Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчёт

К лабораторной работе № 4

на тему:

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)**

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Усенко Ф. В.

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Грицков

(подпись) 214302

Минск 2024

**Вариант 6. Композитный объект с динамическими компонентами**

Условие: 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 org.knowm.xchart.SwingWrapper  
import org.knowm.xchart.XYChartBuilder  
import org.knowm.xchart.style.markers.None  
  
class Computer(  
 var cpu: CPU,  
 var gpu: GPU,  
 var memory: Memory,  
 var storage: Storage,  
 var powerSupply: PowerSupply  
) {  
 private val performanceHistory = mutableListOf(evaluatePerformanceScore())  
 private val chart = XYChartBuilder().width(800).height(600).title("Performance Over Time").xAxisTitle("Step").yAxisTitle("Performance").build()  
 private val chartWrapper = SwingWrapper(chart)  
  
 init {  
 chart.addSeries("Performance", listOf(0), performanceHistory).marker = None()  
 chartWrapper.displayChart()  
 }  
  
 private fun evaluatePerformanceScore(): Double {  
 val cpuScore = cpu.cores \* cpu.frequency  
 val gpuScore = gpu.vram \* 10  
 val memoryScore = memory.size  
 val storageScore = if (storage.type == "SSD") storage.size \* 2 else storage.size  
 return cpuScore + gpuScore + memoryScore + storageScore  
 }  
  
 private fun updatePerformanceHistory() {  
 performanceHistory.add(evaluatePerformanceScore())  
 chart.updateXYSeries("Performance", (0 until performanceHistory.size).toList(), performanceHistory, null)  
 chartWrapper.repaintChart()  
 }  
  
 fun replaceCPU(newCPU: CPU) {  
 cpu = newCPU  
 println("\nCPU replaced with ${cpu.model}")  
 updatePerformanceHistory()  
 }  
  
 fun replaceGPU(newGPU: GPU) {  
 gpu = newGPU  
 println("\nGPU replaced with ${gpu.model}")  
 updatePerformanceHistory()  
 }  
  
 fun upgradeMemory(extraMemory: Int) {  
 memory.size += extraMemory  
 println("\nMemory increased to ${memory.size} GB")  
 updatePerformanceHistory()  
 }  
  
 fun replaceStorage(newStorage: Storage) {  
 storage = newStorage  
 println("\nStorage replaced with ${storage.type} of ${storage.size} GB")  
 updatePerformanceHistory()  
 }  
  
 fun evaluatePerformance(): String {  
 val totalScore = evaluatePerformanceScore()  
 return when {  
 totalScore > 100 -> "\nHigh performance"  
 totalScore > 50 -> "\nMedium performance"  
 else -> "\nLow performance"  
 }  
 }  
  
 fun displayConfiguration() {  
 println()  
 println("Computer Configuration:")  
 println(cpu)  
 println(gpu)  
 println(memory)  
 println(storage)  
 println(powerSupply)  
 }  
}  
  
class CPU(val model: String, val cores: Int, val frequency: Double) {  
 override fun toString(): String {  
 return "CPU: $model, $cores cores, $frequency GHz"  
 }  
}  
  
class GPU(val model: String, val vram: Int) {  
 override fun toString(): String {  
 return "GPU: $model, $vram GB VRAM"  
 }  
}  
  
class Memory(var size: Int) {  
 override fun toString(): String {  
 return "Memory: $size GB"  
 }  
}  
  
class Storage(val type: String, val size: Int) {  
 override fun toString(): String {  
 return "Storage: $type, $size GB"  
 }  
}  
  
class PowerSupply(val wattage: Int) {  
 override fun toString(): String {  
 return "Power Supply: $wattage W"  
 }  
}  
  
fun main() {  
 println("---COMPUTER BUILD---:")  
  
 val cpuModel = promptString("Enter CPU model:")  
 val cpuCores = promptInt("Enter number of CPU cores:")  
 val cpuFrequency = promptDouble("Enter CPU frequency (in GHz):")  
 val gpuModel = promptString("Enter GPU model:")  
 val gpuVram = promptInt("Enter VRAM size (in GB):")  
 val memorySize = promptInt("Enter memory size (in GB):")  
 val storageType = promptStorageType("Enter storage type (SSD or HDD):")  
 val storageSize = promptInt("Enter storage size (in GB):")  
 val powerWattage = promptInt("Enter power supply wattage (in W):")  
  
 val myComputer = Computer(  
 cpu = CPU(cpuModel, cpuCores, cpuFrequency),  
 gpu = GPU(gpuModel, gpuVram),  
 memory = Memory(memorySize),  
 storage = Storage(storageType, storageSize),  
 powerSupply = PowerSupply(powerWattage)  
 )  
  
 myComputer.displayConfiguration()  
 println("Initial performance evaluation: ${myComputer.evaluatePerformance()}")  
  
 while (true) {  
 println("\nChoose a component to replace or enter '5' to exit:")  
 println("1 - Replace CPU")  
 println("2 - Replace GPU")  
 println("3 - Upgrade memory")  
 println("4 - Replace storage")  
 println("5 - Exit")  
  
 when (readLine()) {  
 "1" -> {  
 val newCpuModel = promptString("Enter new CPU model:")  
 val newCpuCores = promptInt("Enter new number of cores:")  
 val newCpuFrequency = promptDouble("Enter new CPU frequency:")  
 myComputer.replaceCPU(CPU(newCpuModel, newCpuCores, newCpuFrequency))  
 }  
 "2" -> {  
 val newGpuModel = promptString("Enter new GPU model:")  
 val newGpuVram = promptInt("Enter new VRAM size (in GB):")  
 myComputer.replaceGPU(GPU(newGpuModel, newGpuVram))  
 }  
 "3" -> {  
 val extraMemory = promptInt("Enter amount of memory to add (in GB):")  
 myComputer.upgradeMemory(extraMemory)  
 }  
 "4" -> {  
 val newStorageType = promptStorageType("Enter new storage type (SSD or HDD):")  
 val newStorageSize = promptInt("Enter new storage size (in GB):")  
 myComputer.replaceStorage(Storage(newStorageType, newStorageSize))  
 }  
 "5" -> {  
 println("Exiting program.")  
 break  
 }  
 else -> println("Invalid choice. Please try again.")  
 }  
  
 println("Updated performance evaluation: ${myComputer.evaluatePerformance()}")  
 myComputer.displayConfiguration()  
 }  
}  
  
fun promptString(message: String): String {  
 while (true) {  
 println(message)  
 val input = readLine()  
 if (!input.isNullOrBlank()) return input  
 println("Error: Input cannot be empty.")  
 }  
}  
  
fun promptInt(message: String): Int {  
 while (true) {  
 println(message)  
 val input = readLine()?.toIntOrNull()  
 if (input != null) return input  
 println("Error: Please enter an integer.")  
 }  
}  
  
fun promptDouble(message: String): Double {  
 while (true) {  
 println(message)  
 val input = readLine()?.toDoubleOrNull()  
 if (input != null) return input  
 println("Error: Please enter a number.")  
 }  
}  
  
fun promptStorageType(message: String): String {  
 while (true) {  
 println(message)  
 val input = readLine()?.uppercase()  
 if (input == "SSD" || input == "HDD") return input  
 println("Error: Enter 'SSD' or 'HDD'.")  
 }  
}

**Контрольные вопросы:**

**Контрольные вопросы**

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

– Класс — это шаблон для создания объектов, объединяющих данные и методы для работы с этими данными. В *Kotlin* класс объявляется с помощью ключевого слова *class*.

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

*val* объект = ПримерКласса()

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

– Свойства класса — это переменные, принадлежащие классу. Их можно объявить внутри класса с использованием ключевых слов *val* (для неизменяемых свойств) или *var* (для изменяемых).

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

– Методы класса объявляются внутри класса с использованием ключевого слова *fun*.

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

– Первичный конструктор объявляется прямо в заголовке класса и используется для инициализации свойств класса

1. Как в *Kotlin* создать вторичный конструктор, и зачем он может понадобиться?

– Вторичный конструктор объявляется внутри класса с помощью ключевого слова *constructor* и может использоваться для создания объектов с разными наборами параметров.

1. Что такое наследование, и как его реализовать в *Kotlin*? Приведите пример.

– Наследование позволяет создать новый класс на основе существующего. В *Kotlin* для этого используется ключевое слово *open* для базового класса и : для наследования

1. Как переопределить метод базового класса в подклассе? Приведите пример.

– Метод базового класса переопределяется с помощью ключевого слова *override*.

1. В чем разница между интерфейсом и абстрактным классом в *Kotlin*?

– Интерфейс определяет контракт, который должны реализовывать классы, и не содержит состояния. Абстрактный класс может содержать реализацию методов и состояние. В *Kotlin* интерфейсы объявляются с помощью ключевого слова *interface*, а абстрактные классы — с помощью *abstract*.

1. Как реализовать множественное наследование через интерфейсы? Приведите пример.

class Duck : Flyable, Swimmable

1. Что такое абстрактный класс, и как объявить абстрактный метод?

– Абстрактный класс — это класс, который не может быть инстанциирован и может содержать абстрактные методы без реализации. Абстрактные методы объявляются с помощью ключевого слова *abstract*.

1. Как реализовать полиморфизм в *Kotlin*? Приведите пример использования.

– Полиморфизм — это возможность использовать один интерфейс или класс для работы с различными типами объектов. В *Kotlin* полиморфизм достигается с помощью наследования и интерфейсов. Полиморфизм позволяет вызывать методы, которые могут иметь разные реализации в разных классах

1. Что такое компаньон-объект, и как его использовать для создания единственного экземпляра класса?

– Компаньон-объект (*companion* *object*) в *Kotlin* — это специальный объект, который создается внутри класса и может содержать методы и свойства, которые можно вызывать без создания экземпляра этого класса. Компаньон-объект предоставляет возможность реализовать паттерн *Singleton*, что позволяет гарантировать существование только одного экземпляра класса.

1. Как в *Kotlin* реализовать инкапсуляцию данных? Приведите пример с использованием приватных полей.

– Это достигается с помощью модификаторов доступа, таких как *private*, *protected*, и *public*.

1. Как объединить объекты разных классов в один объект? Приведите пример.

– Объединение объектов разных классов может быть выполнено с использованием составного паттерна проектирования (*Composite* *Pattern*) или через использование коллекций (например, списков или множеств) для хранения объектов

1. Как переопределить метод *toString* в классе для предоставления настраиваемого строкового представления объекта?

– *@override fun toString() {System.out.println(“….”)}*

1. Как создать класс с несколькими конструкторами, использующимися в разных случаях?

– Используя перегруженные конструкторы.

1. Как реализовать событие в классе с использованием интерфейса в *Kotlin*?

– Используя интерфейсы и делегаты. Интерфейс будет описывать событие, а класс-издатель будет уведомлять подписчиков о возникновении этого события.

1. В чем разница между статическим методом в *Java* и методом компаньон-объекта в *Kotlin*?

– Компаньон-объект вызывается через имя класса или напрямую, может быть переопределен и имеет доступ как к статическим, так и к нестатическим полям