Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №6

з дисципліни
«Об'єктно орієнтоване програмування»

Виконав:
Студент групи ІМ-21
Сірик Максим Олександрович
номер у списку групи: 22

Перевірив:
Порєв Віктор Миколайович

Київ 2023

Зміст

1 Мета:

Мета роботи: отримати вміння та навички використовувати засоби обміну інформацією та запрограмувати взаємодію незалежно працюючих програмних компонентів

2 Завдання:

1. Вимоги щодо організації системи Для виконання лабораторної роботи потрібно створити три незалежні програми, для чого можна створити три проекта у одному рішенні (Solution) Microsoft Visual Studio C++. У цьому випадку усі виконувані файли будуть знаходитися у спільній папці { }Debug або { }Release. Головний проект – програма Lab6 має бути менеджером, який керує двома іншими програмами – Object2 та Object3. Програма Lab6 повинна автоматично, без участі користувача налагоджувати співпрацю та виконувати обмін повідомленнями з програмами Object2 та Object3 для виконання потрібного завдання згідно наведеному вище алгоритму. У кожній програмі передбачити автоматичну обробку потрібних повідомлень і переходів з одного стану в інший. Далі вимоги щодо функціонування системи об'єктів.
2. 1. Для початку роботи користувач програми вибирає потрібний пункт меню програми Lab6. Далі з'являється вікно діалогу, у якому потрібно ввести параметри згідно варіанту завдання. У вікні діалогу користувач натискує кнопку "Так"(або "Виконати") і на цьому місія користувача закінчується – далі він тільки спостерігає, як програма сама автоматично виконає усе, що потрібно для отримання результату. Виклик інших програм – Object2 та Object3 головна програма Lab6 повинна робити без участі користувача.
3. 2. Обмін повідомленнями та масивами даних між програмами Lab6, Object2 та Object3 повинен відбуватися автоматично, без участі користувача. Програма Lab6 повинна автоматично у певній послідовності знаходити та викликати програми Object2 та Object3.
4. 3. У результаті одного сеансу роботи користувач повинен бачити головні вікна програм Object2 та Object3, у яких відображатимуться потрібні результати відповідно варіанту завдань. Для цього вікна програм повинні автоматично розташуватися так, щоб усі результати було видно. Програма Lab6 повинна залишатися у активному стані, щоб користувач мав можливість повторно виконати роботу.
5. 4. Передбачити варіанти успішної роботи у випадках, коли програми Object2 та Object3 (одна або обидві) до того вже були викликані.
6. 5. По завершенні роботи програми Lab6 повинні автоматично завершуватися і програми Object2 та Object3.

$$G = (22) \bmod 4 = 2 \tag{1}$$

2.1 Варіант 2:

1. (а) Користувач вводить значення n, Min, Max у діалоговому вікні.
   (б) Програма викликає програми Object2, 3 і забезпечує обмін повідомленнями для передавання та отримання інформації.
   (в) Створює вектор n дробових (double) чисел у діапазоні Min – Max
2. (а) Створює вектор n дробових (double) чисел у діапазоні Min – Max
   (б) Показує числові значення у декількох стовпчиках та рядках у власному головному вікні
   (в) Записує дані в Clipboard Windows у текстовому форматі
3. (а) Зчитує дані з Clipboard Windows
   (б) Виконує сортування масиву чисел і відображає його у декількох стовпчиках та рядках у власному головному вікні

3 Текст програми:

3.1 Module: com.github.erotourtes.main.view

Лістинг 1: DialogView.kt

```
package com.github.erotourtes.main.view

import com.github.erotourtes.data.MainModel
import tornadofx.*

class DialogView : View("Set Properties") {
    private val model by inject<MainModel>()

    override val root = vbox {
        label("Dialog")

        form {
            fieldset {
                field("Input n value") {
                    textfield {
                        bind(model.nProp)
                        filterInput { it.controlNewText.isInt() }
                    }
                }
                field("Input min value") {
                    textfield {
                        bind(model.minProp)
                        filterInput { it.controlNewText.isDouble() }
                    }
                }
                field("Input max value") {
                    textfield {
                        bind(model.maxProp)
                        filterInput { it.controlNewText.isDouble() }
                    }
                }
            }
        }

        button("Apply").action {
            with(model) {
                if (min > max) minProp.value = max.apply { maxProp.value = min }

                commit()
            }
            close()
        }
    }

    init {
        primaryStage.isAlwaysOnTop = true
    }
}
```

Лістинг 2: MainView.kt

```kotlin
package com.github.erotourtes.main.view

import com.github.erotourtes.data.MainModel
import com.github.erotourtes.data.MainState
import com.github.erotourtes.utils.*
import com.github.erotourtes.utils.EventEmitter
import com.github.erotourtes.utils.process_stream.ProcessReceiver
import com.github.erotourtes.utils.process_stream.ProcessSender
import javafx.beans.property.SimpleStringProperty
import javafx.geometry.Pos
import javafx.stage.FileChooser
import javafx.stage.StageStyle
import tornadofx.*
import java.io.File
import kotlin.reflect.KMutableProperty0
import kotlin.reflect.KProperty0

class MainController : Controller(), Closable {
    private var state = MainState()
    private val model: MainModel by inject()
    private val pathsModel: PathsModel by inject()
    private var program1: ProcessState? = null
    private var program2: ProcessState? = null

    init {
        model.item = state

        pathsModel.path1.onChange { program1?.close() }
        pathsModel.path2.onChange { program2?.close() }
    }

    fun showDialog() {
        val dialog = find<DialogView>()
        dialog.openModal(stageStyle = StageStyle.UTILITY)
    }

    fun send() {
        if (!isAlive(program1)) initChildProcessProgram1()
        program1?.sender?.send(MessageType.DATA, state.toString())
    }

    fun chooseFile(property: KProperty0<SimpleStringProperty>) {
        val file = chooseFile("Choose␣file", arrayOf(FileChooser.ExtensionFilter("JAR", "*
        if (file.isNotEmpty()) {
            val path = file.first().absolutePath
            property.get().value = path
            pathsModel.commit()
        }
    }

    override fun close() {
        Logger.log("dispose")
        program1?.close()
```

```
        program2?.close()
    }

    private fun initChildProcessProgram1() {
        Logger.log("child process 1 init")
        val path = pathsModel.path1.value

        runJar(path, ::program1)

        program1!!.ee.subscribe(MessageType.DATA) {
            val items = ListConverter.toList<Double>(it, String::toDouble)
            if (items.isEmpty()) return@subscribe

            if (!isAlive(program2)) initChildProcessProgram2()
            program2?.sender?.send(MessageType.DATA, ListConverter.toString(items))
        }
    }

    private fun initChildProcessProgram2() {
        Logger.log("child process 2 init")
        val path = pathsModel.path2.value
        runJar(path, ::program2)
    }

    private fun isAlive(program: ProcessState?) = program?.alive ?: false

    private fun runJar(path: String, property: KMutableProperty0<ProcessState?>) {
        runCatching {
            val file = File(path)
            runJarFile(file)?.let {
                property.get()?.close()
                property.set(ProcessState(it, ProcessSender(it, EventEmitter.getFormatter()
            }
        }.onFailure {
            Logger.log("runJar: ${it.message}", Logger.InfoType.ERROR)
        }
    }
}

class MainView : View("Main") {
    private val ctrl: MainController by inject()
    private val pathsModel: PathsModel by inject()

    override val root = vbox {
        menubar {
            menu("Lab") {
                item("Show dialog").action(ctrl::showDialog)
            }
        }

        borderpane {
            center = vbox {
                alignment = Pos.BASELINE_CENTER
                button("RUN") {
```

```
                disableWhen(pathsModel.path1.isEmpty or pathsModel.path2.isEmpty)
                action { ctrl.send() }
            }
        }

        bottom = form {
            fieldset {
                field("program 1") {
                    label().textProperty().bind(pathsModel.path1.stringBinding {
                        "Path: ${formatPath(it)}"
                    })
                    button("Choose file").action {
                        ctrl.chooseFile(pathsModel::path1)
                    }
                }
                field("program 2") {
                    label().textProperty().bind(pathsModel.path2.stringBinding {
                        "Path: ${formatPath(it)}"
                    })
                    button("Choose file").action {
                        ctrl.chooseFile(pathsModel::path2)
                    }
                }
            }
        }
    }
}
```

Лістинг 3: PathsModel.kt

```
package com.github.erotourtes.main.view

import javafx.beans.property.SimpleStringProperty
import tornadofx.*

class PathsModel : ViewModel() {
    private val KEY_FIRST_PATH = "path1"
    private val KEY_SECOND_PATH = "path2"

    val path1 = bind { SimpleStringProperty("", "", config.string(KEY_FIRST_PATH).orEmpty()
    val path2 = bind { SimpleStringProperty("", "", config.string(KEY_SECOND_PATH).orEmpty

    override fun onCommit() {
        with(config) {
            set(KEY_FIRST_PATH to path1.value)
            set(KEY_SECOND_PATH to path2.value)
            save()
        }
    }
}
```

3.2 Module: com.github.erotourtes.utils.process_stream

Лістинг 4: ProcessReceiver.kt

```kotlin
package com.github.erotourtes.utils.process_stream

import com.github.erotourtes.utils.*
import javafx.beans.property.SimpleStringProperty
import javafx.beans.value.ObservableValue
import kotlin.concurrent.thread

class ProcessReceiver(process: Process) : Closable, StringObservable {
    private val inputStream = process.inputStream.bufferedReader()
    private var thread: Thread? = null
    private val input = SimpleStringProperty()

    init {
        thread = thread {
            while (!Thread.currentThread().isInterrupted) {
                if (!inputStream.ready()) {
                    val res = runCatching { Thread.sleep(PROCESS_UPDATE_TIME) }
                    if (res.isFailure) break
                    continue
                }

                val inputData = inputStream.readLine()
                if (input.value == inputData) input.value = MessageType.EMPTY.type
                input.value = inputData

                Logger.log("INPUT: ${input.value}")
            }
            Logger.log("ProcessReceiver(interrupted-end)", Logger.InfoType.WARNING)
        }
    }

    override fun close() {
        inputStream.close()
        thread?.interrupt()
        thread?.join()
    }

    override fun getObservable(): ObservableValue<String> = input
}
```

Лістинг 5: ProcessSender.kt

```kotlin
package com.github.erotourtes.utils.process_stream

import com.github.erotourtes.utils.Closable
import com.github.erotourtes.utils.Logger
import com.github.erotourtes.utils.MessageType

class ProcessSender(process: Process, private val formatter: (MessageType, String) -> Strin
    private val outputStream = process.outputStream.bufferedWriter()

    fun send(type: MessageType, message: String = "") {
        runCatching {
            val combined = formatter(type, message)
            outputStream.write(combined)
```

```
            outputStream.newLine()
            outputStream.flush()
            Logger.log("write: $combined")
        }.onFailure {
            Logger.log("ProcessSender(writeMessage): ${it.message}", Logger.InfoType.ERROR)
        }
    }

    override fun close() {
        runCatching { outputStream.close() }.onFailure {
            Logger.log("ProcessSender(close): ${it.message}", Logger.InfoType.ERROR)
        }
    }
}
```

3.3 Module: com.github.erotourtes.data

Лістинг 6: MainState.kt

```
package com.github.erotourtes.data

import tornadofx.*
import java.lang.IllegalStateException

class MainModel : ItemViewModel<MainState>() {
    val nProp = bind(MainState::n)
    val minProp = bind(MainState::minValue)
    val maxProp = bind(MainState::maxValue)

    override fun onCommit() {
        super.onCommit()
        if (min > max) throw IllegalStateException("min > max")
        if (n < 0) throw IllegalStateException("n < 0")
    }

    val min by minProp
    val max by maxProp
    val n by nProp
}

data class MainState(var n: Int = 0, var minValue: Double = 0.0, var maxValue: Double = 0.0
    override fun toString(): String {
        return "$n $minValue $maxValue"
    }

    companion object {
        fun fromString(string: String): MainState? {
            return runCatching {
                val values = string.split(" ")
                MainState(values[0].toInt(), values[1].toDouble(), values[2].toDouble())
            }.getOrNull()
        }
    }
}
```

3.4 Module: com.github.erotourtes.utils

Лістинг 7: EventEmitter.kt

```kt
package com.github.erotourtes.utils

class EventEmitter<T>(private val source: T) where T : Closable, T : StringObservable {
    private val listeners = mutableMapOf<MessageType, MutableList<(String) -> Unit>>()

    init {
        val observable = source.getObservable()
        observable.addListener { _, _, newMsg ->
            if (newMsg.isEmpty()) return@addListener

            val type = MessageType.getTypeOf(newMsg)
            listeners[type]?.forEach { it(newMsg.removePrefix(type.type).trim()) }
        }
    }

    fun subscribe(event: MessageType, listener: (String) -> Unit) {
        listeners.getOrPut(event) { mutableListOf() }.add(listener)
    }

    fun close() {
        listeners.clear()
        source.close()
    }

    companion object {
        fun format(event: MessageType, message: String = "") = "${event.type} $message"

        fun getFormatter() = { event: MessageType, message: String? -> format(event, messag
    }
}
```

Лістинг 8: ProcessState.kt

```kt
package com.github.erotourtes.utils

import com.github.erotourtes.utils.process_stream.ProcessReceiver
import com.github.erotourtes.utils.process_stream.ProcessSender

class ProcessState(
    private val process: Process,
    val sender: ProcessSender,
    receiver: ProcessReceiver,
) : Closable {
    val alive get() = isAlive && process.isAlive
    val ee = EventEmitter(receiver)

    private var isAlive: Boolean = true
    private var isDestroyedMsgReceived: Boolean = false
    private var isClosed: Boolean = false

    init {
```

```
        ee.subscribe(MessageType.ON_DESTROY) { isDestroyedMsgReceived = true }
    }

    override fun close() {
        if (isClosed) {
            Logger.log("ProcessState(close): process already closed", Logger.InfoType.WARN
            return
        }
        runCatching {
            sender.send(MessageType.DESTROY)
            // To see the logs
            while (!isDestroyedMsgReceived) Thread.sleep(PROCESS_UPDATE_TIME)
            Logger.log("process has been closed isAlive:${process.isAlive}")
            sender.close()
            ee.close()
            process.destroy()
            isAlive = false
            isClosed = true
        }.onFailure {
            Logger.log("ProcessState(close): ${it.message}", Logger.InfoType.ERROR)
        }
    }
}
```

Лістинг 9: Utils.kt

```
package com.github.erotourtes.utils

import javafx.beans.value.ObservableValue
import java.io.File

fun runJarFile(file: File): Process? {
    val java = "${System.getProperty("java.home")}/bin/java"
    val path = file.absolutePath
    return Runtime.getRuntime().exec("$java -jar $path")
}

object Logger {
    var isLogging = true
    var preMessage = ""

    enum class InfoType {
        INFO, ERROR, WARNING
    }

    val color = mapOf(
        InfoType.INFO to "\u001B[32m",
        InfoType.ERROR to "\u001B[31m",
        InfoType.WARNING to "\u001B[33m",
    )

    val reset = "\u001B[0m"

    fun log(message: String, type: InfoType = InfoType.INFO) {
        if (!isLogging) return
```

```
        println("${color[type]}$preMessage: $message$reset")
    }
}

object ListConverter {
    inline fun <reified T> toString(list: List<T>): String = list.joinToString(",")

    inline fun <reified T> toList(string: String, converter: (String) -> T): List<T> {
        if (string.isEmpty()) return emptyList()
        return string.split(",").map {
            converter(it.trim())
        }
    }
}

const val PROCESS_UPDATE_TIME = 100L

enum class MessageType(val type: String) {
    EMPTY("__EMPTY__"), DATA("__DATA__"), DESTROY("__DESTROY__"), ON_DESTROY("__ON_DESTROY_


    companion object {
        fun getTypeOf(message: String): MessageType = MessageType.entries.find { isOfType(i

        fun isOfType(type: MessageType, message: String) = message.startsWith(type.type)
    }
}

interface Closable {
    fun close()
}

interface StringObservable {
    fun getObservable(): ObservableValue<String>
}

fun formatPath(path: String?, takeLast: Int = 2): String {
    if (path == null) return ""
    var count = 0
    for (i in path.indices.reversed()) {
        val char = path[i]
        if (char == '/') count++

        if (count == takeLast) return path.substring(i + 1)
    }

    return path
}
```

3.5 Module: com.github.erotourtes.utils.self_stream

Листинг 10: SelfInputStreamReceiver.kt

```
package com.github.erotourtes.utils.self_stream
```

```
import com.github.erotourtes.utils.*
import javafx.application.Platform
import javafx.beans.property.SimpleStringProperty
import java.io.*
import java.util.concurrent.atomic.AtomicBoolean
import kotlin.concurrent.thread

class SelfInputStreamReceiver : Closable, StringObservable {
    private val input = SimpleStringProperty()
    private var isReading = AtomicBoolean(true)
    private var readerThread: Thread? = null
    private val inputStream = BufferedReader(InputStreamReader(System.`in`))

    init {
        readerThread = thread(start = true, isDaemon = false) {
            while (isReading.get() && !Thread.currentThread().isInterrupted) {
                if (!inputStream.ready()) {
                    val res = runCatching { Thread.sleep(PROCESS_UPDATE_TIME) }
                    if (res.isFailure) break
                    continue
                }
                val input = inputStream.readLine()
                Platform.runLater {
                    if (input == this.input.value) this.input.value = MessageType.EMPTY.typ
                    this.input.value = input
                    Logger.log("INPUT: $input")
                }
            }
            Logger.log("SelfInputStreamReceiver(interrupt-end)", Logger.InfoType.WARNING)
        }
    }

    override fun close() {
        isReading.set(false)
        inputStream.close()
        readerThread?.interrupt()
        readerThread?.join()
        input.value = MessageType.EMPTY.type
        /*
            used to receive `null` (now EMPTY) msg in the inputMessage listener
            because FirstController.dispose when close stream sets inputMessage to null
            and then from the JavaFX thread I call Platform.exit()
            if I don't call Platform.exit() then the app will not close and the process wil
            even if stop() method is called

            So the sequence is:
            1. App.close() -> 2. FirstController.dispose() -> 3. SelfInputStreamReceiver.cl

            All because the documentation says:
            "The implementation of this (app.stop()) method provided by the Application cla
            which is lie because it stops the app
            (maybe it stops the javafx thread, but I don't have my own daemon threads, so t
         */
```

```
    }

    override fun getObservable() = input
}
```

Лістинг 11: SelfOutputStreamSender.kt

```
package com.github.erotourtes.utils.self_stream

import com.github.erotourtes.utils.Logger
import com.github.erotourtes.utils.MessageType

class SelfOutputStreamSender(private val formatter: (MessageType, String) -> String) {
    fun send(message: String) {
        Logger.log("OUTPUT: $message")
        println(formatter(MessageType.DATA, message))
    }

    fun send(type: MessageType) {
        Logger.log("OUTPUT: $type")
        println(formatter(type, ""))
    }
}
```

3.6 Module: com.github.erotourtes.second

Лістинг 12: SecondApp.kt

```
package com.github.erotourtes.second

import com.github.erotourtes.utils.*
import com.github.erotourtes.utils.EventEmitter
import com.github.erotourtes.utils.self_stream.SelfInputStreamReceiver
import com.github.erotourtes.utils.self_stream.SelfOutputStreamSender
import javafx.application.Platform
import javafx.collections.FXCollections
import javafx.collections.ObservableList
import tornadofx.*
import java.lang.management.ManagementFactory


class SecondApp : App(SecondView::class) {
    override fun init() {
        super.init()
        val pid = ManagementFactory.getRuntimeMXBean().name
        Logger.preMessage = "SecondApp($pid)"
    }

    override fun stop() {
        Logger.log("stop method")
        find<SecondController>().close()
        Logger.log("stop method end", Logger.InfoType.WARNING)
        super.stop()
    }
}
```

```
class SecondController : Controller(), Closable {
    private val ee = EventEmitter(SelfInputStreamReceiver())
    private val pSender = SelfOutputStreamSender(EventEmitter.getFormatter())
    val randoms: ObservableList<Double> = FXCollections.observableArrayList()

    init {
        ee.subscribe(MessageType.DESTROY) { Platform.exit() }
        ee.subscribe(MessageType.DATA) {
            runCatching {
                val list = ListConverter.toList(it, String::toDouble).sorted()
                randoms.clear()
                randoms.addAll(list)
            }.onFailure {
                Logger.log("SecondController: ${it.message}", Logger.InfoType.ERROR)
                Platform.exit()
            }
        }
    }

    override fun close() {
        Logger.log("dispose")
        pSender.send(MessageType.ON_DESTROY)
        ee.close()
    }
}

class SecondView : View("Second View") {
    private val ctrl by inject<SecondController>()

    override val root = vbox {
        listview(ctrl.randoms)
    }
}
```

3.7 Module: com.github.erotourtes.first

Лістинг 13: FirstApp.kt

```
package com.github.erotourtes.first

import com.github.erotourtes.data.MainState
import com.github.erotourtes.utils.*
import com.github.erotourtes.utils.EventEmitter
import com.github.erotourtes.utils.self_stream.SelfInputStreamReceiver
import com.github.erotourtes.utils.self_stream.SelfOutputStreamSender
import javafx.application.Platform
import javafx.collections.FXCollections
import javafx.collections.ObservableList
import tornadofx.*
import java.lang.management.ManagementFactory

class FirstApp : App(FirstView::class) {
    override fun init() {
        super.init()
        val pid = ManagementFactory.getRuntimeMXBean().name
```

```
        Logger.preMessage = "FirstApp($pid)"
    }

    override fun stop() {
        Logger.log("stop method")
        find<FirstController>().close()
        Logger.log("stop method end", Logger.InfoType.WARNING)
        super.stop()
    }
}

class FirstController : Controller(), Closable {
    private val ee = EventEmitter(SelfInputStreamReceiver())
    private val pSender = SelfOutputStreamSender(EventEmitter.getFormatter())
    private var state = MainState()

    val randoms: ObservableList<Double> = FXCollections.observableArrayList()

    init {
        ee.subscribe(MessageType.DESTROY) { Platform.exit() }
        ee.subscribe(MessageType.DATA) {
            runCatching {
                Logger.log("INPUT: $it")
                state = MainState.fromString(it) ?: return@subscribe
                regenerateDiapason()

                pSender.send(ListConverter.toString(randoms))
            }.onFailure {
                Logger.log("FirstController: ${it.message}", Logger.InfoType.ERROR)
                Platform.exit()
            }
        }
    }

    private fun regenerateDiapason() {
        val (n, min, max) = state

        randoms.clear()
        for (i in 0 until n)
            randoms.add(min + (max - min) * Math.random())
    }

    override fun close() {
        Logger.log("dispose")
        pSender.send(MessageType.ON_DESTROY)
        ee.close()
    }
}

class FirstView : View("First View") {
    private val ctrl: FirstController by inject()

    override val root = vbox {
        listview(ctrl.randoms)
```

```
    }
}
```

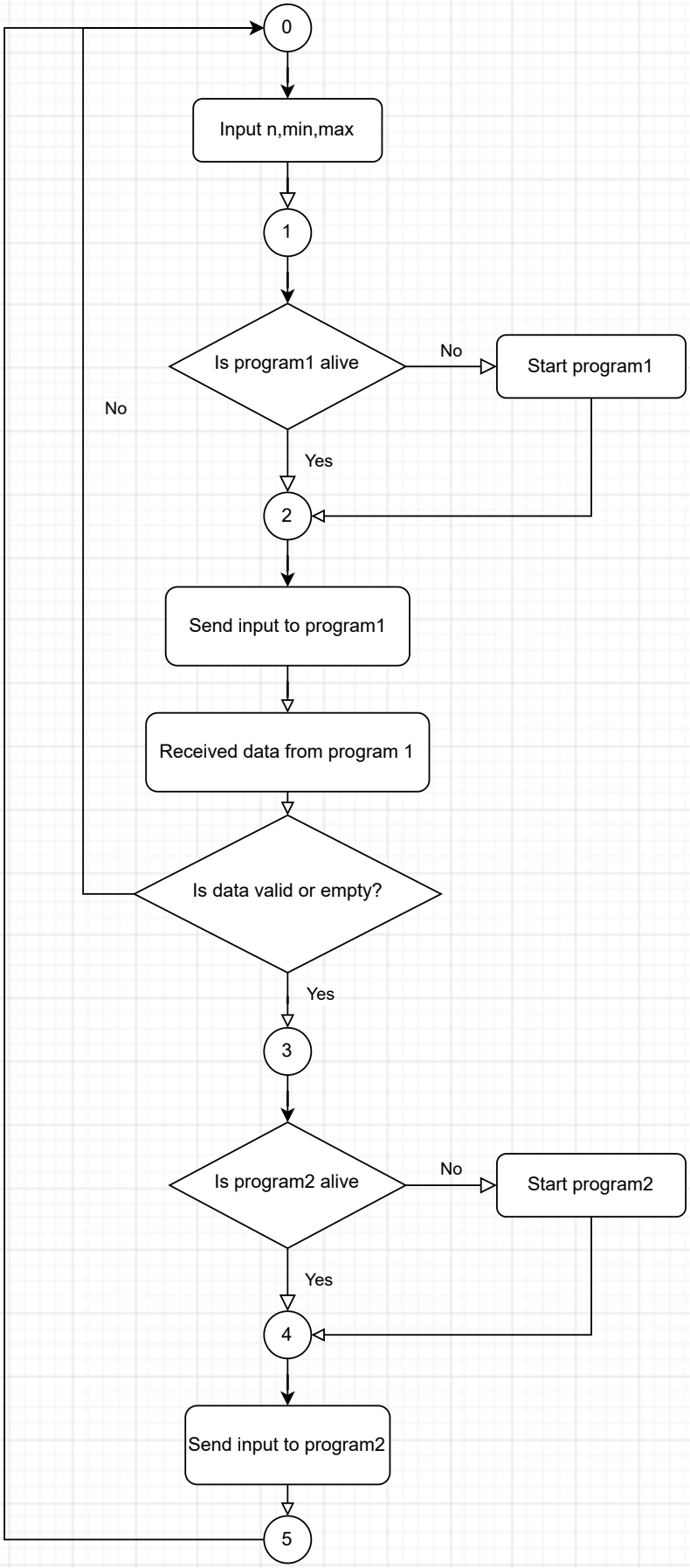
3.8 Module: com.github.erotourtes.main

Лістинг 14: MainApp.kt

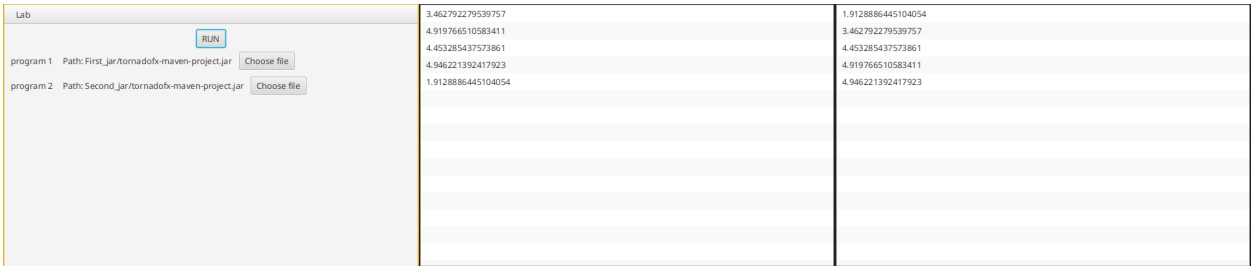
```
package com.github.erotourtes.main

import com.github.erotourtes.main.view.MainController
import com.github.erotourtes.main.view.MainView
import com.github.erotourtes.utils.Logger
import tornadofx.*
import java.lang.management.ManagementFactory

class MainApp : App(MainView::class) {
    override fun init() {
        super.init()
        val pid = ManagementFactory.getRuntimeMXBean().name
        Logger.preMessage = "MainApp($pid)"
    }

    override fun stop() {
        Logger.log("stop method")
        find<MainController>().close()
        Logger.log("stop method end", Logger.InfoType.WARNING)
        super.stop()
    }
}
```

4 Ілюстрації:

4.1 State diagram:


0
Input n,min,max
1
Is program1 alive
No
Start program1
No
Yes
2
Send input to program1
Received data from program 1
Is data valid or empty?
Yes
3
Is program2 alive
No
Start program2
Yes
4
Send input to program2
5

4.2 Images:



5 Висновки:

Отже, я отримав вміння та навички використання обміну інформацією та запрограмував взаємодію незалежно працюючих програмних компонентів