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:

ст. гр. 214302

Кранцевич В.С.

Проверил:

Усенко Ф. В.

Минск 2024

**Цель**: 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 17**: Комплексная система управления транспортом: Напишите систему для управления автопарком, включая классы Vehicle, Car, Truck, и Bus. Реализуйте методы для учета и планирования технического обслуживания, расчета маршрутов, и контроля за расходами на топливо и ремонт.

**КОД ПРОГРАММЫ**

open class Vehicle(

val make: String,

val model: String,

val year: Int,

var fuelConsumption: Double, // Литры на 100 км

var lastServiceDate: String

) {

// расходы

var repairCosts: Double = 0.0

open fun addRepairCost(cost: Double) {

repairCosts += cost

println("Added repair cost of $cost to $make $model. Total repair costs: $repairCosts")

}

// планирование техобслуживания

open fun scheduleService(serviceDate: String) {

lastServiceDate = serviceDate

println("Scheduled service for $make $model on $serviceDate")

}

fun calculateFuelConsumption(distance: Double): Double {

return (distance / 100) \* fuelConsumption

}

fun printRepairCosts() {

println("$make $model has total repair costs: $repairCosts")

}

}

class Car(

make: String,

model: String,

year: Int,

fuelConsumption: Double,

lastServiceDate: String,

val numOfDoors: Int

) : Vehicle(make, model, year, fuelConsumption, lastServiceDate) {

}

class Truck(

make: String,

model: String,

year: Int,

fuelConsumption: Double,

lastServiceDate: String,

val cargoCapacity: Double // Грузоподъемность в тоннах

) : Vehicle(make, model, year, fuelConsumption, lastServiceDate) {

// загрузка грузовика

fun loadCargo(weight: Double) {

if (weight <= cargoCapacity) {

println("Loaded $weight tons of cargo into $make $model")

} else {

println("Cannot load $weight tons, exceeding the cargo capacity!")

}

}

}

class Bus(

make: String,

model: String,

year: Int,

fuelConsumption: Double,

lastServiceDate: String,

val passengerCapacity: Int // Вместимость пассажиров

) : Vehicle(make, model, year, fuelConsumption, lastServiceDate) {

fun boardPassengers(numberOfPassengers: Int) {

if (numberOfPassengers <= passengerCapacity) {

println("Boarded $numberOfPassengers passengers into $make $model")

} else {

println("Cannot board $numberOfPassengers passengers, exceeding the capacity!")

}

}

}

// менеджер

class FleetManager {

private val vehicles = mutableListOf<Vehicle>()

fun addVehicle(vehicle: Vehicle) {

vehicles.add(vehicle)

println("Added ${vehicle.make} ${vehicle.model} to the fleet.")

}

fun calculateTotalFuelConsumption(distance: Double): Double {

return vehicles.sumOf { it.calculateFuelConsumption(distance) }

}

// Планирование обслуживания для всех транспортных средств

fun scheduleFleetService(serviceDate: String) {

vehicles.forEach { it.scheduleService(serviceDate) }

}

fun printFleetInfo() {

vehicles.forEach {

println("${it.make} ${it.model}, Year: ${it.year}, Last service: ${it.lastServiceDate}, Fuel consumption: ${it.fuelConsumption}L/100km")

}

}

fun printRepairCosts() {

vehicles.forEach { it.printRepairCosts() }

}

}

fun main() {

val car1 = Car("Toyota", "Corolla", 2020, 7.5, "2023-01-10", 4)

val truck1 = Truck("Volvo", "FH16", 2022, 30.0, "2023-03-15", 20.0)

val bus1 = Bus("Mercedes", "Sprinter", 2021, 15.0, "2023-05-20", 40)

val fleetManager = FleetManager()

fleetManager.addVehicle(car1)

fleetManager.addVehicle(truck1)

fleetManager.addVehicle(bus1)

car1.addRepairCost(250.0)

truck1.addRepairCost(1000.0)

bus1.addRepairCost(500.0)

fleetManager.printRepairCosts()

fleetManager.printFleetInfo()

val totalFuel = fleetManager.calculateTotalFuelConsumption(200.0)

println("Total fuel consumption for the fleet on a 200 km route: $totalFuel liters")

fleetManager.scheduleFleetService("2024-01-01")

truck1.loadCargo(15.0)

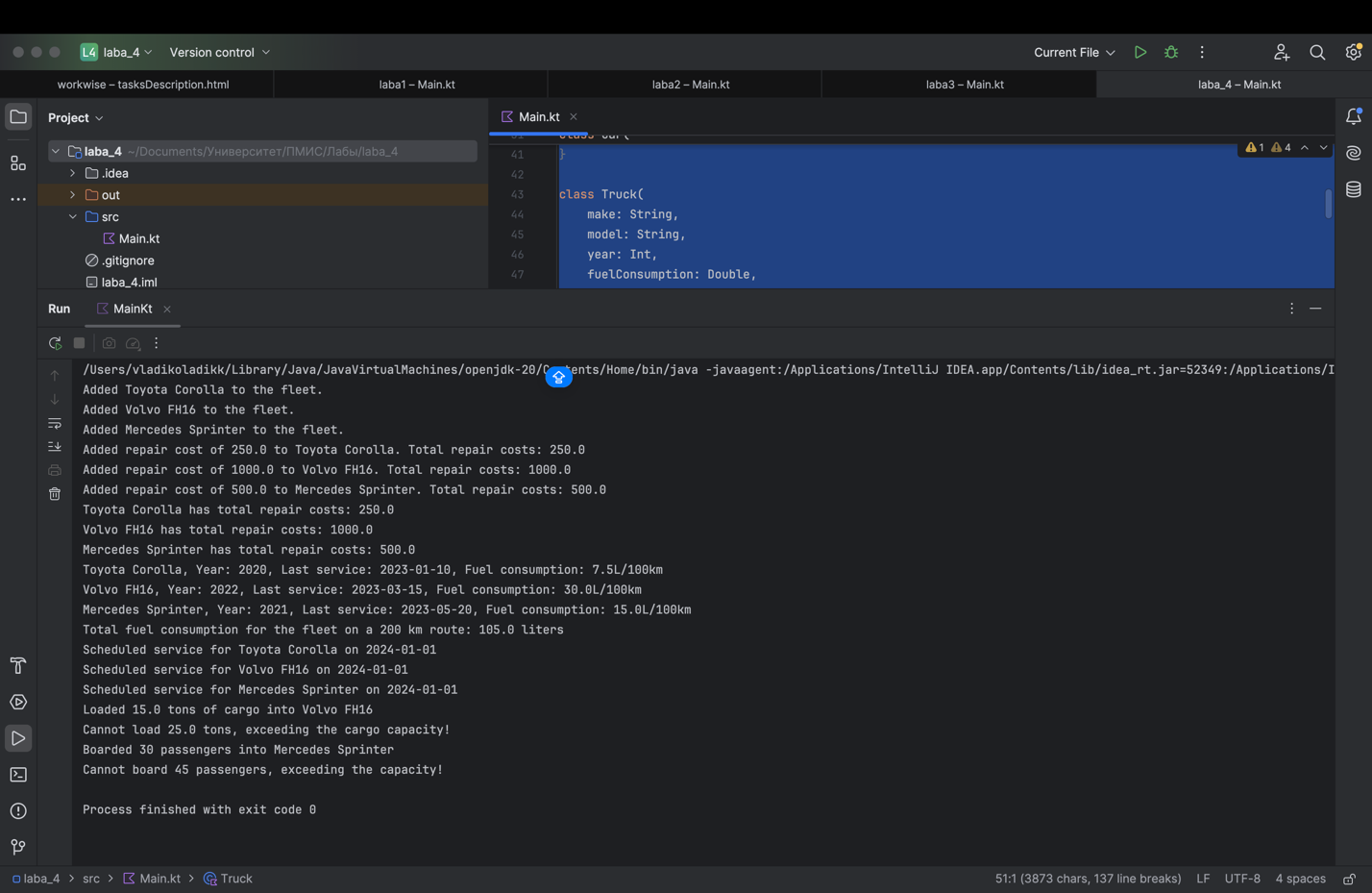
truck1.loadCargo(25.0)

bus1.boardPassengers(30)

bus1.boardPassengers(45)

}

**СКРИНШОТЫ РАБОЧИХ ОКОН**

****

**ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. **Что такое класс в Kotlin, и как он объявляется?**  
   Класс в Kotlin — это основная структура, используемая для описания объектов с состоянием (свойствами) и поведением (методами). Объявляется с помощью ключевого слова class:

class Person(val name: String, var age: Int)

1. **Как создать объект класса в Kotlin? Приведите пример.**  
   Объект класса создается с использованием ключевого слова val или var и вызова конструктора:

val person = Person("Alice", 25)

println(person.name) // Alice

1. **Что такое свойства класса, и как их объявить в Kotlin?**  
   Свойства — это переменные внутри класса, которые хранят состояние объекта. Они объявляются как val или var:

class Car(val make: String, var speed: Int)

1. **Как объявить и использовать метод класса? Приведите пример.**  
   Метод класса — это функция, объявленная внутри класса:

class Calculator {

fun add(a: Int, b: Int): Int = a + b

}

val calculator = Calculator()

println(calculator.add(3, 5)) // 8

1. **Что такое первичный конструктор, и как он используется для инициализации свойств класса?**  
   Первичный конструктор объявляется после имени класса и используется для инициализации свойств:

class Person(val name: String, val age: Int)

val person = Person("Bob", 30)

1. **Как в Kotlin создать вторичный конструктор, и зачем он может понадобиться?**  
   Вторичный конструктор используется для предоставления альтернативных способов создания объекта:

class Book(val title: String) {

var pages: Int = 0

constructor(title: String, pages: Int) : this(title) {

this.pages = pages

}

}

val book = Book("Kotlin Guide", 300)

1. **Что такое наследование, и как его реализовать в Kotlin? Приведите пример.**  
   Наследование позволяет классу унаследовать свойства и методы другого класса. Базовый класс должен быть open:

open class Animal {

open fun speak() = println("Animal sound")

}

class Dog : Animal() {

override fun speak() = println("Bark")

}

val dog = Dog()

dog.speak() // Bark

1. **Как переопределить метод базового класса в подклассе? Приведите пример.**  
   Метод в базовом классе должен быть объявлен с open, а в подклассе — с override:

open class Shape {

open fun draw() = println("Drawing shape")

}

class Circle : Shape() {

override fun draw() = println("Drawing circle")

}

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

* Интерфейсы не хранят состояние, а абстрактные классы могут.
* Интерфейсы поддерживают множественное наследование.
* Абстрактный класс может иметь реализацию методов и свойства.

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

interface Clickable {

fun click()

}

interface Focusable {

fun focus()

}

class Button : Clickable, Focusable {

override fun click() = println("Button clicked")

override fun focus() = println("Button focused")

}

1. **Что такое абстрактный класс, и как объявить абстрактный метод?**  
   Абстрактный класс — это класс, который нельзя создать напрямую. Абстрактные методы не имеют реализации:

abstract class Animal {

abstract fun makeSound()

}

class Cat : Animal() {

override fun makeSound() = println("Meow")

}

1. **Как реализовать полиморфизм в Kotlin? Приведите пример использования.**  
   Полиморфизм позволяет объекту вести себя по-разному в зависимости от типа:

open class Shape {

open fun draw() = println("Drawing shape")

}

class Circle : Shape() {

override fun draw() = println("Drawing circle")

}

fun render(shape: Shape) {

shape.draw()

}

val circle = Circle()

render(circle) // Drawing circle

1. **Что такое компаньон-объект, и как его использовать для создания единственного экземпляра класса?**  
   Компаньон-объект (companion object) используется для создания методов и свойств, доступных без создания экземпляра:

class Factory {

companion object {

fun create(): Factory = Factory()

}

}

val factory = Factory.create()

1. **Как в Kotlin реализовать инкапсуляцию данных? Приведите пример с использованием приватных полей.**  
   Инкапсуляция достигается с помощью модификатора private и геттеров/сеттеров:

class BankAccount {

private var balance: Double = 0.0

fun deposit(amount: Double) {

if (amount > 0) balance += amount

}

fun getBalance() = balance

}

1. **Как объединить объекты разных классов в один объект? Приведите пример.**  
   Можно использовать объединяющий класс или композицию:

class Engine(val type: String)

class Car(val engine: Engine, val brand: String)

val car = Car(Engine("V8"), "Ford")

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

class Person(val name: String, val age: Int) {

override fun toString(): String {

return "Name: $name, Age: $age"

}

}

val person = Person("Alice", 25)

println(person) // Name: Alice, Age: 25

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

class Rectangle(val width: Int, val height: Int) {

constructor(size: Int) : this(size, size)

}

val square = Rectangle(5)

val rectangle = Rectangle(5, 10)

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

interface EventListener {

fun onEvent()

}

class Button(val listener: EventListener) {

fun click() {

listener.onEvent()

}

}

class Handler : EventListener {

override fun onEvent() = println("Button clicked")

}

val button = Button(Handler())

button.click()

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

* Статические методы в Java принадлежат классу и доступны через имя класса.
* Методы компаньон-объекта в Kotlin доступны через объект companion и могут быть расширены.

1. **Как использовать обобщенные классы для создания универсальных контейнеров? Приведите пример.**

class Box<T>(val item: T)

val intBox = Box(123)

val stringBox = Box("Hello")

println(intBox.item) // 123

println(stringBox.item) // Hello

**ВЫВОД**

В ходе лабораторной работы были изучены принципы объектно-ориентированного программирования (ООП) в *Kotlin*, включая классы, объекты, наследование, полиморфизм, интерфейсы и абстрактные классы. Изучено создание и использование собственных классов, а также принципов ООП на практике.