Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Учебная дисциплина «Современные языки программирования»

Отчёт

по лабораторной работе №4

«Использование языка программирования Swift: наследование, протоколы»

Вариант 15

Выполнил студент группы 310101

Криницкий М.А.

Проверил:

Усенко Ф.В.

Минск 2024

Цель работы: выполнить разработку приложения с использованием языка программирования Swift: наследование, протоколы.

Класс может наследовать методы, свойства и другие характеристики другого класса. Когда один класс наследует у другого класса, то наследующий класс называется подклассом, класс у которого наследуют - суперклассом. Наследование - фундаментальное поведение, которое отделяет классы от других типов Swift.

Протокол определяет образец методов, свойств или другие требования, которые соответствуют определенному конкретному заданию или какой-то функциональности. Протокол фактически не предоставляет реализацию для любого из этих требований, он только описывает как реализация должна выглядеть. Протокол может быть принят классом, структурой или перечислением для обеспечения фактической реализации этих требований. Любой тип, который удовлетворяет требованиям протокола, имеет указание соответствовать этому протоколу или другими словами реализовать данный протокол.

Задание:

Протокол Создания уникальных чисел. Реализовать метод, который генерирует уникальное число. Создать класс, который будет иметь в себе генератор. Добавить несколько различных алгоритмов создания уникальных чисел. Создать несколько уникальных последовательностей чисел. (Листинг 1)

Листинг 1

import Foundation

func generateRandomNumber(min: Int, max: Int) -> Int {

// Генерация случайного числа на основе текущего времени

let randomSeed = Int(Date().timeIntervalSince1970 \* 1000) % (max - min + 1)

return min + randomSeed

}

// Протокол для генерации уникальных чисел

protocol UniqueNumberGenerator {

func generateUniqueNumber() -> Int

}

// Генератор случайных чисел

class RandomNumberGenerator: UniqueNumberGenerator {

private var generatedNumbers: Set<Int> = []

private let min: Int = 1

private let max: Int = 1000

func generateUniqueNumber() -> Int {

var uniqueNumber: Int

repeat {

uniqueNumber = generateRandomNumber(min: min, max: max) // Генерация числа от 1 до 1000

} while generatedNumbers.contains(uniqueNumber)

generatedNumbers.insert(uniqueNumber)

return uniqueNumber

}

}

class SequentialNumberGenerator: UniqueNumberGenerator {

private var currentNumber: Int = 0

func generateUniqueNumber() -> Int {

currentNumber += 1

return currentNumber

}

}

class FibonacciNumberGenerator: UniqueNumberGenerator {

private var previous: Int = 0

private var current: Int = 1

func generateUniqueNumber() -> Int {

let nextNumber = previous + current

previous = current

current = nextNumber

return nextNumber

}

}

class UniqueNumberGeneratorClient {

private var generator: UniqueNumberGenerator

init(generator: UniqueNumberGenerator) {

self.generator = generator

}

func generateUniqueNumbers(count: Int) -> [Int] {

guard count > 0 else {

fatalError("count должен быть больше 0")

}

var uniqueNumbers: Set<Int> = []

while uniqueNumbers.count < count {

let number = generator.generateUniqueNumber()

uniqueNumbers.insert(number)

}

return Array(uniqueNumbers)

}

}

let randomGenerator = RandomNumberGenerator()

let randomClient = UniqueNumberGeneratorClient(generator: randomGenerator)

let randomUniqueNumbers = randomClient.generateUniqueNumbers(count: 5)

print("Случайные уникальные числа: \(randomUniqueNumbers)")

let sequentialGenerator = SequentialNumberGenerator()

let sequentialClient = UniqueNumberGeneratorClient(generator: sequentialGenerator)

let sequentialUniqueNumbers = sequentialClient.generateUniqueNumbers(count: 5)

print("Последовательные уникальные числа: \(sequentialUniqueNumbers)")

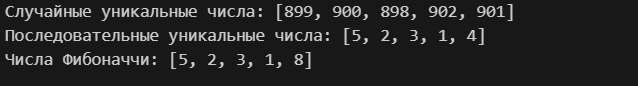
let fibonacciGenerator = FibonacciNumberGenerator()

let fibonacciClient = UniqueNumberGeneratorClient(generator: fibonacciGenerator)

let fibonacciUniqueNumbers = fibonacciClient.generateUniqueNumbers(count: 5)

print("Числа Фибоначчи: \(fibonacciUniqueNumbers)")

Скриншоты работы программы представлены на рисунке 1.



**Вывод**

В лабораторной работе реализована система генерации уникальных чисел с использованием различных подходов: случайной генерации, последовательной генерации и генерации чисел Фибоначчи. Протокол UniqueNumberGenerator определяет интерфейс для генераторов, что обеспечивает легкость добавления новых типов. Классы-генераторы следят за уникальностью создаваемых чисел, а клиентский класс UniqueNumberGeneratorClient позволяет запрашивать заданное количество уникальных чисел. Эта архитектура демонстрирует принципы инкапсуляции и полиморфизма, обеспечивая гибкость и удобство модификации системы.