Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

Информатики и радиоэлектроники»

Учебная дисциплина: «Программирование мобильных информационных систем»

Отчет

по выполнению лабораторной работы «Объектно-ориентированное программирование (ООП)»

Выполнил студент: гр. 210101 Войтович Е.А.

Проверил: Усенко Ф.В.

Минск 2024

**1. 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.

**2. 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.

**3. Оснащение работы:** техническое задание, технический проект, ЭВМ, IntelliJ IDEA.

package roles  
  
interface UserRole {  
 val username: String  
 val age: Int?  
 val email: String  
  
 fun viewContent(): String  
 fun editContent(): String  
}

package roles  
  
class Guest : UserRole {  
  
 override val username: String = "Guest"  
 override val age: Int? = null  
 override val email: String = ""  
  
 override fun viewContent(): String {  
 return "Guest (Age: not provided, Email: not provided): Viewing limited content as a guest."  
 }  
  
 override fun editContent(): String {  
 return "Guest: Guests cannot edit content."  
 }  
}

package roles  
  
class Moderator(  
 override val username: String,  
 override val age: Int?,  
 override val email: String  
) : UserRole {  
  
 override fun viewContent(): String {  
 return "$username (Age: ${age ?: "not provided"}, Email: $email): Viewing and moderating content as a moderator."  
 }  
  
 override fun editContent(): String {  
 return "$username: Editing limited content as a moderator."  
 }  
}

package roles  
  
class User(  
 override val username: String,  
 override val age: Int?,  
 override val email: String  
) : UserRole {  
  
 constructor(username: String, email: String) : this(username, null, email)  
  
 constructor(username: String) : this(username, null, "no-email@example.com")  
  
 override fun viewContent(): String {  
 return "$username (Age: ${age ?: "not provided"}, Email: $email): Viewing content as a regular user."  
 }  
  
 override fun editContent(): String {  
 return "$username: You don't have permission to edit content."  
 }  
}

package roles  
  
class Admin(  
 override val username: String,  
 override val age: Int?,  
 override val email: String  
) : UserRole {  
  
 override fun viewContent(): String {  
 return "$username (Age: ${age ?: "not provided"}, Email: $email): Viewing all content as an admin."  
 }  
  
 override fun editContent(): String {  
 return "$username: Editing content as an admin."  
 }  
  
 fun changeUserRole(user: UserRole, newRole: UserRole): UserRole {  
 println("$username: Changing role of ${user.username} to ${newRole::class.simpleName}")  
 return newRole  
 }  
}

import roles.\*

fun main() {

var user1: UserRole = User("User")

var user2: UserRole = Guest()

var user3: UserRole = Moderator("ModUser", 35, "mod@example.com")

val admin: Admin = Admin("AdminUser", 40, "admin@example.com")

println(user1.viewContent())

println(user1.editContent())

println()

user1 = admin.changeUserRole(user1, Moderator(user1.username, user1.age, user1.email))

println(user1.viewContent())

println(user1.editContent())

println()

user2 = admin.changeUserRole(user2, User("GuestToUser", 30, "asd@gmail.com"))

println(user2.viewContent())

println(user2.editContent())

println()

user3 = admin.changeUserRole(user3, Guest())

println(user3.viewContent())

println(user3.editContent())

}

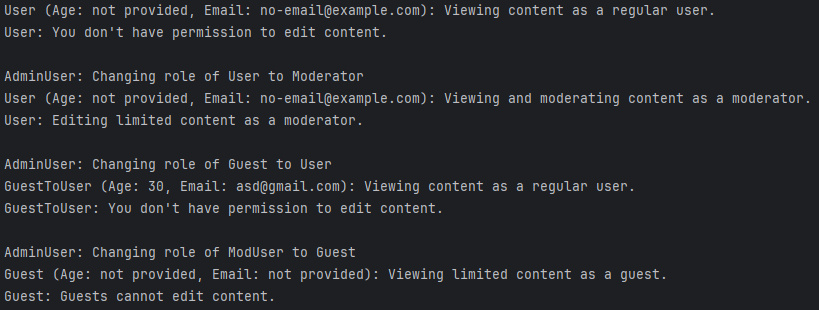


Рисунок 1 – Результат исполнения кода программы

**Вывод:** в ходе выполнения данной лабораторной работы была разработана многомодульная система управления пользователями на языке программирования Kotlin. В системе были реализованы классы для различных ролей: User, Admin, Guest, Moderator. Каждый класс обладает различными уровнями доступа и правами, которые зависят от роли пользователя. Администратор (Admin) имеет расширенные права, в том числе возможность динамически изменять роли других пользователей.