Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Программирование мобильных информационных систем

ОТЧЁТ по лабораторной работе №4 на тему:

**«Объектно-ориентированное программирование (ООП)»**

Выполнил: Проверил:

ст. гр. 214301 Усенко Ф.В.

Кузнецов Е.В.

Минск 2024

**Цель работы:** изучить принципы объектно-ориентированного программирования (ООП) в Kotlin, включая классы, объекты, наследование, полиморфизм, интерфейсы и абстрактные классы. Научиться создавать и использовать собственные классы, а также применять принципы ООП на практике.

**Вариант 13. Композитный объект с динамическими компонентами:** Напишите класс Computer, который состоит из объектов CPU, GPU, Memory, Storage, и PowerSupply. Реализуйте методы для динамического изменения конфигурации компьютера (например, замена процессора или увеличение объема памяти) и оценки его производительности.

# КОД ПРОГРАММЫ

import java.util.Scanner  
  
data class CPU(val name: String, val cores: Int, val frequencyGHz: Double)  
data class GPU(val name: String, val memoryGB: Int)  
data class Memory(val capacityGB: Int)  
data class Storage(val type: String, val capacityGB: Int)  
data class PowerSupply(val powerWatts: Int)  
  
class Computer(  
 var cpu: CPU,  
 var gpu: GPU,  
 var memory: Memory,  
 var storage: Storage,  
 var powerSupply: PowerSupply  
) {  
 fun replaceCPU(newCPU: CPU) {  
 *println*("Замена процессора на ${newCPU.name}")  
 cpu = newCPU  
 }  
  
 fun replaceGPU(newGPU: GPU) {  
 *println*("Замена видеокарты на ${newGPU.name}")  
 gpu = newGPU  
 }  
  
 fun upgradeMemory(additionalGB: Int) {  
 memory = memory.copy(capacityGB = memory.capacityGB + additionalGB)  
 *println*("Оперативная память увеличена до ${memory.capacityGB} ГБ")  
 }  
  
 fun replaceStorage(newStorage: Storage) {  
 *println*("Замена хранилища на ${newStorage.type} объемом ${newStorage.capacityGB} ГБ")  
 storage = newStorage  
 }  
  
 fun replacePowerSupply(newPowerSupply: PowerSupply) {  
 *println*("Замена блока питания на модель мощностью ${newPowerSupply.powerWatts} Вт")  
 powerSupply = newPowerSupply  
 }  
  
 fun evaluatePerformance(): Double {  
 val cpuScore = cpu.cores \* cpu.frequencyGHz  
 val gpuScore = gpu.memoryGB \* 2  
 val memoryScore = memory.capacityGB / 4.0  
 val storageScore = if (storage.type == "SSD") 1.5 else 1.0  
 val totalScore = (cpuScore + gpuScore + memoryScore) \* storageScore  
 *println*("Текущая оценка производительности: %.2f".*format*(totalScore))  
 return totalScore  
 }  
  
 override fun toString(): String {  
 return """  
 Текущая конфигурация компьютера:  
 Процессор: ${cpu.name}, ${cpu.cores} ядра(-ер) @ ${cpu.frequencyGHz} ГГц  
 Видеокарта: ${gpu.name}, ${gpu.memoryGB} ГБ  
 Оперативная память: ${memory.capacityGB} ГБ  
 Хранилище: ${storage.type}, ${storage.capacityGB} ГБ  
 Блок питания: ${powerSupply.powerWatts} Вт  
 """.*trimIndent*()  
 }  
}  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 val computer = Computer(  
 cpu = CPU("Intel i5", 6, 3.5),  
 gpu = GPU("NVIDIA GTX 1660", 6),  
 memory = Memory(16),  
 storage = Storage("SSD", 512),  
 powerSupply = PowerSupply(550)  
 )  
  
 while (true) {  
 *println*(  
 """  
 --- Меню конфигурации компьютера ---  
 1. Посмотреть текущую конфигурацию  
 2. Заменить процессор  
 3. Заменить видеокарту  
 4. Увеличить оперативную память  
 5. Заменить хранилище  
 6. Заменить блок питания  
 7. Оценить производительность  
 8. Выйти из программы  
 """.*trimIndent*()  
 )  
 *print*("Выберите действие: ")  
 when (scanner.nextInt()) {  
 1 -> *println*(computer)  
 2 -> {  
 *print*("Введите название процессора: ")  
 scanner.nextLine()  
 val name = scanner.nextLine()  
 *print*("Введите количество ядер: ")  
 val cores = scanner.nextInt()  
 *print*("Введите частоту (ГГц): ")  
 val frequency = scanner.nextDouble()  
 computer.replaceCPU(CPU(name, cores, frequency))  
 }  
 3 -> {  
 *print*("Введите название видеокарты: ")  
 scanner.nextLine()  
 val name = scanner.nextLine()  
 *print*("Введите объем памяти (ГБ): ")  
 val memory = scanner.nextInt()  
 computer.replaceGPU(GPU(name, memory))  
 }  
 4 -> {  
 *print*("Введите объем памяти ОЗУ для добавления (ГБ): ")  
 val additionalGB = scanner.nextInt()  
 computer.upgradeMemory(additionalGB)  
 }  
 5 -> {  
 *print*("Введите тип хранилища (SSD/HDD): ")  
 scanner.nextLine()  
 val type = scanner.nextLine()  
 *print*("Введите объем хранилища (ГБ): ")  
 val capacity = scanner.nextInt()  
 computer.replaceStorage(Storage(type, capacity))  
 }  
 6 -> {  
 *print*("Введите мощность БП (Вт): ")  
 val power = scanner.nextInt()  
 computer.replacePowerSupply(PowerSupply(power))  
 }  
 7 -> computer.evaluatePerformance()  
 8 -> {  
 *println*("Выход.")  
 break  
 }  
 else -> *println*("Неверный выбор. Попробуйте снова.")  
 }  
 }  
}

# ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое класс в Kotlin, и как он объявляется?

Класс в Kotlin — это шаблон для создания объектов. Он может содержать свойства (переменные) и методы (функции).  
**Объявление класса:**

class Person {

var name: String = ""

var age: Int = 0

}

1. Как создать объект класса в Kotlin? Приведите пример.

Создание объекта осуществляется с помощью ключевого слова val или var и вызова конструктора класса.  
Пример:

val person = Person()

person.name = "John"

person.age = 30

println("${person.name} is ${person.age} years old")

1. Что такое свойства класса, и как их объявить в Kotlin?

Свойства класса — это переменные, определённые внутри класса. Они могут быть var (изменяемые) или val (только для чтения).

1. Как объявить и использовать метод класса? Приведите пример.

Методы класса — это функции, объявленные внутри класса.  
Пример:

kotlin

Копировать код

class Calculator {

fun add(a: Int, b: Int): Int {

return a + b

}

}

fun main() {

val calculator = Calculator()

println(calculator.add(5, 3)) // 8

}

1. Что такое первичный конструктор, и как он используется для инициализации свойств класса?

Первичный конструктор объявляется сразу после имени класса. Он используется для инициализации свойств класса.

# ВЫВОД

В ходы выполнения лабораторной работы были изучены принципы объектно-ориентированного программирования (ООП) в Kotlin, включая классы, объекты, наследование, полиморфизм, интерфейсы и абстрактные классы.