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.

Приходько Н.Н.

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 3. Реализация интерфейсов с множественным наследованием:** Напишите интерфейсы для различных типов транспорта (Drivable, Flyable, Sailable). Создайте классы, которые реализуют несколько интерфейсов (например, AmphibiousCar, FlyingBoat) и добавляют дополнительные методы для переключения режимов и управления.

# КОД ПРОГРАММЫ

import java.util.Scanner  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 *println*("Выберите транспортное средство:")  
 *println*("1. AmphibiousCar")  
 *println*("2. FlyingBoat")  
 *println*("3. UniversalVehicle")  
  
 val choice = scanner.nextInt()  
  
 val vehicle: Any = when (choice) {  
 1 -> AmphibiousCar()  
 2 -> FlyingBoat()  
 3 -> UniversalVehicle()  
 else -> {  
 *println*("Неверный выбор. Завершение программы.")  
 return  
 }  
 }  
  
 *println*("Выбранное ТС: ${vehicle::class.simpleName}")  
  
 loop@ while (true) {  
 *println*("\nВыберите действие:")  
 *println*("1. Переключить режим")  
 *println*("2. Выполнить действие")  
 *println*("3. Выйти")  
  
 when (scanner.nextInt()) {  
 1 -> {  
 if (vehicle is AmphibiousCar) {  
 *println*("Выберите режим (1: Ехать, 2: Плыть):")  
 val mode = when (scanner.nextInt()) {  
 1 -> AmphibiousCar.Mode.*DRIVE* 2 -> AmphibiousCar.Mode.*SAIL* else -> {  
 *println*("Неверный режим.")  
 continue  
 }  
 }  
 vehicle.switchMode(mode)  
 } else if (vehicle is FlyingBoat) {  
 *println*("Выберите режим (1: Лететь, 2: Плыть):")  
 val mode = when (scanner.nextInt()) {  
 1 -> FlyingBoat.Mode.*FLY* 2 -> FlyingBoat.Mode.*SAIL* else -> {  
 *println*("Неверный режим.")  
 continue  
 }  
 }  
 vehicle.switchMode(mode)  
 } else if (vehicle is UniversalVehicle) {  
 *println*("Выберите режим (1: Ехать, 2: Лететь, 3: Плыть):")  
 val mode = when (scanner.nextInt()) {  
 1 -> UniversalVehicle.Mode.*DRIVE* 2 -> UniversalVehicle.Mode.*FLY* 3 -> UniversalVehicle.Mode.*SAIL* else -> {  
 *println*("Неверный режим.")  
 continue  
 }  
 }  
 vehicle.switchMode(mode)  
 }  
 }  
 2 -> {  
 *println*("Выберите действие (1: Ехать, 2: Лететь, 3: Плыть):")  
 when (scanner.nextInt()) {  
 1 -> if (vehicle is Drivable) vehicle.drive() else *println*("Это ТС не может ездить.")  
 2 -> if (vehicle is Flyable) vehicle.fly() else *println*("Это ТС не может летать.")  
 3 -> if (vehicle is Sailable) vehicle.sail() else *println*("Это ТС не может плавать.")  
 else -> *println*("Неверное действие.")  
 }  
 }  
 3 -> {  
 *println*("Выход из программы.")  
 break@loop  
 }  
 else -> *println*("Неверный выбор.")  
 }  
 }  
}

class AmphibiousCar : Drivable, Sailable {  
 private var mode: Mode = Mode.*DRIVE* enum class Mode {  
 *DRIVE*, *SAIL* }  
  
 fun switchMode(newMode: Mode) {  
 mode = newMode  
 *println*("Переключено на ${mode.name.*lowercase*()} mode.")  
 }  
   
 override fun drive() {  
 if (mode == Mode.*DRIVE*) {  
 *println*("Едет.")  
 } else {  
 *println*("Нельзя ехать в режиме плаванья!")  
 }  
 }  
  
 override fun sail() {  
 if (mode == Mode.*SAIL*) {  
 *println*("Плывет по воде.")  
 } else {  
 *println*("Нельзя плыть в режиме езды!")  
 }  
 }  
}

class FlyingBoat : Flyable, Sailable {  
 private var mode: Mode = Mode.*SAIL* enum class Mode {  
 *FLY*, *SAIL* }  
  
 fun switchMode(newMode: Mode) {  
 mode = newMode  
 *println*("Переключено на ${mode.name.*lowercase*()} mode.")  
 }  
  
 override fun fly() {  
 if (mode == Mode.*FLY*) {  
 *println*("Летит по небу.")  
 } else {  
 *println*("Нельзя лететь в режиме плаванья!")  
 }  
 }  
  
 override fun sail() {  
 if (mode == Mode.*SAIL*) {  
 *println*("Плывет по воде.")  
 } else {  
 *println*("Нельзя плыть в режиме полета!")  
 }  
 }  
}

class UniversalVehicle : Drivable, Flyable, Sailable {  
 private var mode: Mode = Mode.*DRIVE* enum class Mode {  
 *DRIVE*, *FLY*, *SAIL* }  
  
 fun switchMode(newMode: Mode) {  
 mode = newMode  
 *println*("Переключено на ${mode.name.*lowercase*()} mode.")  
 }  
  
 override fun drive() {  
 if (mode == Mode.*DRIVE*) {  
 *println*("Едет по дороге.")  
 } else {  
 *println*("Нельзя ехать в ${mode.name.*lowercase*()} mode!")  
 }  
 }  
  
 override fun fly() {  
 if (mode == Mode.*FLY*) {  
 *println*("Летит по небу.")  
 } else {  
 *println*("Нельзя лететь в ${mode.name.*lowercase*()} mode!")  
 }  
 }  
  
 override fun sail() {  
 if (mode == Mode.*SAIL*) {  
 *println*("Плывет по воде.")  
 } else {  
 *println*("Нельзя плыть ${mode.name.*lowercase*()} mode!")  
 }  
 }  
}

interface Sailable {  
 fun sail()  
}

interface Flyable {  
 fun fly()  
}

interface Drivable {  
 fun drive()  
}

# 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.