Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчёт

по лабораторной работе №4

на тему:

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)**

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

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. Ю. Позняк

(подпись) гр. 214302

Минск, 2024

**ЗАДАНИЕ**

Создание многомодульной системы управления пользователями: 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.

**РЕШЕНИЕ**

package core

import java.io.Serializable

interface Role : Serializable {

val name: String

}

class User : Role {

override val name = "Пользователь"

}

class Admin : Role {

override val name = "Администратор"

}

class Guest : Role {

override val name = "Гость"

}

class Moderator : Role {

override val name = "Модератор"

}

package core

import java.io.Serializable

class UserAccount(val username: String, var role: Role) : Serializable {

fun changeRole(newRole: Role) {

role = newRole

}

override fun toString(): String {

return "Логин : '$username'\nРоль : ${role.name})"

}

}

package userManagement

import core.\*

import java.io.\*

class UserManager {

private val users = mutableListOf<UserAccount>()

fun addUser(user: UserAccount) {

users.add(user)

saveUsersToFile()

}

fun removeUser(username: String) {

users.removeIf { it.username == username }

saveUsersToFile()

}

fun findUser(username: String): UserAccount? {

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

}

fun changeUserRole(username: String, newRole: Role) {

val user = findUser(username)

user?.changeRole(newRole)

saveUsersToFile()

}

fun listUsers(): List<UserAccount> {

return users

}

fun authenticate(username: String): UserAccount? {

return findUser(username)

}

private fun saveUsersToFile() {

ObjectOutputStream(FileOutputStream("users.dat")).use { it.writeObject(users) }

}

fun loadUsersFromFile() {

if (File("users.txt").exists()) {

ObjectInputStream(FileInputStream("users.txt")).use {

@Suppress("UNCHECKED\_CAST")

users.addAll(it.readObject() as List<UserAccount>)

}

}

}

}

import core.\*

import userManagement.UserManager

import java.util.Scanner

fun main() {

val scanner = Scanner(System.`in`)

val userManager = UserManager()

userManager.loadUsersFromFile()

var currentUser: UserAccount? = null

while (currentUser == null) {

println("Выберите опцию: [1] Зарегистрироваться, [2] Войти, [3] Выйти")

when (scanner.nextInt()) {

1 -> {

println("Введите имя пользователя для регистрации:")

val username = scanner.next()

println("Выберите роль: [1] Пользователь, [2] Администратор, [3] Модератор, [4] Гость")

val role = when (scanner.nextInt()) {

1 -> User()

2 -> Admin()

3 -> Moderator()

4 -> Guest()

else -> Guest()

}

userManager.addUser(UserAccount(username, role))

println("Пользователь зарегистрирован")

}

2 -> {

println("Введите имя пользователя для входа:")

val username = scanner.next()

currentUser = userManager.authenticate(username)

if (currentUser != null) {

println("Добро пожаловать, ${currentUser.username}!")

} else {

println("Пользователь не найден")

}

}

3 -> return

else -> println("Неверная опция")

}

}

while (true) {

when (currentUser?.role?.name) {

"Администратор" -> {

println("Выберите опцию: [1] Добавить пользователя, [2] Удалить пользователя, [3] Список пользователей, [4] Удалить свой аккаунт, [5] Выйти")

when (scanner.nextInt()) {

1 -> {

println("Введите имя пользователя:")

val username = scanner.next()

println("Выберите роль: [1] Пользователь, [2] Администратор, [3] Модератор, [4] Гость")

val role = when (scanner.nextInt()) {

1 -> User()

2 -> Admin()

3 -> Moderator()

4 -> Guest()

else -> Guest()

}

userManager.addUser(UserAccount(username, role))

println("Пользователь добавлен")

}

2 -> {

println("Введите имя пользователя для удаления:")

val username = scanner.next()

if (username != currentUser.username && userManager.findUser(username)?.role !is Moderator) {

userManager.removeUser(username)

println("Пользователь удален")

} else {

println("Нельзя удалить себя или модератора")

}

}

3 -> {

println("Список пользователей:")

userManager.listUsers().filter {

it.username != currentUser!!.username && it.role !is Moderator

}.forEach { println(it) }

}

4 -> {

userManager.removeUser(currentUser.username)

println("Ваш аккаунт удален")

break

}

5 -> break

else -> println("Неверная опция")

}

}

"Пользователь" -> {

println("Выберите опцию: [1] Список пользователей, [2] Удалить свой аккаунт, [3] Выйти")

when (scanner.nextInt()) {

1 -> {

println("Список пользователей:")

userManager.listUsers().filter {

it.username != currentUser!!.username

}.forEach { println(it) }

}

2 -> {

userManager.removeUser(currentUser.username)

println("Ваш аккаунт удален")

break

}

3 -> break

else -> println("Неверная опция")

}

}

"Модератор" -> {

println("Выберите опцию: [1] Добавить пользователя, [2] Удалить пользователя, [3] Изменить роль, [4] Список пользователей, [5] Удалить свой аккаунт, [6] Выйти")

when (scanner.nextInt()) {

1 -> {

println("Введите имя пользователя:")

val username = scanner.next()

println("Выберите роль: [1] Пользователь, [2] Администратор, [3] Модератор, [4] Гость")

val role = when (scanner.nextInt()) {

1 -> User()

2 -> Admin()

3 -> Moderator()

4 -> Guest()

else -> User()

}

userManager.addUser(UserAccount(username, role))

println("Пользователь добавлен")

}

2 -> {

println("Введите имя пользователя для удаления:")

val username = scanner.next()

if (username != currentUser.username) {

userManager.removeUser(username)

println("Пользователь удален")

} else {

println("Нельзя удалить себя")

}

}

3 -> {

println("Введите имя пользователя для изменения роли:")

val username = scanner.next()

println("Выберите новую роль: [1] Пользователь, [2] Администратор, [3] Модератор, [4] Гость")

val newRole = when (scanner.nextInt()) {

1 -> User()

2 -> Admin()

3 -> Moderator()

4 -> Guest()

else -> Guest()

}

if (username != currentUser.username) {

userManager.changeUserRole(username, newRole)

println("Роль пользователя изменена")

} else {

println("Нельзя изменить свою роль")

}

}

4 -> {

println("Список пользователей:")

userManager.listUsers().filter {

it.username != currentUser!!.username

}.forEach { println(it) }

}

5 -> {

userManager.removeUser(currentUser.username)

println("Ваш аккаунт был удален.")

currentUser = null

break

}

6 -> break

else -> println("Неверная опция")

}

}

"Гость" -> {

println("Выберите опцию: [1] Удалить свой аккаунт, [2] Выйти")

when (scanner.nextInt()) {

1 -> {

userManager.removeUser(currentUser.username)

println("Ваш аккаунт был удален.")

currentUser = null

break

}

2 -> break

else -> println("Неверная опция")

}

}

else -> println("Неизвестная роль пользователя")

}

}

}

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

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

Класс в *kotlin* – это шаблон для создания объектов (экземпляров), который может содержать свойства и методы. Класс объявляется с помощью ключевого слова *class*, после чего следует имя класса и его тело в фигурных скобках.

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

Чтобы создать объект класса, используйте оператор *val* или *var*, за которым следует имя объекта и вызов конструктора класса.

Class person(val name: string, val age: int)

Fun main() {

val person = person("alice", 30) // создание объекта

println("${person.name} is ${person.age} years old.")

}

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

Свойства класса – это переменные, которые хранят состояние объекта. Они объявляются внутри класса с помощью ключевых слов *val* (для неизменяемых свойств) или *var* (для изменяемых).

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

Метод класса – это функция, определенная внутри класса. Методы могут использоваться для выполнения действий или возвращения значений.