Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерных технологий

Лабораторная работа №8

Вариант 213788

Выполнила:

Павличенко Софья Алексеевна, P3115

Проверил:

Вербовой Александр Александрович

Санкт-Петербург 2024г.

Оглавление

Задание....3
Диаграмма классов реализованной объектной модели 4
Решение 5
Исходный код программы....5
Заключение....21

Задание

1. Интерфейс должен быть реализован с помощью библиотеки Swing
2. Графический интерфейс клиентской части должен поддерживать **русский**, **румынский**, **болгарский** и **английский (Новая Зеландия)** языки / локали. Должно обеспечиваться корректное отображение чисел, даты и времени в соответствие с локалью. Переключение языков должно происходить без перезапуска приложения. Локализованные ресурсы должны храниться в **классе**.

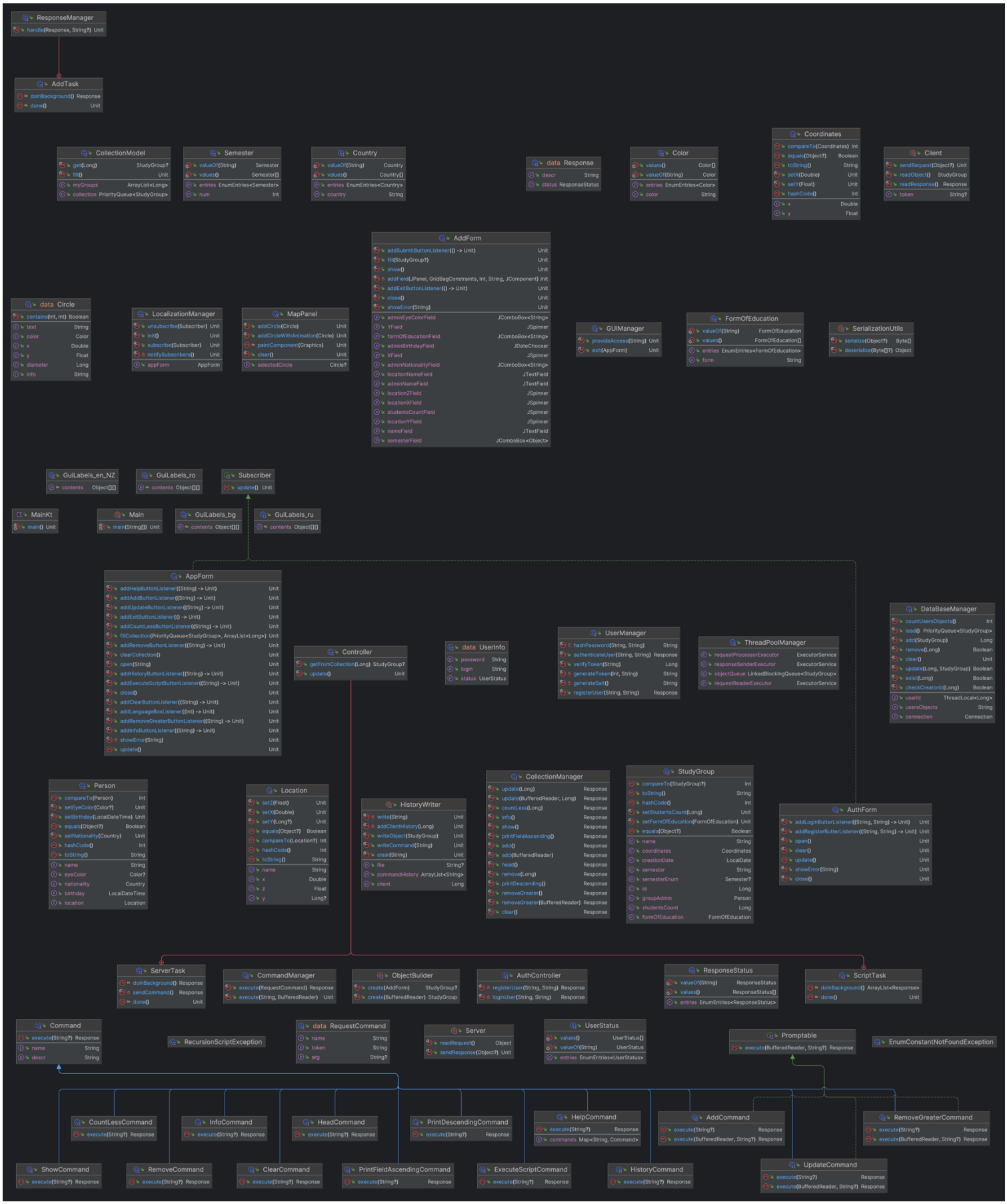
Доработать программу из лабораторной работы №7 следующим образом:

Заменить консольный клиент на клиент с графическим интерфейсом пользователя (GUI). В функционал клиента должно входить:

1. Окно с авторизацией/регистрацией.
2. Отображение текущего пользователя.
3. Таблица, отображающая все объекты из коллекции
    a. Каждое поле объекта - отдельная колонка таблицы.
    b. Строки таблицы можно фильтровать/сортировать по значениям любой из колонок. Сортировку и фильтрацию значений столбцов реализовать с помощью Streams API.
4. Поддержка всех команд из предыдущих лабораторных работ.
5. Область, визуализирующую объекты коллекции
    a. Объекты должны быть нарисованы с помощью графических примитивов с использованием Graphics, Canvas или аналогичных средств графической библиотеки.
    b. При визуализации использовать данные о координатах и размерах объекта.
    c. Объекты от разных пользователей должны быть нарисованы разными цветами.
    d. При нажатии на объект должна выводиться информация об этом объекте.
    e. При добавлении/удалении/изменении объекта, он должен **автоматически** появиться/исчезнуть/измениться на области как владельца, так и всех других клиентов.
    f. При отрисовке объекта должна воспроизводиться согласованная с преподавателем **анимация**.
6. Возможность редактирования отдельных полей любого из объектов (принадлежащего пользователю). Переход к редактированию объекта возможен из таблицы с общим списком объектов и из области с визуализацией объекта.
7. Возможность удаления выбранного объекта (даже если команды remove ранее не было).

Перед непосредственной разработкой приложения **необходимо** согласовать прототип интерфейса с преподавателем. Прототип интерфейса должен быть создан с помощью средства для построения прототипов интерфейсов (mockplus, draw.io, etc.)

Диаграмма классов реализованной объектной модели



Решение

Исходный код программы

client.Main.kt

```
package client

import client.controllers.AuthController
import client.managers.GUIManager
import client.views.AuthForm
import java.awt.Color
import javax.swing.SwingUtilities
import javax.swing.UIManager


fun main() {
    UIManager.put("ToolTip.background", Color.WHITE)
    SwingUtilities.invokeLater {
        val authForm = AuthForm()
        val guiManager = GUIManager(authForm)
        val authController = AuthController(authForm, guiManager)
    }
}
```

client.GUIManager.kt

```
package client.managers

import client.controllers.Controller
import client.localization.LocalizationManager
import client.models.CollectionModel
import client.views.AppForm
import client.views.AuthForm
import javax.swing.SwingUtilities

class GUIManager(private val authForm: AuthForm) {

    private val localizationManager = LocalizationManager()

    fun provideAccess(user: String) {
        SwingUtilities.invokeLater {
            val appForm = AppForm()
            val collectionModel = CollectionModel()
            val controller = Controller(appForm, collectionModel, this)
            localizationManager.appForm = appForm
            localizationManager.init()
            localizationManager.subscribe(authForm)
            localizationManager.subscribe(appForm)
            appForm.open(user)
        }
    }

    fun exit(appForm: AppForm) {
        SwingUtilities.invokeLater {
            appForm.close()
            authForm.open()
        }
```

```
    }
}
```

client.AuthForm.kt

```
package client.views

import client.localization.Subscriber
import java.awt.Color
import java.util.*
import javax.swing.*

class AuthForm: Subscriber {
    private var r = ResourceBundle.getBundle("client.localization.GuiLabels",
Locale("ru", "RU"))

    private val frame = JFrame("Authentication")
    private val groupLayout = GroupLayout(frame.contentPane)

    private val loginLabel = JLabel("${r.getString("login")}:")
    private val passwordLabel = JLabel("${r.getString("password")}:")
    private val loginField = JTextField(18)
    private val passwordField = JPasswordField(18)
    private val errorLabel = JLabel().apply { foreground = Color.RED; text =
" " }
    private val loginButton = JButton(r.getString("signIn"))
    private val registerButton = JButton(r.getString("signUp"))

    init {
        frame.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
        frame.setLocationRelativeTo(null)

        frame.contentPane.layout = groupLayout
        groupLayout.autoCreateGaps = true
        groupLayout.autoCreateContainerGaps = true

        groupLayout.setHorizontalGroup(
            groupLayout.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addGroup(
                    groupLayout.createSequentialGroup()
                        .addGroup(

groupLayout.createParallelGroup(GroupLayout.Alignment.TRAILING)
                                .addComponent(loginLabel)
                                .addComponent(passwordLabel)
                        )
                        .addGroup(

groupLayout.createParallelGroup(GroupLayout.Alignment.LEADING)
                                .addComponent(
                                    loginField,
                                    GroupLayout.DEFAULT_SIZE,
                                    GroupLayout.DEFAULT_SIZE,
                                    Int.MAX_VALUE
                                )
                                .addComponent(
                                    passwordField,
                                    GroupLayout.DEFAULT_SIZE,
                                    GroupLayout.DEFAULT_SIZE,
                                    Int.MAX_VALUE
                                )
```

```
                        )
                )
                .addComponent(errorLabel, GroupLayout.DEFAULT_SIZE,
GroupLayout.DEFAULT_SIZE, Int.MAX_VALUE)
                .addComponent(loginButton, GroupLayout.DEFAULT_SIZE,
GroupLayout.DEFAULT_SIZE, Int.MAX_VALUE)
                .addComponent(registerButton, GroupLayout.DEFAULT_SIZE,
GroupLayout.DEFAULT_SIZE, Int.MAX_VALUE)
        )

        groupLayout.setVerticalGroup(
            groupLayout.createSequentialGroup()
                .addGroup(

groupLayout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                        .addComponent(loginLabel)
                        .addComponent(loginField)
                )
                .addGroup(

groupLayout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                        .addComponent(passwordLabel)
                        .addComponent(passwordField)
                )
                .addComponent(errorLabel, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, Int.MAX_VALUE)
                .addComponent(loginButton, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, Int.MAX_VALUE)
                .addComponent(registerButton, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, Int.MAX_VALUE)
        )

        frame.pack()
        frame.isVisible = true
    }

    fun showError(message: String) {
        errorLabel.text = message
    }

    fun addLoginButterListener(listener: (login: String, password: String) ->
Unit) {
        loginButton.addActionListener {
            listener(loginField.text, String(passwordField.password))
        }
    }

    fun addRegisterButterListener(listener: (login: String, password: String)
-> Unit) {
        registerButton.addActionListener {
            listener(loginField.text, String(passwordField.password))
        }
    }

    fun clear() {
            loginField.text = null
            passwordField.text = null
    }

    fun open() {
            frame.isVisible = true
    }

    fun close() {
```

```
            errorLabel.text = " "
            frame.dispose()
    }

    override fun update() {
        r = ResourceBundle.getBundle("client.localization.GuiLabels",
Locale.getDefault())

        loginLabel.text = "${r.getString("login")}:"
        passwordLabel.text = "${r.getString("password")}:"
        errorLabel.text = " "

        loginButton.text = r.getString("signIn")
        registerButton.text = r.getString("signUp")

        SwingUtilities.updateComponentTreeUI(frame)
    }

}
```

client.AuthController.kt

```
package client.controllers

import client.managers.GUIManager
import client.network.Client
import client.views.AuthForm
import common.communication.*
import javax.swing.SwingUtilities

class AuthController(authForm: AuthForm, guiManager: GUIManager) {

    init {
        SwingUtilities.invokeLater {
            authForm.addLoginButterListener { login, password ->
                if (login == "") authForm.showError("Логин не может быть
пустым!")
                else if (password == "") authForm.showError("Пароль не может
быть пустым!")
                else {
                    val response = loginUser(login, password)
                    if (response.status == ResponseStatus.SUCCESS) {
                        authForm.close()
                        guiManager.provideAccess(login)
                    } else {
                        authForm.showError(response.descr)
                    }
                    authForm.clear()
                }
            }
            authForm.addRegisterButterListener { login, password ->
                if (login == "") authForm.showError("Логин не может быть
пустым!")
                else if (password == "") authForm.showError("Пароль не может
быть пустым!")
                else {
                    val response = registerUser(login, password)
                    if (response.status == ResponseStatus.SUCCESS) {
                        authForm.close()
                        guiManager.provideAccess(login)
                    } else {
```

```
                        authForm.showError(response.descr)
                    }
                    authForm.clear()
                }
            }
        }
    }

    private fun registerUser(login: String, password: String): Response {
        Client.sendRequest(UserInfo(login, password,
UserStatus.REGISTRATION))
        val response = Client.readResponse()
        if (response.status == ResponseStatus.SUCCESS) {
            Client.token = Client.readResponse().descr
        }
        return response
    }

    private fun loginUser(login: String, password: String): Response {
        Client.sendRequest(UserInfo(login, password,
UserStatus.AUTHENTICATION))
        val response = Client.readResponse()
        if (response.status == ResponseStatus.SUCCESS) {
            Client.token = Client.readResponse().descr
        }
        return response
    }
}
```

client.AppForm.kt

```
package client.views

import client.graphics.Circle
import client.localization.Subscriber
import common.models.StudyGroup
import java.awt.*
import java.io.File
import java.net.URL
import java.util.*
import javax.swing.*
import javax.swing.WindowConstants.EXIT_ON_CLOSE
import javax.swing.table.DefaultTableModel
import javax.swing.table.TableRowSorter
import kotlin.collections.ArrayList

class AppForm: Subscriber {

    private var r = ResourceBundle.getBundle("client.localization.GuiLabels",
Locale.getDefault())

    private val userLabel = JLabel("${r.getString("user")}: ")
    private val userLogin = JLabel()
    private val languageLabel = JLabel("${r.getString("language")}: ")
    private val languageComboBox = JComboBox(arrayOf("Русский", "Română",
"Български", "English (NZ)")).apply {
        selectedIndex = when (Locale.getDefault()) {
            Locale("ru", "RU") -> 0
            Locale("ro", "RO") -> 1
            Locale("bg", "BG") -> 2
            Locale("en", "NZ") -> 3
```

```
            else -> 4
        }
    }

    private val exitButton = JButton(r.getString("exit"))
    private val helpButton = JButton(r.getString("help"))
    private val header = JPanel().apply {
        layout = BorderLayout()

        add(JPanel().apply {
            layout = FlowLayout(FlowLayout.LEFT)
            add(userLabel)
            add(userLogin)
        }, BorderLayout.WEST)

        add(JPanel().apply {
            layout = FlowLayout(FlowLayout.RIGHT)
            add(languageLabel)
            add(languageComboBox)
            add(helpButton)
            add(exitButton)
        }, BorderLayout.EAST)
    }

    private val addButton = JButton(r.getString("add"))
    private val removeButton = JButton(r.getString("remove"))
    private val updateButton = JButton(r.getString("update"))
    private val clearButton = JButton(r.getString("clear"))
    private val removeGreaterButton = JButton(r.getString("removeGreater"))

    private val modifierButtons = JPanel().apply {
        layout = FlowLayout(FlowLayout.CENTER)
        add(addButton)
        add(removeButton)
        add(updateButton)
        add(clearButton)
        add(removeGreaterButton)
    }
    private val tableModel = object : DefaultTableModel(arrayOf("id",
r.getString("groupName"), "X", "Y", r.getString("creationDate"),
r.getString("studentsCount"), r.getString("formOfEducation"),
r.getString("semester"), r.getString("name") + " " + r.getString("ofAdmin"),
r.getString("birthday") + " " + r.getString("ofAdmin"),
r.getString("eyeColor") + " " + r.getString("ofAdmin"),
r.getString("originCountry") + " " + r.getString("ofAdmin"),
r.getString("location") + " " + r.getString("ofAdmin"), "x", "y", "z"), 0) {
        override fun isCellEditable(row: Int, column: Int): Boolean {
            return false
        }
    }

    private val table = JTable(tableModel).apply {
        rowSorter = TableRowSorter(tableModel).apply {
            setComparator(0, Comparator<Long> {n1, n2 -> n1.compareTo(n2)})
            setComparator(2, Comparator<Double> {n1, n2 -> n1.compareTo(n2)})
            setComparator(3, Comparator<Float> {n1, n2 -> n1.compareTo(n2)})
            setComparator(5, Comparator<Long> {n1, n2 -> n1.compareTo(n2)})
            setComparator(13, Comparator<Double> {n1, n2 ->
n1.compareTo(n2)})
            setComparator(14, Comparator<Long> {n1, n2 -> n1.compareTo(n2)})
            setComparator(15, Comparator<Float> {n1, n2 -> n1.compareTo(n2)})
        }
        columnModel.getColumn(0).preferredWidth = 30
        columnModel.getColumn(1).preferredWidth = 80
```

```
        columnModel.getColumn(2).preferredWidth = 40
        columnModel.getColumn(3).preferredWidth = 40
        columnModel.getColumn(4).preferredWidth = 100
        columnModel.getColumn(5).preferredWidth = 50
        columnModel.getColumn(6).preferredWidth = 100
        columnModel.getColumn(7).preferredWidth = 50
        columnModel.getColumn(8).preferredWidth = 60
        columnModel.getColumn(9).preferredWidth = 100
        columnModel.getColumn(10).preferredWidth = 60
        columnModel.getColumn(11).preferredWidth = 60
        columnModel.getColumn(12).preferredWidth = 60
        columnModel.getColumn(13).preferredWidth = 40
        columnModel.getColumn(14).preferredWidth = 40
        columnModel.getColumn(15).preferredWidth = 40
    }

    private val scrollTable = JScrollPane(table)

    private val visualization = MapPanel()
    private val displayCollection = JTabbedPane().apply {
        addTab(r.getString("table"), scrollTable)
        addTab(r.getString("map"), visualization)
    }
    private val body = JPanel().apply {
        layout = BoxLayout(this, BoxLayout.Y_AXIS)
        add(displayCollection)
        add(modifierButtons)
    }

    private val countLessLabel = JLabel(r.getString("countLess"))
    private val countLessNField = JTextField(2)
    private val countLessStudentsLabel = JLabel(r.getString("lessStudents"))
    private val countLessButton = JButton(r.getString("count"))
    private val countLessForm = JPanel().apply {
        layout = FlowLayout(FlowLayout.LEFT)
        add(countLessLabel)
        add(countLessNField)
        add(countLessStudentsLabel)
    }

    private val executeScriptButton = JButton(r.getString("executeScript"))
    private val historyButton = JButton(r.getString("history"))
    private val infoButton = JButton(r.getString("info"))

    private val otherButtons = JPanel(GridBagLayout()).apply {
        val gbc = GridBagConstraints().apply {
            fill = GridBagConstraints.HORIZONTAL
            anchor = GridBagConstraints.NORTH
            weightx = 1.0
        }
        gbc.gridy = 0
        gbc.insets = Insets(29, 2, 2, 5)
        add(infoButton, gbc)
        gbc.gridy = 1
        gbc.insets = Insets(2, 2, 2, 5)
        add(historyButton, gbc)
        gbc.gridy = 2
        add(executeScriptButton, gbc)
        gbc.gridy = 3
        add(countLessForm, gbc)
        gbc.gridy = 4
        add(countLessButton, gbc)
        gbc.gridy = 5
        gbc.weighty = 1.0
```

```
        add(Box.createVerticalGlue(), gbc)
    }

    private val frame = JFrame("meow").apply {
        isVisible = false
        defaultCloseOperation = EXIT_ON_CLOSE
        setLocationRelativeTo(null)
        add(header, BorderLayout.NORTH)
        add(body, BorderLayout.CENTER)
        add(otherButtons, BorderLayout.EAST)
        setSize(1200, 600)
    }


    fun open(user: String) {
            frame.isVisible = true
            userLogin.text = user
    }

    fun close() {
            frame.dispose()
    }

    fun addExitButtonListener(listener: () -> Unit) {
        exitButton.addActionListener {
            listener()
        }
    }

    fun addHelpButtonListener(listener: (command: String) -> Unit) {
        helpButton.addActionListener {
            listener("help")
        }
    }

    fun addLanguageBoxListener(listener: (Int) -> Unit) {
        languageComboBox.addItemListener {
            listener(languageComboBox.selectedIndex)
        }
    }

    fun addAddButtonListener(listener: (command: String) -> Unit) {
        addButton.addActionListener {listener("add")}
    }

    fun addRemoveButtonListener(listener: (command: String) -> Unit) {
        removeButton.addActionListener {
            if (displayCollection.selectedIndex == 0) {
                val selectedGroup = table.selectedRow
                if (selectedGroup != -1) {
                    listener("remove_by_id ${table.getValueAt(selectedGroup,
0)}")
                } else {
                    showError(r.getString("itemIsNotSelected"))
                }
            }
            else {
                val selectedGroup = visualization.selectedCircle
                if (selectedGroup != null) {
                    listener("remove_by_id ${selectedGroup.text.toLong()}")
                } else {
                    showError(r.getString("itemIsNotSelected"))
                }
            }
```

```
        }
    }

    fun addUpdateButtonListener(listener: (command: String) -> Unit) {
        updateButton.addActionListener {
            if (displayCollection.selectedIndex == 0) {
                val selectedGroup = table.selectedRow
                if (selectedGroup != -1) {
                    listener("update ${table.getValueAt(selectedGroup, 0)}")
                } else {
                    showError(r.getString("itemIsNotSelected"))
                }
            }
            else {
                val selectedGroup = visualization.selectedCircle
                if (selectedGroup != null) {
                    listener("update ${selectedGroup.text.toLong()}")
                } else {
                    showError(r.getString("itemIsNotSelected"))
                }
            }
        }
    }

    fun addClearButtonListener(listener: (command: String) -> Unit) {
        clearButton.addActionListener {listener("clear")}
    }

    fun addRemoveGreaterButtonListener(listener: (command: String) -> Unit) {
        removeGreaterButton.addActionListener {listener("remove_greater")}
    }

    fun addCountLessButtonListener(listener: (command: String) -> Unit) {
        countLessButton.addActionListener {
            if (countLessNField.text.isNotEmpty())
                listener("count_less_than_students_count
${countLessNField.text}")
            else showError(r.getString("enterNumberOfStudents"))
        }
    }

    fun addExecuteScriptButtonListener(listener: (command: String) -> Unit) {
        executeScriptButton.addActionListener {
            val fileChooser = JFileChooser()
            val currentDir = File(".").canonicalFile
            fileChooser.currentDirectory = currentDir
            val result = fileChooser.showOpenDialog(frame)
            if (result == JFileChooser.APPROVE_OPTION) {
                val selectedFile = fileChooser.selectedFile.canonicalFile
                val relativePath =
currentDir.toPath().relativize(selectedFile.toPath()).toString()
                listener("execute_script $relativePath")
            }
        }
    }

    fun addHistoryButtonListener(listener: (command: String) -> Unit) {
        historyButton.addActionListener {listener("history")}
    }

    fun addInfoButtonListener(listener: (command: String) -> Unit) {
        infoButton.addActionListener {listener("info")}
    }
```

```
    private fun showError(message: String) {
        val icon =
ImageIcon(URL("https://i.pinimg.com/originals/37/bb/47/37bb47c7dcc6d5b13a2646
9270dd4472.gif"))
        val scaledImage = icon.image.getScaledInstance(100, 100,
Image.SCALE_DEFAULT)
        JOptionPane.showMessageDialog(JFrame(), message, "Error",
JOptionPane.ERROR_MESSAGE, ImageIcon(scaledImage))
    }

    private var nowCollection = PriorityQueue<StudyGroup>()
    private var nowMyGroups = ArrayList<Long>()

    fun fillCollection(collection: PriorityQueue<StudyGroup>, myGroups:
ArrayList<Long>) {
        val tempCollection = PriorityQueue(collection)
        while (!tempCollection.isEmpty()) {
            val studyGroup = tempCollection.poll()
            tableModel.addRow(
                arrayOf(
                    studyGroup.id,
                    studyGroup.name,
                    studyGroup.coordinates.x,
                    studyGroup.coordinates.y,
                    studyGroup.creationDate,
                    studyGroup.studentsCount,
                    studyGroup.formOfEducation.form,
                    studyGroup.semesterEnum?.name,
                    studyGroup.groupAdmin.name,
                    studyGroup.groupAdmin.birthday.toLocalDate(),
                    studyGroup.groupAdmin.eyeColor?.color,
                    studyGroup.groupAdmin.nationality.country,
                    studyGroup.groupAdmin.location.name,
                    studyGroup.groupAdmin.location.x,
                    studyGroup.groupAdmin.location.y,
                    studyGroup.groupAdmin.location.z
                )
            )

            val info = """
                <html>
                ${r.getString("groupName")}: ${studyGroup.name}<br>
                X: ${studyGroup.coordinates.x}<br>
                Y: ${studyGroup.coordinates.y}<br>
                ${r.getString("creationDate")}:
${studyGroup.creationDate}<br>
                ${r.getString("studentsCount")}:
${studyGroup.studentsCount}<br>
                ${r.getString("formOfEducation")}:
${studyGroup.formOfEducation.form}<br>
                ${r.getString("semester")}:
${studyGroup.semesterEnum?.name}<br>
                ${r.getString("name")} ${r.getString("ofAdmin")}:
${studyGroup.groupAdmin.name}<br>
                ${r.getString("birthday")} ${r.getString("ofAdmin")}:
${studyGroup.groupAdmin.birthday.toLocalDate()}<br>
                ${r.getString("eyeColor")} ${r.getString("ofAdmin")}:
${studyGroup.groupAdmin.eyeColor?.color}<br>
                ${r.getString("originCountry")} ${r.getString("ofAdmin")}:
${studyGroup.groupAdmin.nationality.country}<br>
                ${r.getString("location")} ${r.getString("ofAdmin")}:
${studyGroup.groupAdmin.location.name}<br>
                x: ${studyGroup.groupAdmin.location.x}<br>
                y: ${studyGroup.groupAdmin.location.y}<br>
```

```
                z: ${studyGroup.groupAdmin.location.z}
                </html>
            """.trimIndent()

            if (studyGroup.id in myGroups) {
                if (nowCollection.contains(studyGroup)) {
                    visualization.addCircle(
                        Circle(
                            studyGroup.coordinates.x,
                            studyGroup.coordinates.y,
                            studyGroup.studentsCount,
                            Color(32, 227, 49, 175),
                            "${studyGroup.id}",
                            info
                        )
                    )
                }
                else {
                    visualization.addCircleWithAnimation(
                        Circle(
                            studyGroup.coordinates.x,
                            studyGroup.coordinates.y,
                            studyGroup.studentsCount,
                            Color(32, 227, 49, 175),
                            "${studyGroup.id}",
                            info
                        )
                    )
                }
            } else {
                if (nowCollection.contains(studyGroup)) {
                    visualization.addCircle(
                        Circle(
                            studyGroup.coordinates.x,
                            studyGroup.coordinates.y,
                            studyGroup.studentsCount,
                            Color(255, 66, 66, 175),
                            "${studyGroup.id}",
                            info
                        )
                    )
                }
                else {
                    visualization.addCircleWithAnimation(
                        Circle(
                            studyGroup.coordinates.x,
                            studyGroup.coordinates.y,
                            studyGroup.studentsCount,
                            Color(255, 66, 66, 175),
                            "${studyGroup.id}",
                            info
                        )
                    )
                }
            }
        }
        nowCollection = PriorityQueue(collection)
        nowMyGroups = ArrayList(myGroups)
    }

    fun clearCollection() {
        tableModel.rowCount = 0
        visualization.clear()
    }
```

```
    override fun update() {
        r = ResourceBundle.getBundle("client.localization.GuiLabels",
Locale.getDefault())

        frame.title = "meow"
        userLabel.text = "${r.getString("user")}: "
        languageLabel.text = "${r.getString("language")}: "

        exitButton.text = r.getString("exit")
        helpButton.text = r.getString("help")
        addButton.text = r.getString("add")
        removeButton.text = r.getString("remove")
        updateButton.text = r.getString("update")
        clearButton.text = r.getString("clear")
        removeGreaterButton.text = r.getString("removeGreater")
        countLessButton.text = r.getString("count")
        executeScriptButton.text = r.getString("executeScript")
        historyButton.text = r.getString("history")
        infoButton.text = r.getString("info")

        countLessLabel.text = r.getString("countLess")
        countLessStudentsLabel.text = r.getString("lessStudents")

        val columnNames = arrayOf(
            "id",
            r.getString("groupName"),
            "X",
            "Y",
            r.getString("creationDate"),
            r.getString("studentsCount"),
            r.getString("formOfEducation"),
            r.getString("semester"),
            "${r.getString("name")} ${r.getString("ofAdmin")}",
            "${r.getString("birthday")} ${r.getString("ofAdmin")}",
            "${r.getString("eyeColor")} ${r.getString("ofAdmin")}",
            "${r.getString("originCountry")} ${r.getString("ofAdmin")}",
            "${r.getString("location")} ${r.getString("ofAdmin")}",
            "x",
            "y",
            "z"
        )

        for (i in columnNames.indices) {
            table.columnModel.getColumn(i).headerValue = columnNames[i]
        }

        displayCollection.setTitleAt(0, r.getString("table"))
        displayCollection.setTitleAt(1, r.getString("map"))

        clearCollection()
        fillCollection(nowCollection, nowMyGroups)

        SwingUtilities.updateComponentTreeUI(frame)
    }
}
```

client.Controller.kt

```
package client.controllers

import client.managers.GUIManager
import client.managers.ResponseManager
import client.models.CollectionModel
import client.network.Client
import client.views.AppForm
import common.communication.RequestCommand
import common.communication.Response
import common.communication.ResponseStatus
import common.models.StudyGroup
import javax.swing.SwingUtilities
import javax.swing.SwingWorker

class Controller(private val view: AppForm, private val model:
CollectionModel, private val guiManager: GUIManager) {
    private val responseManager = ResponseManager(this)

    init {
        SwingUtilities.invokeLater {
            view.fillCollection(model.collection, model.myGroups)
            view.addExitButtonListener { guiManager.exit(view) }
            view.addHelpButtonListener { command ->
ServerTask(command).execute() }
            view.addAddButtonListener { command ->
ServerTask(command).execute() }
            view.addRemoveButtonListener { command ->
ServerTask(command).execute() }
            view.addClearButtonListener { command ->
ServerTask(command).execute() }
            view.addUpdateButtonListener { command ->
ServerTask(command).execute() }
            view.addRemoveGreaterButtonListener { command ->
ServerTask(command).execute() }
            view.addInfoButtonListener { command ->
ServerTask(command).execute() }
            view.addHistoryButtonListener { command ->
ServerTask(command).execute() }
            view.addCountLessButtonListener { command ->
ServerTask(command).execute() }
            view.addExecuteScriptButtonListener { command ->
ScriptTask(command).execute() }
        }
    }

    inner class ServerTask(command: String): SwingWorker<Response, Void>() {
        private val tokens = command.split(' ')
        private val arg = if (tokens.size > 1) tokens[1] else null

        override fun doInBackground(): Response {
            return sendCommand()
        }

        private fun sendCommand(): Response {
            Client.sendRequest(RequestCommand(tokens[0], arg))
            return Client.readResponse()
        }

        override fun done() {
            try {
                val response = get()
                responseManager.handle(response, arg)
            } catch (e: Exception) {
                e.printStackTrace()
```

```
            }
        }
    }

    inner class ScriptTask(command: String): SwingWorker<ArrayList<Response>,
Void>() {
        private val tokens = command.split(' ')
        private val arg = if (tokens.size > 1) tokens[1] else null

        override fun doInBackground(): ArrayList<Response> {
            var response: Response? = null
            val responses = ArrayList<Response>()
            Client.sendRequest(RequestCommand(tokens[0], arg))
            while (response == null || (response.status !=
ResponseStatus.SUCCESS && response.status != ResponseStatus.ERROR)) {
                response = Client.readResponse()
                responses.add(response)
            }
            return responses
        }

        override fun done() {
            try {
                val responses = get()
                for (response in responses) {
                    responseManager.handle(response, arg)
                }
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
    }

    fun update() {
        model.fill()
        SwingUtilities.invokeLater {
            view.clearCollection()
            view.fillCollection(model.collection, model.myGroups)
        }
    }

    fun getFromCollection(id: Long): StudyGroup? {
        return model.get(id)
    }
}
```

client.CollectionModel.kt

```
package client.models

import client.network.Client
import common.communication.RequestCommand
import common.communication.ResponseStatus
import common.models.StudyGroup
import java.util.PriorityQueue

class CollectionModel {
    val collection: PriorityQueue<StudyGroup> = PriorityQueue()
    val myGroups = ArrayList<Long>()

    init {
```

```
        fill()
    }

    fun fill() {
        collection.clear()
        Client.sendRequest(RequestCommand("show", null))
        val response = Client.readResponse()
        if (response.status == ResponseStatus.SUCCESS) {
            val collectionSize = response.descr.toInt()
            for (i in 0 until collectionSize) {
                collection.add(Client.readObject())
            }
        }
        val myGroupsResponse = Client.readResponse().descr.trim()
        if (myGroupsResponse.isNotBlank()) {
            for (group in myGroupsResponse.split(' ')) {
                myGroups.add(group.toLong())
            }
        }
    }

    fun get(id: Long): StudyGroup? {
        for (studyGroup in collection) {
            if (studyGroup.id == id) {
                return studyGroup
            }
        }
        return null
    }
}
```

client.LocalizationManager.kt

```
package client.localization

import client.views.AppForm
import java.util.*

class LocalizationManager {
    lateinit var appForm: AppForm
    private val subscribers = mutableListOf<Subscriber>()

    fun init() {
        appForm.addLanguageBoxListener { index ->
            val locale = when (index) {
                0 -> Locale("ru", "RU")
                1 -> Locale("ro", "RO")
                2 -> Locale("bg", "BG")
                3 -> Locale("en", "NZ")
                else -> Locale.getDefault()
            }
            Locale.setDefault(locale)
            notifySubscribers()
        }
    }

    fun subscribe(subscriber: Subscriber) {
        subscribers.add(subscriber)
    }

    fun unsubscribe(subscriber: Subscriber) {
```

```
        subscribers.remove(subscriber)
    }

    private fun notifySubscribers() {
        subscribers.forEach { subscriber -> subscriber.update()}
    }
}
```

Заключение

В результате выполнения лабораторной работы я научилась создавать графические интерфейсы с использованием библиотеки Swing в Java, обрабатывать события в этих интерфейсах и осуществлять их локализацию.