**Министерство образования и науки Российской Федерации**

ФГАОУ ВО «Уральский федеральный университет   
имени первого Президента России Б. Н. Ельцина»

ИРИТ-РТФ

Центр ускоренного обучения

Оценка проекта

Члены комиссии

**РАЗРАБОТКА ПРОГРАММЫ НА JAVAFX ДЛЯ РЕШЕНИЯ ЗАДАЧИ «ЗАДАЧА О КАНАДСКИХ АВИАЛИНИЯХ»**

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Пояснительная записка

09.03.04 58.29.29 006 ПЗ

Руководитель В. И. Суханов

Студент гр. РИВ-270027у (662) Э. Ю. Азымов

Екатеринбург 2018

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 3](#_Toc515888864)

[ВВЕДЕНИЕ 4](#_Toc515888865)

[1 Постановка задачи 6](#_Toc515888866)

[2 Анализ поставленной задачи 7](#_Toc515888867)

[2.1 Основные объекты задачи и их взаимодействие 7](#_Toc515888868)

[2.2 Структуры данных 7](#_Toc515888869)

[2.3 Алгоритм 8](#_Toc515888870)

[2.4 Схема работы системы 8](#_Toc515888871)

[3 Описание результатов разработки 10](#_Toc515888872)

[4. Руководство пользователя 30](#_Toc515888884)

[4.1 Минимальные системные требования 30](#_Toc515888885)

[4.2 Установка и удаление программы 30](#_Toc515888886)

[4.3 Использование программы 30](#_Toc515888887)

[ЗАКЛЮЧЕНИЕ 33](#_Toc515888888)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc515888889)

[ПРИЛОЖЕНИЕ A (справочное) Код программы 35](#_Toc515888890)

# ВВЕДЕНИЕ

Целью курсовой работы является разработка программы, позволяющей составлять маршрут для путешествия по разным городам. Для разработки программы будем использовать язык программирования Java и фреймворк JavaFX. Задача состоит в следующем: дан список городов и их координаты; найти кратчайший маршрут через все города, при этом ни один город нельзя посещать больше одного раза.

Качество написанной программы определяется соблюдением поставленных условий.

Пояснительная записка имеет следующую структуру:

1. во введении описывается краткая постановка задачи и содержание разделов;
2. основная часть состоит из нескольких разделов:
3. постановка задачи – описание требований к написанию курсовой работы;
4. анализ поставленной задачи – взгляд на программу «изнутри», рассмотрение методов решения, предложения по программной реализации (структур данных, алгоритмов);
5. описание результатов разработки – рассказывается о порядке написания программы, в том числе тестирующих модулей. В конце раздела приводится листинг окончательного варианта программы;
6. руководство пользователя – содержит пояснения по использованию программы;
7. в заключении проводится анализ проделанной работы;
8. список использованных источников – содержит перечень использованной при разработке литературы;
9. в приложении содержится руководство пользователя.

# 1 Постановка задачи

Опишем задачу более подробно. Дан список городов и их координаты. Так же есть начальный и конечный города. В процессе путешествия нужно посетить все города по одному разу.

Необходимо разработать программу с графическим интерфейсом, которая составляет маршрут всегда удовлетворяющий вышеуказанным правилам. Ввод/вывод параметров программы осуществляется через графический интерфейс или через файл. Программа должна содержать 4 раздела. Первый раздел – ввод информации о городах и их координатах. Второй раздел – карта на которой отображены города и путь между ними. Третий раздел – ввод начального и конечного города. Четвертый раздел – помощь, информация о пользованием программой.

Программа должна быть написана на языке программирования Java и использовать фреймворк JavaFX. Описание результатов разработки оформить в виде пояснительной записки, в соответствии с ГОСТ 2.105-90, 7.32-2001 и 7.1-2003. При описании программы использовать схемы работы системы (ГОСТ 19.701-90) и диаграммы классов по правилам UML.

# 2 Анализ поставленной задачи

## 2.1 Основные объекты задачи и их взаимодействие

Рассмотрим поставленную задачу. Есть сущность «Город», у неё есть следующие свойства: название, координата X, координата Y. Из городов формируется список. Вариантом этого списка является кратчайший путь который нужно найти. Так же есть начальный и конечный города.

Каким образом найти кратчайший путь из всех возможных вариантов? Можно воспользоваться методом полного перебора. С помощью вспомогательной функции из заданного списка городов формируется список всех возможных путей, после этого список фильтруется так, чтобы остались только пути начинающиеся с начального города и заканчивающиеся конечным городом. После этого из списка выбирается кратчайший путь путем подсчета стоимости каждого пути.

## 2.2 Структуры данных

Какие структуры данных потребуются при решении задачи? Основная и единственная структура данных – это модель города. Она имеет три поля: строковое поле «name» содержащее название города, целочисленное поле «posX» содержащее координату X и целочисленное поле «posY» содержащее координату Y.

## 2.3 Алгоритм

Определив структуры данных, перейдем к проектированию алгоритма задачи. Одним из вариантов решения задачи может быть следующий:

* 1. Запуск программы;
  2. Ввод списка городов через файл или через графический интерфейс.
  3. Выбор начального и конечного города.
  4. Составление списка всех возможных путей.
  5. Отсеивание путей которые не начинаются с начального города и не заканчиваются конечным городом.
  6. Вычисление стоимости каждого пути.
  7. Вывод на карту самого короткого пути.

## 2.4 Схема работы системы

Оформим описанный алгоритм в виде схемы работы системы по ГОСТ 19.701-90 (Рисунок 1).

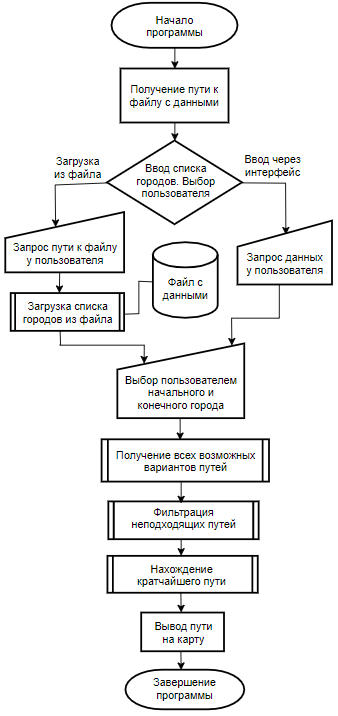


Рисунок 1 – Схема работы системы

# 3 Описание результатов разработки

## 3.1 Структура проекта

Для начала организовываем структуру проекта.

Все классы представляющие собой страницы приложения будем хранить в папке «tabs», структуры данных положим в папку «models», общие графические элементы в папку «controls», перечисления в папку «enums», вспомогательные классы в папку «lib».

Полученная структура проекта изображена ниже (Рисунок 2).

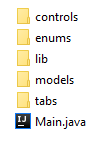


Рисунок 2 – Структура проекта

## 3.2 Класс City

Класс «City» служит для создания объектов городов. У каждого объекта есть следующие поля:

1. Поле «name». Название города. Строковый тип.
2. Поле «posX». Координата города по оси X. Целочисленный тип.
3. Поле «posY». Координата города по оси Y. Целочисленный тип.

Для каждого поля определены геттеры и сеттеры.

Изобразим диаграмму класса City, согласно правилам UML (Рисунок 3).

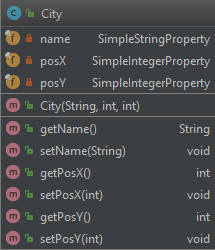


Рисунок 3 – Диаграмма UML класса City

Исходный код класса City приведен в листинге.

package models;

import javafx.beans.property.SimpleIntegerProperty;

import javafx.beans.property.SimpleStringProperty;

public class City {

private final SimpleStringProperty name;

private final SimpleIntegerProperty posX;

private final SimpleIntegerProperty posY;

public City(String name, int posX, int posY) {

this.name = new SimpleStringProperty(name);

this.posX = new SimpleIntegerProperty(posX);

this.posY = new SimpleIntegerProperty(posY);

}

public String getName() {

return this.name.get();

}

public void setName(String fName) {

this.name.set(fName);

}

public int getPosX() {

return this.posX.get();

}

public void setPosX(int posX) {

this.posX.set(posX);

}

public int getPosY() {

return this.posY.get();

}

public void setPosY(int posY) {

this.posY.set(posY);

}

}

## 3.3 Разработка класса CitiesTab

## Этот класс отвечает за создание страницы «Список городов». На этой странице есть таблица в которой выводится список городов. Над таблицей есть панель управления со следующими кнопками:

1. Кнопка «Удалить выбранный». Удаляет город который на момент удаления выбран в таблице.
2. Кнопка «Загрузить список». Открывает окно выбора файла и, если файл выбран, загружает из него список городов в таблицу.
3. Кнопка «Сохранить список». Сохраняет список городов из таблицы в файл в формате json.

Под таблицей есть форма добавления нового города. В этой форме есть три инпута для названия города, координаты X и координаты Y. Так же на форме есть кнопка добавления при нажатии на которую город добавляется в таблицу.

Изобразим диаграмму класса CitiesTab, согласно правилам UML (Рисунок 4).

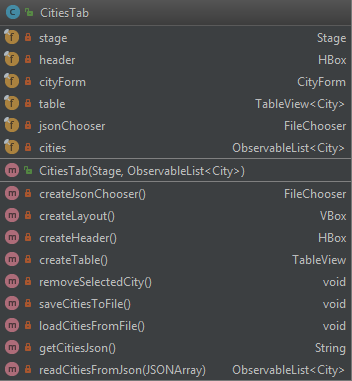


Рисунок 4 – Диаграмма UML класса CitiesTab

Исходный код класса CitiesTab приведен в листинге.

package tabs;  
  
import controls.CityForm;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.stage.Stage;  
import javafx.geometry.Insets;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Font;  
import javafx.stage.FileChooser;  
import javafx.stage.FileChooser.ExtensionFilter;  
import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;  
import org.json.simple.parser.JSONParser;  
  
import java.io.File;  
import java.io.FileReader;  
import java.io.FileWriter;  
  
import models.City;  
import enums.CityJsonFields;  
  
public class CitiesTab extends Tab {  
 private final Stage stage;  
 private final HBox header;  
 private final CityForm cityForm;  
 private final TableView<City> table;  
 private final FileChooser jsonChooser;  
 private final ObservableList<City> cities;  
  
 public CitiesTab(Stage stage, ObservableList<City> cities) {  
 this.stage = stage;  
 this.cities = cities;  
 this.header = this.createHeader();  
 this.table = this.createTable();  
 this.cityForm = new CityForm(this.cities);  
 this.jsonChooser = this.createJsonChooser();  
  
 VBox layout = this.createLayout();  
  
 layout.getChildren().addAll(this.header, this.table, this.cityForm);  
  
 this.setText("Список городов");  
 this.setClosable(false);  
 this.setContent(layout);  
 }  
  
 private FileChooser createJsonChooser() {  
 FileChooser jsonChooser = new FileChooser();  
 ExtensionFilter extFilter = new ExtensionFilter("JSON файлы","\*.json");  
  
 jsonChooser.getExtensionFilters().add(extFilter);  
 jsonChooser.setTitle("Выберите json файл");  
  
 return jsonChooser;  
 }  
  
 private VBox createLayout() {  
 VBox vBox = new VBox();  
 Insets insets = new Insets(10, 10, 10, 10);  
  
  
 vBox.setSpacing(10);  
 vBox.setPadding(insets);  
  
 return vBox;  
 }  
  
 private HBox createHeader() {  
 HBox hBox = new HBox();  
 Button delBtn = new Button("Удалить выбранный");  
 Button loadCitiesBtn = new Button("Загрузить список");  
 Button saveCitiesBtn = new Button("Сохранить список");  
  
 delBtn.setOnAction(e -> this.removeSelectedCity());  
 loadCitiesBtn.setOnAction(e -> this.loadCitiesFromFile());  
 saveCitiesBtn.setOnAction(e -> this.saveCitiesToFile());  
  
 hBox.setSpacing(10);  
 hBox.getChildren().addAll(delBtn, loadCitiesBtn, saveCitiesBtn);  
  
 return hBox;  
 }  
  
 private TableView createTable() {  
 TableView table = new TableView();  
 TableColumn cityNameColumn = new TableColumn("Название");  
 TableColumn xPosColumn = new TableColumn("Координата X");  
 TableColumn yPosColumn = new TableColumn("Координата Y");  
  
 cityNameColumn.setPrefWidth(250);  
 xPosColumn.setPrefWidth(105);  
 yPosColumn.setPrefWidth(105);  
  
 cityNameColumn.setCellValueFactory(new PropertyValueFactory<City, String>("name"));  
 xPosColumn.setCellValueFactory(new PropertyValueFactory<City, String>("posX"));  
 yPosColumn.setCellValueFactory(new PropertyValueFactory<City, String>("posY"));  
  
 table.setEditable(true);  
 table.setItems(cities);  
 table.getColumns().addAll(cityNameColumn, xPosColumn, yPosColumn);  
  
 return table;  
 }  
  
 private void removeSelectedCity() {  
 City selectedCity = table.getSelectionModel().getSelectedItem();  
 table.getItems().remove(selectedCity);  
 }  
  
 private void saveCitiesToFile() {  
 File file = this.jsonChooser.showSaveDialog(this.stage);  
  
 if (file == null) {  
 return;  
 }  
  
 try {  
 FileWriter fileWriter = new FileWriter(file);  
 String citiesJson = this.getCitiesJson();  
  
 fileWriter.write(citiesJson);  
 fileWriter.flush();  
 } catch (Exception ex) {  
 System.out.print(ex.getMessage());  
 }  
 }  
  
 private void loadCitiesFromFile() {  
 File file = this.jsonChooser.showOpenDialog(this.stage);  
  
 if (file == null) {  
 return;  
 }  
  
 try {  
 JSONParser parser = new JSONParser();  
 Object text = parser.parse(new FileReader(file));  
 JSONArray json = (JSONArray) text;  
 ObservableList<City> cities = this.readCitiesFromJson(json);  
  
 this.cities.clear();  
 this.cities.addAll(cities);  
 } catch (Exception ex) {  
 System.out.print(ex.getMessage());  
 }  
 }  
  
 private String getCitiesJson() {  
 JSONArray citiesJson = new JSONArray();  
  
 for (City city : this.cities) {  
 JSONObject cityJson = new JSONObject();  
  
 cityJson.put(CityJsonFields.NAME.toString(), city.getName());  
 cityJson.put(CityJsonFields.POS\_X.toString(), city.getPosX());  
 cityJson.put(CityJsonFields.POS\_Y.toString(), city.getPosY());  
  
 citiesJson.add(cityJson);  
 }  
  
 return citiesJson.toJSONString();  
 }  
  
 private ObservableList<City> readCitiesFromJson(JSONArray citiesJson) {  
 ObservableList<City> cities = FXCollections.observableArrayList();  
  
 for (JSONObject cityJson : (Iterable<JSONObject>) citiesJson) {  
 String cityName = (String) cityJson.get(CityJsonFields.NAME.toString());  
 Long cityPosX = (Long) cityJson.get(CityJsonFields.POS\_X.toString());  
 Long cityPosY = (Long) cityJson.get(CityJsonFields.POS\_Y.toString());  
 City city = new City(cityName, cityPosX.intValue(), cityPosY.intValue());  
  
 cities.add(city);  
 }  
  
 return cities;  
 }  
}

**3.4 Разработка класса MapTab**

Этот класс отвечает за создание страницы «Карта». Если введено меньше одного города, то отображается надпись «Необходимо указать данные о более чем одном городе для построения карты», иначе отображается карта с координатами городов. Города представлены своими названиями. Если построен маршрут, то города на карте соединяются зеленой линией соответственно построенному маршруту.

Изобразим диаграмму класса MapTab, согласно правилам UML (Рисунок 5).

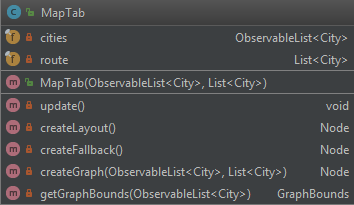


Рисунок 5 – Диаграмма UML класса MapTab

Исходный код класса MapTab приведен в листинге.

package tabs;  
  
import javafx.collections.ObservableList;  
import javafx.geometry.Insets;  
import javafx.scene.Node;  
import javafx.scene.chart.NumberAxis;  
import javafx.scene.chart.LineChart;  
import javafx.scene.chart.XYChart;  
import javafx.scene.control.Label;  
import javafx.scene.control.Tab;  
import javafx.scene.text.Font;  
import models.City;  
import models.GraphBounds;  
  
import java.util.List;  
  
public class MapTab extends Tab {  
 private final ObservableList<City> cities;  
 private final List<City> route;  
  
 public MapTab(ObservableList<City> cities, List<City> route) {  
 this.cities = cities;  
 this.route = route;  
 this.setText("Карта");  
 this.setClosable(false);  
 this.setOnSelectionChanged(e -> this.update());  
 }  
  
 private void update() {  
 this.setContent(this.createLayout());  
 }  
  
 private Node createLayout() {  
 if (this.cities.size() > 1) {  
 return this.createGraph(this.cities, this.route);  
 }  
  
 return this.createFallback();  
 }  
  
 private Node createFallback() {  
 Label label = new Label("Необходимо указать данные о более чем одном городе для построения карты");  
  
 label.setWrapText(true);  
 label.setFont(new Font("Arial", 18));  
 label.setPadding(new Insets(10));  
  
 return label;  
 }  
  
 private Node createGraph(ObservableList<City> cities, List<City> route) {  
 final GraphBounds graphBounds = this.getGraphBounds(cities);  
 final NumberAxis xAxis = new NumberAxis(graphBounds.getMinX(), graphBounds.getMaxX(), 1);  
 final NumberAxis yAxis = new NumberAxis(graphBounds.getMinY(), graphBounds.getMaxY(), 1);  
 final LineChart<Number, Number> lineChart = new LineChart(xAxis, yAxis);  
 final XYChart.Series seriesCities = new XYChart.Series();  
 final Font labelFont = new Font("Arial", 12);  
  
 seriesCities.setName("Город");  
  
 lineChart.getXAxis().setTickMarkVisible(false);  
 lineChart.getYAxis().setTickMarkVisible(false);  
 lineChart.setHorizontalZeroLineVisible(false);  
 lineChart.setVerticalZeroLineVisible(false);  
 lineChart.setLegendVisible(false);  
 lineChart.getData().addAll(seriesCities);  
 lineChart.lookup(".default-color0.chart-series-line").setStyle("-fx-stroke: transparent");  
  
 for (int idx = 1; idx < route.size(); idx++) {  
 City prevCity = route.get(idx - 1);  
 City curCity = route.get(idx);  
 XYChart.Series series = new XYChart.Series();  
 XYChart.Data firstDot = new XYChart.Data(prevCity.getPosX(), prevCity.getPosY());  
 XYChart.Data secondDot = new XYChart.Data(curCity.getPosX(), curCity.getPosY());  
 String selector = ".default-color" + idx + ".chart-series-line";  
  
 series.getData().addAll(firstDot, secondDot);  
 lineChart.getData().add(series);  
 lineChart.lookup(selector).setStyle("-fx-stroke: green");  
 }  
  
 for (City city : cities) {  
 final XYChart.Data data = new XYChart.Data(city.getPosX(), city.getPosY());  
 final Label label = new Label(city.getName());  
  
 label.setFont(labelFont);  
 data.setNode(label);  
 data.getNode().toFront();  
  
 seriesCities.getData().add(data);  
 }  
  
 return lineChart;  
 }  
  
 private GraphBounds getGraphBounds(ObservableList<City> cities) {  
 Integer minX = Integer.MAX\_VALUE;  
 Integer maxX = Integer.MIN\_VALUE;  
 Integer minY = Integer.MAX\_VALUE;  
 Integer maxY = Integer.MIN\_VALUE;  
 final Integer padding = 2;  
  
 for (City city : cities) {  
 Integer posX = city.getPosX();  
 Integer posY = city.getPosY();  
  
 if (posX > maxX) {  
 maxX = posX;  
 }  
  
 if (posX < minX) {  
 minX = posX;  
 }  
  
 if (posY > maxY) {  
 maxY = posY;  
 }  
  
 if (posY < minY) {  
 minY = posY;  
 }  
 }  
  
 return new GraphBounds(minX - padding, maxX + padding, minY - padding, maxY + padding);  
 }  
}

**3.5 Разработка класса RouteTab**

Этот класс отвечает за создание страницы «Маршрут». На этой странице есть два выпадающих меню, для начального и конечного города. Выпадающие меню содержат списки всех доступных городов. Так же есть кнопка для создания маршрута между двумя выбранными городами.

Изобразим диаграмму класса RouteTab, согласно правилам UML (Рисунок 6).

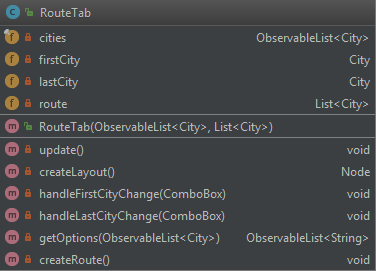


Рисунок 6 – Диаграмма UML класса RouteTab

Исходный код класса RouteTab приведен в листинге.

package tabs;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.geometry.Insets;  
import javafx.scene.Node;  
import javafx.scene.control.Alert;  
import javafx.scene.control.Alert.AlertType;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.Tab;  
import javafx.scene.layout.HBox;  
import models.City;  
import lib.ShortestRouteFinder;  
  
import java.util.List;  
  
public class RouteTab extends Tab {  
 private final ObservableList<City> cities;  
 private City firstCity;  
 private City lastCity;  
 private List<City> route;  
  
 public RouteTab(ObservableList<City> cities, List<City> route) {  
 this.cities = cities;  
 this.route = route;  
 this.setText("Маршрут");  
 this.setClosable(false);  
 this.setContent(this.createLayout());  
 this.setOnSelectionChanged(e -> this.update());  
 }  
  
 private void update() {  
 this.setContent(this.createLayout());  
 }  
  
 private Node createLayout() {  
 HBox hBox = new HBox();  
 ObservableList<String> options = this.getOptions(this.cities);  
 ComboBox firstCitySelect = new ComboBox(options);  
 ComboBox lastCitySelect = new ComboBox(options);  
 Button submitBtn = new Button("Построить маршрут");  
  
 submitBtn.setOnAction(e -> this.createRoute());  
  
 firstCitySelect.setPromptText("Начальный город");  
 lastCitySelect.setPromptText("Конечный город");  
  
 if (this.firstCity != null) {  
 firstCitySelect.setValue(this.firstCity.getName());  
 }  
  
 if (this.lastCity != null) {  
 lastCitySelect.setValue(this.lastCity.getName());  
 }  
  
 firstCitySelect.setOnAction(e -> this.handleFirstCityChange(firstCitySelect));  
 lastCitySelect.setOnAction(e -> this.handleLastCityChange(lastCitySelect));  
  
 hBox.setSpacing(10);  
 hBox.setPadding(new Insets(10));  
 hBox.getChildren().addAll(firstCitySelect, lastCitySelect, submitBtn);  
  
 return hBox;  
 }  
  
 private void handleFirstCityChange(ComboBox comboBox) {  
 Integer index = comboBox.getSelectionModel().getSelectedIndex();  
 City selectedCity = this.cities.get(index);  
  
 if (selectedCity == null) {  
 return;  
 }  
  
 this.firstCity = selectedCity;  
 }  
  
 private void handleLastCityChange(ComboBox comboBox) {  
 Integer index = comboBox.getSelectionModel().getSelectedIndex();  
 City selectedCity = this.cities.get(index);  
  
 if (selectedCity == null) {  
 return;  
 }  
  
 this.lastCity = selectedCity;  
 }  
  
 private ObservableList<String> getOptions(ObservableList<City> cities) {  
 ObservableList<String> options = FXCollections.observableArrayList();  
  
 for (City city : cities) {  
 options.add(city.getName());  
 }  
  
 return options;  
 }  
  
 private void createRoute() {  
 Alert alert = new Alert(AlertType.INFORMATION);  
  
 alert.setTitle("Ошибка");  
 alert.setHeaderText("Ошибка");  
  
 if (this.firstCity == null) {  
 alert.setContentText("Не выбран начальный город");  
 alert.show();  
  
 return;  
 }  
  
 if (this.lastCity == null) {  
 alert.setContentText("Не выбран конечный город");  
 alert.show();  
  
 return;  
 }  
  
 ShortestRouteFinder routeFinder = new ShortestRouteFinder();  
  
 routeFinder.setCities(this.cities);  
 routeFinder.setFirstCity(this.firstCity);  
 routeFinder.setLastCity(this.lastCity);  
  
 this.route.clear();  
 this.route.addAll(routeFinder.find());  
 }  
}

**3.6 Разработка класса HelpTab**

Этот класс отвечает за создание страницы «Помощь». На этой странице пользователь может увидеть справочную информацию о том как ему пользоваться программой.

Изобразим диаграмму класса HelpTab, согласно правилам UML (Рисунок 7).

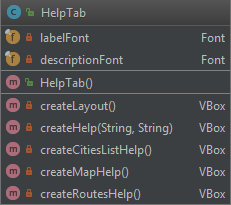


Рисунок 7 – Диаграмма UML класса HelpTab

Исходный код класса HelpTab приведен в листинге.

package tabs;  
  
import javafx.geometry.Insets;  
import javafx.scene.control.Label;  
import javafx.scene.control.Tab;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Font;  
  
public class HelpTab extends Tab {  
 private final Font labelFont = new Font("Arial", 20);  
 private final Font descriptionFont = new Font("Arial", 14);  
  
 public HelpTab() {  
 this.setText("Помощь");  
 this.setClosable(false);  
 this.setContent(this.createLayout());  
 }  
  
 private VBox createLayout() {  
 VBox vBox = new VBox();  
  
 vBox.setSpacing(20);  
 vBox.setPadding(new Insets(10, 10, 10, 10));  
 vBox.getChildren().addAll(  
 this.createCitiesListHelp(),  
 this.createMapHelp(),  
 this.createRoutesHelp()  
 );  
  
 return vBox;  
 }  
  
 private VBox createHelp(String labelText, String descriptionText) {  
 VBox vBox = new VBox();  
 Label label = new Label(labelText);  
 Label description = new Label(descriptionText);  
  
 label.setFont(labelFont);  
 label.setPadding(new Insets(0, 0, 0, 27));  
 description.setWrapText(true);  
 description.setLineSpacing(3.5);  
 description.setFont(descriptionFont);  
  
 vBox.setSpacing(10);  
 vBox.getChildren().addAll(label, description);  
  
 return vBox;  
 }  
  
 private VBox createCitiesListHelp() {  
 return this.createHelp(  
 "Список городов",  
 " На этой вкладке можно задать список городов. " +  
 "В нижней части окна находится форма добавления города. " +  
 "Если необходимо удалить город то нужно выбрать его в таблице и нажать кнопку «Удалить выбранный». " +  
 "\n Список городов можно сохранить в файл, для этого нужно нажать на кнопку «Загрузить список». " +  
 "В дальнейшем можно загрузить данные из этого файла с помощью кнопки «Сохранить список»."  
 );  
 }  
  
 private VBox createMapHelp() {  
 return this.createHelp(  
 "Карта",  
 " На этой вкладке показывается маршрут между городами, от начального до конечного."  
 );  
 }  
  
 private VBox createRoutesHelp() {  
 return this.createHelp(  
 "Маршрут",  
 " На этой вкладке строится маршрут между городами. Для этого в выпадающих " +  
 "списках нужно выбрать начальный и конечный города, после нажать кнопку «Построить маршрут». " +  
 "Результат можно увидеть на вкладке «Карта»."  
 );  
 }  
}

**3.7 Разработка класса Main**

Этот класс отвечает за инициализацию приложения и создание графического интерфейса.

Изобразим диаграмму класса Main, согласно правилам UML (Рисунок 8).

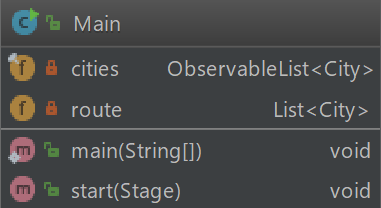


Рисунок 8 – Диаграмма UML класса Main

Исходный код класса Main приведен в листинге.

import javafx.application.Application;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.control.TabPane;  
import javafx.scene.layout.BorderPane;  
import javafx.scene.paint.Color;  
import javafx.stage.Stage;  
import models.City;  
import tabs.CitiesTab;  
import tabs.HelpTab;  
import tabs.MapTab;  
import tabs.RouteTab;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Main extends Application {  
 private final ObservableList<City> cities = FXCollections.observableArrayList();  
 private List<City> route = new ArrayList();  
  
 public static void main(String[] args) {  
 Application.launch(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) {  
 primaryStage.setTitle("Найти кратчайший маршрут");  
 Group root = new Group();  
 Scene scene = new Scene(root, 700, 450, Color.WHITE);  
 TabPane tabPane = new TabPane();  
 BorderPane borderPane = new BorderPane();  
  
 MapTab mapTab = new MapTab(this.cities, this.route);  
 HelpTab helpTab = new HelpTab();  
 RouteTab routesTab = new RouteTab(this.cities, this.route);  
 CitiesTab citiesTab = new CitiesTab(primaryStage, this.cities);  
  
 tabPane.getTabs().addAll(citiesTab, mapTab, routesTab, helpTab);  
  
 borderPane.prefHeightProperty().bind(scene.heightProperty());  
 borderPane.prefWidthProperty().bind(scene.widthProperty());  
 borderPane.setCenter(tabPane);  
  
 root.getChildren().add(borderPane);  
  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

## 3.8 Общая схема системы

Общая схема системы, она же диаграмма классов, в формате UML представлена ниже (Рисунок 9).

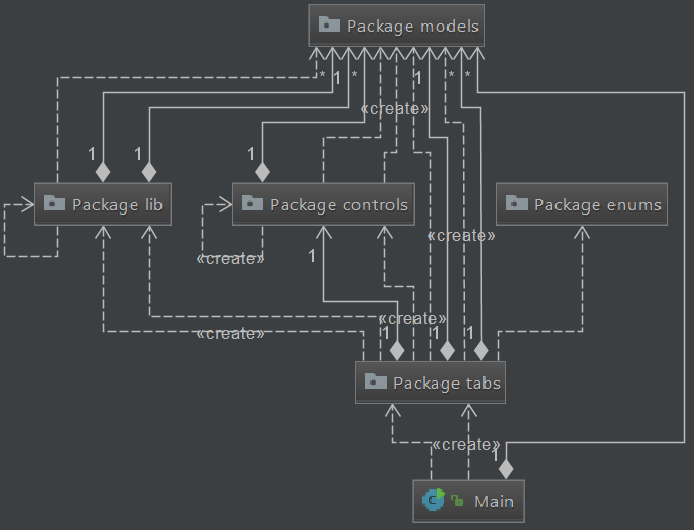


Рисунок 9 – Диаграмма классов

# 4. Руководство пользователя

## 4.1 Минимальные системные требования

Для установки и запуска программы необходимо выполнение следующих минимальных требований:

1. операционная система MS Windows (любая версия);
2. установленная виртуальная машина Java (Java 2 Platform, Standard Edition), не ниже пятой версии. Установочные файлы Java можно загрузить с сервера компании Sun, по адресу: http://java.sun.com.

## 4.2 Установка и удаление программы

Программа поставляется исполняемым файлом App.exe. Пользователь может поместить файл в любую папку и вызывать оттуда. Для установки программы не требуется никаких дополнительных действий.

Чтобы запустить программу нужно дважды кликнуть мышкой по файлу App.exe.

Для удаления программы нужно удалить файл App.exe.

## 4.3 Использование программы

При запуске программы пользователь увидит приложение с графическим интерфейсом (Рисунок 10).

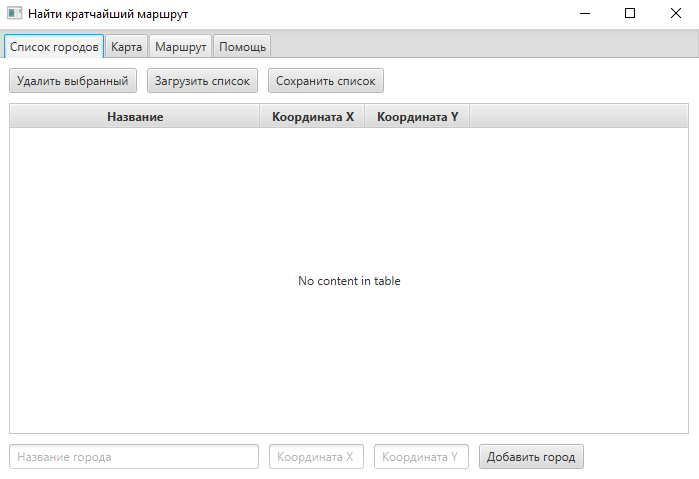


Рисунок 10 – Консольное приложение

Нажмем на кнопку «Загрузить список и в окне выбора файла выберем файл со списком городов в формате json. Пример содержимого файла можно увидеть в листинге ниже.

[  
 { "posX": 0, "posY": -10, "name": "Vancouver" },  
 { "posX": 3, "posY": 3, "name": "Yellowknife" },  
 { "posX": 2, "posY": -4, "name": "Edmonton" },  
 { "posX": -6, "posY": -1, "name": "Calgary" },  
 { "posX": -5, "posY": 5, "name": "Winnipeg" },  
 { "posX": 4, "posY": -6, "name": "Toronto" },  
 { "posX": 4, "posY": 1, "name": "Montreal" },  
 { "posX": -6, "posY": 2, "name": "Halifax" }  
]

После загрузки файла видим что таблица заполнилась данными (Рисунок 11).

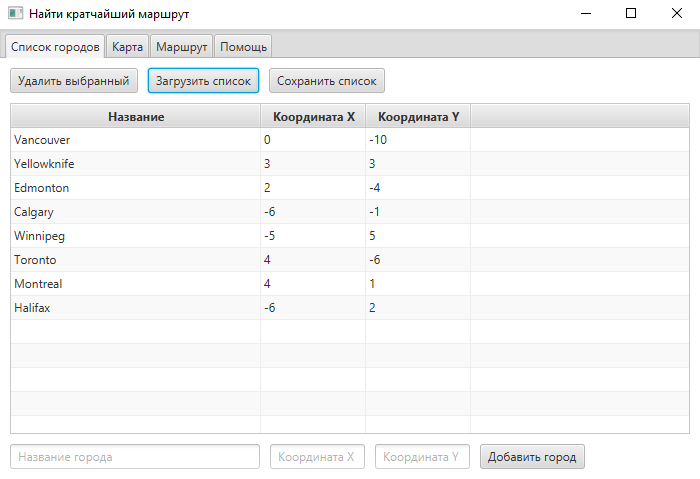


Рисунок 11 – Таблица с данными

Перейдем на страницу «Карта» и увидим карту с отображенными на ней городами (Рисунок 12).

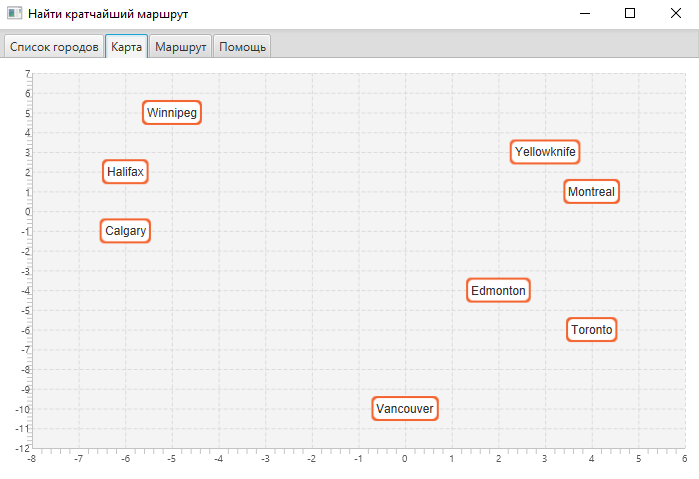


Рисунок 12 – Карта с городами

Перейдем на страницу «Маршрут» и в выпадающих меню выберем начальный и конечный города (Рисунок 13)

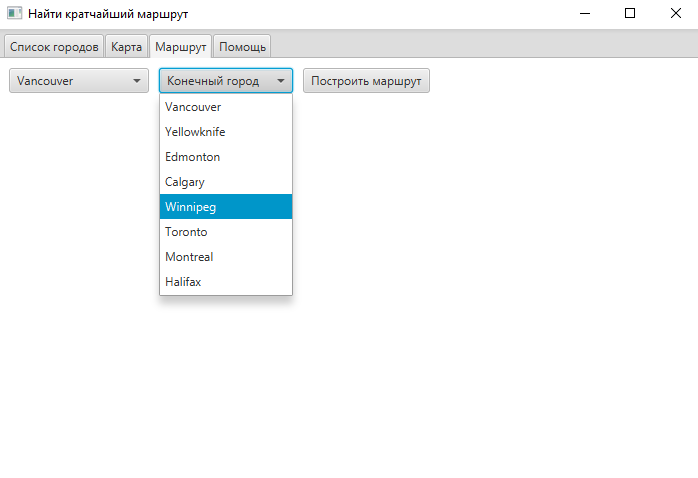


Рисунок 13 – Страница «Маршрут»

Предварительно нажав на кнопку «Построить маршрут» перейдем обратно на страницу «Карта» и увидем что маршрут отобразился на карте (Рисунок 14).

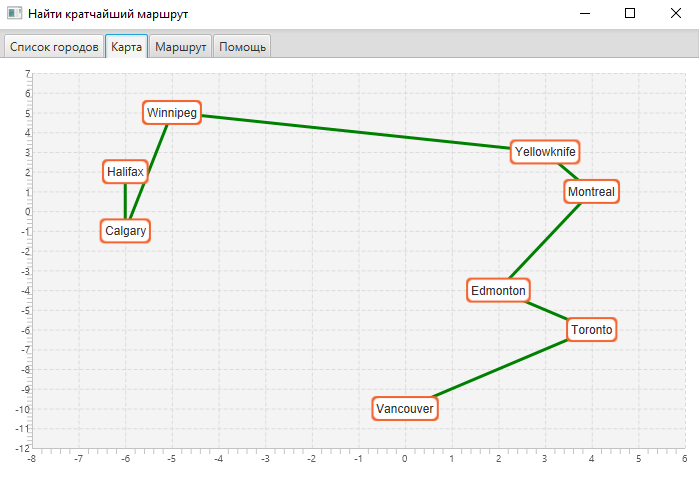


Рисунок 14 – Маршрут на карте

Для того чтобы закрыть программу нужно нажать кнопку с иконкой крестика в правом верхнем углу.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы получены навыки по работе с языком программирования Java и фреймворком JavaFX. Разработано решение задачи «задача о канадских авиалиниях». Составлена схема работы системы, для каждого класса сформированы схемы UML. Составлено руководство пользователя, включающее в себя системные требования, описание процесса по установке и удалению программы, а так же по ее использованию. Приложение A содержит полный текст программы.

Окончательная версия программы – консольное Windows приложение для Java платформы. В ходе выполнения, программа использует файловый ввод и ввод через графический интерфейс.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Окулов С.М. Программирование в алгоритмах [Текст] / С. М. Окулов. – М. : Бином. Лаборатория знаний, 2002. – 341 с.
2. Документация по языку программирования Java [Электронный ресурс] // Сайт Oracle. – [Б. м], [2018]. – Режим доступа: https://docs.oracle.com/javase/8/docs/api/
3. Документация по языку фреймворку JavaFX [Электронный ресурс] // Сайт Oracle. – [Б. м], [2018]. – Режим доступа: https://docs.oracle.com/javafx/2/
4. Ноутон П. JavaTM 2 [Текст]: пер. с англ. / П. Ноутон, Г. Шилдт. – СПб.: БХВ-Петербург, 2005. – 1072 с.
5. Кичигин В.Н. Оформление курсовых и дипломных проектов [Текст]: методические указания для студентов технических специальностей / В. Н. Кичигин, И. Е. Мясников, С. И. Тимошенко. – Екатеринбург: ГОУ ВПО «УГТУ-УПИ», 2005. – 80 с.
6. Тимошенко С. И. Порядок выполнения и требования к оформлению курсовой работы по дисциплине «Программирование на языках высокого уровня» [Текст]: методи-ческие указания / С. И. Тимошенко. – Екатеринбург: изд. ИПК УГТУ, 2004. – 16 с.

# **ПРИЛОЖЕНИЕ A** (справочное) Код программы

## A.1 Класс Main

import javafx.application.Application;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.control.TabPane;  
import javafx.scene.layout.BorderPane;  
import javafx.scene.paint.Color;  
import javafx.stage.Stage;  
import models.City;  
import tabs.CitiesTab;  
import tabs.HelpTab;  
import tabs.MapTab;  
import tabs.RouteTab;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Main extends Application {  
 private final ObservableList<City> cities = FXCollections.observableArrayList();  
 private List<City> route = new ArrayList();  
  
 public static void main(String[] args) {  
 Application.launch(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) {  
 primaryStage.setTitle("Найти кратчайший маршрут");  
 Group root = new Group();  
 Scene scene = new Scene(root, 700, 450, Color.WHITE);  
 TabPane tabPane = new TabPane();  
 BorderPane borderPane = new BorderPane();  
  
 MapTab mapTab = new MapTab(this.cities, this.route);  
 HelpTab helpTab = new HelpTab();  
 RouteTab routesTab = new RouteTab(this.cities, this.route);  
 CitiesTab citiesTab = new CitiesTab(primaryStage, this.cities);  
  
 tabPane.getTabs().addAll(citiesTab, mapTab, routesTab, helpTab);  
  
 borderPane.prefHeightProperty().bind(scene.heightProperty());  
 borderPane.prefWidthProperty().bind(scene.widthProperty());  
 borderPane.setCenter(tabPane);  
  
 root.getChildren().add(borderPane);  
  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

## A.2 Класс CitiesTab

package tabs;  
  
import controls.CityForm;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.stage.Stage;  
import javafx.geometry.Insets;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Font;  
import javafx.stage.FileChooser;  
import javafx.stage.FileChooser.ExtensionFilter;  
import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;  
import org.json.simple.parser.JSONParser;  
  
import java.io.File;  
import java.io.FileReader;  
import java.io.FileWriter;  
  
import models.City;  
import enums.CityJsonFields;  
  
public class CitiesTab extends Tab {  
 private final Stage stage;  
 private final HBox header;  
 private final CityForm cityForm;  
 private final TableView<City> table;  
 private final FileChooser jsonChooser;  
 private final ObservableList<City> cities;  
  
 public CitiesTab(Stage stage, ObservableList<City> cities) {  
 this.stage = stage;  
 this.cities = cities;  
 this.header = this.createHeader();  
 this.table = this.createTable();  
 this.cityForm = new CityForm(this.cities);  
 this.jsonChooser = this.createJsonChooser();  
  
 VBox layout = this.createLayout();  
  
 layout.getChildren().addAll(this.header, this.table, this.cityForm);  
  
 this.setText("Список городов");  
 this.setClosable(false);  
 this.setContent(layout);  
 }  
  
 private FileChooser createJsonChooser() {  
 FileChooser jsonChooser = new FileChooser();  
 ExtensionFilter extFilter = new ExtensionFilter("JSON файлы","\*.json");  
  
 jsonChooser.getExtensionFilters().add(extFilter);  
 jsonChooser.setTitle("Выберите json файл");  
  
 return jsonChooser;  
 }  
  
 private VBox createLayout() {  
 VBox vBox = new VBox();  
 Insets insets = new Insets(10, 10, 10, 10);  
  
  
 vBox.setSpacing(10);  
 vBox.setPadding(insets);  
  
 return vBox;  
 }  
  
 private HBox createHeader() {  
 HBox hBox = new HBox();  
 Button delBtn = new Button("Удалить выбранный");  
 Button loadCitiesBtn = new Button("Загрузить список");  
 Button saveCitiesBtn = new Button("Сохранить список");  
  
 delBtn.setOnAction(e -> this.removeSelectedCity());  
 loadCitiesBtn.setOnAction(e -> this.loadCitiesFromFile());  
 saveCitiesBtn.setOnAction(e -> this.saveCitiesToFile());  
  
 hBox.setSpacing(10);  
 hBox.getChildren().addAll(delBtn, loadCitiesBtn, saveCitiesBtn);  
  
 return hBox;  
 }  
  
 private TableView createTable() {  
 TableView table = new TableView();  
 TableColumn cityNameColumn = new TableColumn("Название");  
 TableColumn xPosColumn = new TableColumn("Координата X");  
 TableColumn yPosColumn = new TableColumn("Координата Y");  
  
 cityNameColumn.setPrefWidth(250);  
 xPosColumn.setPrefWidth(105);  
 yPosColumn.setPrefWidth(105);  
  
 cityNameColumn.setCellValueFactory(new PropertyValueFactory<City, String>("name"));  
 xPosColumn.setCellValueFactory(new PropertyValueFactory<City, String>("posX"));  
 yPosColumn.setCellValueFactory(new PropertyValueFactory<City, String>("posY"));  
  
 table.setEditable(true);  
 table.setItems(cities);  
 table.getColumns().addAll(cityNameColumn, xPosColumn, yPosColumn);  
  
 return table;  
 }  
  
 private void removeSelectedCity() {  
 City selectedCity = table.getSelectionModel().getSelectedItem();  
 table.getItems().remove(selectedCity);  
 }  
  
 private void saveCitiesToFile() {  
 File file = this.jsonChooser.showSaveDialog(this.stage);  
  
 if (file == null) {  
 return;  
 }  
  
 try {  
 FileWriter fileWriter = new FileWriter(file);  
 String citiesJson = this.getCitiesJson();  
  
 fileWriter.write(citiesJson);  
 fileWriter.flush();  
 } catch (Exception ex) {  
 System.out.print(ex.getMessage());  
 }  
 }  
  
 private void loadCitiesFromFile() {  
 File file = this.jsonChooser.showOpenDialog(this.stage);  
  
 if (file == null) {  
 return;  
 }  
  
 try {  
 JSONParser parser = new JSONParser();  
 Object text = parser.parse(new FileReader(file));  
 JSONArray json = (JSONArray) text;  
 ObservableList<City> cities = this.readCitiesFromJson(json);  
  
 this.cities.clear();  
 this.cities.addAll(cities);  
 } catch (Exception ex) {  
 System.out.print(ex.getMessage());  
 }  
 }  
  
 private String getCitiesJson() {  
 JSONArray citiesJson = new JSONArray();  
  
 for (City city : this.cities) {  
 JSONObject cityJson = new JSONObject();  
  
 cityJson.put(CityJsonFields.NAME.toString(), city.getName());  
 cityJson.put(CityJsonFields.POS\_X.toString(), city.getPosX());  
 cityJson.put(CityJsonFields.POS\_Y.toString(), city.getPosY());  
  
 citiesJson.add(cityJson);  
 }  
  
 return citiesJson.toJSONString();  
 }  
  
 private ObservableList<City> readCitiesFromJson(JSONArray citiesJson) {  
 ObservableList<City> cities = FXCollections.observableArrayList();  
  
 for (JSONObject cityJson : (Iterable<JSONObject>) citiesJson) {  
 String cityName = (String) cityJson.get(CityJsonFields.NAME.toString());  
 Long cityPosX = (Long) cityJson.get(CityJsonFields.POS\_X.toString());  
 Long cityPosY = (Long) cityJson.get(CityJsonFields.POS\_Y.toString());  
 City city = new City(cityName, cityPosX.intValue(), cityPosY.intValue());  
  
 cities.add(city);  
 }  
  
 return cities;  
 }  
}

## A.3 Класс HelpTab

package tabs;  
  
import javafx.geometry.Insets;  
import javafx.scene.control.Label;  
import javafx.scene.control.Tab;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Font;  
  
public class HelpTab extends Tab {  
 private final Font labelFont = new Font("Arial", 20);  
 private final Font descriptionFont = new Font("Arial", 14);  
  
 public HelpTab() {  
 this.setText("Помощь");  
 this.setClosable(false);  
 this.setContent(this.createLayout());  
 }  
  
 private VBox createLayout() {  
 VBox vBox = new VBox();  
  
 vBox.setSpacing(20);  
 vBox.setPadding(new Insets(10, 10, 10, 10));  
 vBox.getChildren().addAll(  
 this.createCitiesListHelp(),  
 this.createMapHelp(),  
 this.createRoutesHelp()  
 );  
  
 return vBox;  
 }  
  
 private VBox createHelp(String labelText, String descriptionText) {  
 VBox vBox = new VBox();  
 Label label = new Label(labelText);  
 Label description = new Label(descriptionText);  
  
 label.setFont(labelFont);  
 label.setPadding(new Insets(0, 0, 0, 27));  
 description.setWrapText(true);  
 description.setLineSpacing(3.5);  
 description.setFont(descriptionFont);  
  
 vBox.setSpacing(10);  
 vBox.getChildren().addAll(label, description);  
  
 return vBox;  
 }  
  
 private VBox createCitiesListHelp() {  
 return this.createHelp(  
 "Список городов",  
 " На этой вкладке можно задать список городов. " +  
 "В нижней части окна находится форма добавления города. " +  
 "Если необходимо удалить город то нужно выбрать его в таблице и нажать кнопку «Удалить выбранный». " +  
 "\n Список городов можно сохранить в файл, для этого нужно нажать на кнопку «Загрузить список». " +  
 "В дальнейшем можно загрузить данные из этого файла с помощью кнопки «Сохранить список»."  
 );  
 }  
  
 private VBox createMapHelp() {  
 return this.createHelp(  
 "Карта",  
 " На этой вкладке показывается маршрут между городами, от начального до конечного."  
 );  
 }  
  
 private VBox createRoutesHelp() {  
 return this.createHelp(  
 "Маршрут",  
 " На этой вкладке строится маршрут между городами. Для этого в выпадающих " +  
 "списках нужно выбрать начальный и конечный города, после нажать кнопку «Построить маршрут». " +  
 "Результат можно увидеть на вкладке «Карта»."  
 );  
 }  
}

## A.4 Класс MapTab

package tabs;  
  
import javafx.collections.ObservableList;  
import javafx.geometry.Insets;  
import javafx.scene.Node;  
import javafx.scene.chart.NumberAxis;  
import javafx.scene.chart.LineChart;  
import javafx.scene.chart.XYChart;  
import javafx.scene.control.Label;  
import javafx.scene.control.Tab;  
import javafx.scene.text.Font;  
import models.City;  
import models.GraphBounds;  
  
import java.util.List;  
  
public class MapTab extends Tab {  
 private final ObservableList<City> cities;  
 private final List<City> route;  
  
 public MapTab(ObservableList<City> cities, List<City> route) {  
 this.cities = cities;  
 this.route = route;  
 this.setText("Карта");  
 this.setClosable(false);  
 this.setOnSelectionChanged(e -> this.update());  
 }  
  
 private void update() {  
 this.setContent(this.createLayout());  
 }  
  
 private Node createLayout() {  
 if (this.cities.size() > 1) {  
 return this.createGraph(this.cities, this.route);  
 }  
  
 return this.createFallback();  
 }  
  
 private Node createFallback() {  
 Label label = new Label("Необходимо указать данные о более чем одном городе для построения карты");  
  
 label.setWrapText(true);  
 label.setFont(new Font("Arial", 18));  
 label.setPadding(new Insets(10));  
  
 return label;  
 }  
  
 private Node createGraph(ObservableList<City> cities, List<City> route) {  
 final GraphBounds graphBounds = this.getGraphBounds(cities);  
 final NumberAxis xAxis = new NumberAxis(graphBounds.getMinX(), graphBounds.getMaxX(), 1);  
 final NumberAxis yAxis = new NumberAxis(graphBounds.getMinY(), graphBounds.getMaxY(), 1);  
 final LineChart<Number, Number> lineChart = new LineChart(xAxis, yAxis);  
 final XYChart.Series seriesCities = new XYChart.Series();  
 final Font labelFont = new Font("Arial", 12);  
  
 seriesCities.setName("Город");  
  
 lineChart.getXAxis().setTickMarkVisible(false);  
 lineChart.getYAxis().setTickMarkVisible(false);  
 lineChart.setHorizontalZeroLineVisible(false);  
 lineChart.setVerticalZeroLineVisible(false);  
 lineChart.setLegendVisible(false);  
 lineChart.getData().addAll(seriesCities);  
 lineChart.lookup(".default-color0.chart-series-line").setStyle("-fx-stroke: transparent");  
  
 for (int idx = 1; idx < route.size(); idx++) {  
 City prevCity = route.get(idx - 1);  
 City curCity = route.get(idx);  
 XYChart.Series series = new XYChart.Series();  
 XYChart.Data firstDot = new XYChart.Data(prevCity.getPosX(), prevCity.getPosY());  
 XYChart.Data secondDot = new XYChart.Data(curCity.getPosX(), curCity.getPosY());  
 String selector = ".default-color" + idx + ".chart-series-line";  
  
 series.getData().addAll(firstDot, secondDot);  
 lineChart.getData().add(series);  
 lineChart.lookup(selector).setStyle("-fx-stroke: green");  
 }  
  
 for (City city : cities) {  
 final XYChart.Data data = new XYChart.Data(city.getPosX(), city.getPosY());  
 final Label label = new Label(city.getName());  
  
 label.setFont(labelFont);  
 data.setNode(label);  
 data.getNode().toFront();  
  
 seriesCities.getData().add(data);  
 }  
  
 return lineChart;  
 }  
  
 private GraphBounds getGraphBounds(ObservableList<City> cities) {  
 Integer minX = Integer.MAX\_VALUE;  
 Integer maxX = Integer.MIN\_VALUE;  
 Integer minY = Integer.MAX\_VALUE;  
 Integer maxY = Integer.MIN\_VALUE;  
 final Integer padding = 2;  
  
 for (City city : cities) {  
 Integer posX = city.getPosX();  
 Integer posY = city.getPosY();  
  
 if (posX > maxX) {  
 maxX = posX;  
 }  
  
 if (posX < minX) {  
 minX = posX;  
 }  
  
 if (posY > maxY) {  
 maxY = posY;  
 }  
  
 if (posY < minY) {  
 minY = posY;  
 }  
 }  
  
 return new GraphBounds(minX - padding, maxX + padding, minY - padding, maxY + padding);  
 }  
}

## A.5 Класс RouteTab

package tabs;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.geometry.Insets;  
import javafx.scene.Node;  
import javafx.scene.control.Alert;  
import javafx.scene.control.Alert.AlertType;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.Tab;  
import javafx.scene.layout.HBox;  
import models.City;  
import lib.ShortestRouteFinder;  
  
import java.util.List;  
  
public class RouteTab extends Tab {  
 private final ObservableList<City> cities;  
 private City firstCity;  
 private City lastCity;  
 private List<City> route;  
  
 public RouteTab(ObservableList<City> cities, List<City> route) {  
 this.cities = cities;  
 this.route = route;  
 this.setText("Маршрут");  
 this.setClosable(false);  
 this.setContent(this.createLayout());  
 this.setOnSelectionChanged(e -> this.update());  
 }  
  
 private void update() {  
 this.setContent(this.createLayout());  
 }  
  
 private Node createLayout() {  
 HBox hBox = new HBox();  
 ObservableList<String> options = this.getOptions(this.cities);  
 ComboBox firstCitySelect = new ComboBox(options);  
 ComboBox lastCitySelect = new ComboBox(options);  
 Button submitBtn = new Button("Построить маршрут");  
  
 submitBtn.setOnAction(e -> this.createRoute());  
  
 firstCitySelect.setPromptText("Начальный город");  
 lastCitySelect.setPromptText("Конечный город");  
  
 if (this.firstCity != null) {  
 firstCitySelect.setValue(this.firstCity.getName());  
 }  
  
 if (this.lastCity != null) {  
 lastCitySelect.setValue(this.lastCity.getName());  
 }  
  
 firstCitySelect.setOnAction(e -> this.handleFirstCityChange(firstCitySelect));  
 lastCitySelect.setOnAction(e -> this.handleLastCityChange(lastCitySelect));  
  
 hBox.setSpacing(10);  
 hBox.setPadding(new Insets(10));  
 hBox.getChildren().addAll(firstCitySelect, lastCitySelect, submitBtn);  
  
 return hBox;  
 }  
  
 private void handleFirstCityChange(ComboBox comboBox) {  
 Integer index = comboBox.getSelectionModel().getSelectedIndex();  
 City selectedCity = this.cities.get(index);  
  
 if (selectedCity == null) {  
 return;  
 }  
  
 this.firstCity = selectedCity;  
 }  
  
 private void handleLastCityChange(ComboBox comboBox) {  
 Integer index = comboBox.getSelectionModel().getSelectedIndex();  
 City selectedCity = this.cities.get(index);  
  
 if (selectedCity == null) {  
 return;  
 }  
  
 this.lastCity = selectedCity;  
 }  
  
 private ObservableList<String> getOptions(ObservableList<City> cities) {  
 ObservableList<String> options = FXCollections.observableArrayList();  
  
 for (City city : cities) {  
 options.add(city.getName());  
 }  
  
 return options;  
 }  
  
 private void createRoute() {  
 Alert alert = new Alert(AlertType.INFORMATION);  
  
 alert.setTitle("Ошибка");  
 alert.setHeaderText("Ошибка");  
  
 if (this.firstCity == null) {  
 alert.setContentText("Не выбран начальный город");  
 alert.show();  
  
 return;  
 }  
  
 if (this.lastCity == null) {  
 alert.setContentText("Не выбран конечный город");  
 alert.show();  
  
 return;  
 }  
  
 ShortestRouteFinder routeFinder = new ShortestRouteFinder();  
  
 routeFinder.setCities(this.cities);  
 routeFinder.setFirstCity(this.firstCity);  
 routeFinder.setLastCity(this.lastCity);  
  
 this.route.clear();  
 this.route.addAll(routeFinder.find());  
 }  
}

## A.6 Класс City

package models;  
  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
  
public class City {  
 private final SimpleStringProperty name;  
 private final SimpleIntegerProperty posX;  
 private final SimpleIntegerProperty posY;  
  
 public City(String name, int posX, int posY) {  
 this.name = new SimpleStringProperty(name);  
 this.posX = new SimpleIntegerProperty(posX);  
 this.posY = new SimpleIntegerProperty(posY);  
 }  
  
 public String getName() {  
 return this.name.get();  
 }  
  
 public void setName(String fName) {  
 this.name.set(fName);  
 }  
  
 public int getPosX() {  
 return this.posX.get();  
 }  
  
 public void setPosX(int posX) {  
 this.posX.set(posX);  
 }  
  
 public int getPosY() {  
 return this.posY.get();  
 }  
  
 public void setPosY(int posY) {  
 this.posY.set(posY);  
 }  
}

## A.7 Класс GraphBounds

package models;  
  
public class GraphBounds {  
 private final Integer minX;  
 private final Integer maxX;  
 private final Integer minY;  
 private final Integer maxY;  
  
 public GraphBounds(Integer minX, Integer maxX, Integer minY, Integer maxY) {  
 this.minX = minX;  
 this.maxX = maxX;  
 this.minY = minY;  
 this.maxY = maxY;  
 }  
  
 public Integer getMaxX() {  
 return this.maxX;  
 }  
  
 public Integer getMinX() {  
 return this.minX;  
 }  
  
 public Integer getMaxY() {  
 return this.maxY;  
 }  
  
 public Integer getMinY() {  
 return this.minY;  
 }  
}

## A.8 Класс Helpers

package lib;  
  
import java.util.ArrayList;  
import java.util.List;  
  
class Helpers {  
  
 /\*\*  
 \* Возвращает список всех возможных вариаций расстановки элементов списка  
 \*  
 \* @param <T> Тип элемента в массиве  
 \* @param list массив элементов  
 \* @return {@code List<List<T>}  
 \*/  
 public static <T> List<List<T>> getListPermutations(List<T> list) {  
 List<List<T>> result = new ArrayList<>();  
  
 if (list.isEmpty()) {  
 result.add(new ArrayList<>());  
  
 return result;  
 }  
  
 T firstElement = list.remove(0);  
 List<List<T>> permutations = getListPermutations(list);  
  
 permutations.forEach((List<T> smallerPermutated) -> {  
 for (int idx = 0; idx <= smallerPermutated.size(); idx++) {  
 List<T> temp = new ArrayList<>(smallerPermutated);  
  
 temp.add(idx, firstElement);  
 result.add(temp);  
 }  
 });  
  
 return result;  
 }  
}

## A.9 Класс ShortestRouteFinder

package lib;  
  
import models.City;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class ShortestRouteFinder {  
 private City firstCity;  
 private City lastCity;  
 private List<City> cities;  
  
 public ShortestRouteFinder() {  
 this.cities = new ArrayList<>();  
 }  
  
 public void setCities(List<City> cities) {  
 this.cities = cities;  
 }  
  
 public void setFirstCity(City firstCity) {  
 this.firstCity = firstCity;  
 }  
  
 public void setLastCity(City lastCity) {  
 this.lastCity = lastCity;  
 }  
  
 public List<City> find() {  
 List<City> cities = new ArrayList<>(this.cities);  
 List<List<City>> allPaths = Helpers.getListPermutations(cities);  
 List<List<City>> filteredPaths = this.filterPaths(allPaths);  
  
 return this.getShortestPath(filteredPaths);  
 }  
  
 private List<City> getShortestPath(List<List<City>> paths) {  
 List<City> shortestPath = null;  
 Integer lowestCost = Integer.MAX\_VALUE;  
  
 for (List<City> currentPath : paths) {  
 Integer cost = this.getPathCost(currentPath);  
  
 if (cost < lowestCost) {  
 lowestCost = cost;  
 shortestPath = currentPath;  
 }  
 }  
  
 return shortestPath;  
 }  
  
 private Integer getPathCost(List<City> path) {  
 Integer cost = 0;  
  
 for (Integer idx = 1; idx < path.size(); idx++) {  
 City prevCity = path.get(idx - 1);  
 City curCity = path.get(idx);  
  
 if (prevCity == null) {  
 return cost;  
 }  
  
 double distance = this.getDistanceBetween(prevCity, curCity);  
  
 cost += (int) distance;  
 }  
  
 return cost;  
 }  
  
 private double getDistanceBetween(City firstCity, City secondCity) {  
 Integer x1 = firstCity.getPosX();  
 Integer y1 = firstCity.getPosY();  
 Integer x2 = secondCity.getPosX();  
 Integer y2 = secondCity.getPosY();  
  
 return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));  
 }  
  
 private List<List<City>> filterPaths(List<List<City>> paths) {  
 List<List<City>> filteredPaths = new ArrayList<>();  
  
 for (List<City> path : paths) {  
 if (path.isEmpty()) {  
 continue;  
 }  
  
 if (path.get(0) != this.firstCity) {  
 continue;  
 }  
  
 City lastCity = path.get(path.size() - 1);  
  
 if (lastCity != this.lastCity) {  
 continue;  
 }  
  
 filteredPaths.add(path);  
 }  
  
 return filteredPaths;  
 }  
}

## A.10 Перечисление CityJsonFields

package enums;  
  
public enum CityJsonFields {  
 NAME("name"),  
 POS\_X("posX"),  
 POS\_Y("posY");  
  
 private final String text;  
  
 CityJsonFields(final String text) {  
 this.text = text;  
 }  
  
 @Override  
 public String toString() {  
 return this.text;  
 }  
}

## A.11 Класс CityForm

package controls;  
  
import javafx.collections.ObservableList;  
import javafx.scene.control.Button;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.HBox;  
  
import models.City;  
  
public class CityForm extends HBox {  
 private final ObservableList<City> cities;  
 private final TextField xPosField = new NumberField();  
 private final TextField yPosField = new NumberField();  
 private final TextField cityNameField = new TextField();  
 private final Button addCityButton = new Button("Добавить город");  
  
 public CityForm(ObservableList<City> cities) {  
 this.cities = cities;  
 this.setSpacing(10);  
 this.getChildren().addAll(cityNameField, xPosField, yPosField, addCityButton);  
  
 cityNameField.setPromptText("Название города");  
 xPosField.setPromptText("Координата X");  
 yPosField.setPromptText("Координата Y");  
  
 cityNameField.setPrefWidth(250);  
 xPosField.setPrefWidth(95);  
 yPosField.setPrefWidth(95);  
  
 addCityButton.setOnAction(e -> {  
 City newCity = createCity();  
  
 if (newCity != null) {  
 cities.add(newCity);  
 reset();  
 }  
 });  
 }  
  
 private void reset() {  
 cityNameField.clear();  
 xPosField.clear();  
 yPosField.clear();  
 }  
  
 private City createCity() {  
 String cityName = cityNameField.getText();  
 String xPosStr = xPosField.getText();  
 String yPosStr = yPosField.getText();  
  
 if (cityName.isEmpty() || xPosStr.isEmpty() ||yPosStr.isEmpty()) {  
 return null;  
 }  
  
 try {  
 Integer xPos = Integer.parseInt(xPosStr);  
 Integer yPos = Integer.parseInt(yPosStr);  
  
 return new City(cityName, xPos, yPos);  
 } catch (Exception ex) {  
 return null;  
 }  
 }  
}

## A.12 Класс NumberField

package controls;  
  
import javafx.beans.value.ChangeListener;  
import javafx.beans.value.ObservableValue;  
import javafx.scene.control.TextField;  
  
public class NumberField extends TextField {  
 public NumberField() {  
 this.textProperty().addListener(new ChangeListener<String>() {  
 @Override  
 public void changed(  
 ObservableValue<? extends String> observable,  
 String oldValue,  
 String newValue  
 ) {  
 if (!newValue.matches("\\d\*")) {  
 setText(newValue.replaceAll("[^\\d]", ""));  
 }  
 }  
 });  
 }  
}