МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по учебной практике

ТЕМА: AIRBALLOON

Студент гр. 8303 _____ Бородкин Ю.В.

Студент гр. 8303 _____ Дирксен А.А.

Студент гр. 8303 _____ Сенюшкин Е.В.

Руководитель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Бородкин Ю.В. группы 8303

Студент Дирксен А.А. группы 8303

Студент Сенюшкин Е.В. группы 8303

Тема практики: AirBalloon

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Дейкстра.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 12.07.2020

Дата защиты отчета: 12.07.2020

Студент	_____	Бородкин Ю.В.
Студен	_____	Дирксен А.А.
Студент	_____	Сенюшкин Е.В.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

Основной целью работы является получение навыков программирования на языке Java и освоение парадигмы объектно-ориентированного программирования. Цель достигается командной работой путём разработки программы с графическим интерфейсом, использующей указанные в задании алгоритмы. В этой работе необходимо реализовать алгоритм, находящий кратчайший путь от заданной точки (пункт отправления) до отмеченной пользователем точки (пункт прибытия). Все объекты вносятся на карту пользователем взаимодействием с оконным приложением. Для проверки корректности программы используются JUnit тесты.

SUMMARY

The main goal of the work is to get programming skills in the Java language and mastering of object-oriented programming paradigm. The goal is achieved by teamwork by developing a program with a graphical interface that uses the algorithms specified in the task. In this work, it is necessary to implement an algorithm that finds the shortest path from a given point (departure point) to a point marked by the user (arrival point). All objects are brought to the map by the user through interaction with a window application. To verify the correctness of the program, JUnit tests are used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача практики состоит в разработке приложения, позволяющего на карте в оконном приложении рассчитывать время полёта шара. Программа должна позволять пользователю находить кратчайший путь от пункта отправления до пункта прибытия. Для поиска кратчайшего пути используется алгоритм Дейкстры.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Спецификация проекта "Расчёт полёта воздушного шара с двигателем"

1.1. Описание задачи

Пользователь выбирает из предоставленного списка один из регионов мира (нужно подобрать с хорошим ветром). Появляется Карта данного региона на которой можно выбрать отправной пункт для воздушного шара (например, крупный город) и пункт прибытия. Карта разбивается на квадраты. Программа рассчитывает кратчайший путь и визуализирует его на карте. При расчете пути учитывается собственная скорость шара, а также скорость и направление ветра. При достижении путь визуализируется и выводится время полёта. Для получения данных о ветре используется web-сервис, предоставляющий api для получения данных о погоде.

1.2. Интерфейс

Интерфейс программы состоит из:

- Карты мира, на которой пользователь выбирает регион, а затем координаты начала и конца пути.
- Поля с выбором собственной скорости шара
- Кнопки начала построения пути

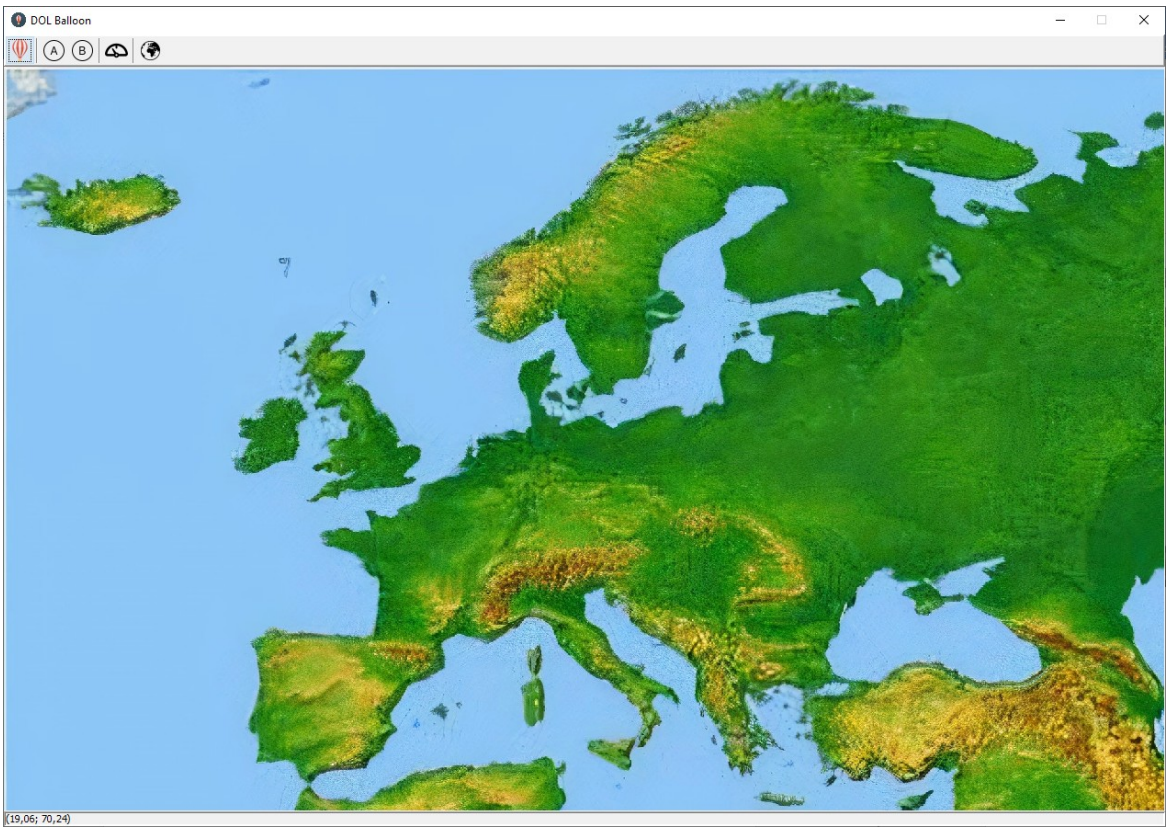


Рисунок 1. Меню приложения

Рисунок 2. Установка точек в регионе

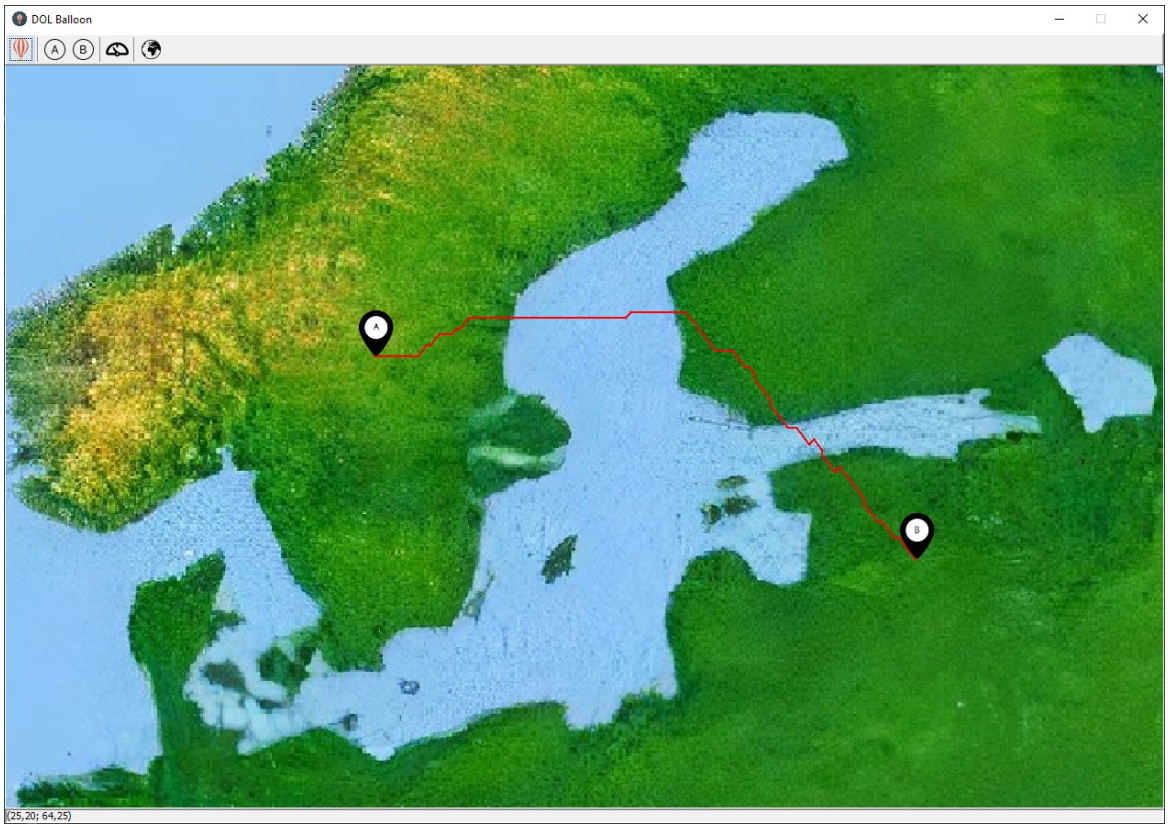


Рисунок 3. Найденный путь

1.3. Входные данные

Входные данные для алгоритма формируются соответственно следующим набором:

- Координаты начала и конца пути
- Скорость шара

1.4. Выходные данные

Алгоритм расчета полета формирует оптимальный путь в виде последовательности пар вершина-время прибытия в вершину, по которым визуализируется путь.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. К 1 июля начать работу над пользовательским интерфейсом, обсудить применения алгоритмов для решения поставленной задачи.
2. К 3 июля подготовить прототип пользовательского интерфейса и спецификацию проекта.
3. К 5 июля добавить возможность отслеживать значения долготы/широты, при изменении положения курсора на карте
4. К 7 июля прописать поиск кратчайшего пути без влияния ветра.
5. К 9 июля организовать пробное соединение интерфейса и логики, перевести контекстное меню в меню инструментов.
6. К 10 июля добавить влияние ветра в логику поиска пути.
7. К 11 июля закончить работу по логике интерфейсу программы
8. К 12 июля завершить разработку проекта.

2.2. Распределение ролей в бригаде

- Бородкин Юрий – фронтенд;
- Дирксен Антон – лидер, тестирование, документация;
- Сенюшкин Егор – алгоритмист.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Application – класс, наследуемый от JFrame, является основным окном, на котором отображаются Status Bar, Tool Bar и карта регионов:

- final int width – ширина окна;
- final int height – высота окна;

ImagePanel – класс, представляющий динамически изменяемую панель, в которой отображена карта:

- private final JFrame frame – главное окно;
- private final ToolBar toolBar – меню инструментов;
- private final JLabel statusBar – статус меню;
- private final Image europeMap – изображение карты Европы;
- private final Image balticSeaMap – изображение карты Балтийского моря;
- private final Image blackSeaMap – изображение карты Чёрного моря;
- private final Image EMBaltic – карта Европы с выделенным Балтийским морем;
- private final Image EMBlack – карта Европы с выделенным Чёрным морем;
- private Image currMap – текущая карта региона;
- private Region region – текущий регион;
- private int widthMap – широта карты;
- private int heightMap – высота карты;
- Point A – точка отправления;
- Point B – точка прибытия;

SpeedControllerWindow – класс, представляющий диалоговое окно, на котором можно выбрать скорость шара:

- private static int currentSpeed – текущее значение скорости;
- private final JSlider slider – ползунок скорости;

ToolBar – класс, представляющий меню инструментов:

- private final int widthAction – ширина кнопки;
- private final int heightAction – длина кнопки;
- public ImagePanelObserver observer – «наблюдатель» над классом ImagePanel;
- Action pressedAction – поле, хранящее последнюю нажатую кнопку;

GeoCoordinates – класс, хранящий широту и долготу:

- private double longitude – долгота;
- private double latitude – широта;

Point – класс точки на карте:

- private GeoCoordinates coordinates – координаты точки;
- private final Image imagePoint – изображение точки;

Region – перечисление регионов.

Dijkstra – класс, хранящий граф и реализующий поиск пути

- Set<Coordinates> A — множество не посещенных вершин
- Set<Coordinates> B — множество посещенных вершин
- Map<Coordinates, Wind> windMap — словарь для хранения данных о ветре в точках
- int speed — скорость шара
- Map<Coordinates, Double> minWay — словарь кратчайщих путей
- Map<Coordinates, Double> minTime — словарь минимальных времен для достижения вершины
- LinkedList<Coordinates> list — список координат, по которым будет лететь шар до цели

- PriorityQueue<Way> priorityQueue — очередь с приоритеом для алгоритмы Дейкстры
- boolean region — переменная региона (Балтийское или Черное море)

3.2. Основные методы

public class Dijkstra:

- private void initRegion() — метод для инициализации графа для определенного региона на основе переменной boolean region
- public void dijkstra(Coordinates from, Coordinates to, int speed, boolean region) — метод нахождения минимального пути, принимает координата начала и конца пути, скорость шара и регион в котором ищется путь.
- private void addCoordinates(Coordinates s, double distance, double time) — метод добавления в приоритетную очередь все смежные вершины, принимает координату, кратчайший путь и минимальное время до этой координаты.
- private Coordinates adjCoordinates(int lat, int lon) — метод проверки на то, что координата не выходит за пределы региона, принимает широту и долготу. В случае успеха возвращает новую координату, иначе null.
- protected double[] setTimeOfFlight(Coordinates c) — метод расчета результирующей скорости из заданной точки во все смежные, принимает координату, возвращает массив из 8 элементов с результирующей скоростью в смежные вершины.
- private double distanceFormula(Coordinates c1, Coordinates c2) — метод расчета расстояния между двумя географическими координатами, принимает координаты и возвращает расстояние

- private double haverSinus(double x) — метод для расчета формулы гаверсинусов, принимает угол в радианах и возвращает значение гаверсинуса.
- public LinkedList<Coordinates> getWayFromCoordinates (Coordinates b) — метод получения кратчайшего пути до точки прибытия, принимает точку прибытия и возвращает связный список вершин из точки начала пути до точки прибытия.

4. ТЕСТИРОВАНИЕ

4.1. Ручное тестирование

Для наглядности приложены скриншоты карты ветров из Яндекс.Погоды на момент тестирования.

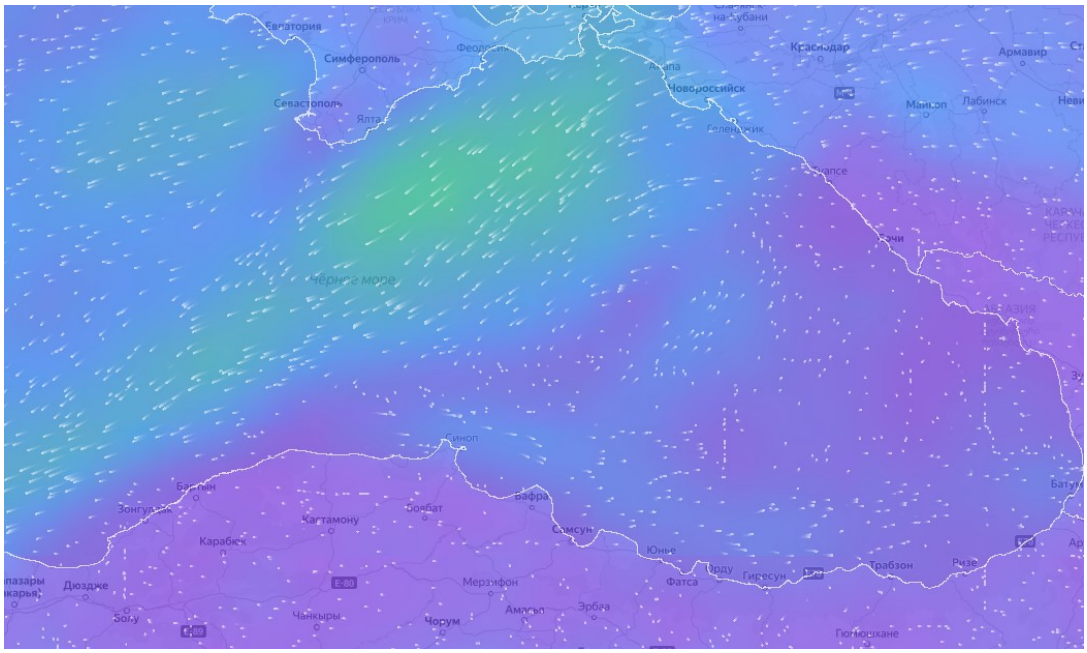


Рисунок 4. Карта ветров на момент тестирования

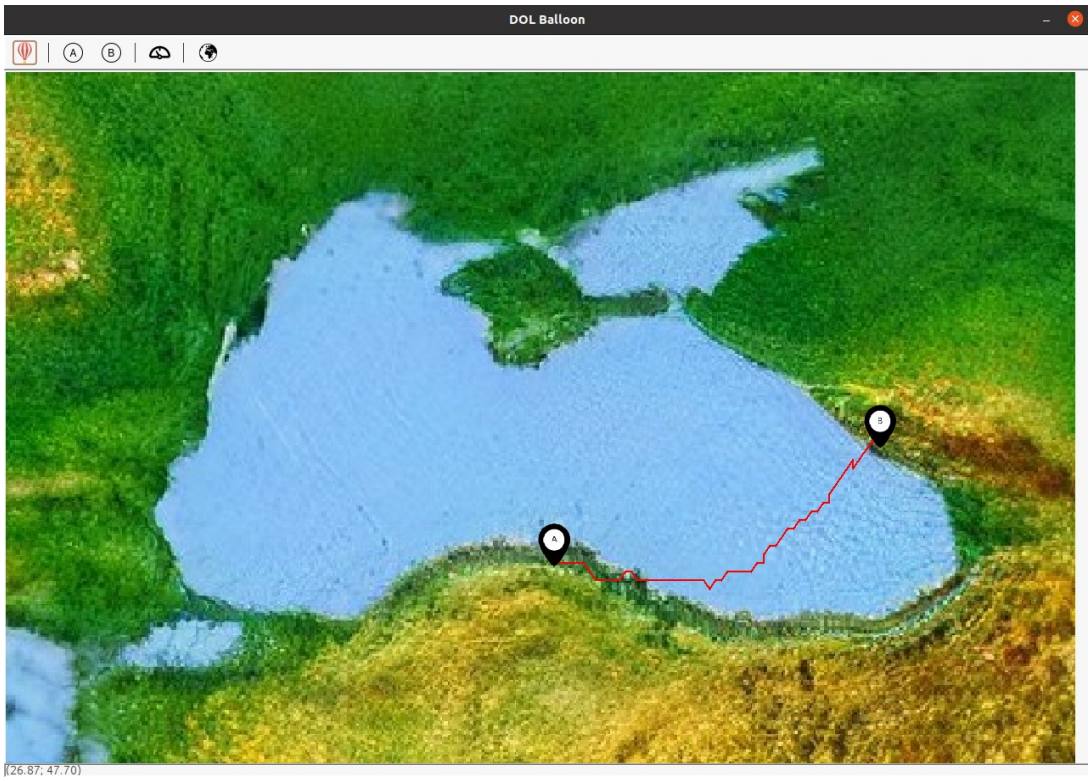


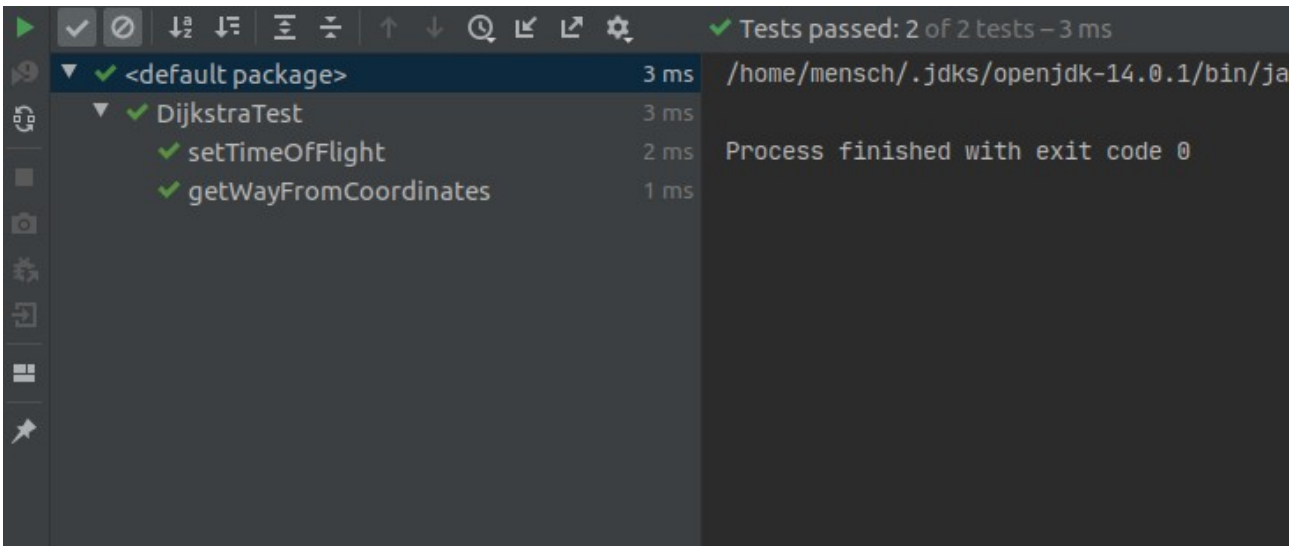
Рисунок 5. Результат ручного тестирования

4.2. Unit-тестирование

Для класса Dijkstra применено Unit-тестирование с использованием библиотеки Junit 4. Создан класс DijkstraTest и реализованы методы для тестирования получения конечного пути (метод getWayFromCoordinates) и расчета результирующей скорости шара (метод setTimeOfFlight).

Для метода getWayFromCoordinates искусственно заполняются словарь переходов между вершинами и связный список конечного пути. Затем, список сравнивается со списком, возвращенным методом getWayFromCoordinates, если они отличаются, выводится сообщение о проваленном тесте.

Для метода setTimeOfFlight искусственно заполняется ветер для одной координаты и вызывается метод setTimeOfFlight для этой точки. Затем, полученный массив сравнивается с массивом заранее посчитанных результирующих скоростей, и если они отличаются, выводится сообщение о проваленном тесте.



ЗАКЛЮЧЕНИЕ

В соответствии с требованиями технического задания была реализована программа для расчета оптимального маршрута и времени полета воздушного шара с двигателем. Разработан пользовательский интерфейс. Он позволяет выбрать регион, начало и конец пути и изменить скорость воздушного шара.

Для поиска пути использовался алгоритм Дейкстры, позволяющий находить минимальный пути из одной точки графа в другие. Для заполнения графа использовались данные о ветре, которые берутся с погодного сервиса.

В результаты работы были освоены методы работы в команде, распределены обязанности, оттренированы навыки работы с системой контроля версий и канбан-доской на примере GitHub. Также команда была ознакомлена с языком программирования Java, в частности с фреймворком Swing для создания графического интефейса, библиотекой JUnit 4 для модульного тестирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Г.Шилдт. Swing. Руководство для начинающих.

2. Java 8. Полное руководство. 9-е издание - Шилдт Г.

3. Руководство по JUnit 4 // https://www.javaguides.net/p/junit-4.html

4. Документация Local Weather API // https://www.worldweatheronline.com/developer/api/docs/local-city-town-weather-api.aspx

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

ImagePanel.java

```
package com.dol.ui.elements;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import javax.imageio.ImageIO;

import com.dol.Coordinates;
import com.dol.Dijkstra;
import com.dol.ui.exceptions.FileNotFoundException;
import com.dol.ui.util.GeoCoordinates;
import com.dol.ui.util.ImagePanelObserver;
import com.dol.ui.util.Point;
import com.dol.ui.util.Region;

public class ImagePanel extends JPanel
        implements MouseListener, MouseMotionListener {

    private final JFrame frame;
    private final ToolBar toolBar;
    private final JLabel statusBar;

    private final Image europeMap;
    private final Image balticSeaMap;
    private final Image blackSeaMap;
    private final Image EMBaltic;
    private final Image EMBlack;

    private Image currMap;
```

```
    private Region region = Region.EUROPE;

    private int widthMap;
    private int heightMap;

    Dijkstra algorithm;
    Point A;
    Point B;

    public ImagePanel(JFrame frame, ToolBar toolBar, JLabel statusBar) throws
FileNotFoundException {
        algorithm = new Dijkstra();
        this.frame = frame;

        toolBar.observer = new ImagePanelObserver(this);
        this.toolBar = toolBar;
        this.statusBar = statusBar;

        addMouseListener(this);
        addMouseMotionListener(this);

        try {
            europeMap = ImageIO.read(new File("resources" + File.separator + "europe map.jpg"));
            currMap = europeMap;
        } catch (IOException e) {
            throw new FileNotFoundException("File \"europe map.jpg\" not found");
        }

        try {
            A = new Point(ImageIO.read(new File("resources" + File.separator + "point A.png")));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"point A.png\" not found");
        }

        try {
```

```
            B = new Point(ImageIO.read(new File("resources" + File.separator + "point B.png")));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"point B.png\" not found");
        }

        try {
            balticSeaMap = ImageIO.read(new File("resources" + File.separator + "baltic sea.jpg"));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"baltic sea.jpg\" not found");
        }

        try {
            blackSeaMap = ImageIO.read(new File("resources" + File.separator + "black sea.jpg"));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"black sea.jpg\" not found");
        }

        try {
            EMBaltic = ImageIO.read(new File("resources" + File.separator + "EM with baltic.jpg"));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"EM with baltic.jpg\" not found");
        }

        try {
            EMBlack = ImageIO.read(new File("resources" + File.separator + "EM with black.jpg"));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"EM with black.jpg\" not found");
        }
    }

    protected void paintComponent(Graphics g) {
        //выравнивание до пропорций исходного изображения
        widthMap = frame.getWidth() - 17;
        heightMap = frame.getHeight() - 91;
        g.drawImage(currMap, 0, 0, widthMap, heightMap, null);
    }
```

```
    private void paintPoint(Point point) {
        if (!point.isWorth()) return;
        getGraphics().drawImage(point.getImagePoint(), point.getX() - 25, point.getY() - 50, 50, 50,
null);
    }

    private void clearMap() {
        paintComponent(getGraphics());
    }

    public Point getPointA() {
        return A;
    }

    public Point getPointB() {
        return B;
    }

    public Region getRegion() {
        return region;
    }

    public void openSpeedControllerWindow() {
        new SpeedControllerWindow(frame).setVisible(true);
    }
    public void setDefaultRegion() {
        A.deletePoint();
        B.deletePoint();
        region = Region.EUROPE;
        currMap = europeMap;
        paintComponent(getGraphics());
    }

    public void run() {
```

```
        if (currMap.equals(europeMap)) {
            JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }
        clearMap();
        paintPoint(A);
        paintPoint(B);
        if (!A.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select a departure point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        if (!B.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select an arrival point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        Coordinates from = new Coordinates((int) (A.getLatitude() * 10), (int) (A.getLongitude() *
10));
        Coordinates to = new Coordinates((int) (B.getLatitude() * 10), (int) (B.getLongitude() * 10));


        switch (region) {
            case BALTIC_SEA :
                algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), true);
                break;

            case BLACK_SEA :
                algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), false);
                break;
        }

        LinkedList <Coordinates> temp = algorithm.getWayFromCoordinates(to);
```

```
        if (temp == null) {
            JOptionPane.showMessageDialog(frame, "Невозможно добраться до пункта назначения!", "Warning!", JOptionPane.WARNING_MESSAGE);
            return;
        }

        JOptionPane.showMessageDialog(frame, String.format("Distance = %.2f km\nTime = %dh: %dm", algorithm.getDistance(to), (int) algorithm.getTime(to), (int) ((algorithm.getTime(to) - (int) algorithm.getTime(to)) * 60)), "Complete!", JOptionPane.INFORMATION_MESSAGE);
        Coordinates[] result = new Coordinates[temp.size()];
        temp.toArray(result);

        for (int i = 1; i < result.length; i++) {

            Coordinates prev = result[i-1];
            Coordinates curr = result[i];

            int prevX = (int) ((prev.getLon() / 10.0 + 25) * widthMap / 75);
            int prevY = (int) ((71 - prev.getLat() / 10.0) * heightMap / 36);

            int currX = (int) ((curr.getLon() / 10.0 + 25) * widthMap / 75);
            int currY = (int) ((71 - curr.getLat() / 10.0) * heightMap / 36);

            switch (region) {

                case BALTIC_SEA : {
                    prevX = (int) ((prev.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                    prevY = (int) ((66.5 - prev.getLat() / 10.0) * heightMap / 13.5);
                    currX = (int) ((curr.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                    currY = (int) ((66.5 - curr.getLat() / 10.0) * heightMap / 13.5);
                    break;
                }

                case BLACK_SEA : {
                    prevX = (int) ((prev.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
```

```
                prevY = (int) ((47.7 - prev.getLat() / 10.0) * heightMap / 8.1) - 6;
                currX = (int) ((curr.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                currY = (int) ((47.7 - curr.getLat() / 10.0) * heightMap / 8.1) - 6;
                break;
            }
        }

        Graphics2D g = (Graphics2D) getGraphics();
        g.setColor(Color.RED);
        g.setStroke(new BasicStroke(2));
        g.drawLine(prevX, prevY, currX, currY);
    }

    paintPoint(A);
    paintPoint(B);
}

@Override
public void mouseClicked(MouseEvent e) {

    GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

    if (region == Region.EUROPE) {


        if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

            region = Region.BALTIC_SEA;
            currMap = balticSeaMap;

            toolBar.resetPressedAction();
            paintComponent(getGraphics());

        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
```

```
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

            region = Region.BLACK_SEA;
            currMap = blackSeaMap;

            toolBar.resetPressedAction();
            paintComponent(getGraphics());

        } else {
            if (toolBar.getPressedAction() == ToolBar.Action.SET_A || toolBar.getPressedAction()
== ToolBar.Action.SET_B) {
                JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
                toolBar.resetPressedAction();
            }
        }

        return;
    }

    switch (toolBar.getPressedAction()) {

        case NOTHING:
            break;

        case SET_A:
            clearMap();

            A.move(point);
            paintPoint(A);
            paintPoint(B);
            break;

        case SET_B:
            clearMap();
```

```
            B.move(point);
            paintPoint(A);
            paintPoint(B);
            break;
        }

        toolBar.resetPressedAction();
    }

    @Override
    public void mousePressed(MouseEvent e) {}

    @Override
    public void mouseReleased(MouseEvent e) {}

    @Override
    public void mouseEntered(MouseEvent e) {}

    @Override
    public void mouseExited(MouseEvent e) {}

    @Override
    public void mouseDragged(MouseEvent e) {}

    @Override
    public void mouseMoved(MouseEvent e) {

        GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

        if (region == Region.EUROPE) {
            if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

                if (!currMap.equals(EMBaltic)) {
                    currMap = EMBaltic;
```

```
                paintComponent(getGraphics());
            }

        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

            if (!currMap.equals(EMBlack)) {
                currMap = EMBlack;
                paintComponent(getGraphics());
            }

        } else if (!currMap.equals(europeMap)) {
            currMap = europeMap;
            paintComponent(getGraphics());
        }
    }
    statusBar.setText(String.format("(%.2f; %.2f)", point.getLongitude(), point.getLatitude()));
  }
}
```

SpeedControllerWindow.java

```
package com.dol.ui.elements;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import javax.imageio.ImageIO;

import com.dol.Coordinates;
import com.dol.Dijkstra;
import com.dol.ui.exceptions.FileNotFoundException;
import com.dol.ui.util.GeoCoordinates;
import com.dol.ui.util.ImagePanelObserver;
```

```
import com.dol.ui.util.Point;
import com.dol.ui.util.Region;

public class ImagePanel extends JPanel
        implements MouseListener, MouseMotionListener {

    private final JFrame frame;
    private final ToolBar toolBar;
    private final JLabel statusBar;

    private final Image europeMap;
    private final Image balticSeaMap;
    private final Image blackSeaMap;
    private final Image EMBaltic;
    private final Image EMBlack;

    private Image currMap;

    private Region region = Region.EUROPE;

    private int widthMap;
    private int heightMap;

    Dijkstra algorithm;
    Point A;
    Point B;

    public ImagePanel(JFrame frame, ToolBar toolBar, JLabel statusBar) throws
FileNotFoundException {
        algorithm = new Dijkstra();
        this.frame = frame;

        toolBar.observer = new ImagePanelObserver(this);
        this.toolBar = toolBar;
        this.statusBar = statusBar;
```

```
addMouseListener(this);
addMouseMotionListener(this);

try {
   europeMap = ImageIO.read(new File("resources" + File.separator + "europe map.jpg"));
   currMap = europeMap;
} catch (IOException e) {
   throw new FileNotFoundException("File \"europe map.jpg\" not found");
}

try {
   A = new Point(ImageIO.read(new File("resources" + File.separator + "point A.png")));
} catch (IOException e) {
   throw new FileNotFoundException("File \"point A.png\" not found");
}

try {
   B = new Point(ImageIO.read(new File("resources" + File.separator + "point B.png")));
} catch (IOException e) {
   throw new FileNotFoundException("File \"point B.png\" not found");
}

try {
   balticSeaMap = ImageIO.read(new File("resources" + File.separator + "baltic sea.jpg"));
} catch (IOException e) {
   throw new FileNotFoundException("File \"baltic sea.jpg\" not found");
}

try {
   blackSeaMap = ImageIO.read(new File("resources" + File.separator + "black sea.jpg"));
} catch (IOException e) {
   throw new FileNotFoundException("File \"black sea.jpg\" not found");
}

try {
   EMBaltic = ImageIO.read(new File("resources" + File.separator + "EM with baltic.jpg"));
```

```
        } catch (IOException e) {
            throw new FileNotFoundException("File \"EM with baltic.jpg\" not found");
        }

        try {
            EMBlack = ImageIO.read(new File("resources" + File.separator + "EM with black.jpg"));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"EM with black.jpg\" not found");
        }
    }

    protected void paintComponent(Graphics g) {
        //выравнивание до пропорций исходного изображения
        widthMap = frame.getWidth() - 17;
        heightMap = frame.getHeight() - 91;
        g.drawImage(currMap, 0, 0, widthMap, heightMap, null);
    }

    private void paintPoint(Point point) {
        if (!point.isWorth()) return;
        getGraphics().drawImage(point.getImagePoint(), point.getX() - 25, point.getY() - 50, 50, 50,
null);
    }

    private void clearMap() {
        paintComponent(getGraphics());
    }

    public Point getPointA() {
        return A;
    }

    public Point getPointB() {
        return B;
    }
```

```
    public Region getRegion() {
        return region;
    }

    public void openSpeedControllerWindow() {
        new SpeedControllerWindow(frame).setVisible(true);
    }
    public void setDefaultRegion() {
        A.deletePoint();
        B.deletePoint();
        region = Region.EUROPE;
        currMap = europeMap;
        paintComponent(getGraphics());
    }

    public void run() {

        if (currMap.equals(europeMap)) {
            JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }
        clearMap();
        paintPoint(A);
        paintPoint(B);
        if (!A.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select a departure point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        if (!B.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select an arrival point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }
```

```
Coordinates from = new Coordinates((int) (A.getLatitude() * 10), (int) (A.getLongitude() *
10));
Coordinates to = new Coordinates((int) (B.getLatitude() * 10), (int) (B.getLongitude() * 10));



switch (region) {
   case BALTIC_SEA :
      algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), true);
      break;

   case BLACK_SEA :
      algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), false);
      break;
}

LinkedList <Coordinates> temp = algorithm.getWayFromCoordinates(to);

if (temp == null) {
   JOptionPane.showMessageDialog(frame, "Невозможно добраться до пункта
назначения!", "Warning!", JOptionPane.WARNING_MESSAGE);
   return;
}

JOptionPane.showMessageDialog(frame, String.format("Distance = %.2f km\nTime = %dh:
%dm", algorithm.getDistance(to), (int) algorithm.getTime(to), (int) ((algorithm.getTime(to) - (int)
algorithm.getTime(to)) * 60)), "Complete!", JOptionPane.INFORMATION_MESSAGE);
Coordinates[] result = new Coordinates[temp.size()];
temp.toArray(result);

for (int i = 1; i < result.length; i++) {

   Coordinates prev = result[i-1];
   Coordinates curr = result[i];

   int prevX = (int) ((prev.getLon() / 10.0 + 25) * widthMap / 75);
```

```
        int prevY = (int) ((71 - prev.getLat() / 10.0) * heightMap / 36);

        int currX = (int) ((curr.getLon() / 10.0 + 25) * widthMap / 75);
        int currY = (int) ((71 - curr.getLat() / 10.0) * heightMap / 36);

        switch (region) {

            case BALTIC_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                prevY = (int) ((66.5 - prev.getLat() / 10.0) * heightMap / 13.5);
                currX = (int) ((curr.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                currY = (int) ((66.5 - curr.getLat() / 10.0) * heightMap / 13.5);
                break;
            }

            case BLACK_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                prevY = (int) ((47.7 - prev.getLat() / 10.0) * heightMap / 8.1) - 6;
                currX = (int) ((curr.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                currY = (int) ((47.7 - curr.getLat() / 10.0) * heightMap / 8.1) - 6;
                break;
            }
        }

        Graphics2D g = (Graphics2D) getGraphics();
        g.setColor(Color.RED);
        g.setStroke(new BasicStroke(2));
        g.drawLine(prevX, prevY, currX, currY);
    }

    paintPoint(A);
    paintPoint(B);
}

@Override
public void mouseClicked(MouseEvent e) {
```

```
        GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

        if (region == Region.EUROPE) {


            if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                    53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

                region = Region.BALTIC_SEA;
                currMap = balticSeaMap;

                toolBar.resetPressedAction();
                paintComponent(getGraphics());

            } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                    39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

                region = Region.BLACK_SEA;
                currMap = blackSeaMap;

                toolBar.resetPressedAction();
                paintComponent(getGraphics());

            } else {
                if (toolBar.getPressedAction() == ToolBar.Action.SET_A || toolBar.getPressedAction()
== ToolBar.Action.SET_B) {
                    JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
                    toolBar.resetPressedAction();
                }
            }

            return;
        }
```

```
    switch (toolBar.getPressedAction()) {

        case NOTHING:
            break;

        case SET_A:
            clearMap();

            A.move(point);
            paintPoint(A);
            paintPoint(B);
            break;

        case SET_B:
            clearMap();

            B.move(point);
            paintPoint(A);
            paintPoint(B);
            break;
    }

    toolBar.resetPressedAction();
}

@Override
public void mousePressed(MouseEvent e) {}

@Override
public void mouseReleased(MouseEvent e) {}

@Override
public void mouseEntered(MouseEvent e) {}

@Override
```

```
    public void mouseExited(MouseEvent e) {}

    @Override
    public void mouseDragged(MouseEvent e) {}

    @Override
    public void mouseMoved(MouseEvent e) {

        GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

        if (region == Region.EUROPE) {
            if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                 53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

                if (!currMap.equals(EMBaltic)) {
                    currMap = EMBaltic;
                    paintComponent(getGraphics());
                }

            } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                       39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

                if (!currMap.equals(EMBlack)) {
                    currMap = EMBlack;
                    paintComponent(getGraphics());
                }

            } else if (!currMap.equals(europeMap)) {
                currMap = europeMap;
                paintComponent(getGraphics());
            }
        }
        statusBar.setText(String.format("(%.2f; %.2f)", point.getLongitude(), point.getLatitude()));
    }
}
```

ToolBar.java

```
package com.dol.ui.elements;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import javax.imageio.ImageIO;

import com.dol.Coordinates;
import com.dol.Dijkstra;
import com.dol.ui.exceptions.FileNotFoundException;
import com.dol.ui.util.GeoCoordinates;
import com.dol.ui.util.ImagePanelObserver;
import com.dol.ui.util.Point;
import com.dol.ui.util.Region;

public class ImagePanel extends JPanel
        implements MouseListener, MouseMotionListener {

    private final JFrame frame;
    private final ToolBar toolBar;
    private final JLabel statusBar;

    private final Image europeMap;
    private final Image balticSeaMap;
    private final Image blackSeaMap;
    private final Image EMBaltic;
    private final Image EMBlack;

    private Image currMap;

    private Region region = Region.EUROPE;
```

```
    private int widthMap;
    private int heightMap;

    Dijkstra algorithm;
    Point A;
    Point B;

    public ImagePanel(JFrame frame, ToolBar toolBar, JLabel statusBar) throws
FileNotFoundException {
        algorithm = new Dijkstra();
        this.frame = frame;

        toolBar.observer = new ImagePanelObserver(this);
        this.toolBar = toolBar;
        this.statusBar = statusBar;

        addMouseListener(this);
        addMouseMotionListener(this);

        try {
            europeMap = ImageIO.read(new File("resources" + File.separator + "europe map.jpg"));
            currMap = europeMap;
        } catch (IOException e) {
            throw new FileNotFoundException("File \"europe map.jpg\" not found");
        }

        try {
            A = new Point(ImageIO.read(new File("resources" + File.separator + "point A.png")));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"point A.png\" not found");
        }

        try {
            B = new Point(ImageIO.read(new File("resources" + File.separator + "point B.png")));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"point B.png\" not found");
```

```
    }

    try {
        balticSeaMap = ImageIO.read(new File("resources" + File.separator + "baltic sea.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"baltic sea.jpg\" not found");
    }

    try {
        blackSeaMap = ImageIO.read(new File("resources" + File.separator + "black sea.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"black sea.jpg\" not found");
    }

    try {
        EMBaltic = ImageIO.read(new File("resources" + File.separator + "EM with baltic.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"EM with baltic.jpg\" not found");
    }

    try {
        EMBlack = ImageIO.read(new File("resources" + File.separator + "EM with black.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"EM with black.jpg\" not found");
    }
}

protected void paintComponent(Graphics g) {
    //выравнивание до пропорций исходного изображения
    widthMap = frame.getWidth() - 17;
    heightMap = frame.getHeight() - 91;
    g.drawImage(currMap, 0, 0, widthMap, heightMap, null);
}

private void paintPoint(Point point) {
    if (!point.isWorth()) return;
```

```
        getGraphics().drawImage(point.getImagePoint(), point.getX() - 25, point.getY() - 50, 50, 50,
null);
    }

    private void clearMap() {
        paintComponent(getGraphics());
    }

    public Point getPointA() {
        return A;
    }

    public Point getPointB() {
        return B;
    }

    public Region getRegion() {
        return region;
    }

    public void openSpeedControllerWindow() {
        new SpeedControllerWindow(frame).setVisible(true);
    }
    public void setDefaultRegion() {
        A.deletePoint();
        B.deletePoint();
        region = Region.EUROPE;
        currMap = europeMap;
        paintComponent(getGraphics());
    }

    public void run() {

        if (currMap.equals(europeMap)) {
            JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
```

```
            return;
        }
        clearMap();
        paintPoint(A);
        paintPoint(B);
        if (!A.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select a departure point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        if (!B.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select an arrival point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        Coordinates from = new Coordinates((int) (A.getLatitude() * 10), (int) (A.getLongitude() *
10));
        Coordinates to = new Coordinates((int) (B.getLatitude() * 10), (int) (B.getLongitude() * 10));



        switch (region) {
            case BALTIC_SEA :
                algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), true);
                break;

            case BLACK_SEA :
                algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), false);
                break;
        }

        LinkedList <Coordinates> temp = algorithm.getWayFromCoordinates(to);

        if (temp == null) {
```

```
        JOptionPane.showMessageDialog(frame, "Невозможно добраться до пункта назначения!", "Warning!", JOptionPane.WARNING_MESSAGE);
        return;
    }

    JOptionPane.showMessageDialog(frame, String.format("Distance = %.2f km\nTime = %dh: %dm", algorithm.getDistance(to), (int) algorithm.getTime(to), (int) ((algorithm.getTime(to) - (int) algorithm.getTime(to)) * 60)), "Complete!", JOptionPane.INFORMATION_MESSAGE);
    Coordinates[] result = new Coordinates[temp.size()];
    temp.toArray(result);

    for (int i = 1; i < result.length; i++) {

        Coordinates prev = result[i-1];
        Coordinates curr = result[i];

        int prevX = (int) ((prev.getLon() / 10.0 + 25) * widthMap / 75);
        int prevY = (int) ((71 - prev.getLat() / 10.0) * heightMap / 36);

        int currX = (int) ((curr.getLon() / 10.0 + 25) * widthMap / 75);
        int currY = (int) ((71 - curr.getLat() / 10.0) * heightMap / 36);

        switch (region) {

            case BALTIC_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                prevY = (int) ((66.5 - prev.getLat() / 10.0) * heightMap / 13.5);
                currX = (int) ((curr.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                currY = (int) ((66.5 - curr.getLat() / 10.0) * heightMap / 13.5);
                break;
            }

            case BLACK_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                prevY = (int) ((47.7 - prev.getLat() / 10.0) * heightMap / 8.1) - 6;
                currX = (int) ((curr.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
```

```
                currY = (int) ((47.7 - curr.getLat() / 10.0) * heightMap / 8.1) - 6;
                break;
            }
        }

        Graphics2D g = (Graphics2D) getGraphics();
        g.setColor(Color.RED);
        g.setStroke(new BasicStroke(2));
        g.drawLine(prevX, prevY, currX, currY);
    }

    paintPoint(A);
    paintPoint(B);
}

@Override
public void mouseClicked(MouseEvent e) {

    GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

    if (region == Region.EUROPE) {


        if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

            region = Region.BALTIC_SEA;
            currMap = balticSeaMap;

            toolBar.resetPressedAction();
            paintComponent(getGraphics());

        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {
```

```
                region = Region.BLACK_SEA;
                currMap = blackSeaMap;

                toolBar.resetPressedAction();
                paintComponent(getGraphics());

            } else {
                if (toolBar.getPressedAction() == ToolBar.Action.SET_A || toolBar.getPressedAction()
== ToolBar.Action.SET_B) {
                    JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
                    toolBar.resetPressedAction();
                }
            }

            return;
        }

        switch (toolBar.getPressedAction()) {

            case NOTHING:
                break;

            case SET_A:
                clearMap();

                A.move(point);
                paintPoint(A);
                paintPoint(B);
                break;

            case SET_B:
                clearMap();

                B.move(point);
                paintPoint(A);
```

```
                paintPoint(B);
                break;
            }

            toolBar.resetPressedAction();
        }

        @Override
        public void mousePressed(MouseEvent e) {}

        @Override
        public void mouseReleased(MouseEvent e) {}

        @Override
        public void mouseEntered(MouseEvent e) {}

        @Override
        public void mouseExited(MouseEvent e) {}

        @Override
        public void mouseDragged(MouseEvent e) {}

        @Override
        public void mouseMoved(MouseEvent e) {

            GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

            if (region == Region.EUROPE) {
                if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                    53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

                    if (!currMap.equals(EMBaltic)) {
                        currMap = EMBaltic;
                        paintComponent(getGraphics());
                    }
```

```
        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

            if (!currMap.equals(EMBlack)) {
                currMap = EMBlack;
                paintComponent(getGraphics());
            }

        } else if (!currMap.equals(europeMap)) {
            currMap = europeMap;
            paintComponent(getGraphics());
        }
      }
      statusBar.setText(String.format("(%.2f; %.2f)", point.getLongitude(), point.getLatitude()));
   }
}
```

FileNotFoundException.java

```
package com.dol.ui.elements;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import javax.imageio.ImageIO;

import com.dol.Coordinates;
import com.dol.Dijkstra;
import com.dol.ui.exceptions.FileNotFoundException;
import com.dol.ui.util.GeoCoordinates;
import com.dol.ui.util.ImagePanelObserver;
import com.dol.ui.util.Point;
import com.dol.ui.util.Region;
```

```
public class ImagePanel extends JPanel
        implements MouseListener, MouseMotionListener {

    private final JFrame frame;
    private final ToolBar toolBar;
    private final JLabel statusBar;

    private final Image europeMap;
    private final Image balticSeaMap;
    private final Image blackSeaMap;
    private final Image EMBaltic;
    private final Image EMBlack;

    private Image currMap;

    private Region region = Region.EUROPE;

    private int widthMap;
    private int heightMap;

    Dijkstra algorithm;
    Point A;
    Point B;

    public ImagePanel(JFrame frame, ToolBar toolBar, JLabel statusBar) throws
FileNotFoundException {
        algorithm = new Dijkstra();
        this.frame = frame;

        toolBar.observer = new ImagePanelObserver(this);
        this.toolBar = toolBar;
        this.statusBar = statusBar;

        addMouseListener(this);
        addMouseMotionListener(this);
```

```
try {
    europeMap = ImageIO.read(new File("resources" + File.separator + "europe map.jpg"));
    currMap = europeMap;
} catch (IOException e) {
    throw new FileNotFoundException("File \"europe map.jpg\" not found");
}

try {
    A = new Point(ImageIO.read(new File("resources" + File.separator + "point A.png")));
} catch (IOException e) {
    throw new FileNotFoundException("File \"point A.png\" not found");
}

try {
    B = new Point(ImageIO.read(new File("resources" + File.separator + "point B.png")));
} catch (IOException e) {
    throw new FileNotFoundException("File \"point B.png\" not found");
}

try {
    balticSeaMap = ImageIO.read(new File("resources" + File.separator + "baltic sea.jpg"));
} catch (IOException e) {
    throw new FileNotFoundException("File \"baltic sea.jpg\" not found");
}

try {
    blackSeaMap = ImageIO.read(new File("resources" + File.separator + "black sea.jpg"));
} catch (IOException e) {
    throw new FileNotFoundException("File \"black sea.jpg\" not found");
}

try {
    EMBaltic = ImageIO.read(new File("resources" + File.separator + "EM with baltic.jpg"));
} catch (IOException e) {
    throw new FileNotFoundException("File \"EM with baltic.jpg\" not found");
```

```
        }

        try {
            EMBlack = ImageIO.read(new File("resources" + File.separator + "EM with black.jpg"));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"EM with black.jpg\" not found");
        }
    }

    protected void paintComponent(Graphics g) {
        //выравнивание до пропорций исходного изображения
        widthMap = frame.getWidth() - 17;
        heightMap = frame.getHeight() - 91;
        g.drawImage(currMap, 0, 0, widthMap, heightMap, null);
    }

    private void paintPoint(Point point) {
        if (!point.isWorth()) return;
        getGraphics().drawImage(point.getImagePoint(), point.getX() - 25, point.getY() - 50, 50, 50,
null);
    }

    private void clearMap() {
        paintComponent(getGraphics());
    }

    public Point getPointA() {
        return A;
    }

    public Point getPointB() {
        return B;
    }

    public Region getRegion() {
        return region;
```

```
    }

    public void openSpeedControllerWindow() {
        new SpeedControllerWindow(frame).setVisible(true);
    }
    public void setDefaultRegion() {
        A.deletePoint();
        B.deletePoint();
        region = Region.EUROPE;
        currMap = europeMap;
        paintComponent(getGraphics());
    }

    public void run() {

        if (currMap.equals(europeMap)) {
            JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }
        clearMap();
        paintPoint(A);
        paintPoint(B);
        if (!A.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select a departure point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        if (!B.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select an arrival point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }
```

```
Coordinates from = new Coordinates((int) (A.getLatitude() * 10), (int) (A.getLongitude() * 10));
Coordinates to = new Coordinates((int) (B.getLatitude() * 10), (int) (B.getLongitude() * 10));



switch (region) {
   case BALTIC_SEA :
      algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), true);
      break;

   case BLACK_SEA :
      algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), false);
      break;
}

LinkedList <Coordinates> temp = algorithm.getWayFromCoordinates(to);

if (temp == null) {
   JOptionPane.showMessageDialog(frame, "Невозможно добраться до пункта назначения!", "Warning!", JOptionPane.WARNING_MESSAGE);
   return;
}

JOptionPane.showMessageDialog(frame, String.format("Distance = %.2f km\nTime = %dh: %dm", algorithm.getDistance(to), (int) algorithm.getTime(to), (int) ((algorithm.getTime(to) - (int) algorithm.getTime(to)) * 60)), "Complete!", JOptionPane.INFORMATION_MESSAGE);
Coordinates[] result = new Coordinates[temp.size()];
temp.toArray(result);

for (int i = 1; i < result.length; i++) {

   Coordinates prev = result[i-1];
   Coordinates curr = result[i];

   int prevX = (int) ((prev.getLon() / 10.0 + 25) * widthMap / 75);
   int prevY = (int) ((71 - prev.getLat() / 10.0) * heightMap / 36);
```

```
        int currX = (int) ((curr.getLon() / 10.0 + 25) * widthMap / 75);
        int currY = (int) ((71 - curr.getLat() / 10.0) * heightMap / 36);

        switch (region) {

            case BALTIC_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                prevY = (int) ((66.5 - prev.getLat() / 10.0) * heightMap / 13.5);
                currX = (int) ((curr.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                currY = (int) ((66.5 - curr.getLat() / 10.0) * heightMap / 13.5);
                break;
            }

            case BLACK_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                prevY = (int) ((47.7 - prev.getLat() / 10.0) * heightMap / 8.1) - 6;
                currX = (int) ((curr.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                currY = (int) ((47.7 - curr.getLat() / 10.0) * heightMap / 8.1) - 6;
                break;
            }
        }

        Graphics2D g = (Graphics2D) getGraphics();
        g.setColor(Color.RED);
        g.setStroke(new BasicStroke(2));
        g.drawLine(prevX, prevY, currX, currY);
    }

    paintPoint(A);
    paintPoint(B);
}

@Override
public void mouseClicked(MouseEvent e) {
```

```
GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

    if (region == Region.EUROPE) {


        if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

            region = Region.BALTIC_SEA;
            currMap = balticSeaMap;

            toolBar.resetPressedAction();
            paintComponent(getGraphics());

        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

            region = Region.BLACK_SEA;
            currMap = blackSeaMap;

            toolBar.resetPressedAction();
            paintComponent(getGraphics());

        } else {
            if (toolBar.getPressedAction() == ToolBar.Action.SET_A || toolBar.getPressedAction()
== ToolBar.Action.SET_B) {
                JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
                toolBar.resetPressedAction();
            }
        }

        return;
    }
```

```
    switch (toolBar.getPressedAction()) {

        case NOTHING:
            break;

        case SET_A:
            clearMap();

            A.move(point);
            paintPoint(A);
            paintPoint(B);
            break;

        case SET_B:
            clearMap();

            B.move(point);
            paintPoint(A);
            paintPoint(B);
            break;
    }

    toolBar.resetPressedAction();
}

@Override
public void mousePressed(MouseEvent e) {}

@Override
public void mouseReleased(MouseEvent e) {}

@Override
public void mouseEntered(MouseEvent e) {}

@Override
public void mouseExited(MouseEvent e) {}
```

```
    @Override
    public void mouseDragged(MouseEvent e) {}

    @Override
    public void mouseMoved(MouseEvent e) {

        GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

        if (region == Region.EUROPE) {
            if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                 53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

                if (!currMap.equals(EMBaltic)) {
                    currMap = EMBaltic;
                    paintComponent(getGraphics());
                }

            } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                      39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

                if (!currMap.equals(EMBlack)) {
                    currMap = EMBlack;
                    paintComponent(getGraphics());
                }

            } else if (!currMap.equals(europeMap)) {
                currMap = europeMap;
                paintComponent(getGraphics());
            }
        }
        statusBar.setText(String.format("(%.2f; %.2f)", point.getLongitude(), point.getLatitude()));
    }
}
```

GeoCoordinates.java

```
package com.dol.ui.elements;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import javax.imageio.ImageIO;

import com.dol.Coordinates;
import com.dol.Dijkstra;
import com.dol.ui.exceptions.FileNotFoundException;
import com.dol.ui.util.GeoCoordinates;
import com.dol.ui.util.ImagePanelObserver;
import com.dol.ui.util.Point;
import com.dol.ui.util.Region;

public class ImagePanel extends JPanel
        implements MouseListener, MouseMotionListener {

    private final JFrame frame;
    private final ToolBar toolBar;
    private final JLabel statusBar;

    private final Image europeMap;
    private final Image balticSeaMap;
    private final Image blackSeaMap;
    private final Image EMBaltic;
    private final Image EMBlack;

    private Image currMap;

    private Region region = Region.EUROPE;
```

```
    private int widthMap;
    private int heightMap;

    Dijkstra algorithm;
    Point A;
    Point B;

    public ImagePanel(JFrame frame, ToolBar toolBar, JLabel statusBar) throws
FileNotFoundException {
        algorithm = new Dijkstra();
        this.frame = frame;

        toolBar.observer = new ImagePanelObserver(this);
        this.toolBar = toolBar;
        this.statusBar = statusBar;

        addMouseListener(this);
        addMouseMotionListener(this);

        try {
            europeMap = ImageIO.read(new File("resources" + File.separator + "europe map.jpg"));
            currMap = europeMap;
        } catch (IOException e) {
            throw new FileNotFoundException("File \"europe map.jpg\" not found");
        }

        try {
            A = new Point(ImageIO.read(new File("resources" + File.separator + "point A.png")));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"point A.png\" not found");
        }

        try {
            B = new Point(ImageIO.read(new File("resources" + File.separator + "point B.png")));
        } catch (IOException e) {
            throw new FileNotFoundException("File \"point B.png\" not found");
```

```
    }

    try {
        balticSeaMap = ImageIO.read(new File("resources" + File.separator + "baltic sea.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"baltic sea.jpg\" not found");
    }

    try {
        blackSeaMap = ImageIO.read(new File("resources" + File.separator + "black sea.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"black sea.jpg\" not found");
    }

    try {
        EMBaltic = ImageIO.read(new File("resources" + File.separator + "EM with baltic.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"EM with baltic.jpg\" not found");
    }

    try {
        EMBlack = ImageIO.read(new File("resources" + File.separator + "EM with black.jpg"));
    } catch (IOException e) {
        throw new FileNotFoundException("File \"EM with black.jpg\" not found");
    }
}

protected void paintComponent(Graphics g) {
    //выравнивание до пропорций исходного изображения
    widthMap = frame.getWidth() - 17;
    heightMap = frame.getHeight() - 91;
    g.drawImage(currMap, 0, 0, widthMap, heightMap, null);
}

private void paintPoint(Point point) {
    if (!point.isWorth()) return;
```

```
        getGraphics().drawImage(point.getImagePoint(), point.getX() - 25, point.getY() - 50, 50, 50,
null);
    }

    private void clearMap() {
        paintComponent(getGraphics());
    }

    public Point getPointA() {
        return A;
    }

    public Point getPointB() {
        return B;
    }

    public Region getRegion() {
        return region;
    }

    public void openSpeedControllerWindow() {
        new SpeedControllerWindow(frame).setVisible(true);
    }
    public void setDefaultRegion() {
        A.deletePoint();
        B.deletePoint();
        region = Region.EUROPE;
        currMap = europeMap;
        paintComponent(getGraphics());
    }

    public void run() {

        if (currMap.equals(europeMap)) {
            JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
```

```
            return;
        }
        clearMap();
        paintPoint(A);
        paintPoint(B);
        if (!A.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select a departure point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        if (!B.isWorth()) {
            JOptionPane.showMessageDialog(frame, "Please select an arrival point!", "Warning!",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        Coordinates from = new Coordinates((int) (A.getLatitude() * 10), (int) (A.getLongitude() *
10));
        Coordinates to = new Coordinates((int) (B.getLatitude() * 10), (int) (B.getLongitude() * 10));



        switch (region) {
            case BALTIC_SEA :
                algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), true);
                break;

            case BLACK_SEA :
                algorithm.dijkstra(from, to, SpeedControllerWindow.getCurrentSpeed(), false);
                break;
        }

        LinkedList <Coordinates> temp = algorithm.getWayFromCoordinates(to);

        if (temp == null) {
```

```
        JOptionPane.showMessageDialog(frame, "Невозможно добраться до пункта назначения!", "Warning!", JOptionPane.WARNING_MESSAGE);
        return;
    }

    JOptionPane.showMessageDialog(frame, String.format("Distance = %.2f km\nTime = %dh: %dm", algorithm.getDistance(to), (int) algorithm.getTime(to), (int) ((algorithm.getTime(to) - (int) algorithm.getTime(to)) * 60)), "Complete!", JOptionPane.INFORMATION_MESSAGE);
    Coordinates[] result = new Coordinates[temp.size()];
    temp.toArray(result);

    for (int i = 1; i < result.length; i++) {

        Coordinates prev = result[i-1];
        Coordinates curr = result[i];

        int prevX = (int) ((prev.getLon() / 10.0 + 25) * widthMap / 75);
        int prevY = (int) ((71 - prev.getLat() / 10.0) * heightMap / 36);

        int currX = (int) ((curr.getLon() / 10.0 + 25) * widthMap / 75);
        int currY = (int) ((71 - curr.getLat() / 10.0) * heightMap / 36);

        switch (region) {

            case BALTIC_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                prevY = (int) ((66.5 - prev.getLat() / 10.0) * heightMap / 13.5);
                currX = (int) ((curr.getLon() / 10.0 - 5.30) * widthMap / 27.20);
                currY = (int) ((66.5 - curr.getLat() / 10.0) * heightMap / 13.5);
                break;
            }

            case BLACK_SEA : {
                prevX = (int) ((prev.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
                prevY = (int) ((47.7 - prev.getLat() / 10.0) * heightMap / 8.1) - 6;
                currX = (int) ((curr.getLon() / 10.0 - 25.3) * widthMap / 17.9) + 6;
```

```
                currY = (int) ((47.7 - curr.getLat() / 10.0) * heightMap / 8.1) - 6;
                break;
            }
        }

        Graphics2D g = (Graphics2D) getGraphics();
        g.setColor(Color.RED);
        g.setStroke(new BasicStroke(2));
        g.drawLine(prevX, prevY, currX, currY);
    }

    paintPoint(A);
    paintPoint(B);
}

@Override
public void mouseClicked(MouseEvent e) {

    GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

    if (region == Region.EUROPE) {


        if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

            region = Region.BALTIC_SEA;
            currMap = balticSeaMap;

            toolBar.resetPressedAction();
            paintComponent(getGraphics());

        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {
```

```
                region = Region.BLACK_SEA;
                currMap = blackSeaMap;

                toolBar.resetPressedAction();
                paintComponent(getGraphics());

            } else {
                if (toolBar.getPressedAction() == ToolBar.Action.SET_A || toolBar.getPressedAction()
== ToolBar.Action.SET_B) {
                    JOptionPane.showMessageDialog(frame, "Please select a region!", "Warning!",
JOptionPane.WARNING_MESSAGE);
                    toolBar.resetPressedAction();
                }
            }

            return;
        }

        switch (toolBar.getPressedAction()) {

            case NOTHING:
                break;

            case SET_A:
                clearMap();

                A.move(point);
                paintPoint(A);
                paintPoint(B);
                break;

            case SET_B:
                clearMap();

                B.move(point);
                paintPoint(A);
```

```
                paintPoint(B);
                break;
            }

            toolBar.resetPressedAction();
        }

        @Override
        public void mousePressed(MouseEvent e) {}

        @Override
        public void mouseReleased(MouseEvent e) {}

        @Override
        public void mouseEntered(MouseEvent e) {}

        @Override
        public void mouseExited(MouseEvent e) {}

        @Override
        public void mouseDragged(MouseEvent e) {}

        @Override
        public void mouseMoved(MouseEvent e) {

            GeoCoordinates point = new GeoCoordinates(e.getX(), e.getY(), widthMap, heightMap,
region);

            if (region == Region.EUROPE) {
                if (5.3 < point.getLongitude() && point.getLongitude() < 32.5 &&
                    53 < point.getLatitude() && point.getLatitude() < 66.5 ) {

                    if (!currMap.equals(EMBaltic)) {
                        currMap = EMBaltic;
                        paintComponent(getGraphics());
                    }
```

```
        } else if (25.3 < point.getLongitude() && point.getLongitude() < 43.2 &&
                39.6 < point.getLatitude() && point.getLatitude() < 47.7 ) {

            if (!currMap.equals(EMBlack)) {
                currMap = EMBlack;
                paintComponent(getGraphics());
            }

        } else if (!currMap.equals(europeMap)) {
            currMap = europeMap;
            paintComponent(getGraphics());
        }
    }
    statusBar.setText(String.format("(%.2f; %.2f)", point.getLongitude(), point.getLatitude()));
  }
}
```

ImagePanelObserver.java

```
package com.dol.ui.util;

import com.dol.ui.elements.ImagePanel;

public class ImagePanelObserver {

  private final ImagePanel panel;

  public ImagePanelObserver(ImagePanel panel) {
    this.panel = panel;
  }

  public Point getPointA() {
    return panel.getPointA();
  }

  public Point getPointB() {
```

```
        return panel.getPointB();
    }

    public Region getRegion() {
        return panel.getRegion();
    }

    public void run() {
        panel.run();
    }

    public void setDefaultRegion() {
        panel.setDefaultRegion();
    }

    public void openSpeedControllerWindow() {
        panel.openSpeedControllerWindow();
    }
}
```

Point.java

```
package com.dol.ui.util;

import java.awt.*;

public class Point {

    public GeoCoordinates coordinates;
    private final Image imagePoint;

    public Point(Image image) {
        imagePoint = image;
        coordinates = null;
    }

    public Image getImagePoint() {
```

```
        return imagePoint;
    }

    public boolean isWorth() {
        return coordinates != null;
    }

    public void deletePoint() {
        coordinates = null;
    }

    public void move(GeoCoordinates coordinates) {
        this.coordinates = coordinates;
    }

    public void move(int x, int y, int width, int height, Region region) {
        coordinates = new GeoCoordinates(x, y, width, height, region);
    }

    public int getX() {
        return coordinates.getX();
    }

    public int getY() {
        return coordinates.getY();
    }
    public double getLongitude() {
        return coordinates.getLongitude();
    }
    public double getLatitude() {
        return coordinates.getLatitude();
    }
}
```

Region.java

```
package com.dol.ui.util;

public enum Region {
   EUROPE,
   BALTIC_SEA,
   BLACK_SEA
}
```

Application.java

```
package com.dol.ui;

import com.dol.ui.elements.ImagePanel;
import com.dol.ui.elements.ToolBar;
import com.dol.ui.exceptions.FileNotFoundException;

import javax.swing.*;
import javax.swing.border.BevelBorder;
import java.awt.*;
import java.io.File;


public class Application extends JFrame {

   final int width = 1280;
   final int height = 900;

   Application() {

      super("DOL Balloon");
      setSize(width, height);
      setIconImage(new ImageIcon("resources" + File.separator + "icon.png").getImage());

      JPanel mainPanel = new JPanel(new BorderLayout());
```

```
        ImagePanel mapPanel;
        ToolBar toolBar;
        JPanel statusBar = new JPanel();

        statusBar.setBorder(new BevelBorder(BevelBorder.LOWERED));
        statusBar.setPreferredSize(new Dimension(width, 16));
        statusBar.setLayout(new BoxLayout(statusBar, BoxLayout.X_AXIS));
        JLabel statusLabel = new JLabel();
        statusLabel.setHorizontalAlignment(SwingConstants.LEFT);
        statusBar.add(statusLabel);

        try {
            toolBar = new ToolBar();
            mapPanel = new ImagePanel(this, toolBar, statusLabel);
        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error!",
JOptionPane.ERROR_MESSAGE);
            System.exit(1);
            return;
        }

        rootPane.setContentPane(mainPanel);
        add(toolBar, BorderLayout.NORTH);
        add(mapPanel, BorderLayout.CENTER);
        add(statusBar, BorderLayout.SOUTH);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setVisible(true);
    }

    public static void main(String[] args) {

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
```

```
            e.printStackTrace();
            System.exit(1);
        }

        new Application();
    }
}
```

Coordinates.java

```
package com.dol;

import java.util.Objects;

public class Coordinates {
    private final int lat;
    private final int lon;

    public Coordinates(int lat, int lon) {
        this.lat = lat;
        this.lon = lon;
    }

    public int getLat() { return lat; }
    public int getLon() { return lon; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Coordinates that = (Coordinates) o;
        return lat == that.lat &&
                lon == that.lon;
    }

    @Override
```

```
    public int hashCode() {
        return Objects.hash(lat, lon);
    }
}
```

Dijkstra.java

```
package com.dol;

import java.io.IOException;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.text.MessageFormat;
import java.util.*;
import org.apache.commons.io.*;
import org.json.JSONObject;

public class Dijkstra {
    Set<Coordinates> A; // мно-во не посещенных вершин
    Set<Coordinates> B; // мно-во посещенных вершин
    Map<Coordinates, Wind> windMap;
    int speed;

    Map<Coordinates, Double> minWay; // словарь кратчайщих путей
    Map<Coordinates, Double> minTime; // словарь минимальных времен
    Map<Coordinates, Coordinates> parentCord;

    LinkedList<Coordinates> list; // Тут хранится список координат, по которым будет лететь шар до цели

    PriorityQueue<Way> priorityQueue;

    boolean region;

    public Dijkstra() {
        windMap = new HashMap<>();
```

```
}

private void initRegion() {
    if (region) { // Baltic sea
        for (int lat = 530; lat <= 665; lat++) { // вот тут задается карта la - latitude(широта)
            for (int lon = 53; lon <= 325; lon++) { // lo - longitude(долгота)
                Coordinates a = new Coordinates(lat, lon);
                A.add(a); // Множества не посещенных вершин
                minWay.put(a, Double.MAX_VALUE); // Минимальное расстояние до координат
                minTime.put(a, Double.MAX_VALUE);
                parentCord.put(a, null);
            }
        }
    } else { // Black sea
        for (int lat = 396; lat <= 477; lat++) { // вот тут задается карта la - latitude(широта)
            for (int lon = 253; lon <= 432; lon++) { // lo - longitude(долгота)
                Coordinates a = new Coordinates(lat, lon);
                A.add(a); // Множества не посещенных вершин
                minWay.put(a, Double.MAX_VALUE); // Минимальное расстояние до координат
                minTime.put(a, Double.MAX_VALUE);
                parentCord.put(a, null);
            }
        }
    }
}

public void dijkstra(Coordinates from, Coordinates to, int speed, boolean region){
    A = new HashSet<>();
    B = new HashSet<>();
    minWay = new HashMap<>();
    minTime = new HashMap<>();
    parentCord = new HashMap<>();
    priorityQueue = new PriorityQueue<>();
    this.speed = speed;
    this.region = region;
```

```
        initRegion();
        addCoordinates(from, 0.0, 0); // Начальная точка

        while (!priorityQueue.isEmpty()) { // пока есть непосещенные вершины продолжаем крутиться
            Way a = priorityQueue.poll();
            if (B.contains(a.getCoordinates())) // Если вершина, которая вышла из очереди уже была рассмотренна в множество B, скипаем ее
                continue;
            addCoordinates(a.getCoordinates(), a.getDistance(), a.getTime());
        }
    }

    private void addCoordinates(Coordinates s, double distance, double time) {
        int currLat = s.getLat();
        int currLon = s.getLon();

        int direction = -1;
        double [] wind = setTimeOfFlight(s);


        minWay.replace(s, distance); // Фиксируем найденное минимальное расстояние до точки
        minTime.replace(s, time);
        A.remove(s); // Удаляем вершину из не рассмотренных
        B.add(s); // Помечаем как рассмотенную

        for(int lat = -1; lat <= 1; lat++) {
            for(int lon = -1; lon <= 1; lon++) {
                if (!(lat == 0 && lon == 0)) {
                    direction++;
                }
                Coordinates newS = adjCoordinates(currLat + lat, currLon + lon); // Проверка на то, что смежная клетка не вышла за границы области

                if (newS == null || B.contains(newS)) { // если вершина не вышла за границы и уже не была рассмотрена(B.contains(newS)), то продолжаем с ней танцевать
```

```
                continue;
            }

            if (wind[direction] <= 0) {
                continue;
            }

            double dist = distanceFormula(s, newS);
            double oldTime = minTime.get(newS);
            double newTime = time + dist/wind[direction];
           // double newTime = time + 5;
            if ((oldTime > newTime)) {
                double newDistance = distance + dist;

                minWay.replace(newS, newDistance);
                minTime.replace(newS, newTime);
                parentCord.replace(newS, s);

                priorityQueue.add(new Way(newS, newDistance, newTime));
            }
        }
    }
}

private Coordinates adjCoordinates(int lat, int lon) {
    if (region) {
        if (lat >= 530 && lat <= 665 && 53 <= lon && lon <= 325) {
            return new Coordinates(lat, lon);
        }
    } else {
        if (lat >= 396 && lat <= 477 && 253 <= lon && lon <= 432) {
            return new Coordinates(lat, lon);
        }
    }
    return null;
}
```

```
    protected double[] setTimeOfFlight(Coordinates c) {
        Coordinates coordinates = new Coordinates(c.getLat()/10, c.getLon()/10);
        Wind wind;

        double[] resultingSpeed = new double[8];

        if ((wind = windMap.get(coordinates)) == null) {
            String urlS = MessageFormat.format("http://api.worldweatheronline.com/premium/v1/
weather.ashx?key=716224505d374e6e98f142540201207&q={0},
{1}&mca=no&tp=1&num_of_days=1&format=json",
                    coordinates.getLat(), coordinates.getLon());
            try {
                JSONObject json = new JSONObject(IOUtils.toString(new URL(urlS),
StandardCharsets.UTF_8));
                JSONObject weatherOnTime =
json.getJSONObject("data").getJSONArray("current_condition").getJSONObject(0);
                int windDegree = (weatherOnTime.getInt("winddirDegree") + 180) % 360;
                int windSpeed = weatherOnTime.getInt("windspeedKmph");
                windDegree = 90 - windDegree % 90 - 90 * (windDegree / 90);
                wind = new Wind(windSpeed, windDegree);
            } catch (IOException e) {
                e.printStackTrace();
            }
            windMap.put(coordinates, wind);
        }

        wind.degree = 90 - wind.degree % 90 - 90 * (wind.degree / 90);
        for (int i = 0; i < 8; i++) {
            double windProjection = wind.speed*Math.cos(Math.toRadians(wind.degree + 45 * i));
            double engineProjection = Math.sqrt(Math.pow(speed, 2) - Math.pow(windProjection, 2));
            resultingSpeed[i] = (int)
(engineProjection+wind.speed*Math.sin(Math.toRadians(wind.degree + 45 * i)));
//            System.out.println(resultingSpeed[i]);
        }
        return resultingSpeed;
```

```
    }

    // НЕ СМОТРИ СЮДА, только если решил развлечься с точностью
    /*
    широта и долгота задается через целые числа, потому что для них куда проще считать хэши и нет коллизий с точностью
    у меня точность до 0.1, так что все координаты выгляд как-то так: 85.5 == 855
     */

    private double distanceFormula(Coordinates c1, Coordinates c2) {
        double fi1 = ((double)c1.getLat())/10.0;
        fi1 = Math.toRadians(fi1);

        double fi2 = ((double)c2.getLat())/10.0;
        fi2 = Math.toRadians(fi2);

        double ly1 = ((double)c1.getLon())/10.0;
        ly1 = Math.toRadians(ly1);

        double ly2 = ((double)c2.getLon()/10.0);
        ly2 = Math.toRadians(ly2);
        // если решишь, что тебе зачем-то нужно расстояние в м, то 6363.564 замени на 6363564.0
        return 2 * 6363.564 * Math.asin(Math.sqrt(haverSinus(fi2 - fi1) + Math.cos(fi1) * Math.cos(fi2) * haverSinus(ly2 - ly1)));
    }

    private double haverSinus(double x) {
        return Math.pow(Math.sin(x/2.0),2);
    }

    public LinkedList<Coordinates> getWayFromCoordinates(Coordinates b) {
        list = new LinkedList<>();

        Coordinates c = b;
        list.addFirst(c);
```

```
        while ((c = parentCord.get(c)) != null) {
            list.addFirst(c);
        }

        if (list.size() == 1)
            return null;

        return list;
    }

    public double getTime(Coordinates b) {
        return minTime.get(b);
    }

    public double getDistance(Coordinates b) {
        return minWay.get(b);
    }
}
```

Way.java

```
package com.dol;

import java.util.Objects;

public class Way implements Comparable<Way> {
    private final Coordinates c;
    private final double distance;
    private final double time;

    public Way(Coordinates c, double distance, double time) {
        this.c = c;
        this.distance = distance;
        this.time = time;
    }
```

```
    public double getDistance() { return distance; }

    public Coordinates getCoordinates() { return this.c; }

    public double getTime() { return this.time; }


    @Override
    public int compareTo(Way way) {
        if (time < way.time) {
            return -1;
        } else if (time > way.time) {
            return 1;
        }
        return 0;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Way way = (Way) o;
        return Double.compare(way.distance, distance) == 0 &&
                Double.compare(way.time, time) == 0 &&
                Objects.equals(c, way.c);
    }

    @Override
    public int hashCode() {
        return Objects.hash(c, distance, time);
    }
}
```

Wind.java

```
package com.dol;

public class Wind {
    int speed;
    int degree;

    public Wind(int speed, int degree) {
        this.speed = speed;
        this.degree = degree;
    }
}
```

ПРИЛОЖЕНИЕ В

UNIT-ТЕСТИРОВАНИЕ

DijkstraTest.java

```
package com.dol;

import org.junit.Assert;
import org.junit.Test;

import java.util.HashMap;
import java.util.LinkedList;

public class DijkstraTest {

    @Test
    public void getWayFromCoordinates() {
        Dijkstra dijkstraVert = new Dijkstra();
        Dijkstra dijkstraHor = new Dijkstra();
        dijkstraVert.parentCord = new HashMap<>();
        dijkstraHor.parentCord = new HashMap<>();
        LinkedList<Coordinates> listVert = new LinkedList<>();
        LinkedList<Coordinates> listHor = new LinkedList<>();
        Coordinates start = new Coordinates(455, 242);
        listVert.add(start);
        listHor.add(start);
        for (int i = 0; i < 10; i++) {
            Coordinates coordVert1 = new Coordinates(456+i, 242);
            Coordinates coordVert2 = new Coordinates(455+i, 242);
            Coordinates coordHor1 = new Coordinates(455, 243+i);
            Coordinates coordHor2 = new Coordinates(455, 242+i);
            dijkstraVert.parentCord.put(coordVert1, coordVert2);
            dijkstraHor.parentCord.put(coordHor1, coordHor2);
            listVert.add(coordVert1);
            listHor.add(coordHor1);
        }
```

```
        Assert.assertEquals(listVert, dijkstraVert.getWayFromCoordinates(new
Coordinates(465,242)));
        Assert.assertEquals(listHor, dijkstraHor.getWayFromCoordinates(new Coordinates(455,252)));
    }


    @Test
    public void setTimeOfFlight() {
        Dijkstra dijkstra = new Dijkstra();
        dijkstra.windMap = new HashMap<>();
        dijkstra.speed = 10;
        Coordinates coordinates = new Coordinates(420, 270);
        Wind wind = new Wind(15, 180);
        dijkstra.windMap.put(coordinates, wind);
        double[] expected = {0, -11, -7, 0, 0, 16, 26, 0};
        for (int i = 0; i < 8; i++) {
            Assert.assertEquals(expected[i], dijkstra.setTimeOfFlight(new Coordinates(425, 277))[i],
1e-6);
        }
    }
}
```

ПРИЛОЖЕНИЕ С

UML-ДИАГРАММА

https://github.com/mensch25/airBalloon_DOL/blob/LaFinale/DOL_UML.png