ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

старший преподаватель		В. А. Ушаков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Разработка классов, наследование и полиморфизм

по курсу: ИТ-модуль "Разработка мобильных приложений"

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4142		К.С. Некрасов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Вариант 57

$(4142 + 14) \% 82 + 1 = 57$

Задание

Общее задание

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Наследование применять только в тех заданиях, в которых это логически обосновано. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString().

Задание по варианту

Создать объект класса Дерево, используя классы Лист, Ветка. Методы: зацвести, опасть листьям, покрыться инеем, пожелтеть листьям.

Листинг кода

Main.kt

```
fun main(args: Array<String>) {
    // делаем ствол
    val trunk = Branch(10f, 1f)
    // делаем ветки
    val branches = mutableListOf<Branch>()
    for (i in 1..3) {
        branches.add(Branch(1f, .1f, trunk))
    }

    // делаем дерево
    val tree = Tree(”Берёза”, trunk, branches)

    // выводим информацию о дереве
    println(”Дерево до цветения:”)
    println(tree)
    println(tree.branches)

    // делаем листья
    tree.bloom()

    // выводим информацию о дереве
```

```
    println(”Дерево после цветения:”)
    println(tree)
    println(tree.branches)

    // делаем листья жёлтыми
    tree.turnLeavesYellow()

    // выводим информацию о дереве
    println(”Дерево после желтения:”)
    println(tree)
    println(tree.branches)

    // сбрасываем листья
    tree.dropLeaves()

    // выводим информацию о дереве
    println(”Дерево после сброса листьев:”)
    println(tree)
    println(tree.branches)

    // замораживаем дерево
    tree.frostbite()

    // выводим информацию о дереве
    println(”Дерево после заморозки:”)
    println(tree)
    println(tree.branches)
}
```

Tree.kt

```
class Tree(val name: String, val trunk: Branch, val branches: List<Bra
    fun dropLeaves() {
        trunk.dropLeaves()
        branches.forEach { it.dropLeaves() }
    }

    fun turnLeavesYellow() {
        trunk.turnLeavesYellow()
        branches.forEach { it.turnLeavesYellow() }
    }
```

```
    fun bloom() {
        trunk.growLeaves()
        branches.forEach { it.growLeaves() }
    }

    fun frostbite() {
        trunk.frostbite()
        branches.forEach { it.frostbite() }
    }

    override fun toString(): String {
        return "Дерево: $name, " +
                "толщина ствола: ${trunk.thickness}, " +
                "высота: ${trunk.length}, " +
                "количество веток: ${branches.size}, " +
           "количество листьев: ${branches.sumOf { it.leavesCount() }
    }

    override fun equals(other: Any?): Boolean {
        if (other is Tree) {
        return name == other.name && trunk == other.trunk && branches =
        }
        return false
    }

    override fun hashCode(): Int {
        var result = name.hashCode()
        result = 31 * result + trunk.hashCode()
        result = 31 * result + branches.hashCode()
        return result
    }
}
```

Leaf.kt

```
class Leaf(private var initialBranch: Branch) {

    private var currentBranch: Branch? = initialBranch

    private var color = "зелёный"
```

```
    init {
        currentBranch?.addLeaf(this)
    }

    fun turnYellow() {
        color = ”жёлтый”
    }

    fun fall() {
        currentBranch = null
    }

    override fun equals(other: Any?): Boolean {
        if (other is Leaf) {
        return currentBranch == other.currentBranch && color == other.
        }
        return false
    }

    override fun hashCode(): Int {
        var result = currentBranch.hashCode()
        result = 31 * result + color.hashCode()
        return result
    }

    override fun toString(): String {
        return ”Лист: цвет $color”
    }
}
```

Branch.kt

```
class Branch(val length: Float, val thickness: Float, val parent: Bran
    private val leaves: MutableList<Leaf> = mutableListOf();
    private var frostbitten = false

    fun addLeaf(leaf: Leaf) {
        leaves.add(leaf)
    }
```

```
fun dropLeaves() {
    leaves.forEach { it.fall() }
    leaves.clear()
}

fun turnLeavesYellow() {
    leaves.forEach { it.turnYellow() }
}

fun growLeaves() {
    for (i in 1..10) {
        Leaf(this)
    }
}

fun leavesCount(): Int {
    return leaves.size
}

fun frostbite() {
    frostbitten = true
}

override fun toString(): String {
    val parentInfo = if (parent != null) {
    "ветка растёт из ветки длиной ${parent.length} и толщиной ${pa
    } else {
        "эта ветка - ствол"
    }

    val isFrostbitten = if (frostbitten) {
        "заиндевела"
    } else {
        "в норме"
    }

    val leavesInfo = if (leaves.size > 0) {
    "количество листьев: ${leaves.size}, листья: $leaves"
    } else {
        "листьев нет"
```

```
        }

        return "Ветка: $parentInfo, " +
                "длина: $length, " +
                "толщина: $thickness, " +
                "состояние: $isFrostbitten, " +
                leavesInfo
    }

    override fun equals(other: Any?): Boolean {
        if (other is Branch) {
        return length == other.length && thickness == other.thickness
        }
        return false
    }

    override fun hashCode(): Int {
        var result = length.hashCode()
        result = 31 * result + thickness.hashCode()
        result = 31 * result + leaves.hashCode()
        return result
    }
}
```

Результат работы программы