Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

Современные языки программирования

Отчет по лабораторной работе №4

**«Использование языка программирования *Swift*: наследование, протоколы»**

Выполнил: Еремейко А. А.

Студент группы 310901

Проверил: Усенко Ф.В.

Минск 2024

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

**Вариант 2:** Создать класс, который проверяет число и определяет, что это – простое число, число Фибоначчи, комплексное число, целое, вещественное. Каждый метод относится к отдельному протоколу.

**Текст программы:**

protocol ComplexCheckable {

    func isComplex(real: Double, imaginary: Double) -> Bool

}

import Foundation

class ComplexChecker: ComplexCheckable {

    func isComplex(real: Double, imaginary: Double) -> Bool {

        // Комплексное число, если мнимая часть не равна нулю

        return imaginary != 0

    }

}

protocol FibonacciCheckable {

    func isFibonacci(\_ number: Int) -> Bool

}

import Foundation

class FibonacciChecker: FibonacciCheckable {

    func isFibonacci(\_ number: Int) -> Bool {

        func isPerfectSquare(\_ x: Int) -> Bool {

            let s = Int(sqrt(Double(x)))

            return s \* s == x

        }

        // Число Фибоначчи, если 5 \* n^2 + 4 или 5 \* n^2 - 4 — полные квадраты

        return isPerfectSquare(5 \* number \* number + 4) || isPerfectSquare(5 \* number \* number - 4)

    }

}

protocol FloatCheckable {

    func isFloat(\_ number: Double) -> Bool

}

import Foundation

class FloatChecker: FloatCheckable {

    func isFloat(\_ number: Double) -> Bool {

        // Вещественное число — это число с дробной частью

        return number != floor(number)

    }

}

protocol IntegerCheckable {

    func isInteger(\_ number: Double) -> Bool

}

import Foundation

class IntegerChecker: IntegerCheckable {

    func isInteger(\_ number: Double) -> Bool {

        // Целое число — это число, не имеющее дробной части

        return number == floor(number)

    }

}

protocol PrimeCheckable {

    func isPrime(\_ number: Int) -> Bool

}

import Foundation

class PrimeChecker: PrimeCheckable {

    func isPrime(\_ number: Int) -> Bool {

        if number <= 1 {

            return false

        }

        for i in 2..<Int(sqrt(Double(number)) + 1) {

            if number % i == 0 {

                return false

            }

        }

        return true

    }

}

import Foundation

print("\n")

// Создание экземпляров через протоколы

let primeChecker: PrimeCheckable = PrimeChecker()

let fibonacciChecker: FibonacciCheckable = FibonacciChecker()

let complexChecker: ComplexCheckable = ComplexChecker()

let integerChecker: IntegerCheckable = IntegerChecker()

let floatChecker: FloatCheckable = FloatChecker()

// Примеры использования

let number1 = 7

print("Число \(number1) простое? \(primeChecker.isPrime(number1))") // true

print("Число \(number1) является числом Фибоначчи? \(fibonacciChecker.isFibonacci(number1))") // true

let real = 3.0

let imaginary = 4.0

print("Число (\(real), \(imaginary)) комплексное? \(complexChecker.isComplex(real: real, imaginary: imaginary))") // true

let number2: Double = 5.0

print("Число \(number2) целое? \(integerChecker.isInteger(number2))") // true

print("Число \(number2) вещественное? \(floatChecker.isFloat(number2))") // false

let number3: Double = 5.5

print("Число \(number3) целое? \(integerChecker.isInteger(number3))") // false

print("Число \(number3) вещественное? \(floatChecker.isFloat(number3))") // true

Результат работы программы приведен на рисунке 1.

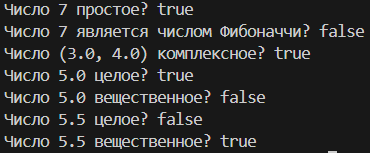


Рисунок 1 – Результат работы программы

**Ответы на вопросы к лабораторной работе:**

1. Что такое наследование?

Наследование в Swift — это механизм объектно-ориентированного программирования, который позволяет одному классу (наследнику) получать свойства, методы и функциональность другого класса (родителя). Это способствует повторному использованию кода, улучшает его структуру и уменьшает дублирование. Класс-наследник может добавлять новые свойства и методы, а также изменять (переопределять) поведение, унаследованное от родительского класса. В Swift классы могут наследоваться только от одного родительского класса.

2. Что такое переопределение? Что можно переопределять?

Переопределение — это процесс изменения поведения свойств, методов или инициализаторов, унаследованных от родительского класса. Переопределённый элемент сохраняет своё имя и сигнатуру, но получает новую реализацию. В Swift можно переопределять методы (включая функции экземпляра и функции класса); свойства (как вычисляемые свойства, так и свойства с наблюдателями); инициализаторы (с учётом правил безопасного наследования). Для переопределения используется ключевое слово override.

3. Что такое протоколы?

Протоколы в Swift — это шаблоны (контракты), которые определяют требования к методам, свойствам или другим характеристикам. Протокол сам по себе не реализует функциональность, а лишь указывает, что должна реализовать структура, класс или перечисление, которые принимают его. Протоколы используются для стандартизации поведения и обеспечения гибкости за счёт использования типов, соответствующих определённым требованиям.

4. Для чего служат протоколы?

Протоколы в Swift служат для:

1. Определения набора требований, которые должны быть реализованы классами, структурами или перечислениями.
2. Поддержки множественного наследования (один тип может соответствовать нескольким протоколам).
3. Создания абстракций, которые упрощают взаимодействие между различными частями программы.
4. Улучшения читаемости и переиспользования кода.

Протоколы часто применяются для делегирования и обеспечения единообразного поведения.

5. Что такое расширение протоколов?

Расширение протоколов позволяет добавлять реализацию методов и свойств в протокол, предоставляя их "по умолчанию" для всех типов, которые соответствуют этому протоколу. Это делает протоколы более мощными и гибкими, позволяя добавлять функциональность без необходимости менять уже существующие типы.

6. Какой синтаксис у протоколов?

Протоколы описываются с помощью ключевого слова protocol. Внутри протокола определяются требования: свойства, методы или другие элементы, которые должны быть реализованы типами, принимающими этот протокол.

Типы (классы, структуры или перечисления) указывают своё соответствие протоколу после имени типа через двоеточие :. Пример синтаксиса:

protocol SomeProtocol {

// определение протокола…

}

7. Что из себя представляет делегирование?

Делегирование — это шаблон проектирования, который позволяет одному объекту передавать часть своих обязанностей другому объекту. Это достигается за счёт использования протоколов: делегат (объект, принимающий на себя задачи) реализует методы протокола, определённого основным объектом.

**Вывод:** Таким образом, в ходе лабораторной работы я изучил наследование и протоколы в языке программирования *Swift* и выполнил разработку приложения с использованием данного языка.