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.В. Зинковская

(подпись) гр. 214301

Минск, 2024

**Цель:** Создать систему управления проектами, в которой можно создавать проекты, назначать задачи и распределять роли среди участников (ProjectManager, Developer, Tester). Реализовать методы для отслеживания выполнения задач и автоматического пересчета сроков в зависимости от изменений в проекте.

**Код**

package Participant

class Admin(override val name: String) : ProjectParticipant {

override val role = "Администратор"

}

package Participant

class Developer(override val name: String) : ProjectParticipant {

override val role = "Developer"

}

package Participant

class ProjectManager(override val name: String) : ProjectParticipant {

override val role = "ProjectManager"

}

package Participant

interface ProjectParticipant {

val name: String

val role: String

}

package Participant

class Tester(override val name: String) : ProjectParticipant {

override val role = "Tester"

}

import Participant.ProjectManager

import Participant.ProjectParticipant

import java.time.LocalDate

class Project(

private val name: String,

private val manager: ProjectManager,

private var tasks: MutableList<Task> = mutableListOf(),

private var dueDate: LocalDate

) {

fun getName() = name

fun getManager() = manager

fun getTasks() = tasks

fun getDueDate() = dueDate

fun addTask(task: Task) {

tasks.add(task)

recalculateDueDate()

println("Задача '${task.getTitle()}' добавлена в проект '$name'.")

}

fun assignTask(taskTitle: String, assignee: ProjectParticipant) {

val task = tasks.find { it.getTitle() == taskTitle }

if (task != null) {

task.setAssignee(assignee)

println("Задача '${task.getTitle()}' назначена ${assignee.name}.")

} else {

println("Задача '$taskTitle' не найдена в проекте '$name'.")

}

}

fun updateTaskStatus(taskTitle: String, newStatus: String) {

val task = tasks.find { it.getTitle() == taskTitle }

if (task != null) {

task.updateStatus(newStatus)

recalculateDueDate()

} else {

println("Задача '$taskTitle' не найдена в проекте '$name'.")

}

}

private fun recalculateDueDate() {

val maxTaskDueDate = tasks.maxOfOrNull { it.getDueDate() }

if (maxTaskDueDate != null && maxTaskDueDate.isAfter(dueDate)) {

dueDate = maxTaskDueDate

println("Срок выполнения проекта '$name' передвинут на $dueDate.")

}

}

}

import Participant.ProjectParticipant

import java.util.\*

class ProjectManagementSystem {

private val users = mutableListOf<ProjectParticipant>()

private val projects = mutableListOf<Project>()

fun addUser(user: ProjectParticipant) {

users.add(user)

println("Пользователь ${user.name} с ролью ${user.role} добавлен")

}

fun findUserByName(name: String): ProjectParticipant? {

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

}

fun addProject(project: Project) {

projects.add(project)

println("Проект '${project.getName()}' создан")

}

fun viewProjects() {

if (projects.isNotEmpty()) {

println("\nСписок проектов:")

projects.forEach { project ->

println("${project.getName()}, Менеджер: ${project.getManager().name}, Срок выполнения: ${project.getDueDate()}")

project.getTasks().forEach { task ->

println("\nЗадача: ${task.getTitle()}, Статус: ${task.getStatus()}, Срок: ${task.getDueDate()}, Исполнитель: ${task.getAssignee()?.name ?: "Не назначен"}")

}

}

} else {

println("Нет проектов")

}

}

fun selectProject(): Project? {

if (projects.isEmpty()) {

println("Нет доступных проектов.")

return null

}

println("\nВыберите проект:")

projects.forEachIndexed { index, project ->

println("${index + 1}. ${project.getName()} (Срок выполнения: ${project.getDueDate()})")

}

return projects[readIndexFromUser(projects.size)]

}

fun selectTask(project: Project): Task? {

if (project.getTasks().isEmpty()) {

println("Нет доступных задач в проекте '${project.getName()}'.")

return null

}

println("\nВыберите задачу:")

project.getTasks().forEachIndexed { index, task ->

println("${index + 1}. ${task.getTitle()} (Статус: ${task.getStatus()})")

}

return project.getTasks()[readIndexFromUser(project.getTasks().size)]

}

fun selectUser(): ProjectParticipant? {

if (users.isEmpty()) {

println("Нет доступных пользователей.")

return null

}

println("\nВыберите пользователя:")

users.forEachIndexed { index, user ->

println("${index + 1}. ${user.name} (${user.role})")

}

return users[readIndexFromUser(users.size)]

}

fun selectStatus(): String {

val statuses = listOf("Ожидает", "В процессе", "Завершено")

println("\nВыберите статус задачи:")

statuses.forEachIndexed { index, status ->

println("${index + 1}. $status")

}

return statuses[readIndexFromUser(statuses.size)]

}

private fun readIndexFromUser(maxIndex: Int): Int {

val scanner = Scanner(System.`in`)

var index: Int

while (true) {

print("Введите номер (1-$maxIndex): ")

index = scanner.nextInt() - 1

if (index in 0 until maxIndex) break

else println("Некорректный выбор, попробуйте снова.")

}

return index

}

}

import Participant.ProjectParticipant

import java.time.LocalDate

class Task(

private val title: String,

private var assignee: ProjectParticipant?,

private var status: String,

private var dueDate: LocalDate

) {

fun getTitle() = title

fun getAssignee() = assignee

fun getStatus() = status

fun getDueDate() = dueDate

fun setAssignee(newAssignee: ProjectParticipant?) {

assignee = newAssignee

}

fun updateStatus(newStatus: String) {

status = newStatus

println("Статус задачи '$title': '$status'.")

}

}

import Participant.\*

import java.time.LocalDate

import java.util.Scanner

fun main() {

val system = ProjectManagementSystem()

val scanner = Scanner(System.`in`)

val admin = Admin("Admin")

system.addUser(admin)

var currentUser: ProjectParticipant? = null

while (currentUser == null) {

println("Введите ваше имя для входа:")

val name = scanner.nextLine()

currentUser = system.findUserByName(name)

if (currentUser == null)

println("Пользователь не найден")

}

if (currentUser is Admin) {

while (true) {

println("\nКоманды администратора:")

println("1. Добавить пользователя")

println("2. Создать проект")

println("3. Добавить задачу в проект")

println("4. Назначить задачу")

println("5. Обновить статус задачи")

println("6. Просмотреть все проекты")

println("7. Выход")

when (scanner.nextInt()) {

1 -> {

println("Введите имя нового пользователя:")

scanner.nextLine()

val userName = scanner.nextLine()

println("Назначьте роль (1 - ProjectManager, 2 - Developer, 3 - Tester):")

when (scanner.nextInt()) {

1 -> system.addUser(ProjectManager(userName))

2 -> system.addUser(Developer(userName))

3 -> system.addUser(Tester(userName))

else -> println("Некорректная роль")

}

}

2 -> {

println("Введите название проекта:")

scanner.nextLine()

val projectName = scanner.nextLine()

println("Введите имя менеджера проекта:")

val manager = system.selectUser() as? ProjectManager

if (manager != null) {

println("Введите срок выполнения проекта (YYYY-MM-DD):")

val dueDate = LocalDate.parse(scanner.nextLine())

val project = Project(projectName, manager, dueDate = dueDate)

system.addProject(project)

} else

println("Менеджер не найден или некорректная роль.")

}

3 -> {

val project = system.selectProject()

if (project != null) {

println("Введите название задачи:")

scanner.nextLine()

val taskTitle = scanner.nextLine()

println("Введите срок задачи (YYYY-MM-DD):")

val taskDueDate = LocalDate.parse(scanner.nextLine())

val task = Task(taskTitle, null, "Ожидание", taskDueDate)

project.addTask(task)

}

}

4 -> {

val project = system.selectProject()

if (project != null) {

val task = system.selectTask(project)

if (task != null) {

val assignee = system.selectUser()

if (assignee != null)

project.assignTask(task.getTitle(), assignee)

}

}

}

5 -> {

val project = system.selectProject()

if (project != null) {

val task = system.selectTask(project)

if (task != null) {

val newStatus = system.selectStatus()

project.updateTaskStatus(task.getTitle(), newStatus)

}

}

}

6 -> system.viewProjects()

7 -> {

println("Выход из системы.")

return

}

else -> println("Некорректный ввод. Попробуйте снова.")

}

}

}

}

**Контрольные вопросы**

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

В Kotlin первичный конструктор объявляется прямо в заголовке класса и используется для инициализации свойств класса.

2 Как в Kotlin создать вторичный конструктор, и зачем он может понадобиться?

Вторичные конструкторы используются для предоставления дополнительных способов инициализации объектов. Они объявляются с использованием ключевого слова constructor.

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

Абстрактные классы используются, когда необходимо создать базовый класс, который нельзя инстанцировать, но который может содержать как абстрактные, так и не абстрактные методы.

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

Компаньон-объект — это специальный объект внутри класса, который ведет себя как статический метод или поле в Java. Компаньон-объекты позволяют обращаться к методам и свойствам класса без создания экземпляра.

Компаньон-объекты полезны, когда нужно предоставить функции для работы с классом, которые не зависят от конкретного объекта.

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

Объекты создаются с помощью вызова конструктора класса. В Kotlin ключевое слово new не требуется.

val myCar = Car("Toyota", "Corolla", 2020)

myCar.startEngine()

**Вывод**: Были изучены и применены на практике основы объектно- ориентированного программирования. Была создана система управления проекктами.