**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: **«Расчёт полёта воздушного шара»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7304 |  | Петруненко Д.А. |
| Студент гр. 7304 |  | Есиков О.И. |
| Студент гр. 7303 |  | Овчинников С.М. |
| Руководитель |  | Фирсов М. А. |

Санкт-Петербург

2019

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Петруненко Д.А. группы 7304 | | |
| Студент Есиков О.И. группы 7304 | | |
| Студент Овчинников С.М. группы 7303  Тема практики: «Расчёт полёта воздушного шара» | | |
| Задание на практику:  Пользователь выбирает из предоставленного списка один из регионов мира. Появляется Карта данного региона на которой можно выбрать отправной пункт для воздушного шара (например, крупный город). Помимо точки пользователь также может задать время и дату полёта (в прошлом). Программа рассчитывает путь и визуализирует его на карте. Существует возможность задать временной промежуток полёта шара. Данные о силе и направлении ветра должны соответствовать временным промежуткам полёта шара. Для получения данных о ветре используется web-сервис, предоставляющий api для получения данных о погоде.  Алгоритм: Алгоритм перемещения шара по карте?. | | |
| Сроки прохождения практики: 01.07.2019 – 14.07.2019 | | |
| Дата сдачи отчета: 12.07.2019 | | |
| Дата защиты отчета: 12.07.2019 | | |
|  | | |
| Студент |  | Петруненко Д.А. |
| Студент |  | Есиков О.И. |
| Студент |  | Овчинников С.М. |
| Руководитель |  | Фирсов М. А. |

**Аннотация**

Целью работы является получения навыков работы с такой парадигмой программирования, как объектно-ориентированное программирование. Для получения данных знаний выполняется один из вариантов мини-проекта. В процессе выполнения мини-проекта необходимо реализовать графический интерфейс к данной задаче, организовать ввод и вывод данных с его помощь, реализовать сам алгоритм, научиться работать в команде. В данной работе в качестве мини-проекта выступает расчёт полёта воздушного шара, с получением данных о характере его перемещения с web-сервиса, который предоставляет api для получения данных о погоде. Также при разработке выполняется написание тестирования, для проверки корректности алгоритма.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Требования к программе | 6 |
| 1.1. | Исходные требования к программе\* | 0 |
| 1.2. | Уточнение требований после сдачи прототипа | 0 |
| 1.3.  1.4 | Уточнение требований после сдачи 1-ой версии  Уточнение требований после сдачи 2-ой версии | 0  0 |
| 2. | План разработки и распределение ролей в бригаде | 0 |
| 2.1. | План разработки | 0 |
| 2.2. | Распределение ролей в бригаде | 0 |
| 3. | Особенности реализации | 0 |
| 3.1. | Использованные структуры данных | 0 |
| 3.2. | Основные методы | 0 |
| 3.3 |  | 0 |
| 4. | Тестирование | 0 |
| 4.1 | Тестирование графического интерфейса | 0 |
| 4.2 | Тестирование кода алгоритма | 0 |
| 4.3 | … | 0 |
|  | Заключение | 0 |
|  | Список использованных источников | 0 |
|  | Приложение А. Исходный код – только в электронном виде | 0 |

**введение**

Основная цель практики – реализация мини-проекта, который является визуализацией алгоритма. В данной работе такой алгоритм – это модель перемещения воздушного шара, направление движения которого зависит только от ветра. Для выполнения этой цели были поставлены задачи: разработка GUI к проекту, отправление запроса на сервер, вычисление новых координат шара. Проект включает в себя 2 алгоритма, из которых выбирает 1 пользователь и получает результат его работы.

Первый алгоритм – перемещение шара в течение выбранного промежутка времени. Для выбранной точки для выбранного дня отправляется запрос на сервер о погоде, после чего в течение выбранного промежутка времени осуществляется движение, после чего происходит новый запрос на сервер. Результатом этого алгоритма является путь шара, который отображается на карте.

Второй алгоритм – поиск пути из выбранной начальной точки в выбранную конечную. Расчёт новых координат шара происходит аналогично. Результатом алгоритма является путь шара до этой точки, который отображается на карте, или вывод сообщения о том, что пути не оказалось. Шар считается достигшим заданной точки, если он попадает в небольшую (≈ 0.5 градуса) окрестность точки.

**1. требования к программе**

**1.1. Исходные требования к программе**

**1.1.1. Требования к входным данным**

Для корректной работы алгоритма требуется:

* выбранная точка на карте;
* промежуток времени расчёта;
* время обновления погодных условий.

**1.1.2 Требования к визуализации**

Программа должна обладать простым и понятным интерфейсом. Прототип интерфейса представлен на рисунке 1.

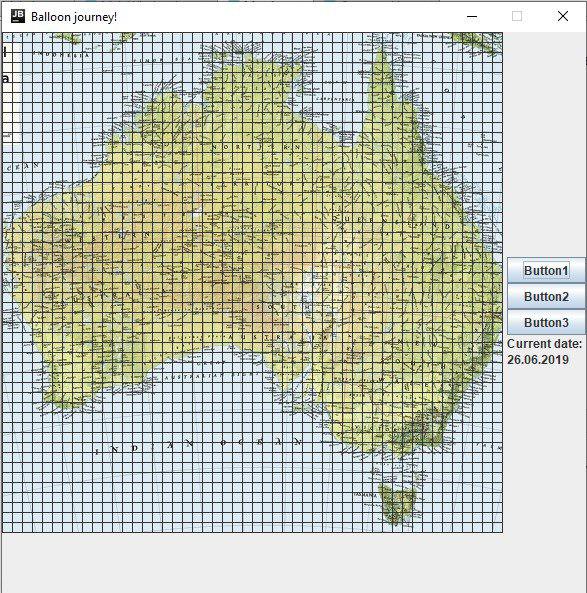


Рисунок 1 — прототип интерфейса.

Программа имеет интерактивную область с картой и с возможностью выбора стартовой точки, также на данную область будет выведен построенный маршрут. С правой стороны расположены функциональные кнопки и поля ввода информации.

**1.1.3 Требования к алгоритму и данным**

**Алгоритм выполняет запросы о состоянии погоды в данной точке в данное время и по полученным данным строит маршрут, который сохраняет в определённую структуру данных и передаёт их на визуализацию.**

**1.1.4 Требования к выходным данным**

**Выходными данными программы является построенный маршрут на карте.**

**1.2. Требования к программе после учточнения работы**

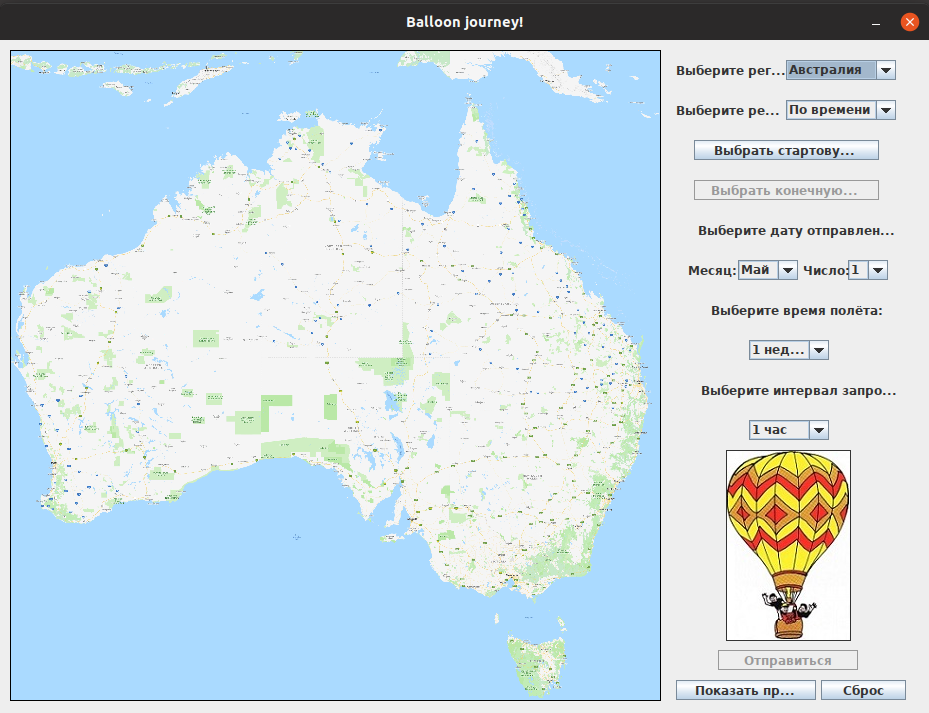
**1.2.1. Требования к входным данным**

Для корректной работы алгоритма требуется:

* выбранная точка на карте или точки для разных режимов;
* промежуток времени расчёта;
* время обновления погодных условий;
* выбор региона для наблюдения;
* время старта;

**1.2.2 Требования к визуализации**

**Интерфейс представленный на рисунке 2, выполняет все поставленный требования и остаётся интуитивно понятным и простым.**

**Рисунок 2 -­ доработанный интерфейс**

**В приложение добавлены новые кнопки и поля для сбора данных, также добавлена функция масштабирования карты и возможность выбора нескольких режимов.**

**1.2.3 Требования к алгоритму и данным**

**Была введена структура данных Vertex:**

координата относительно карты;  
координаты в реальном мире;  
погодные условия в точке во время прихода;

**Алгоритм получает необходимые данные, и выполняет следующие действия:**

**+** последовательно строит точки маршрута; **+** записывает в vertex: **+** погодные условия; **+** координаты в реальном мире; **+** добавляет в линейный список vertex; **+** реализует 2 задачи: **+** свободное блуждание шара; **+** маршрут до конечной точки.

**1.1.4 Требования к выходным данным**

Визуальный вывод постоенного маршрута на карту с выводом следующей информации: **+** общий пройденный маршрут; **+** погодных условий в точках замера; **+** координаты замеров;

**2. План разработки и распределение ролей в бригаде**

**2.1. План разработки**

**2.2. Распределение ролей в бригаде**

**3. Особенности реализации**

**3.1. Использованные структуры данных**

Для реализации проекта потребовалось разработать следующие структуры данных:

* Класс doublePoint, который хранит информацию о координате точки:

Поля: -private double x – первая координата;

-private double y – вторая координата.

Методы: -public double getX() – получить первую координату;

-public double getY() – получить вторую координату;

-public boolean equals(Object obj) – проверить два объекта на равенство;

-public int hashCode() – получить хэш-кода объекта

-public String toString() – представить объект в виде строки.

* Класс Time, который хранит информацию временном промежутке:

Поля: -private int year – количество лет;

-private int month – количество месяцев;

-private int day – количество дней;

-private int hour – количество часов;

Методы: -public boolean TimeNotOut() – проверка на равенство 0;

-public int getDay() – получить количество дней;

-public int getHour() – получить количество часов;

-public int getMonth() – получить количество месяцев;

-public int getYear() – получить количество лет;

-public void setDay(int day) – задать количество дней;

-public void setHour(int hour) – задать количество часов;

-public void setMonth(int month) – задать количество месяцев;

-public String toString() – представить объект в виде строки;

* Класс WeatherParameters, который хранит информацию о ветре в конкретной точке:

Поля: -private final int WindGustKmph – скорость ветра;

-private final int WinddirDegree – угол, который характеризует направление

Методы: -public int getWinddirDegree() – получить угол;

-public int getWindGustKmph() – получить скорость;

-public boolean equals(Object obj) – проверить два объекта на равенство;

-public int hashCode() – получить хэш-код объекта;

-public String toString() – представить объект в виде строки;

* Класс Vertex, который хранит информацию о конкретной точке на карте:

Поля: -private final doublePoint RealCoordinate – координаты в реальности;

-private doublePoint MapCoordinate – координаты на карте в приложении;

-private WeatherParameters weatherInPoint – данные о ветре в этой точке;

Методы: -public void setMapCoordinate(doublePoint mapCoordinate) – задать координаты на карте в приложении;

-public void setWeatherInPoint(WeatherParameters weatherInPoint) – задать данные о ветре в этой точке;

-public doublePoint getRealCoordinate() – получить реальные координаты;

-public doublePoint getMapCoordinate() – получить координаты на карте приложения;

-public WeatherParameters getWeatherInPoint() – получить данные о ветре;

-public boolean equals(Object obj) – проверить два объекта на равенство;

-public int hashCode() – получить хэш-код объекта;

-public String toString() – представить объект в виде строки;

**3.2. Основные методы**

Основные методы для работы были реализованы в классах MoveBalloonAlgorithm, который реализует перемещение воздушного шара, и Parsing, который реализует запрос к серверу и извлечение необходимой информации из него:

* public LinkedList<Vertex> AlgorithmTime(doublePoint startPoint, String startData, int startHour, Time TimeInAir, int step) – осуществляет перемещение воздушного шара из заданной точки startPoint, начиная с даты startData, начиная с часа startHour, в течение промежутка времени TimeInAir, с интервалом запроса к серверу в step часов, возвращает полученный список точек, в которых останавливался шар;
* public LinkedList<Vertex> AlgorithmEndPoint(doublePoint startPoint, doublePoint endPoint, String startData, int startHour, int step, double SizeEpsilon) – осуществляет полёт из точки startPoint в конечную точку endPoint, начиная с даты startData, начиная с часа startHour, с интервалом запроса к серверу в step часов, путь в точку endPoint считается найденным, если он попадет в окрестность этой точки SizeEpsilon, в случае нахождения пути он возвращается в списке;
* private void getMesWeather() – отправляет http запрос на сервер с данными о погоде;
* public WeatherParameters getParameters() – в случае успешного запроса на сервер осуществляет извлечение необходимой информации из полученного ответа и возвращает его в виде класса WeatherParameters.

**заключение**

Разработка поставленной задачи произошла по плану. Было спроектировано и запрограммировано приложения для поиска маршрута реального воздушного шара. Приложения использует реальные или исторические погодные данные, позволяет выбрать любую начальную точку на карте, выбрать различные регионы на планете Земля. Также был разработан графический интерфейс визуально отображающий результаты работы алгоритма и позволяющий управлять возможностями приложения. Основной алгоритм был покрыт Unit тестами, а графический интерфейс был оттестирован Поставленные задачи были выполнены полностью.

Также в приложение были добавлены дополнительные функции: режим поиска маршрута к конечной точке; изменения интервала запроса данных с сервера; расширение количества регионов; установка карт высокого разрешения; реализация интерактивной карты возможностью увеличения; создание логера и логирование работы алгоритмов.

Таким образом разработка приложения произошла успешно с полным выполнением плана и реализацией дополнительного функционала.

**список использованных источников**

1. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления

2. Официальная документация к Java: https://docs.oracle.com/en/java/javase/

3. Java. Эффективное программирование. Блох Джошуа 2014 год

4. Учебный курс по основам Java на Stepik: https://stepik.org/course/187/

5. Википедия : https://ru.wikipedia.org

6. https://ru.stackoverflow.com/

7. https://habr.com/ru/

**приложение А**

**Название приложения**

**приложение А**

**Исходный код**

1)Файл Main.java:

**import** gui.\*;  
**import** logger.Logs;  
  
**import** java.io.IOException;  
**import** java.util.logging.Level;  
  
**public class** Main {  
 **public static void** main(String[] args) **throws** IOException {  
 Logs.*writeLog*(**"start program"**, Level.***INFO***);  
 MainWindow window = **new** MainWindow();  
 window.setVisible(**true**);  
 }  
}

2)Файл MainWindow.java:

**package** gui;  
  
**import** algo.MoveBalloonAlgorithm;  
**import** dateStruct.Time;  
**import** dateStruct.Vertex;  
**import** dateStruct.doublePoint;  
  
**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.util.LinkedList;  
  
**public class** MainWindow **extends** JFrame{  
  
 **private final** Command **commands** = **new** Command();  
 **private final** Map **map** = **new** Map(650, 650);  
 **final** MapMouseAdapter **adapter** = **new** MapMouseAdapter();  
  
 **public** MainWindow() {  
 **super**(**"Balloon journey!"**);  
 **this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 **this**.setSize(930, 710); *//40 пикселей - полоска наверху программы* **this**.setResizable(**false**);  
 **this**.setLayout(**null**);  
 **map**.setLocation(10, 10);  
 **map**.setSize(651, 651);  
 **this**.add(**map**);  
 **commands**.setLocation(661, 10);  
 **commands**.setSize(350,651);  
 **this**.add(**commands**);  
  
 **commands**.**start**.addActionListener(**new** JToggleButtonActionListener());  
 **commands**.**begin**.addActionListener(**new** JButtonActionListener());  
 **commands**.**region**.addActionListener(**new** JComboBoxActionListener());  
 **commands**.**example**.addActionListener(**new** JToggleButtonExampleActionListener());  
 **commands**.**mode**.addActionListener(**new** JComboBoxModeActionListener());  
 **commands**.**clear**.addActionListener(**new** JButtonClearActionListener());  
 **commands**.**end**.addActionListener(**new** JToggleButtonActionListener());  
 **commands**.**day**.addActionListener(**new** JComboBoxDaysActionListener());  
 **commands**.**month**.addActionListener(**new** JComboBoxDaysActionListener());  
 }  
  
 **class** MapMouseAdapter **extends** MouseAdapter **implements** MouseWheelListener{  
 **private int mouseX**;  
 **private int mouseY**;  
  
 **public** MapMouseAdapter(){  
 addMouseWheelListener(**this**);  
 }  
  
 @Override  
 **public void** mouseClicked(MouseEvent event) {  
 **mouseX** = event.getX();  
 **mouseY** = event.getY();  
 **map**.setNullWay();  
 **if**(**commands**.**start**.isSelected()) {  
 **map**.setStartPoint(**new** doublePoint(**mouseX**, **mouseY**));  
 }  
 **if**(**commands**.**end**.isSelected()) {  
 **map**.setEndPoint(**new** doublePoint(**mouseX**, **mouseY**));  
 }  
 **commands**.**begin**.setEnabled(**true**);  
 **if**(**commands**.**mode**.getSelectedIndex() == 1 && !**map**.isPointsInit()){  
 **commands**.**begin**.setEnabled(**false**);  
 }  
 **commands**.**example**.setSelected(**false**);  
 **commands**.**example**.setEnabled(**false**);  
 }  
  
 @Override  
 **public void** mouseDragged(MouseEvent me) {  
 **mouseX** = me.getX();  
 **mouseY** = me.getY();  
 }  
  
 */\*\*  
 \* Событие на вращение колеса мышки  
 \** ***@param e*** *- событие  
 \* Имеет отдельного слушателя т.е. не зависит от включенного режима выбора  
 \*/* @Override  
 **public void** mouseWheelMoved(MouseWheelEvent e){  
 **double** step = 0.05;  
 **if**(e.getWheelRotation()> 0) {  
 step \*= -1;  
 }  
 **map**.MapScale(step,**new** doublePoint(e.getX(),e.getY()));  
 }  
 }  
  
 **class** JToggleButtonActionListener **implements** ActionListener {  
  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 **if**(**commands**.**start**.isSelected() || **commands**.**end**.isSelected()) {  
 **map**.setGrid(**true**);  
 **map**.addMouseListener(**adapter**);  
 } **else** {  
 **map**.setGrid(**false**);  
 **map**.removeMouseListener(**adapter**);  
 }  
 **map**.repaint();  
 }  
 }  
  
 **class** JButtonActionListener **implements** ActionListener {  
  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 **boolean** mode = **false**;  
 **int** index = **commands**.**mode**.getSelectedIndex();  
  
 **if**(index == 0){  
 mode = **true**;  
 }  
 index = **commands**.**month**.getSelectedIndex();  
 String date = **new** String(**"2019-"**);  
 **switch**(index){  
 **case** 0:  
 date += **"05-"**;  
 **break**;  
 **case** 1:  
 date += **"06-"**;  
 **break**;  
 **case** 2:  
 date += **"07-"**;  
 **break**;  
 }  
 index = **commands**.**day**.getSelectedIndex();  
 index++;  
 date += index;  
 index = **commands**.**time**.getSelectedIndex();  
 **int** hour = 0;  
 **switch**(index){  
 **case** 0:  
 hour = 1;  
 **break**;  
 **case** 1:  
 hour = 6;  
 **break**;  
 **case** 2:  
 hour = 12;  
 **break**;  
 **case** 3:  
 hour = 24;  
 **break**;  
 }  
 index = **commands**.**air**.getSelectedIndex();  
 **int** countDay = ++index;  
 countDay \*= 7;  
  
 LinkedList<Vertex> temp = **null**;  
 *//для определения полушарий* **boolean** nordSphere = **true**;  
 **boolean** estSphere = **true**;  
 **double** sizeLatitude = **map**.getUpperRight().getX() - **map**.getBottomLefht().getX();  
 **if**(sizeLatitude < 0){  
 sizeLatitude \*= -1;  
 nordSphere = **false**;  
 }  
 **double** sizeLongitude = **map**.getUpperRight().getY() - **map**.getBottomLefht().getY();  
 **if**(sizeLongitude < 0){  
 sizeLongitude \*= -1;  
 estSphere = **false**;  
 }  
  
 MoveBalloonAlgorithm algo;  
 **try** {  
 *//500 - км в градусе - const* algo = **new** MoveBalloonAlgorithm(**map**.getBottomLefht(), sizeLongitude, sizeLatitude, nordSphere, estSphere, 500);  
 **if**(mode){  
 temp = algo.AlgorithmTime(**map**.getRealCoordinate(**true**), date, 0, **new** Time(0, countDay, 0), hour);  
 }**else**{  
 **double** epsilon = 0.5; *//область вокруг конечной точки в дробных градусах* temp = algo.AlgorithmEndPoint(**map**.getRealCoordinate(**true**), **map**.getRealCoordinate(**false**), date, 0, hour, epsilon);  
 }  
 }**catch** (IOException e) {  
 JOptionPane.*showMessageDialog*(**null**, **"Ошибка записи в лог-файл!"**, **"Error!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 System.*exit*(-1);  
 }**catch** (NullPointerException e) {  
 JOptionPane.*showMessageDialog*(**null**, **"Обращение к null! Выполните сброс и выберите другие данные!"**, **"Error!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 **return**;  
 }  
  
 **map**.setGrid(**false**);  
 **commands**.**start**.setSelected(**false**);  
 **commands**.**end**.setSelected(**false**);  
  
 **if**(temp != **null**){  
 **map**.setAlgorithmDate(temp);  
 }**else**{  
 **if**(mode){  
 JOptionPane.*showMessageDialog*(**null**, **"По таким данным не удалось построить путь!"**, **"Fail!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 }**else**{  
 JOptionPane.*showMessageDialog*(**null**, **"Пути в выбранную точку не нашлось!"**, **"Fail!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 }  
 }  
 **map**.repaint();  
 **commands**.**begin**.setEnabled(**false**);  
 }  
 }  
  
 **class** JComboBoxActionListener **implements** ActionListener {  
  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 String value = **""**;  
 doublePoint bottomLeft = **null**;  
 doublePoint upperRight = **null**;  
 **int** regionIndex = **commands**.**region**.getSelectedIndex();  
 **switch**(regionIndex) {  
 **case** 0:  
 value = **"australia.jpg"**;  
 bottomLeft = **new** doublePoint(43.73333, 112.93333);  
 upperRight = **new** doublePoint(7.36667, 154.31667);  
 **break**;  
 **case** 1:  
 value = **"balkans.jpg"**;  
 bottomLeft = **new** doublePoint(36.06667, 13.43333);  
 upperRight = **new** doublePoint(48.16667, 29.88333);  
 **break**;  
 **case** 2:  
 value = **"china.jpg"**;  
 bottomLeft = **new** doublePoint(17.7, 100.1);  
 upperRight = **new** doublePoint(44.1, 131.35);  
 **break**;  
 **case** 3:  
 value = **"india.jpg"**;  
 bottomLeft = **new** doublePoint(5.36667, 65.4);  
 upperRight = **new** doublePoint(31.08333, 92.58333);  
 **break**;  
 **case** 4:  
 value = **"russia.jpg"**;  
 bottomLeft = **new** doublePoint(52.86667, 28.83333);  
 upperRight = **new** doublePoint(60.63333, 43.1);  
 **break**;  
 **case** 5:  
 value = **"scandinavia.jpg"**;  
 bottomLeft = **new** doublePoint(54.88333, 4.18333);  
 upperRight = **new** doublePoint(71.31667, 41.3);  
 **break**;  
 **case** 6:  
 value = **"usa.jpg"**;  
 bottomLeft = **new** doublePoint(4.86667, 124.1);  
 upperRight = **new** doublePoint(49.33333, 72.05);  
 **break**;  
 **case** 7:  
 value = **"antarctica.jpg"**;  
 bottomLeft = **new** doublePoint(83.1, 96.63333);  
 upperRight = **new** doublePoint(62.46667, 16.65);  
 **break**;  
 }  
 **map**.setRegion(value, bottomLeft, upperRight); *//X - широта, Y - долгота* **map**.setGrid(**false**);  
 **commands**.**start**.setSelected(**false**);  
 **commands**.**end**.setSelected(**false**);  
 **map**.setNullWay();  
 **map**.setNullStartPoint();  
 **map**.setNullEndPoint();  
  
 **map**.repaint();  
 **commands**.**begin**.setEnabled(**false**);  
 **commands**.**example**.setSelected(**false**);  
 **commands**.**example**.setEnabled(**true**);  
 }  
 }  
  
 *//ОБРАБОТЧИК НАЖАТИЯ НА КНОПКУ "Показать пример"* **class** JToggleButtonExampleActionListener **implements** ActionListener {  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 LinkedList<Vertex> temp = **new** LinkedList<>();  
 **if**(**commands**.**example**.isSelected()) {  
 doublePoint garbage = **new** doublePoint(0, 0);  
 **for** (**int** i = 0; i < 10; i++) {  
 temp.add(**new** Vertex(garbage));  
 }  
 temp.get(0).setMapCoordinate(**new** doublePoint(100, 100));  
 temp.get(1).setMapCoordinate(**new** doublePoint(269, 200));  
 temp.get(2).setMapCoordinate(**new** doublePoint(158, 358));  
 temp.get(3).setMapCoordinate(**new** doublePoint(173, 489));  
 temp.get(4).setMapCoordinate(**new** doublePoint(343, 457));  
 temp.get(5).setMapCoordinate(**new** doublePoint(429, 387));  
 temp.get(6).setMapCoordinate(**new** doublePoint(401, 361));  
 temp.get(7).setMapCoordinate(**new** doublePoint(389, 379));  
 temp.get(8).setMapCoordinate(**new** doublePoint(406, 410));  
 temp.get(9).setMapCoordinate(**new** doublePoint(485, 523));  
 }  
 **map**.setAlgorithmDate(temp);  
 }  
 }  
  
 **class** JButtonClearActionListener **implements** ActionListener {  
  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 **map**.setNullWay();  
 **map**.setNullStartPoint();  
 **map**.setNullEndPoint();  
 **map**.setGrid(**false**);  
 **map**.setNullScale();  
 **map**.repaint();  
 **commands**.**example**.setSelected(**false**);  
 **commands**.**end**.setSelected(**false**);  
 **commands**.**start**.setSelected(**false**);  
 **commands**.**example**.setEnabled(**true**);  
 **commands**.**begin**.setEnabled(**false**);  
 }  
 }  
  
 **class** JComboBoxModeActionListener **implements** ActionListener {  
  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 **if**(**commands**.**mode**.getSelectedIndex() == 1){  
 **commands**.**end**.setEnabled(**true**);  
 **commands**.**air**.setEnabled(**false**);  
 }**else**{  
 **commands**.**end**.setEnabled(**false**);  
 **commands**.**air**.setEnabled(**true**);  
 **commands**.**end**.setSelected(**false**);  
 **if**(!**commands**.**start**.isSelected()) {  
 **map**.setGrid(**false**);  
 **map**.setNullEndPoint();  
 **map**.repaint();  
 }  
 }  
 }  
 }  
  
 **class** JComboBoxDaysActionListener **implements** ActionListener {  
  
 @Override  
 **public void** actionPerformed(ActionEvent event) {  
 **int** month = **commands**.**month**.getSelectedIndex();  
 **int** day = **commands**.**day**.getSelectedIndex();  
 **if**(day == 30 && month == 1) {  
 JOptionPane.*showMessageDialog*(**null**, **"Несуществующая дата!"**, **"Warning!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 **commands**.**day**.setSelectedIndex(29);  
 }  
 **int** lastDayJuly = 8; *//последний наступивший день июля(индексация с 0), не забывать менять* **if**(day > lastDayJuly && month == 2) {  
 JOptionPane.*showMessageDialog*(**null**, **"Ненаступившая дата!"**, **"Warning!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 **commands**.**day**.setSelectedIndex(lastDayJuly);  
 }  
 }  
 }  
  
 **class** Command **extends** JPanel{  
  
 **private final** JLabel **labelRegion** = **new** JLabel(**"Выберите регион:"**);  
 **private final** JComboBox **region** = **new** JComboBox(**new** String[] {**"Австралия"**, **"Балканы"**, **"Китай"**, **"Индия"**, **"Россия"**,  
 **"Скандинавия"**, **"США"**, **"Антарктида"**});  
 **private** JLabel **labelMode** = **new** JLabel(**"Выберите режим:"**);  
 **private final** JComboBox **mode** = **new** JComboBox(**new** String[] {**"По времени"**, **"К точке"**});  
 **private final** JToggleButton **start** = **new** JToggleButton(**"Выбрать стартовую точку"**);  
 **private final** JLabel **date** = **new** JLabel(**"Выберите дату отправления:"**);  
 **private final** JLabel **labelMonth** = **new** JLabel(**"Месяц:"**);  
 **private final** JComboBox **month** = **new** JComboBox(**new** String[] {**"Май"**, **"Июнь"**, **"Июль"**});  
 **private final** String[] **days** = **new** String[] {**"1"**, **"2"**, **"3"**, **"4"**, **"5"**, **"6"**, **"7"**, **"8"**, **"9"**, **"10"**, **"11"**, **"12"**, **"13"**, **"14"**, **"15"**,  
 **"16"**, **"17"**, **"18"**, **"19"**, **"20"**, **"21"**, **"22"**, **"23"**, **"24"**, **"25"**, **"26"**, **"27"**, **"28"**, **"29"**, **"30"**, **"31"**};  
 **private final** JLabel **labelDays** = **new** JLabel(**"Число:"**);  
 **private final** JComboBox **day** = **new** JComboBox(**days**);  
 **private final** JButton **begin** = **new** JButton(**"Отправиться"**);  
 **private final** JLabel **labelTime** = **new** JLabel(**"Выберите интервал запроса:"**);  
 **private final** JComboBox **time** = **new** JComboBox(**new** String[] {**"1 час"**, **"6 часов"**, **"12 часов"**, **"24 часа"**});  
 **private final** JToggleButton **end** = **new** JToggleButton(**"Выбрать конечную точку"**);  
 **private final** JLabel **labelAir** = **new** JLabel(**"Выберите время полёта:"**);  
 **private final** JComboBox **air** = **new** JComboBox(**new** String[] {**"1 неделя"**, **"2 недели"**, **"3 недели"**, **"4 недели"**,  
 **"5 недель"**, **"6 недель"**, **"7 недель"**, **"8 недель"**});  
 **private final** JToggleButton **example** = **new** JToggleButton(**"Показать пример"**);  
 **private final** JButton **clear** = **new** JButton(**"Сброс"**);  
 **private** Image **image**;  
  
 **public** Command() {  
 **super**();  
 **this**.setLayout(**null**);  
 **labelRegion**.setSize(110, 40);  
 **labelRegion**.setLocation(15, 0);  
 **this**.add(**labelRegion**);  
 **region**.setSize(110, 20);  
 **region**.setLocation(125, 10);  
 **this**.add(**region**);  
 **labelMode**.setSize(110, 40);  
 **labelMode**.setLocation(15, 40);  
 **this**.add(**labelMode**);  
 **mode**.setSize(110, 20);  
 **mode**.setLocation(125, 50);  
 **this**.add(**mode**);  
 **start**.setSize(185, 20);  
 **start**.setLocation(33, 90);  
 **this**.add(**start**);  
 **end**.setSize(185, 20);  
 **end**.setLocation(33, 130);  
 **this**.add(**end**);  
 **date**.setSize(200, 20);  
 **date**.setLocation(37, 170);  
 **this**.add(**date**);  
 **labelMonth**.setSize(75, 20);  
 **labelMonth**.setLocation(27, 210);  
 **this**.add(**labelMonth**);  
 **month**.setSize(60, 20);  
 **month**.setLocation(77, 210);  
 **this**.add(**month**);  
 **labelDays**.setSize(75,20);  
 **labelDays**.setLocation(142, 210);  
 **this**.add(**labelDays**);  
 **day**.setSize(40, 20);  
 **day**.setLocation(187, 210);  
 **this**.add(**day**);  
 **labelAir**.setSize(200, 20);  
 **labelAir**.setLocation(50, 250);  
 **this**.add(**labelAir**);  
 **air**.setSize(80, 20);  
 **air**.setLocation( 88,290);  
 **this**.add(**air**);  
 **labelTime**.setSize(200, 20);  
 **labelTime**.setLocation(40, 330);  
 **this**.add(**labelTime**);  
 **time**.setSize(80, 20);  
 **time**.setLocation( 88,370);  
 **this**.add(**time**);  
 **begin**.setSize(140, 20);  
 **begin**.setLocation(57, 600);  
 **this**.add(**begin**);  
 **example**.setSize(140, 20);  
 **example**.setLocation(15, 630);  
 **this**.add(**example**);  
 **clear**.setSize(85, 20);  
 **clear**.setLocation(160, 630);  
 **this**.add(**clear**);  
  
 **begin**.setEnabled(**false**);  
 **end**.setEnabled(**false**);  
  
 **try** {  
 String path = **""**;  
 path = **"src"** + File.***separator*** + **"gui"** + File.***separator*** + **"balloon"** + File.***separator*** + **"balloon.jpg"**;  
 **image** = ImageIO.*read*(**new** File(path));  
 } **catch** (IOException e) {  
 JOptionPane.*showMessageDialog*(**null**, **"Balloon image not open!"**, **"Error!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 System.*exit*(-1);  
 }  
 }  
  
 @Override  
 **public void** paint(Graphics g) {  
 **super**.paint(g);  
 g.drawImage(**image**, 65, 400, **this**);  
 g.drawLine(65, 400, 65, 590);  
 g.drawLine(65, 400, 189, 400);  
 g.drawLine(65, 590, 189, 590);  
 g.drawLine(189, 400, 189, 590);  
 }  
 }  
}

3)Файл Map.java:

**package** gui;  
  
**import** dateStruct.Vertex;  
**import** dateStruct.doublePoint;  
*import org.jetbrains.annotations.NotNull;***import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.util.LinkedList;  
  
**public class** Map **extends** JPanel {  
 */\*\*  
 \** ***@value*** *width - ширина окна  
 \** ***@value*** *height - высота окна  
 \** ***@value*** *grid - флаг сетки  
 \** ***@value*** *gridColor - цвет сетки  
 \** ***@value*** *MapImage - фоновое изображение  
 \** ***@value*** *region - имя фонового изображения  
 \** ***@value*** *StartPoint -стартовая точка алгоритма  
 \** ***@value*** *EndPoint - конечная точка алгоритма  
 \** ***@value*** *PointColor - цвет точки  
 \** ***@value*** *ScaleFactor - коэфицент масштаба  
 \** ***@value*** *ScalePoint - точка-центр масштабирования  
 \** ***@value*** *ImageFactor - коэфицент сжатия большого изображения  
 \** ***@value*** *offsetX,offsetY - смещение карты за левый верхний угол  
 \** ***@value*** *bottomLeft - реальные координаты левого нижнего угла карты  
 \** ***@value*** *upperRight - реальные координаты правого верхнего угла карты  
 \*/* **private final int width**;  
 **private final int height**;  
 **private boolean grid** = **false**;  
 **private final** Color **gridColor** = **new** Color(0,0,0, 87);  
 **private** Image **MapImage**;  
 **private** String **region** = **"australia.jpg"**;  
 **private** doublePoint **StartPoint**;  
 **private** doublePoint **EndPoint**;  
 **private final** Color **PointColor** = **new** Color(0, 13, 255);  
 **private double ScaleFactor**;  
 **private** doublePoint **ScalePoint**;  
 **private double ImageFactor**;  
 **private int offsetX**,**offsetY**;  
 **private** doublePoint **bottomLeft** = **new** doublePoint(43.73333, 112.93333); *//значения для Австралии* **private** doublePoint **upperRight** = **new** doublePoint(7.36667, 154.31667); *//для загрузки по умолчанию* **private** LinkedList<Vertex> **BalloonWay**;  
  
  
 */\*\*  
 \* Конструктор класса  
 \** ***@param width*** *- ширина окна карты  
 \** ***@param height*** *- высота окна карты  
 \* загрузка стартового региона через setRegion  
 \* Стартовый масштаб = 1  
 \* Смещение карты = 0  
 \* Путь шара отсутствует  
 \*/* **public** Map(**int** width, **int** height) {  
 **super**();  
 **this**.**width** = width;  
 **this**.**height** = height;  
  
 setRegion(**region**, **bottomLeft**, **upperRight**);  
  
 **ScaleFactor** = 1;  
 **offsetX** = **offsetY** =0;  
 **BalloonWay** = **null**;  
 }  
  
 */\*\*  
 \* изменение флага сетки  
 \** ***@param value*** *- true, если нужна сетка  
 \*/* **public void** setGrid(**boolean** value) {  
 **grid** = value;  
 }  
  
 */\*\*  
 \* установка стартовой точки на карте  
 \** ***@param p*** *- стартовая точка  
 \* обязательная перерисовка области  
 \*/* **public void** setStartPoint(doublePoint p){  
 **StartPoint** = **new** doublePoint((**int**)Math.*abs*((( p.getX()+ **offsetX**) / **ScaleFactor**)),  
 (**int**) Math.*abs*(((p.getY()+ **offsetY**)/ **ScaleFactor**)));  
 repaint();  
 }  
  
 **public void** setEndPoint(doublePoint p){  
 **EndPoint** = **new** doublePoint((**int**)Math.*abs*((( p.getX()+ **offsetX**) / **ScaleFactor**)),  
 (**int**) Math.*abs*(((p.getY()+ **offsetY**)/ **ScaleFactor**)));  
 repaint();  
 }  
  
 */\*\*  
 \* метод для изменения масштаба  
 \** ***@param step*** *- прибавка к масштабу  
 \** ***@param coord*** *- точка вызова изменения масштаба  
 \*/* **public void** MapScale(**double** step, doublePoint coord){  
 **if**( isVisible(coord) && (**ScaleFactor** + step) >= 1.0 && (**ScaleFactor** + step) < 5.0){  
 **ScaleFactor** += step;  
 **ScalePoint** = **new** doublePoint(coord);  
  
 **double** MoveMapFactorX = (**ScalePoint**.getX() / **width**); *// тут изменить коэф смещения для большей плавности* **double** MoveMapFactorY = (**ScalePoint**.getY() / **height**);  
  
 **offsetX** = (**int**) Math.*round*( MoveMapFactorX \* **width** \* (**ScaleFactor** -1));  
  
 **if** ((**MapImage**.getWidth(**null**) \* **ImageFactor** \* **ScaleFactor** - **offsetX**) < **width**) {  
 **offsetX** = (**int**) Math.*round*((**MapImage**.getWidth(**null**) \* **ScaleFactor** \* **ImageFactor** - **width**));  
 }  
  
  
 **offsetY** = (**int**) Math.*round*( MoveMapFactorY \* **height** \* (**ScaleFactor** -1));  
 **if** ((**MapImage**.getHeight(**null**) \* **ScaleFactor** \* **ImageFactor** - **offsetY**) < **height**) {  
 **offsetY** = (**int**) Math.*round*((**MapImage**.getHeight(**null**) \* **ScaleFactor** \* **ImageFactor** - **height**));  
 }  
 repaint();  
 }  
 }  
  
 */\*\*  
 \* проверка вхождения точки в область видимой карты  
 \** ***@param coord*** *- точка проверки  
 \** ***@return*** *true - если входит в область видимой карты  
 \*/* **private boolean** isVisible(*@NotNull* doublePoint coord) {  
 **int** borderX = 10;  
 **int** borderY = 50;  
 **return** (!(coord.getX() <= borderX) && !(coord.getX() >= **width** + borderX)) &&  
 (!(coord.getY() <= borderY) && !(coord.getY() >= **height** + borderY));  
 }  
  
 */\*\*  
 \* вычисление реальных координат в дробных градусах для стартовой или конечной точки  
 \** ***@param value*** *- true, если для стартовой; false, если для конечной  
 \** ***@return*** *- реальные координаты стартовой точки  
 \*/* **public** doublePoint getRealCoordinate(**boolean** value){  
 *//коэффициенты показывают сколько приходится градусов на один пиксель Map* **double** FactorX = (**bottomLeft**.getX() - **upperRight**.getX()) / **height**;  
 **double** FactorY = (**upperRight**.getY() - **bottomLeft**.getY()) / **width**;  
 **double** x, y;  
 **if**(value) {  
 x = **upperRight**.getX() + FactorX \* **StartPoint**.getX(); *//широта* y = **bottomLeft**.getY() + FactorY \* **StartPoint**.getY(); *//долгота* }**else**{  
 x = **upperRight**.getX() + FactorX \* **EndPoint**.getX(); *//широта* y = **bottomLeft**.getY() + FactorY \* **EndPoint**.getY(); *//долгота* }  
 **return new** doublePoint(x, y);  
 }  
  
 */\*\*  
 \* Вычисление координат на карте приложения без масштабирования  
 \** ***@param realCoordinate*** *- реальные географические координаты в дробных градусах  
 \** ***@return*** *- координаты на карте в приложении, возвращаются int, для точного расположения на карте  
 \*/* **public** doublePoint getMapCoordinate(doublePoint realCoordinate){  
  
 **double** FactorX = (**bottomLeft**.getX() - **upperRight**.getX()) / **height**;  
 **double** FactorY = (**upperRight**.getY() - **bottomLeft**.getY()) / **width**;  
 **double** x = (realCoordinate.getX() - **upperRight**.getX()) / FactorX;  
 **double** y = (realCoordinate.getY() - **bottomLeft**.getY()) / FactorY;  
 x = Math.*round*(x);  
 y = Math.*round*(y);  
 **return new** doublePoint(x, y);  
 }  
  
 **public** doublePoint getBottomLefht(){**return bottomLeft**;}  
  
 **public** doublePoint getUpperRight(){**return upperRight**;}  
  
 **public void** setNullWay() { **BalloonWay** = **null**; }  
  
 **public void** setNullStartPoint() { **StartPoint** = **null**; }  
  
 **public void** setNullEndPoint() { **EndPoint** = **null**; }  
  
 **public void** setNullScale() {  
 **ScalePoint** = **null**;  
 **offsetX** = **offsetY** = 0;  
 **ScaleFactor** = 1;  
 }  
  
 */\*\*  
 \* проверка установки стартовой и конечной точки для алогритма к точке  
 \** ***@return*** *- true, если точки установлены; false, если одна из них null  
 \*/* **public boolean** isPointsInit() { **return StartPoint** != **null** && **EndPoint** != **null**;}  
  
 */\*\*  
 \* загрузка данных в класс map  
 \* также высчитывает координаты относительно карты для всех точек  
 \*/* **public void** setAlgorithmDate(LinkedList<Vertex> date){  
 **grid** = **false**;  
 **BalloonWay** = date;  
  
 **for** (Vertex vertex: date) {  
 **if**(vertex.getMapCoordinate() == **null**){  
 doublePoint temp = vertex.getRealCoordinate();  
 doublePoint temp2 = getMapCoordinate(temp);  
 vertex.setMapCoordinate(temp2);  
 }  
 }  
 repaint();  
 }  
  
 */\*\*  
 \* Загружает регион и высчитывает некоторые параметры  
 \** ***@param value*** *- имя региона  
 \** ***@param bottomLeft*** *- реальные координаты левого нижнего угла в дробных градусах  
 \** ***@param upperRight*** *- реальные координаты правого верхнего угла в дробных градусах  
 \* безопасная загрузка  
 \*/* **public void** setRegion(String value, doublePoint bottomLeft, doublePoint upperRight) {  
 **region** = value;  
 **this**.**bottomLeft** = bottomLeft;  
 **this**.**upperRight** = upperRight;  
 **MapImage** = **null**;  
 **try** {  
 String path = **""**;  
 path = **"src"** + File.***separator*** + **"gui"** + File.***separator*** + **"maps"** + File.***separator*** + **region**;  
 **MapImage** = ImageIO.*read*(**new** File(path));  
 } **catch** (IOException e) {  
 JOptionPane.*showMessageDialog*(**null**, **"Map not open!"**, **"Error!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 System.*exit*(-1);  
 }  
  
 **ImageFactor** = **width** / (**double** )**MapImage**.getWidth(**null**);  
 setNullScale();  
 }  
  
 */\*\*  
 \* Метод для отрисовки  
 \** ***@param g*** *- графический контекст  
 \* для получения реальных координат на отрисованной карте умножайте на \* ScaleFactor \* ImageFactor  
 \* и после вычитайте смещение угла карты  
 \*/* @Override  
 **public void** paint(Graphics g) {  
 g.drawImage(**MapImage**,  
 -**offsetX**, -**offsetY**,  
 (**int**)(**MapImage**.getWidth(**null**) \* **ScaleFactor** \* **ImageFactor**),  
 (**int**)(**MapImage**.getHeight(**null**) \* **ScaleFactor** \* **ImageFactor**),**null**);  
  
 **if**(**grid**) {  
 g.setColor(**gridColor**);  
 **for**(**int** i = -**offsetX**; i <= **width**; i += (**int**)(15 \* **ScaleFactor**)) {  
 g.drawLine(i, 0, i, **height**);  
 }  
 **for**(**int** i = -**offsetY**; i <= **height**; i += (**int**)(15 \* **ScaleFactor**)) {  
 g.drawLine(0, i, **width**, i);  
 }  
 } **else** {  
 g.drawLine(0, 0, 0, **height**);  
 g.drawLine(0, 0, **width**, 0);  
 g.drawLine(**width**, 0, **width**, **height**);  
 g.drawLine(0, **height**, **width**, **height**);  
 }  
  
 **int** radius = 3;  
 g.setColor(**PointColor**);  
  
  
 **if**(**BalloonWay** !=**null**){  
 doublePoint prevPoint = **null**;  
 **for** (Vertex vertex: **BalloonWay**) {  
 doublePoint temp = vertex.getMapCoordinate();  
  
 **int** X = (**int**) Math.*round*(temp.getX() \* **ScaleFactor** - **offsetX** - radius);  
 **int** Y = (**int**) Math.*round*(temp.getY() \* **ScaleFactor** - **offsetY** - radius);  
 g.drawOval( X , Y ,2\*radius,2\*radius);  
 g.fillOval( X, Y,2\*radius,2\*radius);  
 g.setColor(Color.***BLACK***);  
  
 g.setFont(**new** Font(**"Serif"**, Font.***PLAIN***, 10));  
  
 **if**(**ScaleFactor** >= 2.2){  
 **if**(vertex.getWeatherInPoint() != **null**){  
 g.drawString(vertex.getWeatherInPoint().toString(), X -30, Y - 15);  
 }  
  
 g.drawString(vertex.getRealCoordinate().toString(), X -30, Y - 5);  
 }  
  
 **if**(prevPoint == **null**){  
 prevPoint = **new** doublePoint(X+radius,Y+radius);  
 **continue**;  
 }  
  
 g.drawLine((**int**)prevPoint.getX(),(**int**)prevPoint.getY(),X+radius,Y+radius);  
 prevPoint = **new** doublePoint(X+radius,Y+radius);  
  
 }  
 }  
  
 **if**(**EndPoint** != **null** && **StartPoint** != **null**){  
 **if** (**StartPoint**.equals(**EndPoint**)) {  
 Color temp = **new** Color(128, 0, 128);  
 g.setColor(temp);  
  
 **int** X = (**int**) Math.*round*(**StartPoint**.getX() \* **ScaleFactor** - **offsetX** - 1.5 \* radius);  
 **int** Y = (**int**) Math.*round*(**StartPoint**.getY() \* **ScaleFactor** - **offsetY** - 1.5 \* radius);  
  
 g.drawOval( X, Y,3\*radius,3\*radius);  
 g.fillOval( X, Y,3\*radius,3\*radius);  
 g.drawString(**"s-t/f-h"**, X -10, Y + 20);  
 **return**;  
 }  
 }  
  
 g.setColor(**PointColor**);  
 **if**(**StartPoint** != **null**){  
  
 **int** X = (**int**) Math.*round*(**StartPoint**.getX() \* **ScaleFactor** - **offsetX** - 1.5 \* radius);  
 **int** Y = (**int**) Math.*round*(**StartPoint**.getY() \* **ScaleFactor** - **offsetY** - 1.5 \* radius);  
  
 g.drawOval( X, Y,3\*radius,3\*radius);  
 g.fillOval( X, Y,3\*radius,3\*radius);  
 g.drawString(**"start"**, X -10, Y + 20);  
 }  
  
 g.setColor(Color.***RED***);  
 **if**(**EndPoint** != **null**){  
  
 **int** X = (**int**) Math.*round*(**EndPoint**.getX() \* **ScaleFactor** - **offsetX** - 1.5 \* radius);  
 **int** Y = (**int**) Math.*round*(**EndPoint**.getY() \* **ScaleFactor** - **offsetY** - 1.5 \* radius);  
  
 g.drawOval( X, Y,3\*radius,3\*radius);  
 g.fillOval( X, Y,3\*radius,3\*radius);  
 g.drawString(**"finish"**, X -10, Y + 20);  
 }  
 }  
}

4)Файл Logs.java:

**package** logger;  
  
**import** com.sun.tools.javac.Main;  
  
**import** java.io.IOException;  
**import** java.util.logging.\*;  
  
**public class** Logs{  
 **private final static** Logger ***logger*** = Logger.*getLogger*(Main.**class**.getName());  
  
 **static** {  
 Handler fileLog = **null**;  
 **try** {  
 fileLog = **new** FileHandler();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 fileLog.setFormatter(**new** Forms());  
 fileLog.setFormatter(**new** Forms());  
 ***logger***.setUseParentHandlers(**false**);  
 ***logger***.addHandler(fileLog);  
 }  
  
 **public static void** writeLog(String Message, Level level) **throws** IOException {  
 ***logger***.log(level, Message);  
 }  
  
 **static class** Forms **extends** Formatter {  
 @Override  
 **public** String format(LogRecord record){  
 **return** record.getLevel() + **":"** + record.getMessage();  
 }  
 }  
}

5)Файл MoveBalloonAlgorithm.java:

**package** algo;  
  
**import** dateStruct.\*;  
**import** logger.Logs;  
  
**import** java.io.IOException;  
**import** java.util.Date;  
**import** java.util.LinkedList;  
**import** java.util.logging.Level;  
  
**public class** MoveBalloonAlgorithm {  
  
 **private** doublePoint **ControlPoint**;  
 **private double sizeMapLongitude**;  
 **private double sizeMapLatitude**;  
 **private int scale**;  
 **private boolean nordHemisphere**;  
 **private boolean estHemisphere**;  
  
 **public** MoveBalloonAlgorithm(doublePoint СontrolPoint, **double** sizeLongitude, **double** sizeLatitude, **boolean** nordSphere, **boolean** estSphere, **int** scale) **throws** IOException {  
 **this**.**ControlPoint** = СontrolPoint;  
 **sizeMapLongitude** = sizeLongitude;  
 **sizeMapLatitude** = sizeLatitude;  
 **this**.**scale** = scale;  
 **nordHemisphere** = nordSphere;  
 **estHemisphere** = estSphere;  
 Date date = **new** Date();  
 Logs.*writeLog*(date.toString()+**"\n"**, Level.***INFO***);  
 Logs.*writeLog*(**" -- Create MoveBalloonAlgorithm instance successful --\n"**, Level.***INFO***);  
 }  
  
 */\*\*  
 \* Метод moveBalloon - метод, который вычисляет координаты следующей точки  
 \*/* **private** doublePoint moveBalloon(WeatherParameters parameters, doublePoint startPoint, **int** step) {  
 **if**(**scale** == 0)  
 **return null**;  
 **double** x = startPoint.getX() + Math.*cos*(parameters.getWinddirDegree()) \* (parameters.getWindGustKmph() \* step / (**double**)**scale**);  
 **double** y = startPoint.getY() + Math.*sin*(parameters.getWinddirDegree()) \* (parameters.getWindGustKmph() \* step / (**double**)**scale**);  
 doublePoint result = **new** doublePoint(x, y);  
 **return** result;  
 }  
  
 */\*\*  
 \* Метод coordsIsCorrect - метод, который проверяет выход за границы карты  
 \*/* **private boolean** coordsIsCorrect(doublePoint tmp, doublePoint Control, **double** sizeLatitude, **double** sizeLongitude) {  
 **if**(**nordHemisphere**){  
 **if**(tmp.getX() > Control.getX() && tmp.getX() < Control.getX() + sizeLatitude){  
 **return** coordsLongitudeIsCorrect(tmp, Control, sizeLongitude);  
 }**else**{  
 **return false**;  
 }  
 }**else**{  
 **if**(tmp.getX() < Control.getX() && tmp.getX() > Control.getX() - sizeLatitude){  
 **return** coordsLongitudeIsCorrect(tmp, Control, sizeLongitude);  
 }**else**{  
 **return false**;  
 }  
 }  
 }  
  
 */\*\*  
 \* Метод coordsLongitudeIsCorrect - метод, который проверяет выход за пределы долготы на карте  
 \*/* **public boolean** coordsLongitudeIsCorrect(doublePoint tmp, doublePoint Control, **double** sizeLongitude){  
 **if**(**estHemisphere**){  
 **if**(tmp.getY() > Control.getY() && tmp.getY() < Control.getY() + sizeLongitude){  
 **return true**;  
 }**else**{  
 **return false**;  
 }  
 }**else** {  
 **if**(tmp.getY() < Control.getY() && tmp.getY() > Control.getY() - sizeLongitude) {  
 **return true**;  
 }  
 **else**{  
 **return false**;  
 }  
 }  
  
 }  
  
 */\*\*  
 \* Метод methodTimeOut - метод, осуществляющий уменьшение времени полёта после каждого шага  
 \*/* **private int** methodTimeOut(Time tmp, **int** step) {  
 **if** ((tmp.getHour() - step) <= 0) {  
 **if** (tmp.getDay() > 0) {  
 tmp.setHour(tmp.getHour() + 24 - step);  
 tmp.setDay(tmp.getDay() - 1);  
 **return** step;  
 } **else** {  
 **int** res = tmp.getHour();  
 tmp.setHour(0);  
 **return** res;  
 }  
 } **else** {  
 tmp.setHour(tmp.getHour() - step);  
 **return** step;  
 }  
 }  
  
 */\*\*  
 \* Метод AlgorithmTime - алгоритм вычисления конечной точки, при задании пользователем варианта программы "Полёт по времени"  
 \*/* **public** LinkedList<Vertex> AlgorithmTime(doublePoint startPoint, String startData, **int** startHour, Time TimeInAir, **int** step) **throws** IOException {  
  
 Logs.*writeLog*(**" -- Start alhorithm -- \n"**, Level.***INFO***);  
 Parsing pars = **new** Parsing(startData, step, startHour);  
 doublePoint tmp = startPoint;  
 Vertex vert;  
 LinkedList<Vertex> List = **new** LinkedList<>();  
 **while** (TimeInAir.TimeNotOut()) {  
 Logs.*writeLog*(**" -- NEXT STEP! --\n"**, Level.***INFO***);  
 **if** (!coordsIsCorrect(tmp, **ControlPoint**, **sizeMapLatitude**, **sizeMapLongitude**)) {  
 Logs.*writeLog*(**" \n-!- Error: out of bounds -!- \n"** +**"Last successful Point:\n"**+List.getLast().getRealCoordinate().toString()+**"\n"** , Level.***WARNING***);  
 **return** List;  
 } **else** {  
 vert = **new** Vertex(tmp);  
 List.addLast(vert);  
 }  
 pars.setLocation(tmp.getX(), tmp.getY());  
 Logs.*writeLog*(**" -- Start getting data from the server --\n"**, Level.***INFO***);  
 WeatherParameters parameters = pars.getParameters();  
 **int** res = methodTimeOut(TimeInAir, step);  
 **if** (res <= 0) {  
 Logs.*writeLog*(**" \n-- So little step! --\n"** + **" -- Step value we can step on:"** + res+ **"\n\n"**, Level.***WARNING***);  
 **return null**;  
 }  
 **if** (parameters != **null**) {  
 Logs.*writeLog*(**" -- Successful receipt of the parameters --\n"**, Level.***INFO***);  
 vert.setWeatherInPoint(parameters);  
 Logs.*writeLog*(**" -- Start moving the balloon --\n"**, Level.***INFO***);  
 tmp = moveBalloon(parameters, tmp, res);  
 **if** (tmp == **null**) {  
 Logs.*writeLog*(**" -!- ERRORR in the movement of the balloon -!- \n"**+**"Last successful Point:\n"**+List.getLast().getRealCoordinate().toString()+**"\n"**, Level.***WARNING***);  
 **return null**;  
 }  
 Logs.*writeLog*(**" -- The successful relocation of the balloon --\n\n"**, Level.***INFO***);  
 } **else** {  
 Logs.*writeLog*(**" -!- An error retrieving the parameters -!- \n"** + **"Last successful weaher parameter:\n"** + List.getLast().getWeatherInPoint().toString()+**"\n"**, Level.***WARNING***);  
 **return** List;  
 }  
 }  
 Logs.*writeLog*(**" -- A successful exit of the algorithm -- \n"**, Level.***INFO***);  
 **return** List;  
 }  
  
 */\*\*  
 \* Метод isNotEnd - метод, проверяющий достиг ли алгоритм конечной точки в вариантре программы "Полёт к конечной координате",  
 \* причём равенство конца вычисляется с учётом какой то области  
 \*/* **private boolean** isNotEnd(doublePoint tmp, doublePoint End, **double** SizeEpsilon) {  
 **double** Control\_x = End.getX() - (SizeEpsilon / 2);  
 **double** Control\_y = End.getY() - (SizeEpsilon / 2);  
 **return** !coordsIsCorrect(tmp, **new** doublePoint(Control\_x, Control\_y), SizeEpsilon, SizeEpsilon);  
 }  
  
 */\*\*  
 \* Метод AlgorithmEndPoint - алгоритм вычисления конечной точки, при задании пользователем варианта программы "Полёт к конечной координате"  
 \*/* **public** LinkedList<Vertex> AlgorithmEndPoint(doublePoint startPoint, doublePoint endPoint, String startData, **int** startHour, **int** step, **double** SizeEpsilon) **throws** IOException {  
  
 Logs.*writeLog*(**" -- Start alhorithm -- "**, Level.***INFO***);  
 Parsing pars = **new** Parsing(startData, step, startHour);  
 doublePoint tmp = startPoint;  
 WeatherParameters tmpW = **null**;  
 Vertex vert;  
 LinkedList<Vertex> List = **new** LinkedList<>();  
 **while** (isNotEnd(tmp, endPoint, SizeEpsilon)) {  
 Logs.*writeLog*(**" -- NEXT STEP! --\n"**, Level.***INFO***);  
 **if** (!coordsIsCorrect(tmp, **ControlPoint**, **sizeMapLatitude**, **sizeMapLongitude**)) {  
 Logs.*writeLog*(**" \n-!- Error: out of bounds -!- \n"** +**"Last successful Point:\n"**+List.getLast().getRealCoordinate().toString()+**"\n"** , Level.***WARNING***);  
 **return** List;  
 }**else** {  
 vert = **new** Vertex(tmp);  
 List.addLast(vert);  
 }  
 pars.setLocation(tmp.getX(), tmp.getY());  
 Logs.*writeLog*(**" -- Start getting data from the server --\n"**, Level.***INFO***);  
 WeatherParameters parameters = pars.getParameters();  
 tmpW = parameters;  
 **if** (parameters != **null**) {  
 Logs.*writeLog*(**" -- Successful receipt of the parameters --\n"**, Level.***INFO***);  
 vert.setWeatherInPoint(parameters);  
 Logs.*writeLog*(**" -- Start moving the balloon --\n"**, Level.***INFO***);  
 tmp = moveBalloon(parameters, tmp, step);  
 **if** (tmp == **null**) {  
 Logs.*writeLog*(**" -!- ERRORR in the movement of the balloon -!- \n"**+**"Last successful Point:\n"**+List.getLast().getRealCoordinate().toString()+**"\n"**, Level.***WARNING***);  
 **return null**;  
 }  
 Logs.*writeLog*(**" -- The successful relocation of the balloon -- \n\n"**, Level.***INFO***);  
 } **else** {  
 Logs.*writeLog*(**" -!- An error retrieving the parameters -!- \n"** + **"Last successful weaher parameter:\n"** + List.getLast().getWeatherInPoint().toString()+**"\n"**, Level.***WARNING***);  
 **return null**;  
 }  
 }  
 vert = **new** Vertex(endPoint);  
 vert.setWeatherInPoint(tmpW);  
 List.addLast(vert);  
 Logs.*writeLog*(**" -- A successful exit of the algorithm -- \n"**, Level.***INFO***);  
 **return** List;  
 }  
}

6)Файл Parsing.java:

**package** algo;  
  
**import** dateStruct.Time;  
**import** dateStruct.WeatherParameters;  
**import** logger.Logs;  
  
**import** javax.swing.\*;  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.net.HttpURLConnection;  
**import** java.net.URL;  
**import** java.util.Calendar;  
**import** java.util.logging.Level;  
  
**public class** Parsing {  
 */\*\*  
 \* Параметры  
 \* source - источник  
 \* key - ключ(обязательное поле для доступа)  
 \* location - местоположение  
 \* format - формат ответа сайта  
 \* dataStart и dataend -начальные и конечные даты для погоды  
 \* timeReload - время обновления погоды для конкретного дня  
 \*/* **private static final** String ***source*** = **"https://api.worldweatheronline.com/premium/v1/past-weather.ashx?"**;  
 **private static final** String ***key*** = **"key=b60f705b24864ca6bc891344190307"**;  
 *//private static final String key = "key=a0fdc17cd5084361937220857190807";  
 //private static final String key = "key=c1d7ed81edf64f0985d155150191007";* **private** String **location** = **"q=Moscow"**;  
 **private static final** String ***format*** = **"format=json"**;  
 **private** String **dataStart**;  
 **private static final** String ***timeReload*** = **"tp=1"**;  
 **private** HttpURLConnection **connection** = **null**;  
 **private** String **answser**;  
 **private int step**;  
 **private** Time **countTime**;  
 **private** Calendar **calendar** = Calendar.*getInstance*();  
 **private int EndDay** = **calendar**.get(Calendar.***DAY\_OF\_MONTH***);  
  
 **public** Parsing(String data, **int** step, **int** startHour) **throws** IOException {  
 **countTime** = **new** Time(data, startHour);  
 **this**.**step** = step;  
 setDataStart(data);  
 Logs.*writeLog*(**" -- Create parsing class instance successful --"**, Level.***INFO***);  
 }  
  
 **public void** setDataStart(String dataStart) {  
 StringBuilder str = **new** StringBuilder(**"date="**);  
 **this**.**dataStart** = str.append(dataStart).toString();  
 }  
  
 **public void** setLocation(**double** x, **double** y) {  
 StringBuilder str = **new** StringBuilder(**"q="**);  
 **this**.**location** = str.append(x + **","** + y).toString();  
 }  
  
 **public void** setLocation(String location) {  
 StringBuilder str = **new** StringBuilder(**"q="**);  
 **this**.**location** = str.append(location).toString();  
 }  
  
 */\*\*  
 \* Отравление HTTP запроса на сервер API для получения необходимой информации  
 \*/* **private void** getMesWeather() **throws** IOException {  
 String request = **new** String(***source*** + ***key*** + **"&"** + **location** + **"&"** + ***format*** + **"&"** + **dataStart** + **"&"** + ***timeReload***);  
  
 **boolean** flagTry = **false**;  
 **for**(**int** i =0;!flagTry&&i!=2;i++){  
 **try** {  
 **connection** = (HttpURLConnection) **new** URL(request).openConnection();  
 **connection**.setRequestMethod(**"GET"**);  
 **connection**.setConnectTimeout(750);  
 **connection**.setReadTimeout(750);  
 **connection**.connect();  
 StringBuilder sb = **new** StringBuilder();  
  
 **if** (HttpURLConnection.***HTTP\_OK*** == **connection**.getResponseCode()) {  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(**connection**.getInputStream()));  
 String line;  
  
 **while** ((line = in.readLine()) != **null**) {  
 sb.append(line + **"\n"**);  
 }  
 }**else**{  
 Logs.*writeLog*(**" -- Fail"** + **connection**.getResponseCode() + **", "** + **connection**.getResponseMessage() + **" --\n"**, Level.***WARNING***);  
 }  
 **answser** = sb.toString();  
 flagTry = **true**;  
 } **catch** (Throwable cause) {  
 Logs.*writeLog*(**" !--! Fatal error sending HTTP request! !--!\n"**, Level.***SEVERE***);  
 } **finally** {  
 **if** (**connection** != **null**)  
 **connection**.disconnect();  
 }  
 }  
 **if**(!flagTry){  
 JOptionPane.*showMessageDialog*(**null**, **"Подключиться к серверу не удалось! Повторите ещё раз!"**, **"Warning!"**, JOptionPane.***PLAIN\_MESSAGE***);  
 }  
 }  
  
 */\*\*  
 \* Метод getParameters - метода, осуществяющий получение данных о погоде путём отправления HTTTP запроса и осуществяющий парсинг ответа, в случае успеха  
 \*/* **public** WeatherParameters getParameters() **throws** IOException {  
 **if** (**countTime**.getMonth() > 7 && **countTime**.getDay() > **EndDay** - 1) {  
 Logs.*writeLog*(**" -- Error, out of available time! --\n"**+ **"Last day: "**+ **EndDay**+**"\n"**, Level.***WARNING***);  
 */\*  
 \* Возможно сделать более красиво  
 \*/* **return null**;  
 }  
  
 **if** (**countTime**.getHour() >= 24) {  
 **countTime**.setDay(**countTime**.getDay() + 1);  
 **countTime**.setHour(**countTime**.getHour() - 24);  
 String dataNext = **new** String(**countTime**.getYear() + **"-"** + **countTime**.getMonth() + **"-"** + **countTime**.getDay());  
 setDataStart(dataNext);  
 }  
  
 **if** (**countTime**.getMonth() == 5 && **countTime**.getDay() >= 31 || **countTime**.getMonth() == 6 && **countTime**.getDay() >= 30) {  
 **countTime**.setMonth(**countTime**.getMonth() + 1);  
 **countTime**.setDay(1);  
 String dataNext = **new** String(**countTime**.getYear() + **"-"** + **countTime**.getMonth() + **"-"** + **countTime**.getDay());  
 setDataStart(dataNext);  
 }  
  
 **if** (**countTime**.getHour() < 24) {  
 **answser** = **""**;  
 getMesWeather();  
 String find = **"\"time\":\""** + (**countTime**.getHour() \* 100) + **"\""**;  
 **int** index = **answser**.indexOf(find);  
  
 **if** (index < 0) {  
 Logs.*writeLog*(**" -- Error: unable to find any matching weather location to the query submitted! --\n"**, Level.***WARNING***);  
 **return null**;  
 }  
 **answser** = **answser**.substring(index);  
 index = **answser**.indexOf(**"windspeedKmph"**);  
 index += 16;  
  
 StringBuilder strBuf = **new** StringBuilder();  
 **for** (**int** i = index; i < index + 5; i++) {  
 **if** (Character.*isDigit*(**answser**.charAt(i))) {  
 strBuf.append(**answser**.charAt(i));  
 }  
 }  
  
 **int** parameter1 = Integer.*parseInt*(strBuf.toString());  
 index += 21;  
 strBuf.setLength(0);  
 **for** (**int** i = index; i < index + 5; i++) {  
 **if** (Character.*isDigit*(**answser**.charAt(i))) {  
 strBuf.append(**answser**.charAt(i));  
 }  
 }  
  
 **int** parameter2 = Integer.*parseInt*(strBuf.toString());  
 WeatherParameters result = **new** WeatherParameters(parameter1, parameter2);  
 **countTime**.setHour(**countTime**.getHour() + **step**);  
 **return** result;  
 }  
 Logs.*writeLog*(**" -- Error at parsing! --\n"**, Level.***WARNING***);  
 **return null**;  
 }  
}

7)Файл doublePoint.java:

**package** dateStruct;  
  
*/\*\*  
 \* стуктура точка основанная на double  
 \*/***public class** doublePoint {  
 **private double x**;  
 **private double y**;  
  
 **public double** getX() { **return x**;}  
  
 **public double** getY() { **return y**;}  
  
 **public** doublePoint(**double** X\_New, **double** Y\_New){  
 **x** = X\_New;  
 **y** = Y\_New;  
 }  
  
 **public** doublePoint(doublePoint NewPoint){  
 **if** (NewPoint != **null**){  
 **x** = NewPoint.getX();  
 **y** = NewPoint.getY();  
 }  
 }  
  
 @Override  
 **public boolean** equals(Object obj){  
 **if**(**this**==obj){  
 **return true**;  
 }  
 **if**(obj **instanceof** dateStruct.doublePoint){  
 dateStruct.doublePoint otherObj = (dateStruct.doublePoint)obj;  
 **if**(Math.*abs*(otherObj.**x**-**x**)<0.1){  
 **if**(Math.*abs*(otherObj.**y**-**y**)<0.1) **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 @Override  
 **public int** hashCode(){  
 **int** result = 31;  
 **long** longBits1 = Double.*doubleToLongBits*(**x**);  
 **long** longBits2 = Double.*doubleToLongBits*(**y**);  
 result = 11 \* result + (**int**)(longBits1 - (longBits1 >>> 32)) + (**int**)(longBits2 - (longBits2 >>> 32));  
 **return** result;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "X: "**+ String.*format*(**"%.2f"**, **x**) +**" Y: "**+ String.*format*(**"%.2f"**, **y**);  
 }  
}

8)Файл Time.java:

**package** dateStruct;  
  
**import** java.io.IOException;  
  
**public class** Time {  
 **private int year**;  
 **private int month**;  
 **private int day**;  
 **private int hour**;  
  
 **public** Time(String data, **int** hour){  
 **this**(data);  
 **this**.**hour** = hour;  
 }  
  
 **public** Time(String data){  
 String[] split = data.split(**"[-]"**);  
 **year** = Integer.*parseInt*(split[0]);  
 **month** = Integer.*parseInt*(split[1]);  
 **day** = Integer.*parseInt*(split[2]);  
 **hour** = 0;  
 }  
  
 */\*\*  
 \* Конструктор от 3х аргументов необходим для задания времени полёта  
 \* \*/* **public** Time(**int** month, **int** day, **int** hour){  
 **this**.**month** = month;  
 **this**.**day** = day;  
 **this**.**hour** = hour;  
 }  
  
 */\*\*  
 \* Метод TimeNotOut необходим для работы алгоритме в режиме полёта по времени, который вычисляет "закончилось ли время полёта?"  
 \* \*/* **public boolean** TimeNotOut() **throws** IOException {  
 **if**(**month**==0&&**day**==0&&**hour**==0){  
 **return false**;}  
 **else**{  
 **return true**;}  
 }  
  
 **public int** getDay() { **return day**; }  
  
 **public int** getHour() { **return hour**; }  
  
 **public int** getMonth() { **return month**;}  
  
 **public int** getYear() {**return year**; }  
  
 **public void** setDay(**int** day) { **this**.**day** = day; }  
  
 **public void** setHour(**int** hour) { **this**.**hour** = hour; }  
  
 **public void** setMonth(**int** month) {**this**.**month** = month; }  
  
 @Override  
 **public** String toString() {  
 StringBuilder str = **new** StringBuilder();  
 str.append(**"Month:\n"**+ **month**);  
 str.append(**"Day:\n"**+**day**);  
 str.append(**"Hour:\n"**+**hour**);  
 **return** str.toString();  
 }  
}

9)Файл Vertex.java:

**package** dateStruct;  
  
*/\*\*  
 \* Структура хранения данных для алгоритма  
 \* RealCoordinate - координаты в реальном мире  
 \* MapCoordinate - координаты на карте  
 \* PointWeather - Погода в точке  
 \*/***public class** Vertex{  
 **private final** doublePoint **RealCoordinate**;  
 **private** doublePoint **MapCoordinate**;  
 **private** WeatherParameters **weatherInPoint**;  
  
 **public** Vertex(doublePoint coord){  
 **RealCoordinate** = coord;  
 **MapCoordinate** = **null**;  
 **weatherInPoint** = **null**;  
 }  
  
 */\*\*  
 \* устанавливает координаты map координаты  
 \** ***@param mapCoordinate*** *\*/* **public void** setMapCoordinate(doublePoint mapCoordinate) {  
 **MapCoordinate** = mapCoordinate;  
 }  
  
 */\*\*  
 \* Устаналивает значение погоды в точке  
 \** ***@param weatherInPoint*** *\*/* **public void** setWeatherInPoint(WeatherParameters weatherInPoint) {  
 **this**.**weatherInPoint** = weatherInPoint;  
 }  
  
 */\*\*  
 \* возвращает реальный координаты точки  
 \** ***@return*** *\*/* **public** doublePoint getRealCoordinate(){  
 **return RealCoordinate**;  
 }  
  
 */\*\*  
 \* возвращает map координаты точки  
 \** ***@return*** *\*/* **public** doublePoint getMapCoordinate(){  
 **return MapCoordinate**;  
 }  
  
 */\*\*  
 \* возвращает текущую погоду  
 \** ***@return*** *\*/* **public** WeatherParameters getWeatherInPoint(){  
 **return weatherInPoint**;  
 }  
  
 @Override  
 **public boolean** equals(Object obj){  
 **if**(**this**==obj){  
 **return true**;  
 }  
 **if**(obj **instanceof** Vertex){  
 Vertex otherObj = (Vertex)obj;  
 **if**(otherObj.**RealCoordinate**.equals(**RealCoordinate**)){  
 **if**(otherObj.**weatherInPoint**.equals(**weatherInPoint**)) **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 @Override  
 **public int** hashCode(){  
 **int** result = 31;  
 result = 11 \* result + **RealCoordinate**.hashCode() + **weatherInPoint**.hashCode();  
 **return** result;  
 }  
  
 @Override  
 **public** String toString() {  
 StringBuilder str = **new** StringBuilder();  
 str.append(**"real coordinate:\n"**).append(**RealCoordinate**.toString());  
 **if** (**MapCoordinate** != **null**) str.append(**"map coordinate:\n"**).append(**MapCoordinate**.toString());  
 **if** (**weatherInPoint** != **null**) str.append(**"\nWeather parameter in this point:\n"**).append(**weatherInPoint**.toString());  
 **return** str.toString();  
 }  
}

10)Файл WeatherParameters.java:

**package** dateStruct;  
  
**public class** WeatherParameters {  
 */\*\*  
 \* WindGustKmph - Порыв ветра в километрах в час  
 \* WinddirDegree - Направление ветра в градусах  
 \*/* **private final int WindGustKmph**;  
 **private final int WinddirDegree**;  
  
 **public** WeatherParameters(**int** input\_WindGustKmph, **int** input\_WinddirDegree) {  
 **WindGustKmph** = input\_WindGustKmph;  
 **WinddirDegree** = input\_WinddirDegree;  
 }  
  
 **public int** getWinddirDegree() {  
 **return WinddirDegree**;  
 }  
  
 **public int** getWindGustKmph() {  
 **return WindGustKmph**;  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (**this** == obj) {  
 **return true**;  
 }  
 **if** (obj **instanceof** WeatherParameters) {  
 WeatherParameters otherObj = (WeatherParameters) obj;  
 **if** (otherObj.**WindGustKmph** == **WindGustKmph**) {  
 **if** (otherObj.**WinddirDegree** == **WinddirDegree**) **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 @Override  
 **public int** hashCode() {  
 **int** result = 31;  
 result = 11 \* result + **WindGustKmph** + **WinddirDegree**;  
 **return** result;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "A: "** + **WinddirDegree** + **" S: "** + **WindGustKmph**;  
 }  
}