«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет

По лабораторной работе №8

Вариант 1674148

Студент:

*Ильин Н. С.*

*Р3110 поток 2.9*

Преподаватель:

*Бойко В.А.*

Санкт-Петербург, 2024 г.

Оглавление

[**Задание:** 3](#_Toc160583382)

[**Выполнение работы:** 4](#_Toc160583383)

[**Выводы**: 8](#_Toc160583384)

# **Задание:**

1. Интерфейс должен быть реализован с помощью библиотеки JavaFX
2. Графический интерфейс клиентской части должен поддерживать **русский**, **нидерландский**, **французкий** и **испанский (Никарагуа)** языки / локали. Должно обеспечиваться корректное отображение чисел, даты и времени в соответстии с локалью. Переключение языков должно происходить без перезапуска приложения. Локализованные ресурсы должны храниться в **файле свойств**.

Доработать программу из [лабораторной работы №7](https://se.ifmo.ru/courses/programming#lab7) 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
   1. Каждое поле объекта - отдельная колонка таблицы.
   2. 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
   1. Объекты должны быть нарисованы с помощью графических примитивов с использованием [Graphics](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html), [Canvas](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/Canvas.html) или аналогичных средств графической библиотеки.
   2. При визуализации использовать данные о координатах и размерах объекта.
   3. Объекты от разных пользователей должны быть нарисованы разными цветами.
   4. При нажатии на объект должна выводиться информация об этом объекте.
   5. При добавлении/удалении/изменении объекта, он должен **автоматически** появиться/исчезнуть/измениться  на области как владельца, так и всех других клиентов.
   6. 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.)

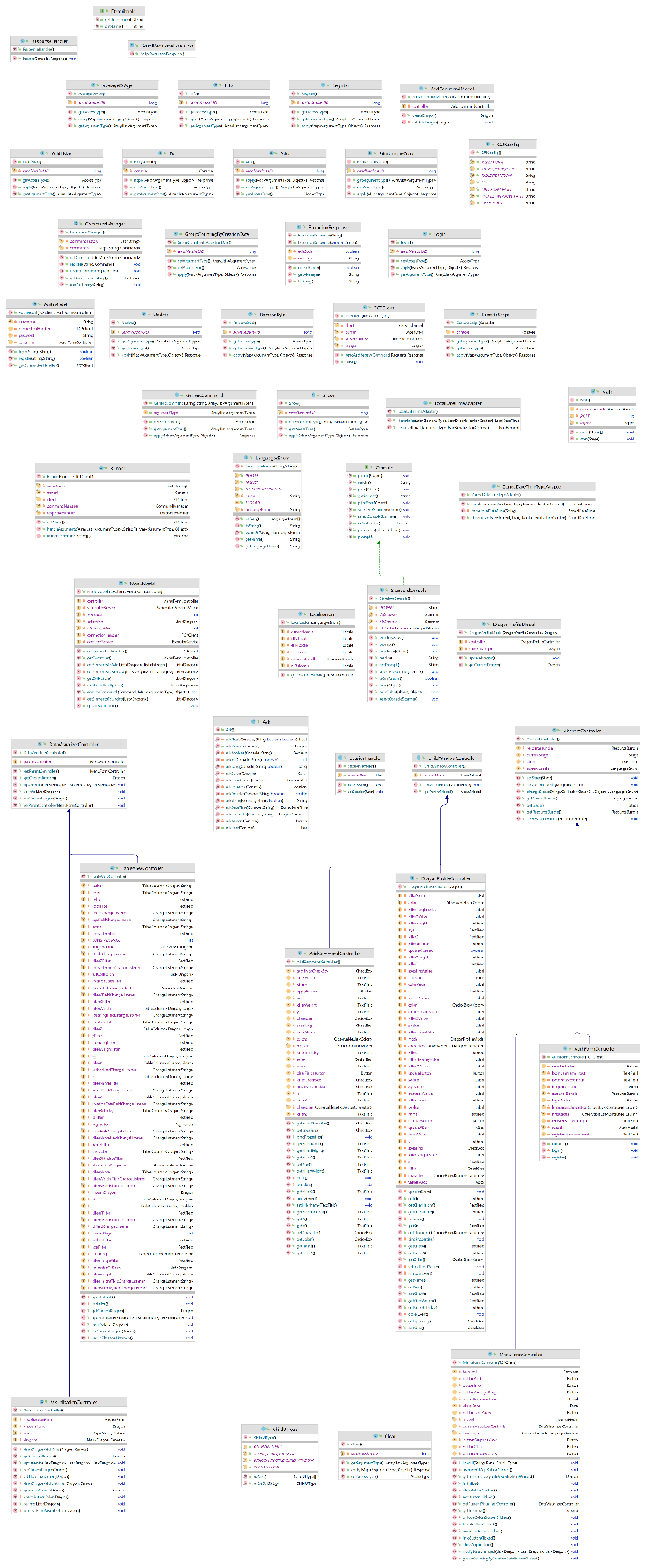
# **Выполнение работы:**

**UML-диаграмма:**

Сервер**:**



Клиент:



**Github:**  
<https://github.com/MrTheFall/study/tree/master/2sem/lab7>

**Наиболее важные части кода:**

Client Main.java

package org.lab6;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Alert;  
import javafx.scene.control.ButtonType;  
import javafx.stage.Stage;  
import org.lab6.network.TCPClient;  
import org.lab6.ui.controllers.AbstractController;  
import org.lab6.ui.controllers.AuthFormController;  
import org.lab6.ui.localization.LanguagesEnum;  
import org.lab6.ui.localization.Localization;  
import org.lab6.utils.Runner;  
import org.lab6.utils.console.StandardConsole;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
import java.io.IOException;  
import java.net.ConnectException;  
import java.net.InetAddress;  
import java.util.Locale;  
import java.util.ResourceBundle;  
  
import static org.lab6.ui.GUIConfig.\*;  
  
public class Main extends Application {  
  
 private static final int *PORT* = 25565;  
 public static final Logger *logger* = LogManager.*getLogger*("ClientLogger");  
 private ResourceBundle resourceBundle;  
  
 public static void main(String[] args) {  
 *launch*(args); *// This will start the JavaFX application* }  
  
 @Override  
 public void start(Stage primaryStage) {  
 try {  
 var console = new StandardConsole();  
 var client = new TCPClient(InetAddress.*getLocalHost*(), *PORT*);  
 var runner = new Runner(console, client);  
  
 *// Set the default locale to Russian  
// Locale.setDefault(new Locale("ru", "RU"));  
// this.resourceBundle = ResourceBundle.getBundle("lang", Locale.getDefault());  
// // Load the login form  
// FXMLLoader loader = new FXMLLoader(getClass().getResource("/forms/AuthForm.fxml"), this.resourceBundle);  
// Parent root = loader.load();  
//  
// Scene scene = new Scene(root);  
// primaryStage.setScene(scene);  
// primaryStage.show();* FXMLLoader mainWindowLoader = new FXMLLoader();  
 mainWindowLoader.setLocation(getClass().getResource(*AUTH\_FORM*));  
 Localization localization = new Localization(LanguagesEnum.*RUSSIAN*);  
 mainWindowLoader.setResources(localization.getResourceBundle());  
  
 mainWindowLoader.setControllerFactory(c -> {  
 AbstractController controller = new AuthFormController(client);  
 controller.setCurrentLocale(LanguagesEnum.*RUSSIAN*);  
 controller.setResourceBundle(localization.getResourceBundle());  
 return controller;  
 });  
  
 Parent root = mainWindowLoader.load();  
 AbstractController controller = mainWindowLoader.getController();  
 controller.setStage(primaryStage);  
 primaryStage.setTitle(*TITLE*);  
 primaryStage.setResizable(false);  
 primaryStage.setScene(new Scene(root));  
 primaryStage.show();  
   
  
  
 } catch (ConnectException e) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*);  
 alert.setTitle("No connection to server");  
 alert.setHeaderText("No connection to server, app will be closed.");  
 alert.showAndWait().ifPresent(rs -> {  
 if (rs == ButtonType.*OK*) {  
 System.*out*.println("Pressed OK.");  
 }  
 });  
  
 *//logger.error(resourceBundle.getString("server.unavailable"));* } catch (IOException e) {  
 throw new RuntimeException(e);  
 } catch (ClassNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

Server TCPServer.java

package org.lab6;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.lab6.commands.\*;  
import org.lab6.handlers.CommandHandler;  
import org.lab6.managers.\*;  
import org.lab6.utils.console.StandardConsole;  
  
import java.io.IOException;  
import java.net.SocketException;  
import java.net.UnknownHostException;  
import java.sql.SQLException;  
  
public class Main {  
 public static final int PORT = 25565;  
  
 public static Logger logger = LogManager.getLogger("ServerLogger");  
  
 public static void main(String[] args) throws SQLException {  
 var console = new StandardConsole();  
  
*// String filename = System.getenv("FILENAME");  
// if (filename == null || filename.isEmpty()) {  
// System.out.println("Задайте имя загружаемого файла с помощью переменной среды FILENAME");  
// System.exit(1);  
// }  
// System.out.println(filename);  
 //var dumpManager = new DumpManager(filename, console);* var dbUrl = System.getenv("PG\_URL");  
 var dbUsername = System.getenv("PG\_USERNAME");  
 var dbPassword = System.getenv("PG\_PASSWORD");  
 var databaseManager = new DatabaseManager(dbUrl, dbUsername, dbPassword);  
 var collectionManager = new CollectionManager(databaseManager);  
 if (!collectionManager.init()) { System.*exit*(1); }  
  
 collectionManager.validateAll(console);  
 *//Runtime.getRuntime().addShutdownHook(new Thread(collectionManager::saveCollection));* var commandManager = new CommandManager() {{  
 register("help", new Help(console, this));  
 register("info", new Info(console, collectionManager));  
 register("add", new Add(console, collectionManager, databaseManager));  
 register("update", new Update(console, collectionManager, databaseManager));  
 register("remove\_by\_id", new RemoveById(console, collectionManager, databaseManager));  
 register("clear", new Clear(console, collectionManager, databaseManager));  
 register("show", new Show(console, collectionManager));  
 register("add\_if\_max", new AddIfMax(console, collectionManager, databaseManager));  
 register("sort", new Sort(console, collectionManager));  
 register("average\_of\_age", new AverageOfAge(console, collectionManager));  
 register("group\_counting\_by\_creation\_date", new GroupCountingByCreationDate(console, collectionManager));  
 register("print\_unique\_color", new PrintUniqueColor(console, collectionManager));  
 register("register", new Register(console, databaseManager));  
 register("login", new Login(console, databaseManager));  
  
 register("exit", new Exit(console)); *// Server-only command* register("save", new Save(console, collectionManager)); *// Server-only command* }};  
  
 try {  
 var server = new TCPServer(*PORT*, new CommandHandler(commandManager, databaseManager));  
 *//server.setAfterHook(collectionManager::saveCollection);* server.run();  
 } catch (SocketException e) {  
 *logger*.fatal("Случилась ошибка сокета", e);  
 } catch (UnknownHostException e) {  
 *logger*.fatal("Неизвестный хост", e);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

MenuModel.java

package org.lab6.ui.models;  
  
import common.models.Dragon;  
import common.models.User;  
import common.network.Request;  
import common.network.Response;  
import common.utils.ArgumentType;  
import common.utils.Command;  
import javafx.concurrent.ScheduledService;  
import javafx.concurrent.Task;  
import javafx.util.Duration;  
import jdk.jshell.spi.ExecutionControl;  
import org.lab6.auth.SessionHandler;  
import org.lab6.commands.Register;  
import org.lab6.commands.Show;  
import org.lab6.network.TCPClient;  
import org.lab6.ui.controllers.MenuFormController;  
  
import javax.management.DescriptorAccess;  
import java.io.Console;  
import java.io.IOException;  
import java.net.SocketTimeoutException;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Map;  
import java.util.concurrent.CopyOnWriteArrayList;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.stream.Collectors;  
  
public class MenuModel {  
  
 private static final int *THREADS* = Runtime.*getRuntime*().availableProcessors();  
 private final static int *UPDATE\_TIME* = 5000;  
 private final TCPClient connectionHandler;  
 private final List<Dragon> collection = new CopyOnWriteArrayList<>();  
 private final MenuFormController controller;  
 private final ExecutorService executorService = Executors.*newFixedThreadPool*(*THREADS*);  
  
 public MenuModel(TCPClient connectionHandler, MenuFormController controller) {  
 this.connectionHandler = connectionHandler;  
 this.controller = controller;  
 scheduledService.setPeriod(Duration.*millis*(*UPDATE\_TIME*));  
 scheduledService.start();  
 }  
  
 public void executeCommand(Command command, Map<ArgumentType, Object> args) {  
 Task<String> task = new Task<String>() {  
 @Override  
 protected String call() throws Exception {  
 String message = connectionHandler.sendAndReceiveCommand(new Request(command, args)).getMessage();  
 return message;  
 }  
 };  
 task.setOnFailed(event -> {  
 ExecutionControl.UserException exception = (ExecutionControl.UserException) task.getException();  
 });  
 task.setOnFailed(event -> {  
 if (task.getException().getClass().equals(SocketTimeoutException.class)) {  
 controller.getTerminal().appendText("От сервера не был получен ответ..." + "\n");  
 }  
 });  
 task.setOnSucceeded(event -> {  
 if (task.getValue() != null) controller.getTerminal().appendText(task.getValue() + "\n");  
 });  
 executorService.execute(task);  
 }  
  
  
  
 private final ScheduledService<Void> scheduledService = new ScheduledService<Void>() {  
  
 @Override  
 protected Task<Void> createTask() {  
 return new Task<Void>() {  
 @Override  
 protected Void call() {  
 updateCollection();  
 return null;  
 }  
 };  
 }  
 };  
 public List<Dragon> getCollection() {  
 return collection;  
 }  
  
 public void updateCollection() {  
 executorService.execute(createTaskForUpdate());  
 }  
  
 private Task<Response> createTaskForUpdate() {  
 Task<Response> taskForUpdate = new Task<Response>() {  
 @Override  
 protected Response call() throws ExecutionControl.UserException {  
 Request request = new Request(new Show(), Map.*of*(ArgumentType.*AUTH\_SESSION*, SessionHandler.*getSession*()));  
 try {  
 return connectionHandler.sendAndReceiveCommand(request);  
 } catch (IOException | ClassNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 };  
 taskForUpdate.setOnSucceeded(event -> {  
 List<Dragon> recievedPeople = ((Response) taskForUpdate.getValue()).getDragons();  
 List<Integer> currentIds = collection.stream().map(Dragon::getId).collect(Collectors.*toList*());  
 if (recievedPeople != null) {  
 controller.notifyDataChanged(getElementsToRemove(recievedPeople, currentIds),  
 getElementsToAdd(recievedPeople, currentIds),  
 getElementsToUpdate(recievedPeople));  
 collection.clear();  
 collection.addAll(recievedPeople);  
 }  
 });  
 taskForUpdate.setOnFailed(event -> {  
 try {  
 System.*out*.println(taskForUpdate.getException());  
 ExecutionControl.UserException e = ((ExecutionControl.UserException) taskForUpdate.getException());  
 controller.closeApplication();  
 controller.getTerminal().appendText(e.getMessage() + "\n");  
 } catch (RuntimeException e) {  
 controller.getTerminal().appendText(taskForUpdate.getException() + "\n");  
 }  
 });  
 return taskForUpdate;  
  
 }  
  
 private List<Dragon> getElementsToAdd(List<Dragon> recievedPeople, List<Integer> currentIds) {  
 List<Integer> recievedIds = recievedPeople.stream().map(Dragon::getId).toList();  
 List<Dragon> newElements = new ArrayList<>();  
 for (Integer id : recievedIds) {  
 if (!currentIds.contains(id)) {  
 recievedPeople.stream().filter(Dragon -> Dragon.getId() == id).findFirst().ifPresent(newElements::add);  
 }  
 }  
 return newElements;  
 }  
  
 public MenuFormController getController() {  
 return controller;  
 }  
  
 private List<Dragon> getElementsToRemove(List<Dragon> recievedPeople, List<Integer> currentIds) {  
 List<Integer> recievedIds = recievedPeople.stream().map(Dragon::getId).toList();  
 List<Dragon> elementsToRemove = new ArrayList<>();  
 for (Integer id : currentIds) {  
 if (!recievedIds.contains(id)) {  
 collection.stream().filter(Dragon -> Dragon.getId() == id).findFirst().ifPresent(elementsToRemove::add);  
 }  
 }  
 return elementsToRemove;  
 }  
  
 private List<Dragon> getElementsToUpdate(List<Dragon> recievedPeople) {  
 List<Dragon> elementsToUpdate = new ArrayList<>();  
 for (Dragon recievedDragon : recievedPeople) {  
 for (Dragon human : collection) {  
 if (human.getId() == recievedDragon.getId() && human.hashCode() != recievedDragon.hashCode()) {  
 elementsToUpdate.add(recievedDragon);  
 }  
 }  
 }  
 return elementsToUpdate;  
 }  
  
 public TCPClient getConnectionHandler() {  
 return connectionHandler;  
 }  
}

MenuFormController.java

package org.lab6.ui.controllers;  
  
import common.models.Dragon;  
import common.utils.ArgumentType;  
import common.utils.Command;  
import javafx.application.Platform;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.ChoiceBox;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextArea;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
import jdk.jshell.spi.ExecutionControl;  
import org.lab6.auth.SessionHandler;  
import org.lab6.commands.\*;  
import org.lab6.network.TCPClient;  
import org.lab6.ui.ChildUIType;  
import org.lab6.ui.GUIConfig;  
import org.lab6.ui.localization.LanguagesEnum;  
import org.lab6.ui.localization.Localization;  
import org.lab6.ui.models.AuthModel;  
import org.lab6.ui.models.MenuModel;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Map;  
  
public class MenuFormController extends AbstractController {  
 private final MenuModel model;  
 @FXML  
 private Pane visualPane;  
  
 @FXML  
 private Button buttonAdd;  
 @FXML  
 private Button buttonClear;  
 @FXML  
 private Button buttonAverageOfAge;  
 @FXML  
 private Button buttonInfo;  
 @FXML  
 private Button buttonUniqueColors;  
  
 @FXML  
 private Label menuUsernameLabel;  
  
 @FXML  
 private Button buttonTableView;  
 @FXML  
 private Button buttonGraphicsView;  
  
  
 @FXML  
 private TextArea terminal;  
 private DataVisualizerController currentVisualizerController;  
  
  
  
 private final ObservableList<LanguagesEnum> languages = FXCollections.*observableArrayList*(LanguagesEnum.*values*());  
  
 public MenuFormController(TCPClient client){  
 this.model = new MenuModel(client, this);  
 }  
  
 public void initialize() {  
 menuUsernameLabel.setText(SessionHandler.*getSession*().getUsername());  
 buttonTableView.setDisable(false);  
 buttonGraphicsView.setDisable(false);  
 }  
  
 public void notifyDataChanged(List<Dragon> elementsToRemove, List<Dragon> elementsToAdd, List<Dragon> elementsToUpdate) {  
 if (currentVisualizerController != null) {  
 currentVisualizerController.updateInfo(elementsToRemove, elementsToAdd, elementsToUpdate);  
 }  
 }  
 public TextArea getTerminal() {  
 return terminal;  
 }  
  
 protected void loadUI(String uiPath, Pane targetPane, ChildUIType uiType) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource(uiPath));  
 Localization localization = new Localization(getCurrentLocale());  
 loader.setResources(localization.getResourceBundle());  
 if (uiType.equals(ChildUIType.*VISUALISATION*)) {  
 targetPane.getChildren().clear();  
 Parent parent = loader.load();  
 currentVisualizerController = loader.getController();  
 model.updateCollection();  
 currentVisualizerController.setInfo(model.getCollection());  
 currentVisualizerController.setParentController(this);  
 targetPane.getChildren().add(parent);  
 } else if (uiType.equals(ChildUIType.*SIMPLE\_CHILD\_WINDOW*)) {  
 Parent parent = loader.load();  
 ChildWindowController controller = loader.getController();  
 controller.setMenuModel(model);  
 Stage stage = new Stage();  
 stage.setTitle(GUIConfig.*TITLE*);  
 stage.setResizable(false);  
 stage.setScene(new Scene(parent));  
 stage.show();  
 } else if (uiType.equals(ChildUIType.*DRAGON\_PROFILE\_CHILD\_WINDOW*)) {  
 loader.setControllerFactory(controllerClass -> new DragonProfileController(getSelectedDragonInVisualisationWindow()));  
 Parent parent = loader.load();  
 ChildWindowController controller = loader.getController();  
 controller.setMenuModel(model);  
 Stage stage = new Stage();  
 stage.setTitle(GUIConfig.*TITLE*);  
 stage.setResizable(false);  
 stage.setScene(new Scene(parent));  
 stage.show();  
 } else if (uiType.equals(ChildUIType.*ON\_MAIN\_MENU*)) {  
 Parent parent = loader.load();  
 ChildWindowController controller = loader.getController();  
 controller.setMenuModel(model);  
 targetPane.getChildren().add(parent);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 private void addButtonClicked() {  
 loadUI(GUIConfig.*ADD\_FORM\_PATH*, null, ChildUIType.*SIMPLE\_CHILD\_WINDOW*);  
 }  
  
 @FXML  
 private void clearButtonClicked() {  
 model.executeCommand(new Clear(), Map.*of*(ArgumentType.*AUTH\_SESSION*, SessionHandler.*getSession*()));  
 model.updateCollection();  
 }  
 @FXML  
 private void infoButtonClicked() {  
 model.executeCommand(new Info(), Map.*of*(ArgumentType.*AUTH\_SESSION*, SessionHandler.*getSession*()));  
 }  
 @FXML  
 private void uniqueColorsButtonClicked() {  
 model.executeCommand(new PrintUniqueColor(), Map.*of*(ArgumentType.*AUTH\_SESSION*, SessionHandler.*getSession*()));  
 }  
 @FXML  
 private void averageOfAgeButtonClicked() {  
 model.executeCommand(new AverageOfAge(), Map.*of*(ArgumentType.*AUTH\_SESSION*, SessionHandler.*getSession*()));  
 }  
  
 @FXML  
 private void groupCountingByCreationDateButtonClicked() {  
 model.executeCommand(new GroupCountingByCreationDate(), Map.*of*(ArgumentType.*AUTH\_SESSION*, SessionHandler.*getSession*()));  
 }  
  
 @FXML  
 private void tableButtonClicked() {  
 loadUI(GUIConfig.*TABLEVIEW\_FORM*, visualPane, ChildUIType.*VISUALISATION*);  
 buttonGraphicsView.setDisable(false);  
 buttonTableView.setDisable(true);  
 }  
 @FXML  
 private void visualizeButtonClicked() {  
 loadUI(GUIConfig.*VISUAL\_PANE\_PATH*, visualPane, ChildUIType.*VISUALISATION*);  
 buttonTableView.setDisable(false);  
 buttonGraphicsView.setDisable(true);  
 }  
 public DataVisualizerController getCurrentVisualizerController() {  
 return currentVisualizerController;  
 }  
 public Dragon getSelectedDragonInVisualisationWindow() {  
 return currentVisualizerController.getChosenDragon();  
 }  
 public void closeApplication() {  
 Platform.*exit*();  
 }  
}

# **Выводы**:

В ходе выполнения данной лабораторной работы были осуществлены значительные изменения в структуре программы. Основное внимание уделялось реализации графического интерфейса пользователя (GUI) с использованием библиотеки JavaFX и поддержке нескольких языков.

Замена консольного клиента на клиента с графическим интерфейсом пользователя позволила сделать работу с программой более удобной и интуитивно понятной. Реализованы все функции, предусмотренные в предыдущих лабораторных работах, включая авторизацию/регистрацию, отображение текущего пользователя и всех объектов из коллекции как в виде таблицы, так и в виде графических объектов.

С помощью JavaFX и Graphics была создана область, визуализирующая объекты коллекции. Реализована возможность редактирования отдельных полей любого из объектов, принадлежащих пользователю, а также удаление выбранного объекта.

Интерфейс поддерживает русский, нидерландский, французский и испанский (Никарагуа) языки / локали. Локализованные ресурсы хранятся в файле свойств, что обеспечивает легкую поддержку и расширение списка поддерживаемых языков.

В целом, данная работа представляет собой важный этап в изучении принципов работы с графическим интерфейсом пользователя в Java, использования технологии JavaFX и поддержки многоязычности. Полученные знания и навыки будут полезны при разработке графических интерфейсов для многопользовательских приложений и при работе с локализацией и интернационализацией в приложениях.