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.

Чехомов Е.Г.

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.

1. **Вариант 10. Система управления проектами с ролями:** Создайте систему управления проектами, в которой можно создавать проекты, назначать задачи и распределять роли среди участников (ProjectManager, Developer, Tester). Реализуйте методы для отслеживания выполнения задач и автоматического пересчета сроков в зависимости от изменений в проекте.

# КОД ПРОГРАММЫ

import kotlin.system.exitProcess  
  
  
enum class Role {  
 *Менеджер\_Проекта*, *Разработчик*, *Тестировщик*}  
  
  
enum class TaskStatus {  
 *Ожидание*, *В\_Процессе*, *Завершено*}  
  
  
  
  
fun main() {  
 val manager = ProjectManager()  
  
 while (true) {  
 *println*(  
 """  
 1. Создать проект  
 2. Добавить участника  
 3. Добавить задачу  
 4. Обновить статус задачи  
 5. Пересчитать сроки задач  
 6. Показать проекты  
 7. Показать задачи проекта  
 8. Выйти  
 """.*trimIndent*()  
 )  
 *print*("Введите номер команды: ")  
 try {  
 when (*readLine*()?.*toIntOrNull*()) {  
 1 -> {  
 *print*("Введите название проекта: ")  
 val name = *readLine*()!!  
 manager.createProject(name)  
 }  
  
 2 -> {  
 *print*("Введите ID проекта: ")  
 val projectId = *readLine*()?.*toIntOrNull*() ?: throw IllegalArgumentException()  
 *print*("Введите имя участника: ")  
 val name = *readLine*()!!  
 *print*("Введите роль участника (Менеджер Проекта, Разработчик, Тестировщик): ")  
 val role = Role.valueOf(*readLine*()!!)  
 manager.addParticipant(projectId, name, role)  
 }  
  
 3 -> {  
 *print*("Введите ID проекта: ")  
 val projectId = *readLine*()?.*toIntOrNull*() ?: throw IllegalArgumentException()  
 *print*("Введите название задачи: ")  
 val name = *readLine*()!!  
 *print*("Введите имя назначенного участника: ")  
 val participantName = *readLine*()!!  
 *print*("Введите количество дней на выполнение задачи: ")  
 val estimatedDays = *readLine*()?.*toLongOrNull*() ?: throw IllegalArgumentException()  
 *print*("Введите дату начала задачи (dd.MM.yyyy): ")  
 val startDate = *readLine*()!!  
 manager.addTask(projectId, name, participantName, estimatedDays, startDate)  
 }  
  
 4 -> {  
 *print*("Введите ID проекта: ")  
 val projectId = *readLine*()?.*toIntOrNull*() ?: throw IllegalArgumentException()  
 *print*("Введите ID задачи: ")  
 val taskId = *readLine*()?.*toIntOrNull*() ?: throw IllegalArgumentException()  
 *print*("Введите новый статус задачи (Ожидание, В Процессе, Завершено): ")  
 val status = TaskStatus.valueOf(*readLine*()!!)  
 manager.updateTaskStatus(projectId, taskId, status)  
 }  
  
 5 -> {  
 *print*("Введите ID проекта для пересчета сроков: ")  
 val projectId = *readLine*()?.*toIntOrNull*() ?: throw IllegalArgumentException()  
 manager.recalculateDeadlines(projectId)  
 }  
  
 6 -> {  
 manager.displayProjects()  
 }  
  
 7 -> {  
 *print*("Введите ID проекта для отображения задач: ")  
 val projectId = *readLine*()?.*toIntOrNull*() ?: throw IllegalArgumentException()  
 manager.displayProjectTasks(projectId)  
 }  
  
 8 -> {  
 *println*("Выход из программы.")  
 *exitProcess*(0)  
 }  
  
 else -> *println*("Неверный выбор. Попробуйте снова.")  
 }  
 } catch (e: Exception) {  
 *println*("Ошибка ввода. Попробуйте снова.")  
 }  
 }  
}

import java.time.temporal.ChronoUnit  
import java.time.LocalDate  
  
class Project(  
 val id: Int,  
 val name: String  
) : ParticipantManager, TaskManager, ProgressTracker {  
 private val participants = *mutableListOf*<Participant>()  
 private val tasks = *mutableListOf*<Task>()  
  
 override fun addParticipant(participant: Participant) {  
 participants.add(participant)  
 *println*("Участник '${participant.name}' с ролью '${participant.role}' добавлен в проект '${name}'.")  
 }  
  
 override fun getParticipants(): List<Participant> = participants  
  
 override fun addTask(task: Task) {  
 tasks.add(task)  
 *println*("Задача '${task.name}' добавлена в проект '${name}'.")  
 }  
  
 override fun getTasks(): List<Task> = tasks  
  
 override fun updateTaskStatus(taskId: Int, newStatus: TaskStatus) {  
 val task = tasks.*find* **{ it**.id == taskId **}** if (task != null) {  
 task.updateStatus(newStatus)  
 *println*("Статус задачи '${task.name}' обновлен на '${newStatus}'.")  
 } else {  
 *println*("Задача с ID $taskId не найдена.")  
 }  
 }  
  
 override fun recalculateDeadlines() {  
 tasks.*forEach* **{** task **->** if (task.status != TaskStatus.*Завершено*) {  
 val daysRemaining = ChronoUnit.*DAYS*.between(LocalDate.now(), task.dueDate)  
 if (daysRemaining < 0) {  
 task.dueDate = LocalDate.now().plusDays(task.estimatedDays)  
 *println*("Срок задачи '${task.name}' пересчитан: новый срок - ${task.dueDate}.")  
 }  
 }  
 **}** }  
  
 override fun trackProgress() {  
 val completedTasks = tasks.*count* **{ it**.status == TaskStatus.*Завершено* **}** val progress = if (tasks.*isNotEmpty*()) {  
 (completedTasks.toDouble() / tasks.size) \* 100  
 } else 0.0  
 *println*("Прогресс проекта '${name}': ${"%.2f".*format*(progress)}%.")  
 }  
  
 override fun displayTasks() {  
 *println*("Список задач проекта '${name}':")  
 tasks.*forEach* **{** task **->** *println*(  
 "ID: ${task.id}, Название: '${task.name}', Назначен: ${task.assignedTo.name}, " +  
 "Статус: ${task.status}, Дата завершения: ${task.dueDate}"  
 )  
 **}** }  
}

import java.time.LocalDate  
import java.time.format.DateTimeFormatter  
  
  
class ProjectManager {  
 private val projects = *mutableListOf*<Project>()  
 private var taskIdCounter = 1  
 private val dateFormatter = DateTimeFormatter.ofPattern("dd.MM.yyyy")  
  
 fun createProject(name: String) {  
 val project = Project(projects.size + 1, name)  
 projects.add(project)  
 *println*("Проект '$name' создан.")  
 }  
  
 fun addParticipant(projectId: Int, name: String, role: Role) {  
 val project = projects.*find* **{ it**.id == projectId **}** if (project != null) {  
 project.addParticipant(Participant(name, role))  
 } else {  
 *println*("Проект с ID $projectId не найден.")  
 }  
 }  
  
 fun addTask(  
 projectId: Int,  
 name: String,  
 participantName: String,  
 estimatedDays: Long,  
 startDate: String  
 ) {  
 val project = projects.*find* **{ it**.id == projectId **}** if (project != null) {  
 val participant = project.getParticipants().*find* **{ it**.name == participantName **}** if (participant != null) {  
 val start = LocalDate.parse(startDate, dateFormatter)  
 val due = start.plusDays(estimatedDays)  
 val task = Task(  
 id = taskIdCounter++,  
 name = name,  
 assignedTo = participant,  
 estimatedDays = estimatedDays,  
 startDate = start,  
 dueDate = due  
 )  
 project.addTask(task)  
 project.trackProgress()  
 } else {  
 *println*("Участник с именем '$participantName' не найден в проекте.")  
 }  
 } else {  
 *println*("Проект с ID $projectId не найден.")  
 }  
 }  
  
 fun updateTaskStatus(projectId: Int, taskId: Int, newStatus: TaskStatus) {  
 val project = projects.*find* **{ it**.id == projectId **}** project?.updateTaskStatus(taskId, newStatus)  
 ?: *println*("Проект с ID $projectId не найден.")  
 }  
  
 fun recalculateDeadlines(projectId: Int) {  
 val project = projects.*find* **{ it**.id == projectId **}** project?.recalculateDeadlines()  
 ?: *println*("Проект с ID $projectId не найден.")  
 }  
  
 fun displayProjects() {  
 if (projects.isEmpty()) {  
 *println*("Проекты отсутствуют.")  
 } else {  
 *println*("Список проектов:")  
 projects.*forEach* **{** project **->** *println*("ID: ${project.id}, Название: '${project.name}'")  
 **}** }  
 }  
  
 fun displayProjectTasks(projectId: Int) {  
 val project = projects.*find* **{ it**.id == projectId **}** project?.displayTasks()  
 ?: *println*("Проект с ID $projectId не найден.")  
 }  
}

import java.time.LocalDate  
  
data class Task(  
 val id: Int,  
 val name: String,  
 val assignedTo: Participant,  
 var status: TaskStatus = TaskStatus.*Ожидание*,  
 val estimatedDays: Long,  
 var startDate: LocalDate,  
 var dueDate: LocalDate  
) {  
 fun updateStatus(newStatus: TaskStatus) {  
 status = newStatus  
 }  
}

# 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.