Министерство образования

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Отчет по лабораторной работе №4

«Использование языка программирования Swift: наследование, протоколы»

Выполнил: Михович И.С.

Студент группы 310902

Проверил: Усенко Ф.В.

Минск 2024

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

Задание. Класс рациональных дробей. класс, представляющий рациональную дробь (num – числитель, den – знаменатель). Класс содержит методы умножения и деления (дроби на дробь и дроби на целое число). Создать протокол, который служит для создания случайной дроби из заданного диапазона целых. Добавить две различные реализации протокола.

**Код на языке Swift:**

**Package.swift**

**// swift-tools-version: 6.0**

**import PackageDescription**

**let package = Package(**

**name: "lab4",**

**platforms: [.macOS(.v13)],**

**products: [**

**.executable(name: "16.4", targets: ["MainModule"]),**

**],**

**dependencies: [],**

**targets: [**

**.target(**

**name: "ValidationModule",**

**dependencies: []**

**),**

**.target(**

**name: "FractionsModule",**

**dependencies: ["ValidationModule"]**

**),**

**.executableTarget(**

**name: "MainModule",**

**dependencies: ["FractionsModule"]**

**)**

**]**

**)**

**FractionsModule\FractionOperations.swift**

**import Foundation**

**public struct FractionOperations {**

**public static func multiply(fraction1: Fraction, fraction2: Fraction) -> Fraction {**

**return try! RationalFraction(**

**numerator: fraction1.numerator \* fraction2.numerator,**

**denominator: fraction1.denominator \* fraction2.denominator**

**)**

**}**

**public static func divide(fraction1: Fraction, fraction2: Fraction) -> Fraction {**

**return try! RationalFraction(**

**numerator: fraction1.numerator \* fraction2.denominator,**

**denominator: fraction1.denominator \* fraction2.numerator**

**)**

**}**

**public static func multiplyByInteger(fraction: Fraction, integer: Int) -> Fraction {**

**return try! RationalFraction(**

**numerator: fraction.numerator \* integer,**

**denominator: fraction.denominator**

**)**

**}**

**public static func divideByInteger(fraction: Fraction, integer: Int) -> Fraction {**

**return try! RationalFraction(**

**numerator: fraction.numerator,**

**denominator: fraction.denominator \* integer**

**)**

**}**

**}**

**FractionsModule\RandomFractionGenerator.swift**

**import Foundation**

**public protocol RandomFractionGenerator {**

**func generateFraction() -> Fraction**

**}**

**public struct RandomPositiveFractionGenerator: RandomFractionGenerator {**

**private let range: ClosedRange<Int>**

**public init(range: ClosedRange<Int>) {**

**self.range = range**

**}**

**public func generateFraction() -> Fraction {**

**let numerator = Int.random(in: range)**

**var denominator: Int**

**repeat {**

**denominator = Int.random(in: range)**

**} while denominator == 0**

**return try! RationalFraction(numerator: numerator, denominator: denominator)**

**}**

**}**

**public struct RandomNegativeFractionGenerator: RandomFractionGenerator {**

**private let range: ClosedRange<Int>**

**public init(range: ClosedRange<Int>) {**

**self.range = range**

**}**

**public func generateFraction() -> Fraction {**

**let numerator = -Int.random(in: range)**

**var denominator: Int**

**repeat {**

**denominator = Int.random(in: range)**

**} while denominator == 0**

**return try! RationalFraction(numerator: numerator, denominator: denominator)**

**}**

**}**

**FractionsModule\RationalFraction.swift**

**import Foundation**

**import ValidationModule**

**public protocol Fraction {**

**var numerator: Int { get }**

**var denominator: Int { get }**

**func toDouble() -> Double**

**}**

**public struct RationalFraction: Fraction {**

**public let numerator: Int**

**public let denominator: Int**

**public init(numerator: Int, denominator: Int) throws {**

**try Validation.validateDenominator(denominator)**

**self.numerator = numerator**

**self.denominator = denominator**

**}**

**public func toDouble() -> Double {**

**return Double(numerator) / Double(denominator)**

**}**

**}**

**MainModule\main.swift**

**import Foundation**

**import FractionsModule**

**do {**

**let fraction1: Fraction = try RationalFraction(numerator: 1, denominator: 2)**

**let fraction2: Fraction = try RationalFraction(numerator: 3, denominator: 4)**

**let product = FractionOperations.multiply(fraction1: fraction1, fraction2: fraction2)**

**let quotient = FractionOperations.divide(fraction1: fraction1, fraction2: fraction2)**

**print("Результат умножения: \(product.numerator)/\(product.denominator)")**

**print("Результат деления: \(quotient.numerator)/\(quotient.denominator)")**

**let randomGenerator1: RandomFractionGenerator = RandomPositiveFractionGenerator(range: 1...10)**

**let randomFraction1 = randomGenerator1.generateFraction()**

**print("Случайная положительная дробь: \(randomFraction1.numerator)/\(randomFraction1.denominator)")**

**let randomGenerator2: RandomFractionGenerator = RandomNegativeFractionGenerator(range: 1...10)**

**let randomFraction2 = randomGenerator2.generateFraction()**

**print("Случайная отрицательная дробь: \(randomFraction2.numerator)/\(randomFraction2.denominator)")**

**} catch {**

**print("Ошибка: \(error)")**

**}**

**ValidationModule\Validation.swift**

**import Foundation**

**public struct Validation {**

**public static func validateDenominator(\_ denominator: Int) throws {**

**guard denominator != 0 else {**

**throw ValidationError.invalidDenominator**

**}**

**}**

**}**

**public enum ValidationError: Error, CustomStringConvertible {**

**case invalidDenominator**

**public var description: String {**

**switch self {**

**case .invalidDenominator:**

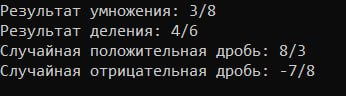
**return "Знаменатель не может быть равен 0."**

**}**

**}**

**}**

**Скриншоты работающей программы:**

****

**Рисунок 1 - Результат выполнения программы**

**Вывод: В этой лабораторной работе мы разработали приложение на языке программирования Swift, используя наследование и протоколы. Функционал приложения был разбит на несколько пакетов, учитывая логику работы, и была сделана валидация вводимых значений, работа программы происходила с экземплярами протоколов, а не экземплярами классов.**