amount}")

}

fun requestNumberChange(newNumber: String, administrator: Administrator) {

administrator.changePhoneNumber(this, newNumber)

}

fun cancelService(service: Service, administrator: Administrator) {

administrator.changeServices(this, service, false)

}

}

Код класса Bill:

data class Bill(val amount: Double, val subscriber: Subscriber)

Код класса Administrator:

class Administrator {

fun changePhoneNumber(subscriber: Subscriber, newNumber: String) {

subscriber.phoneNumber = newNumber

println("Номер абонента ${subscriber.name} изменен на $newNumber")

}

fun changeServices(subscriber: Subscriber, service: Service, addService: Boolean) {

if (addService) {

subscriber.services.add(service)

println("Услуга ${service.name} добавлена абоненту ${subscriber.name}")

} else {

if (subscriber.services.remove(service))

println("Услуга ${service.name} отменена для абонента ${subscriber.name}")

else

println("Услуга не подключена")

}

}

fun temporarilyDisconnect(subscriber: Subscriber) {

println("Абонент ${subscriber.name} временно отключен за неуплату")

}

}

Код класса Service:

data class Service(val name: String)

Код функции Main:

fun main() {

val service = Service("Интернет")

val subscriber = Subscriber("Иван Иванов", "89123456789", mutableListOf(service))

println(subscriber)

val bill = Bill(1500.0, subscriber)

val administrator = Administrator()

subscriber.payBill(bill)

subscriber.requestNumberChange("89123456700", administrator)

subscriber.cancelService(service, administrator)

administrator.changeServices(subscriber, service, true)

administrator.temporarilyDisconnect(subscriber)

}

Результат работы программы представлен на рисунке 8:

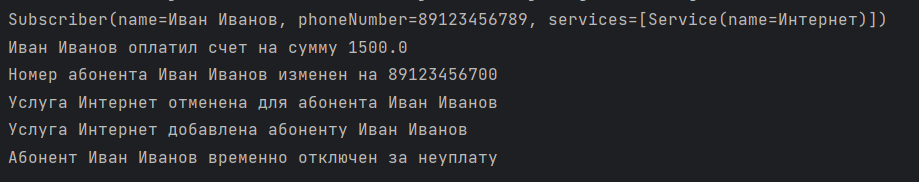


Рисунок 8 – Результат работы программы

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