Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчёт

К лабораторной работе № 4

на тему:

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)**

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Усенко Ф. В.

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.А. Барбук

(подпись) 214302

Минск 2024

**Вариант 1. Создание многомодульной системы управления пользователями**

Условие: Напишите многомодульную систему, которая управляет пользователями в рамках крупного проекта. Реализуйте классы для различных ролей (User, Admin, Guest, Moderator) с разными уровнями доступа и возможностью динамически изменять права.

**Код программы:**

fun main() {

val userManager = UserManager()

val admin = Admin("admin", "admin123")

val moderator = Moderator("moderator", "mod123")

val guest = Guest("guest")

userManager.registerUser(admin)

userManager.registerUser(moderator)

userManager.registerUser(guest)

println("Введите имя пользователя")

var name = readLine()

println("Введите пароль")

var password = readLine()

val user = userManager.login(name.toString(), password.toString())

if (user != null) {

println("Login successful for ${user.username}")

} else {

println("Login failed")

}

userManager.changePermissions(moderator, listOf("READ", "WRITE", "DELETE"))

userManager.changePassword(admin, "newAdminPass")

userManager.printUserPermissions(moderator)

}

class UserManager {

private val users = mutableListOf<User>()

fun registerUser(user: User) {

users.add(user)

}

fun login(username: String, password: String): User? {

return users.find { it.username == username && it.password == password }

}

fun changePassword(user: User, newPassword: String) {

user.password = newPassword

}

fun changePermissions(user: User, newPermissions: List<String>) {

user.permissions = newPermissions

}

fun printUserPermissions(user: User) {

println("${user.username} has permissions: ${user.permissions.joinToString(", ")}")

}

}

open class User(val username: String, var password: String) : PermissionHolder {

override var permissions = listOf("READ")

}

class Admin(username: String, password: String) : User(username, password) {

init {

permissions = listOf("READ", "WRITE", "DELETE")

}

}

class Moderator(username: String, password: String) : User(username, password) {

init {

permissions = listOf("READ", "WRITE")

}

}

class Guest(username: String) : User(username, "guest") {

init {

permissions = listOf("READ")

}

}

**Контрольные вопросы:**

1. Что такое класс в *Kotlin*, и как он объявляется?

– Класс — это шаблон для создания объектов, объединяющих данные и методы для работы с этими данными. В *Kotlin* класс объявляется с помощью ключевого слова *class*.

1. Как создать объект класса в *Kotlin*? Приведите пример.

*val* объект = ПримерКласса()

1. Что такое свойства класса, и как их объявить в *Kotlin*?

– Свойства класса — это переменные, принадлежащие классу. Их можно объявить внутри класса с использованием ключевых слов *val* (для неизменяемых свойств) или *var* (для изменяемых).

1. Как объявить и использовать метод класса? Приведите пример.

– Методы класса объявляются внутри класса с использованием ключевого слова *fun*.

5. Что такое первичный конструктор, и как он используется для инициализации свойств класса?

– Первичный конструктор объявляется прямо в заголовке класса и используется для инициализации свойств класса

1. Как в *Kotlin* создать вторичный конструктор, и зачем он может понадобиться?

– Вторичный конструктор объявляется внутри класса с помощью ключевого слова *constructor* и может использоваться для создания объектов с разными наборами параметров.

1. Что такое наследование, и как его реализовать в *Kotlin*? Приведите пример.

– Наследование позволяет создать новый класс на основе существующего. В *Kotlin* для этого используется ключевое слово *open* для базового класса и : для наследования

1. Как переопределить метод базового класса в подклассе? Приведите пример.

– Метод базового класса переопределяется с помощью ключевого слова *override*.

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

– Интерфейс определяет контракт, который должны реализовывать классы, и не содержит состояния. Абстрактный класс может содержать реализацию методов и состояние. В *Kotlin* интерфейсы объявляются с помощью ключевого слова *interface*, а абстрактные классы — с помощью *abstract*.

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

class Duck : Flyable, Swimmable

1. Что такое абстрактный класс, и как объявить абстрактный метод?

– Абстрактный класс — это класс, который не может быть инстанциирован и может содержать абстрактные методы без реализации. Абстрактные методы объявляются с помощью ключевого слова *abstract*.

1. Как реализовать полиморфизм в *Kotlin*? Приведите пример использования.

– Полиморфизм — это возможность использовать один интерфейс или класс для работы с различными типами объектов. В *Kotlin* полиморфизм достигается с помощью наследования и интерфейсов. Полиморфизм позволяет вызывать методы, которые могут иметь разные реализации в разных классах

1. Что такое компаньон-объект, и как его использовать для создания единственного экземпляра класса?

– Компаньон-объект (*companion* *object*) в *Kotlin* — это специальный объект, который создается внутри класса и может содержать методы и свойства, которые можно вызывать без создания экземпляра этого класса. Компаньон-объект предоставляет возможность реализовать паттерн *Singleton*, что позволяет гарантировать существование только одного экземпляра класса.

1. Как в *Kotlin* реализовать инкапсуляцию данных? Приведите пример с использованием приватных полей.

– Это достигается с помощью модификаторов доступа, таких как *private*, *protected*, и *public*.

1. Как объединить объекты разных классов в один объект? Приведите пример.

– Объединение объектов разных классов может быть выполнено с использованием составного паттерна проектирования (*Composite* *Pattern*) или через использование коллекций (например, списков или множеств) для хранения объектов

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

– *@override fun toString() {System.out.println(“….”)}*

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

– Используя перегруженные конструкторы.

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

– Используя интерфейсы и делегаты. Интерфейс будет описывать событие, а класс-издатель будет уведомлять подписчиков о возникновении этого события.

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

– Компаньон-объект вызывается через имя класса или напрямую, может быть переопределен и имеет доступ как к статическим, так и к нестатическим полям