**Кросплатформенне програмування**

## **Лабораторна робота №3**

«Багатопотокове та конкурентне програмування»

*Виконав Заяць Ілля, група ІПЗ-41*

**Мета:**

*Завдання 4. Симулятор неблокуючого інтерфейсу користувача*

Консольний додаток з неблокуючим ввідом, який записує результати у файл.

Вхід користувача:

task NAME X: виконати завдання X, назвати його NAME і записати результат у файл результатів.

get: вивести останній результат та його назву на консоль.

finish grace: припинити прийом нових завдань, завершити всі очікуючі завдання та завершити роботу додатку.

finish force: завершити роботу додатку без очікування завершення невиконаних завдань.

clean: очистити файл результатів.

help: вивести інструкції на консоль.

Деталі:

Є список попередньо визначених завдань, які блокують потік. X у команді task — це ціле число, індекс завдання, яке потрібно виконати. Кожне завдання є Callable<T>, який блокує потік виконання на 300–5000 мс. Метод T.toString() реалізовано так, щоб T представляло собою одну строку тексту. Виконання завдання означає отримання результату (T) і запис його в файл результатів, який надається додатку при запуску. NAME у команді task не повинно містити пробілів.

Записи у файлі повинні мати вигляд: NAME: T. Виведення на консоль повинно бути виглядати так: T [NAME]. Помилки не повинні записуватись у файл результатів. Ви можете записувати їх у окремий файл або виводити на консоль.

Команда get повинна читати представлення останнього результату і виводити його разом з назвою, яку було надано цьому результату/завданню.

Примітки:

Додаток не повинен використовувати більше 6 потоків.

Консольний ввід повинен бути завжди доступний (не блокуватися), за винятком випадку, коли було викликано команду finish.

Додаток повинен запускатися за 1.5 секунди.

Якщо потік вводу закриється (отримано null), додаток повинен завершитися. Ви самі вирішуєте, чи повинен додаток завершитися коректно чи ні.

Для інструктора: Як оцінити завдання:

Студент може отримати 7 балів загалом. Не має значення, чи використовуються потоки чи корутини.

4 бали за правильну реалізацію завдання (якщо воно працює добре, проходить всі тести і т.д.) та 3 бали за правильний вибір конкурентних структур даних і безпечну роботу з ними.

Перевірка:

Запустіть тести локально: ./gradlew test;

Запустіть Detekt локально: ./gradlew detektCheckAll;

Запустіть Diktat локально: ./gradlew diktatCheckAll;

Запустіть весь Github workflow локально: ./gradlew githubWorkflow.

Хід роботи

1.)- Виконати завдання.

2.)- Виконати тестування створеної програми.

## *Завдання*

## ***Файл Main.kt***

## 1.)- Опис структури

Цей файл містить основний код програми, яка запускається при виконанні. Він включає:

Основну функцію main(), яка ініціалізує основні об'єкти програми та забезпечує взаємодію з користувачем через консоль.

Взаємодія з користувачем здійснюється через Scanner (для зчитування вводу з консолі).

В залежності від введеної команди (наприклад, "task", "finish grace", "get", "help" тощо) викликаються відповідні методи класу TaskExecutor.

Програма підтримує наступні команди:

task NAME X — виконує задачу із зазначеним ідентифікатором X та ім'ям NAME.

get — виводить останній результат виконаного завдання.

finish grace / finish force — зупиняє виконання завдань, дозволяючи або вимушено.

clean — очищує файл результатів.

help — виводить допомогу з описом команд.

## 2.)- Код

package org.jub.kotlin.hometask4

import java.io.File

import java.util.Scanner

fun main() {

val resultsFile = File("results.txt") // Шлях до файлу результатів

val taskExecutor = TaskExecutor(resultsFile)

val scanner = Scanner(System.`in`)

while (scanner.hasNextLine()) {

val command = scanner.nextLine().trim()

// Обробка команд

when {

command.startsWith("task") -> {

val parts = command.split(" ")

if (parts.size != 3) {

println("Invalid task command. Expected format: task NAME X")

continue

}

val taskName = parts[1]

val taskId = try {

parts[2].toInt()

} catch (e: NumberFormatException) {

println("Invalid task ID. It should be an integer.")

continue

}

if (taskName.contains(" ")) {

println("Task name cannot contain spaces.")

continue

}

taskExecutor.executeTask(taskName, taskId)

}

command == "get" -> {

println(taskExecutor.getLastResult())

}

command == "finish grace" -> {

taskExecutor.finish(graceful = true)

break

}

command == "finish force" -> {

taskExecutor.finish(graceful = false)

break

}

command == "clean" -> {

taskExecutor.cleanResultsFile()

}

command == "help" -> {

taskExecutor.showHelp()

}

else -> {

println("Unknown command. Type 'help' for a list of commands.")

}

}

}

}

## ***Файл Tasks.kt***

## 1.)- Опис структури

Цей файл містить опис класів для роботи з завданнями:

Клас Task реалізує інтерфейс Callable<String>, що дозволяє виконувати завдання асинхронно (в окремому потоці). Кожне завдання вимагає певного часу для виконання, який задається випадковим чином у діапазоні від 300 до 5000 мс.

Клас TaskExecutor керує виконанням завдань. Він містить:

ExecutorService (через пул потоків), що дозволяє асинхронно запускати завдання.

LinkedBlockingQueue для черги завдань.

Метод для виконання завдань (executeTask), отримання останнього результату (getLastResult), завершення роботи (finish), очищення результатів (cleanResultsFile), та запису результатів у файл (writeToFile).

Клас TaskExecutor також обробляє синхронізацію доступу до результатів, щоб уникнути проблем з паралельним доступом до спільних даних.

## 2.)- Код

package org.jub.kotlin.hometask4 // Замість цього використовуйте правильний пакет вашого проекту

import java.util.concurrent.Callable

import kotlin.random.Random

// Клас Task, який реалізує Callable для асинхронного виконання

class Task(private val taskId: Int, private val taskTime: Long) : Callable<String> {

override fun call(): String {

Thread.sleep(taskTime)

return "Task $taskId completed in $taskTime ms"

}

}

package org.jub.kotlin.hometask4

import java.io.File

import java.util.concurrent.\*

class TaskExecutor(private val resultsFile: File) {

private val executorService: ExecutorService = Executors.newFixedThreadPool(6)

private val taskQueue: LinkedBlockingQueue<Callable<String>> = LinkedBlockingQueue()

private val results = mutableListOf<String>()

private var isFinished = false

fun executeTask(taskName: String, taskId: Int) {

if (isFinished) {

println("Cannot add new tasks, the program has finished.")

return

}

val task = Task(taskId, Random.nextLong(300, 5000))

taskQueue.add(task)

executorService.submit {

val result = task.call()

synchronized(results) {

results.add("$taskName: $result")

}

writeToFile()

}

}

fun getLastResult(): String {

synchronized(results) {

return if (results.isNotEmpty()) results.last() else "No results yet"

}

}

fun finish(graceful: Boolean) {

isFinished = true

if (graceful) {

executorService.shutdown()

if (!executorService.awaitTermination(60, TimeUnit.SECONDS)) {

executorService.shutdownNow()

}

} else {

executorService.shutdownNow()

}

}

fun cleanResultsFile() {

resultsFile.writeText("")

}

private fun writeToFile() {

synchronized(results) {

try {

resultsFile.printWriter().use { out ->

results.forEach {

out.println(it)

}

}

} catch (e: Exception) {

println("Error writing to file: ${e.message}")

}

}

}

fun showHelp() {

println("""Commands:

task NAME X: Execute task X, name it NAME, and write the result to the results file

get: Output the last result and its name to the console

finish grace: Stop accepting new tasks, finish all pending tasks, and stop the application

finish force: Stop the application without waiting for pending tasks to finish

clean: Clean the results file

help: Output guidelines to the console""")

}

}

## ***Файл Application.kt***

## 1.)- Опис структури

Цей файл містить інтерфейс Application, який передбачає загальну структуру для програм, що працюють з асинхронними завданнями:

Описаний метод waitToFinish(), який дозволяє програмі чекати завершення всіх операцій перед її закриттям.

Статичний метод create(), який, ймовірно, має на меті створити новий екземпляр програми, але реалізація ще не готова (містить TODO()).

Цей файл не використовується безпосередньо у поточній логіці програми, але, ймовірно, він є основою для майбутнього розвитку або для тестування.

## 2.)- Код

package org.jub.kotlin.hometask4

interface Application : Runnable {

/\*\*

\* Wait for your application to stop.

\* You might need this, you might not, that's fine.

\*/

fun waitToFinish() {}

companion object {

/\*\*

\* Creates a new Application, which writes results to the file at the given path.

\* You have to think of what type the `tasks` should be yourself.

\*

\* @param resultsFile path to a file in which results whould be stored

\* @param tasks List of available tasks.

\*/

fun create(resultsFile: String, tasks: Any?): Application = TODO()

}

}

## ***Файл ApplicationTest.kt***

## 1.)- Опис структури

Цей файл відповідає за тестування основної логіки програми (не зовсім коректно названий, бо виглядає як реальний виконавець):

В ньому відбувається ініціалізація об'єкта TaskExecutor і взаємодія з ним через консоль.

Зчитуються команди від користувача та обробляються за аналогією з кодом в Main.kt, але тут немає чіткої організації тестів або моків для тестування. По суті дублює логіку Main.kt.

## 2.)- Код

package org.jub.kotlin.hometask4

import java.io.File

import java.util.Scanner

fun main() {

val resultsFile = File("results.txt")

val taskExecutor = TaskExecutor(resultsFile)

val scanner = Scanner(System.`in`)

while (scanner.hasNextLine()) {

val command = scanner.nextLine().trim()

if (command == "finish grace" || command == "finish force") {

taskExecutor.finish(graceful = command == "finish grace")

break

}

when {

command.startsWith("task") -> {

val parts = command.split(" ")

if (parts.size != 3) {

println("Invalid task command. Expected format: task NAME X")

continue

}

val taskName = parts[1]

val taskId = try {

parts[2].toInt()

} catch (e: NumberFormatException) {

println("Invalid task ID. It should be an integer.")

continue

}

if (taskName.contains(" ")) {

println("Task name cannot contain spaces.")

continue

}

taskExecutor.executeTask(taskName, taskId)

}

command == "get" -> {

println(taskExecutor.getLastResult())

}

command == "finish grace" -> {

taskExecutor.finish(graceful = true)

break

}

command == "finish force" -> {

taskExecutor.finish(graceful = false)

break

}

command == "clean" -> {

taskExecutor.cleanResultsFile()

}

command == "help" -> {

taskExecutor.showHelp()

}

else -> {

println("Unknown command. Type 'help' for a list of commands.")

}

}

}

}

## ***Файл settings.gradle.kts***

1.)- Опис структури

Цей файл містить налаштування Gradle для проекту:

Встановлюється ім'я кореневого проекту: "kotlin\_lab\_3".

Цей файл не виконує жодної іншої функції, крім як вказує на основну інформацію про проект.

2.)- Код

rootProject.name = "kotlin\_lab\_3"

## ***Файл build.gradle.kts***

## 1.)- Опис структури

Файл конфігурації для Gradle:

Визначено використання плагіна для Kotlin (kotlin("jvm")), що дозволяє компілювати проект Kotlin для Java Virtual Machine (JVM).

Визначено залежності:

kotlin("stdlib") — стандартна бібліотека Kotlin.

Залежності для тестування з використанням JUnit 5 (для написання юніт-тестів).

Налаштовано використання JUnit Platform для запуску тестів.

Додаткові плагіни для статичного аналізу коду (Diktat і Detekt), які допомагають підтримувати чистоту та стандарти кодування.

## 2.)- Код

plugins {

kotlin("jvm") version "1.8.0" // або поточна версія Kotlin

application

}

dependencies {

implementation(kotlin("stdlib"))

// Залежність для JUnit 5

testImplementation("org.junit.jupiter:junit-jupiter-api:5.7.0")

testImplementation("org.junit.jupiter:junit-jupiter-engine:5.7.0")

testImplementation("org.junit.jupiter:junit-jupiter-params:5.7.0") // для параметризованих тестів

// Залежність для плагіна Gradle

testImplementation("org.junit.jupiter:junit-jupiter-engine:5.7.0")

}

tasks.test {

useJUnitPlatform() // Увімкнути JUnit 5

}

project.configureDiktat()

project.createDiktatTask()

project.configureDetekt()

project.createDetektTask()

## ***Структури, що використовуються в проекті (просте пояснення):***

**Файли та ресурси:**

Використовується файл для зберігання результатів (results.txt), в який записуються результати виконаних завдань.

**Основні об'єкти та структури даних:**

ExecutorService — для асинхронного виконання завдань.

Callable — інтерфейс, що реалізує клас Task для виконання завдань.

LinkedBlockingQueue — черга для зберігання завдань перед виконанням.

File — для роботи з файлами.

Scanner — для зчитування вводу з консолі.

Thread.sleep — для симуляції часу виконання завдань.

**Синхронізація:**

Використання synchronized для забезпечення безпечного доступу до результатів в умовах багатопотоковості.

**Методи для взаємодії з користувачем:**

Перевірка правильності команд, повідомлення про помилки та виведення результатів через консоль.

### ***Результат виконання програми:***

**На жаль, видається помилка, пов’язана, швидше всього, з версією Gradle і її конфліктом з Kotlin:**

FAILURE: Build failed with an exception.

\* Where:

Build file 'D:\╨Я╤А╨╛╤Ф╨║╤В╨╕\╨Ъ╨Я\_╨Ы╨атДЦ3.2\kotlin\_lab\_3\build.gradle.kts' line: 1

\* What went wrong:

Error resolving plugin [id: 'org.jetbrains.kotlin.jvm', version: '1.8.0']

> The request for this plugin could not be satisfied because the plugin is already on the classpath with an unknown version, so compatibility cannot be checked.

\* Try:

> Run with --stacktrace option to get the stack trace.

> Run with --info or --debug option to get more log output.

> Run with --scan to get full insights.

> Get more help at https://help.gradle.org.

BUILD FAILED in 939ms

3 actionable tasks: 3 up-to-date

**Якщо прибрати версію Kotlin, то наступна помилка буде щодо Detect і Dictat:**FAILURE: Build failed with an exception.

\* Where:

Build file 'D:\╨Я╤А╨╛╤Ф╨║╤В╨╕\╨Ъ╨Я\_╨Ы╨атДЦ3.2\kotlin\_lab\_3\build.gradle.kts' line: 22

\* What went wrong:

Script compilation errors:

Line 22: project.configureDiktat()

^ Unresolved reference: configureDiktat

Line 23: project.createDiktatTask()

^ Unresolved reference: createDiktatTask

Line 24: project.configureDetekt()

^ Unresolved reference: configureDetekt

Line 25: project.createDetektTask()

^ Unresolved reference: createDetektTask

4 errors

\* Try:

> Run with --stacktrace option to get the stack trace.

> Run with --info or --debug option to get more log output.

> Run with --scan to get full insights.

> Get more help at https://help.gradle.org.

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.11/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.

BUILD FAILED in 4s

3 actionable tasks: 3 up-to-date

**Якщо оголосити їх попередньо (id("io.gitlab.arturbosch.detekt") version "1.18.0"; id("com.saveourtool.diktat") version "1.0.0"), то видається:**FAILURE: Build failed with an exception.

\* Where:

Build file 'D:\╨Я╤А╨╛╤Ф╨║╤В╨╕\╨Ъ╨Я\_╨Ы╨атДЦ3.2\kotlin\_lab\_3\build.gradle.kts' line: 1

\* What went wrong:

Error resolving plugin [id: 'io.gitlab.arturbosch.detekt', version: '1.18.0']

> The request for this plugin could not be satisfied because the plugin is already on the classpath with an unknown version, so compatibility cannot be checked.

\* Try:

> Run with --stacktrace option to get the stack trace.

> Run with --info or --debug option to get more log output.

> Run with --scan to get full insights.

> Get more help at https://help.gradle.org.

BUILD FAILED in 2s

3 actionable tasks: 3 up-to-date

## **Висновок:**

Цей проект демонструє базову реалізацію асинхронних завдань із записом результатів у файл та взаємодією з користувачем через командний рядок. Використовуються потоки (через ExecutorService) для виконання завдань, а також базові концепції роботи з файлами та синхронізацією. Основний фокус — це управління завданнями, їх виконання та обробка результатів. На жаль, мені не вдалося його реалізувати через помилки.