ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

СОГЛАСОВАНО
Профессор департамента
прикладной математики
московского института
электроники и математики
им А.Н. Тихонова,
канд. физ.-мат. наук

_____ / В. Ю. Попов /
«___» _____ 2023 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента
программной инженерии,
канд. техн. наук

_____ / В. В. Шилов /
«___» _____ 2023 г.

Подп. и дат	
Инв. № дубл	
Взам. инв. №	
Подп. и дат	
Инв. № подл	

ПРИЛОЖЕНИЕ ДЛЯ ВИЗУАЛИЗАЦИИ ГЕОИНФОРМАЦИОННЫХ ДАННЫХ

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.10.03-01 12 01-1-ЛУ

Исполнитель:

студент группы БПИ205

_____ / И. А. Шагурин /

«____» _____ 2023 г.

Москва 2023

УТВЕРЖДЕН
RU.17701729.10.03-01 12 01-1-ЛУ

ПРИЛОЖЕНИЕ ДЛЯ ВИЗУАЛИЗАЦИИ ГЕОИНФОРМАЦИОННЫХ ДАННЫХ

Текст программы

RU.17701729.10.03-01 12 01-1

Листов 179

Подп. и дат	
Инв. № дубл	
Взам. инв. №	
Подп. и дат	
Инв. № подл	

Москва 2023

СОДЕРЖАНИЕ

1. ТЕКСТ ПРОГРАММЫ

Программа написана на языке java (использована среда разработки IntelliJ IDEA Community Edition 2022.2.2). Программа состоит из 22 файлов (заимствованный код не учтен).

В данном документе содержится только вручную написанный исходный код программы, заимствованный код не представлен.

1.1. Пакет `controller`

1.1.1. Файл `Controller.java`

```
/*
 * В данном файле содержится реализация контроллера
 * главного фрейма для паттерна MVC.
 */

package controller;

import model.GeoinformationDataUnit;
import model.Model;
import view.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.Timer;

/**
```

```
 * Контроллер главного фрейма для паттерна MVC.
 *
 * @author Иван Шагурин
 */
public class Controller {
    /**
     * Главный фрейм.
     */
    private final MainFrame mainFrame;

    /**
     * Модель для паттерна MVC.
     */
    private final Model model;

    /**
     * Флажок, который поднят, если идет процесс обхода директории,
     * которая содержит файлы с данными для визуализации.
     */
    private static boolean processingFlag = false;

    /**
     * Таймер.
     */
    private Timer timer = new Timer();

    /**
     * Конструктор.
     *
```

```
     * @param mainFrame главный фрейм.
     * @param model модель главного фрейма для паттерна MVC.
     */
    public Controller(MainFrame mainFrame, Model model) {
        this.mainFrame = mainFrame;
        this.model = model;
    }

    /**
     * Добавляем элементам GUI обработчики событий.
     */
    public void addEventListeners() {
        VisualizationMouseAdapter visualizationMouseAdapter
                = new VisualizationMouseAdapter(this);

        mainFrame.visualizationComponent
                .addMouseListener(visualizationMouseAdapter);
        mainFrame.visualizationComponent
                .addMouseMotionListener(visualizationMouseAdapter);
        mainFrame.visualizationComponent
                .addMouseWheelListener(visualizationMouseAdapter);

        mainFrame.selectShownFilesTypeButton.addMouseListener(new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                FileTypeFrame fileTypeFrame = new
FileTypeFrame(mainFrame);
                fileTypeFrame.setVisible(true);
            }
```

```
        });

        mainFrame.selectFolderButton
                .addMouseListener(
                        new SelectFolderButtonMouseAdapter(mainFrame,
model));

        mainFrame.selectDefaultFolderButton.addMouseListener(
                new SelectDefaultFolderButtonMouseAdapter(mainFrame,
model));

        mainFrame.gotoPreviousFileButton.addMouseListener(new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (model.geoinformationDataUnits.size() == 0) {
                    return;
                }

                model.gotoPreviousFile();
            }
        });

        mainFrame.gotoNextFileButton.addMouseListener(new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (model.geoinformationDataUnits.size() == 0) {
                    return;
                }
```

```
                model.gotoNextFile();
            }
        });

        mainFrame.defaultOffsetAndScaleButton.addMouseListener(new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {

mainFrame.visualizationComponent.display.adjustOffsetAndSizes(mainFram
e.visualizationComponent);
                mainFrame.visualizationComponent.repaint();
            }
        });

        mainFrame.fileNumberSlider
                .addChangeListener(e ->
model.gotoCertainFile(mainFrame.fileNumberSlider.getValue()));

        mainFrame.gotoFileButton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (model.geoinformationDataUnits.size() == 0) {
                    return;
                }

                int fileNumber;

                try {
```

```
                    fileNumber =
Integer.parseInt(mainFrame.fileNumberTextField.getText());
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null,
                            "Номер файла введен некорректно.");
                    return;
                }

                if ((fileNumber < 1) || (fileNumber >
model.getFilesSize())) {
                    JOptionPane.showMessageDialog(null,
                            "Номер файла должен быть в диапазоне от 1
до "
                                    + model.getFilesSize()
                                    + " включительно.");
                    return;
                }

                model.gotoCertainFile(fileNumber - 1);
                mainFrame.fileNumberTextField.setText("");
            }
        });

        mainFrame.changeColorIndicatorLimitButton.addMouseListener(new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                try {
                    float newColorIndicatorLimit
                            =
Float.parseFloat(mainFrame.colorIndicatorLimitTextField.getText());
```

```
                    if (newColorIndicatorLimit <= 0) {
                        JOptionPane.showMessageDialog(
                                null,
                                "Предельное значение цветового
индикатора должно быть положительным.");
                        return;
                    }


mainFrame.colorIndicatorLimitTextField.setText("");
                    GeoinformationDataUnit.maxValue =
newColorIndicatorLimit;
                    mainFrame.visualize();
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(
                            null,
                            "Новое предельное значение цветового
индикатора введено некорректно.");
                }
            }
        });


        mainFrame.selectDateAndTimeButton.addMouseListener(new
MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (model.geoinformationDataUnits.size() == 0) {
                    return;
                }
```

```
                String[] splitResult =
mainFrame.timeTextField.getText().split(":");


                if (splitResult.length != 2) {
                    JOptionPane.showMessageDialog(null, "Время указано
некорректно.");
                    return;
                }


                int hour;
                int minute;


                try {
                    hour = Integer.parseInt(splitResult[0]);
                    minute = Integer.parseInt(splitResult[1]);
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null, "Время указано
некорректно.");
                    return;
                }


                if ((hour < 0) || (hour > 23) || (minute < 0) ||
(minute > 59)) {
                    JOptionPane.showMessageDialog(null, "Время указано
некорректно.");
                    return;
                }


                int year =
mainFrame.datePickerUtil.sqlDateModel.getYear();
```

```
                int month =
mainFrame.datePickerUtil.sqlDateModel.getMonth();
                int day =
mainFrame.datePickerUtil.sqlDateModel.getDay();


                Calendar calendar = Calendar.getInstance();


                calendar.set(Calendar.YEAR, year);
                calendar.set(Calendar.MONTH, month);
                calendar.set(Calendar.DAY_OF_MONTH, day);
                calendar.set(Calendar.HOUR_OF_DAY, hour);
                calendar.set(Calendar.MINUTE, minute);


                model.gotoFileWithSelectedDateAndTime(calendar);
            }
        });


        mainFrame.paintMarginCheckBox.addActionListener(e -> {
            model.showMarginFlag =
mainFrame.paintMarginCheckBox.isSelected();
            mainFrame.visualize();
        });


        mainFrame.paintHeatmapCheckBox.addActionListener(e -> {
            model.showHeatmapFlag =
mainFrame.paintHeatmapCheckBox.isSelected();
            mainFrame.visualize();
        });


        mainFrame.smoothMarginCheckBox.addActionListener(e -> {
```

```
            model.smoothMarginFlag =
mainFrame.smoothMarginCheckBox.isSelected();
            mainFrame.visualize();
        });

        mainFrame.specifyMarginLevelButton.addActionListener(e -> {
            try {

                mainFrame.model.marginLevel
                        =
Float.parseFloat(mainFrame.marginLevelTextField.getText());
                mainFrame.visualize();
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(
                        null,
                        "Новый уровень границы указан некорректно.");
            }
        });

        mainFrame.changeWaitingTimeButton.addActionListener(e -> {
            try {
                int waitingTimeInSeconds
                        =
Integer.parseInt(mainFrame.changeWaitingTimeTextField.getText());

                if (waitingTimeInSeconds < 0) {
                    JOptionPane.showMessageDialog(null,
                            "Время ожидания не может быть
отрицательным.");
                    return;
```

```
                }


                mainFrame.model.waitingTimeInSeconds =
waitingTimeInSeconds;


                boolean automaticDataDownloadFlag =
mainFrame.automaticDataDownloadCheckBox.isSelected();
                timer.cancel();
                timer = new Timer();
                mainFrame.changeWaitingTimeTextField.setText("");
                mainFrame.visualize();
                if (automaticDataDownloadFlag) {
                    if
(mainFrame.automaticDataDownloadCheckBox.isSelected()) {
                        timer.schedule(new TimerTask() {
                            @Override
                            public void run() {
                                if (model.selectedFolder == null) {

JOptionPane.showMessageDialog(null,
                                                "Папка для загрузки данных
не выбрана.");
                                    return;
                                }


doFilesProcessing(model.selectedFolder, mainFrame, model);
                            }
                        }, model.waitingTimeInSeconds * 1000L,
model.waitingTimeInSeconds * 1000L);
                    } else {
```

```
                        timer.cancel();
                        timer = new Timer();
                    }
                }
            } catch (NumberFormatException exception) {
                JOptionPane.showMessageDialog(null,
                        "Новое время ожидания указано некорректно.");
            }
        });

        mainFrame.automaticDataDownloadCheckBox.addActionListener(e ->
{
            automaticDataDownloadCheckBoxListener();
        });
    }

    /**
     * Адаптер обработки событий мыши для компоненты для визуализации.
     */
    static class VisualizationMouseAdapter extends MouseAdapter {
        /**
         * Флажок, который поднят, если активен режим перемещения
         * изображения компоненты для визуализации.
         */
        private boolean inMove = false;

        /**
         * Контроллер главного фрейма для паттерна MVC.
         */
```

```
        private final Controller controller;


        /**
         * Координата x нажатия мыши на компоненту для визуализации.
         */
        int x;


        /**
         * Координата y нажатия мыши на компоненту для визуализации.
         */
        int y;


        /**
         * Конструктор.
         *
         * @param controller контроллер главного фрейма для паттерна
MVC.
         */
        public VisualizationMouseAdapter(Controller controller) {
            this.controller = controller;
        }


        /**
         * Обработчик события: "мышь зажата".
         *
         * @param e the event to be processed
         */
        @Override
        public void mousePressed(MouseEvent e) {
```

```
        if (e.getButton() == MouseEvent.BUTTON3) {
            inMove = true;
            x = e.getX();
            y = e.getY();
        } else if (e.getButton() == MouseEvent.BUTTON1) {
            controller.model.showInfoFlag = true;
            controller.model.mouseAdapterX = e.getX();
            controller.model.mouseAdapterY = e.getY();

            controller.mainFrame.visualizationComponent.repaint();
        }
    }

    /**
     * Обработчик события перетаскимвания зажатой мыши.
     *
     * @param e the event to be processed
     */
    @Override
    public void mouseDragged(MouseEvent e) {
        if (inMove) {
            int deltaX = e.getX() - x;
            int deltaY = e.getY() - y;

            x = e.getX();
            y = e.getY();

            controller.mainFrame.visualizationComponent
                    .display.changeOffset(deltaX, deltaY);
```

```
            controller.mainFrame.visualizationComponent.repaint();
        } else if (controller.model.showInfoFlag) {
            controller.model.mouseAdapterX = e.getX();
            controller.model.mouseAdapterY = e.getY();

            controller.mainFrame.visualizationComponent.repaint();
        }
    }

    /**
     * Обработчик события: "мышь отжата".
     *
     * @param e the event to be processed
     */
    @Override
    public void mouseReleased(MouseEvent e) {
         inMove = false;
         controller.model.showInfoFlag = false;

         controller.mainFrame.visualizationComponent.repaint();
    }

    /**
     * Обработчик события: "колесико мыши прокрутилось".
     *
     * @param e the event to be processed
     */
    @Override
    public void mouseWheelMoved(MouseWheelEvent e) {
```

```
            Display display =
controller.mainFrame.visualizationComponent.display;
            int a1 = e.getX() - display.getOffsetX();
            int c1 = e.getY() - display.getOffsetY();
            float alpha;

            if (e.getWheelRotation() < 0) {
                if (!display.canIncreaseSizes()) {
                    return;
                }

                display.increaseSizes();
                alpha = Display.INCREASE_SIZE_RATIO;
            } else {
                if (!display.canDecreaseSizes()) {
                    return;
                }

                display.decreaseSizes();
                alpha = Display.DECREASE_SIZE_RATIO;
            }

            display.changeOffset((int) ((1 - alpha) * a1), (int) ((1 -
alpha) * c1));

            controller.mainFrame.visualizationComponent.repaint();
        }
    }

    /**
```

```
     * Адаптер обработки событий мыши для кнопки выбора папки.
     */
    static class FolderButtonMouseAdapter extends MouseAdapter {
        /**
         * Главный фрейм.
         */
        protected MainFrame mainFrame;

        /**
         * Модель главного фрейма.
         */
        protected Model model;

        /**
         * Конструктор.
         *
         * @param mainFrame главный фрейм.
         * @param model модель главного фрейма.
         */
        public FolderButtonMouseAdapter(MainFrame mainFrame, Model
model) {
            this.mainFrame = mainFrame;
            this.model = model;
        }
    }

    /**
     * Адаптер обработки событий для кнопки выбора папки вручную.
     */
```

```
    static class SelectFolderButtonMouseAdapter extends
FolderButtonMouseAdapter {

        /**

         * Конструктор.

         *

         * @param mainFrame главный фрейм.

         * @param model модель главного фрейма.

         */

        public SelectFolderButtonMouseAdapter(MainFrame mainFrame,
Model model) {

            super(mainFrame, model);

        }


        /**

         * Обработчик события: "мышь щелкнута".

         *

         * @param e the event to be processed

         */

        @Override

        public void mouseClicked(MouseEvent e) {

            if (processingFlag) {

                return;

            }


            JFileChooser fileChooser = new JFileChooser();


fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);

            int result = fileChooser.showOpenDialog(mainFrame);


            if (result == JFileChooser.APPROVE_OPTION) {
```

```
                mainFrame.model.selectedFolder =
fileChooser.getSelectedFile();

mainFrame.selectedFolderTextField.setText(fileChooser.getSelectedFile(
).getAbsolutePath());

                new Thread(() -> {
                    doFilesProcessing(fileChooser.getSelectedFile(),
mainFrame, model);
                }).start();
            }
        }
    }

    /**
     * Адаптер обработки событий для кнопки выбора папки по умолчанию.
     */
    static class SelectDefaultFolderButtonMouseAdapter extends
FolderButtonMouseAdapter {
        /**
         * Конструктор.
         *
         * @param mainFrame главный фрейм.
         * @param model модель главного фрейма.
         */
        public SelectDefaultFolderButtonMouseAdapter(MainFrame
mainFrame, Model model) {
            super(mainFrame, model);
        }

        /**
```

```
         * Обработчик события: "мышь щелкнута".
         *
         * @param e the event to be processed
         */
        @Override
        public void mouseClicked(MouseEvent e) {
            BufferedReader reader;

            try {
                reader = new BufferedReader(new
FileReader("default.txt", StandardCharsets.UTF_8));
                String line = reader.readLine();
                File folder;

                try {
                    folder = new File(line);
                } catch (Exception exception) {
                    JOptionPane.showMessageDialog(null,
                            "Ошибка при попытке открыть папку по
умолчанию.");
                    return;
                }

                try {
                    mainFrame.model.selectedFolder = folder;

mainFrame.selectedFolderTextField.setText(folder.getAbsolutePath());

                    new Thread(() -> {
                        doFilesProcessing(folder, mainFrame, model);
```

```
                    }).start();
                } catch (Exception exception) {
                    JOptionPane.showMessageDialog(null,
                            "Ошибка при попытке открыть папку по
умолчанию.");
                }

                reader.close();
            } catch (FileNotFoundException exception) {
                JOptionPane.showMessageDialog(null,
                        "Папка по умолчанию не обнаружена.");
                File file = new File("default.txt");
                try {
                    file.createNewFile();
                } catch (IOException ignored) {
                }
            } catch (IOException exception) {
                JOptionPane.showMessageDialog(null,
                        "Ошибка при попытке открыть папку по
умолчанию.");
            }
        }
    }

    /**
     * Обход дерева папок с целью формирования коллекции .txt файлов,
     * включая подготовительные и заключительные действия.
     *
     * @param file корень дерева папок.
     * @param mainFrame главный фрейм.
```

```
     * @param model модель.
     */
    private static synchronized void doFilesProcessing(File file,
MainFrame mainFrame, Model model) {
        if (file == null) {
            return;
        }

        mainFrame.selectFolderButton.setEnabled(false);
        mainFrame.selectDefaultFolderButton.setEnabled(false);
        processingFlag = true;
        mainFrame.progressBar.setVisible(true);

mainFrame.container.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_C
URSOR));
        model.clear();
        int numberOfTXTFiles = 500000;
        mainFrame.progressBar.setMinimum(0);
        mainFrame.progressBar.setMaximum(numberOfTXTFiles);
        processFilesFromFolder(file, mainFrame, model);
        model.selectFiles();

mainFrame.container.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAUL
T_CURSOR));
        mainFrame.progressBar.setValue(0);
        mainFrame.progressBar.setVisible(false);
        processingFlag = false;
        mainFrame.selectFolderButton.setEnabled(true);
        mainFrame.selectDefaultFolderButton.setEnabled(true);
        model.gotoCertainFile(model.getFilesSize() - 1);
    }
```

```
    /**
     * Формируем коллекцию .txt файлов через обход дерева папок.
     *
     * @param folder корень коллекции.
     * @param mainFrame главный фрейм.
     * @param model модель.
     */
    private static void processFilesFromFolder(File folder, MainFrame
mainFrame, Model model)
    {
        File[] folderEntries = folder.listFiles();

        if (folderEntries == null) {
            return;
        }

        Arrays.sort(folderEntries);

        for (File entry : folderEntries)
        {
            if (entry.isDirectory())
            {
                processFilesFromFolder(entry, mainFrame, model);
                continue;
            }

            Optional<String> fileExtension =
getExtensionByStringHandling(entry.getName());
```

```
            if (fileExtension.isPresent() &&
fileExtension.get().equals("txt")) {
                model.distributeFile(entry);

mainFrame.progressBar.setValue(mainFrame.progressBar.getValue() + 1);
            }
        }
    }

    /**
     * Получаем расширение файла по его имени.
     *
     * @param filename имя файла.
     * @return расширение файла.
     */
    public static Optional<String> getExtensionByStringHandling(String
filename) {
        return Optional.ofNullable(filename)
                .filter(f -> f.contains("."))
                .map(f -> f.substring(filename.lastIndexOf(".") + 1));
    }

    public void automaticDataDownloadCheckBoxListener() {
        if (mainFrame.automaticDataDownloadCheckBox.isSelected()) {
            timer.schedule(new TimerTask() {
                @Override
                public void run() {
                    if (model.selectedFolder == null) {
                        JOptionPane.showMessageDialog(null,
```

```
                    "Папка для загрузки данных не
выбрана.");
                return;
            }

            doFilesProcessing(model.selectedFolder, mainFrame,
model);
        }
    }, model.waitingTimeInSeconds * 1000L,
model.waitingTimeInSeconds * 1000L);
} else {
    timer.cancel();
    timer = new Timer();
}
}
}
```

1.2. Пакет `main`

1.2.1. Файл `Main.java`

```
/*
 * В данном файле содержится точка входа в приложение.
 */

package main;

import view.MainFrame;

/**
 * Класс с точкой входа в приложение.
```

```
 *
 * @author Иван Шагурин
 */
public class Main {
    /**
     * Точка входа в приложение.
     *
     * @param args аргументы командной строки.
     */
    public static void main(String[] args) {
        MainFrame mainFrame = new MainFrame();
        mainFrame.setVisible(true);
    }
}
```

1.3. Пакет `model`

1.3.1. Файл `CastType.java`

```
/*
 * В данном файле содержится перечисление с типом
 * прогноза данных Ovation Prime.
 */

package model;

/**
 * Перечисление с типом прогноза данных Ovation Prime.
 *
 * @author Иван Шагурин
```

```
 */
public enum CastType {
    /**
     * Прогноз данных Ovation Prime.
     */
    FORECAST,
    /**
     * Наблюдаемые данные Ovation Prime.
     */
    NOWCAST
}
```

1.3.2. Файл `EnergyType.java`

```
/*
 * В данной файле содержится перечисление с типом энергии,
 * которому относятся данные Ovation Prime.
 */

package model;

/**
 * Перечисление с типом энергии, к которому
 * относятся данные Ovation Prime.
 */
public enum EnergyType {
    /**
     * Данные о вкладе рассеянного сияния.
     */
```

```
    DIFFUSE,
    /**
     * Данные о вкладе ионов.
     */
    IONS,
    /**
     * Данные о вкладе моноэнергетических пиков.
     */
    MONO,
    /**
     * Данные о вкладе "broadband" ускорения.
     */
    WAVE,
    /**
     * Данные об общем вкладе авроральных компонент.
     */
    TOTAL
}
```

1.3.3. Файл `FileType.java`

```
/*
 * В данном файле содержится класс с информацией о выбранном
 * типе файлов для визуализации.
 */

package model;

/**
```

```
 * Класс с информацией о выбранном типе файлов для визуализации.
 *
 * @param horizonSideType перечисление с типом полусферы.
 * @param castType перечисление с типом прогноза данных Ovation Prime.
 * @param energyType перечисление с типом энергии, к которому
относятся
 *                   данные Ovation Prime.
 *
 * @author Иван Шагурин
 */
public record FileType(HorizonSideType horizonSideType,
                       CastType castType,
                       EnergyType energyType) {
    /**
     * Получаем строковое представление класса.
     *
     * @return описанное строковое представление.
     */
    public String toString() {
        return getHorizonTypeComponent()
                + getCastTypeComponent()
                + getEnergyTypeComponent()
                + "energy flux";
    }

    /**
     * Получаем строковое представление типа полусферы.
     *
     * @return описанное строковое представление.
```

```
 */
private String getHorizonTypeComponent() {
    if (horizonSideType == HorizonSideType.NORTH) {
        return "north ";
    } else if (horizonSideType == HorizonSideType.SOUTH) {
        return "south ";
    } else {
        return "";
    }
}

/**
 * Получаем строковое представление типа прогноза.
 *
 * @return описаное строковое представление.
 */
private String getCastTypeComponent() {
    if (castType == CastType.FORECAST) {
        return "forecast ";
    } else if (castType == CastType.NOWCAST) {
        return "nowcast ";
    } else {
        return "";
    }
}

/**
 * Получаем строковое представление типа энергии.
 *
```

```
     * @return описанное строковое представление.
     */
    private String getEnergyTypeComponent() {
        if (energyType == EnergyType.DIFFUSE) {
            return "diffuse ";
        } else if (energyType == EnergyType.IONS) {
            return "ions ";
        } else if (energyType == EnergyType.MONO) {
            return "mono ";
        } else if (energyType == EnergyType.WAVE) {
            return "wave ";
        } else if (energyType == EnergyType.TOTAL) {
            return "";
        } else {
            return "";
        }
    }
}
```

1.3.4. Файл GeoinformationDataUnit.java

```
/**
 * В данном файле содержится реализация единицы геоинформационных данных.
 */

package model;

import view.PointF;
import view.RadiusVector;
```

```
import java.awt.*;
import java.util.Arrays;

/**
 * Единица геоинформационных данных.
 *
 * @author Иван Шагурин
 */
public class GeoinformationDataUnit {
    /**
     * Первый граничный цвет для цветового индикатора.
     */
    private final static Color FIRST_COLOR = new Color(255, 0, 0);

    /**
     * Второй граничный цвет для цветового индикатора.
     */
    private final static Color SECOND_COLOR = new Color(255, 165, 0);

    /**
     * Третий граничный цвет для цветового индикатора.
     */
    private final static Color THIRD_COLOR = new Color(255, 255, 0);

    /**
     * Четвертый граничный цвет для цветового индикатора.
     */
    private final static Color FOURTH_COLOR = new Color(0, 255, 0);
```

```
    /**
     * Пятый граничный цвет для цветового индикатора.
     */
    private final static Color FIFTH_COLOR = new Color(0, 0, 255);

    /**
     * Шестой граничный цвет для цветового индикатора.
     */
    private final static Color SIXTH_COLOR = new Color(100, 0, 100);

    /**
     * Седьмой граничный цвет для цветового индикатора.
     */
    private final static Color SEVENTH_COLOR = new Color(0, 0, 0);

    /**
     * Значение единицы геоинформационных данных, которое
соответствует
     * первому граничному цвету для цветового индикатора.
     */
    public static float maxValue = 1.5f;

    /**
     * Полярный угол единицы геоинформационных данных.
     */
    private final float polarAngle;

    /**
```

```
     * Полярное расстояние единицы геоинформационных данных.
     */
    private final float polarDistance;

    /**
     * Значение единицы геоинформационных данных.
     */
    private final float value;

    /**
     * Конструктор.
     *
     * @param polarAngle полярный угол.
     * @param polarDistance полярное расстояние.
     * @param value значение.
     */
    public GeoinformationDataUnit(
            float polarAngle,
            float polarDistance,
            float value
    ) {
        this.polarAngle = polarAngle;
        this.polarDistance = polarDistance;
        this.value = value;
    }

    /**
     * Получаем координаты единицы геоинформационных данных.
     *
```

```
     * @return упомянутые координаты.
     */
    public GeoinformationDataUnitCoordinates getCoordinates() {
        return new GeoinformationDataUnitCoordinates(
                polarAngle,
                polarDistance
        );
    }

    /**
     * Получаем полярный угол.
     *
     * @return полярный угол.
     */
    public float getPolarAngle() {
        return polarAngle;
    }

    /**
     * Получаем полярный угол в радианах.
     *
     * @return полярный угол в радианах.
     */
    public float getStandardPolarAngle() {
        return (float)((polarAngle - 6) * Math.PI / 12);
    }

    /**
     * Получаем полярное расстояние.
```

```
     *
     * @return полярное расстояние.
     */
    public float getPolarDistance() {
        return polarDistance;
    }

    /**
     * Получаем стандартное полярное расстояние.
     *
     * @param displayWidth ширина экрана.
     * @return стандартное полярное расстояние.
     */
    public float getStandardPolarDistance(int displayWidth) {
        return (90 - polarDistance) / 80 * displayWidth;
    }

    /**
     * Получаем стандартную координату x.
     *
     * @param displayWidth ширина экрана.
     * @return стандартная координата x.
     */
    public float getStandardX(int displayWidth) {
        return (float)(getStandardPolarDistance(displayWidth)
                * Math.cos(getStandardPolarAngle()));
    }

    /**
```

```
     * Получаем стандартную координату y.
     *
     * @param displayWidth ширина экрана.
     * @return стандартная координата y.
     */
    public float getStandardY(int displayWidth) {
        return (float)(getStandardPolarDistance(displayWidth)
                * Math.sin(getStandardPolarAngle()));
    }

    /**
     * Получаем значение.
     *
     * @return значение.
     */
    public float getValue() {
        return value;
    }

    /**
     * Получаем цвет.
     *
     * @return цвет.
     */
    public Color getColor() {
        return getColor(value);
    }

    /**
```

```
     * Получаем первую вершину четырехугольника, который
     * приближает изображение единицы геоинформационных данных.
     *
     * @param displayWidth ширина дисплея.
     * @return упомянутая вершина.
     */
    public PointF getFirstPoint(int displayWidth) {
        float x1 = getStandardX(displayWidth);
        float y1 = getStandardY(displayWidth);

        return new PointF(x1, y1);
    }

    /**
     * Получаем вторую вершину четырехугольника, который
     * приближает изображение единицы геоинформационных данных.
     *
     * @param displayWidth ширина дисплея.
     * @return упомянутая вершина.
     */
    public PointF getSecondPoint(int displayWidth) {
        PointF firstPoint = getFirstPoint(displayWidth);
        RadiusVector radiusVector
                = new RadiusVector(0, 0, firstPoint.x, firstPoint.y);
        radiusVector.applyRotation(Math.PI / 48);

        float x2 = radiusVector.getArrowheadX();
        float y2 = radiusVector.getArrowheadY();
```

```
        return new PointF(x2, y2);
    }

    /**
     * Получаем третью вершину четырехугольника, который
     * приближает изображение единицы геоинформационных данных.
     *
     * @param displayWidth ширина дисплея.
     * @return упомянутая вершина.
     */
    public PointF getThirdPoint(int displayWidth) {
        PointF secondPoint = getSecondPoint(displayWidth);
        RadiusVector radiusVector
                = new RadiusVector(0, 0, secondPoint.x,
secondPoint.y);

        double l0 = displayWidth / 160f;
        double ratio = (radiusVector.getLength() - l0) /
radiusVector.getLength();

        radiusVector.applyHomothety(ratio);

        float x3 = radiusVector.getArrowheadX();
        float y3 = radiusVector.getArrowheadY();

        return new PointF(x3, y3);
    }

    /**
     * Получаем четвертую вершину четырехугольника, который
```

```
     * приближает изображение единицы геоинформационных данных.
     *
     * @param displayWidth ширина дисплея.
     * @return упомянутая вершина.
     */
    public PointF getFourthPoint(int displayWidth) {
        PointF thirdPoint = getThirdPoint(displayWidth);
        RadiusVector radiusVector
                = new RadiusVector(0, 0, thirdPoint.x, thirdPoint.y);

        radiusVector.applyRotation(-Math.PI / 48);

        float x4 = radiusVector.getArrowheadX();
        float y4 = radiusVector.getArrowheadY();

        return new PointF(x4, y4);
    }

    /**
     * Получаем цвет, который соответсвтует указанному значению.
     *
     * @param value указанное значение.
     * @return цвет, который соответсвтует указанному значению.
     */
    public static Color getColor(double value) {
        if (value > maxValue) {
            return FIRST_COLOR;
        } else if (value > maxValue / 3 * 2.5) {
            return getColorGradient(
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.10.03-01 12 01-1				
Инв. № подл.	Подп. и	Взам. инв.	Инв. №	Подп. и

```
                maxValue / 3 * 2.5, SECOND_COLOR,
                maxValue, FIRST_COLOR,
                value
        );
    } else if (value > maxValue / 3 * 2) {
        return getColorGradient(
                maxValue / 3 * 2, THIRD_COLOR,
                maxValue / 3 * 2.5, SECOND_COLOR,
                value
        );
    } else if (value > maxValue / 3 * 1.5) {
        return getColorGradient(
                maxValue / 3 * 1.5, FOURTH_COLOR,
                maxValue / 3 * 2, THIRD_COLOR,
                value
        );
    } else if (value > maxValue / 3) {
        return getColorGradient(
                maxValue / 3, FIFTH_COLOR,
                maxValue / 3 * 1.5, FOURTH_COLOR,
                value
        );
    } else if (value > maxValue / 3 * 0.05) {
        return getColorGradient(
                maxValue / 3 * 0.05, SIXTH_COLOR,
                maxValue / 3, FIFTH_COLOR,
                value
        );
    } else if (value > 0) {
```

```
            return getColorGradient(
                    0, SEVENTH_COLOR,
                    maxValue / 3 * 0.05, SIXTH_COLOR,
                    value
            );
        } else {
            return SEVENTH_COLOR;
        }
    }

    /**
     * Имеется отрезок на оси действительных чисел. Его концам
     * соответствуют цвета. Также имеется точка. Мы возвращаем
     * градиентный цвет, который определяется отношением, в котором
     * указанная точка делит указанный отрезок.
     *
     * @param leftBound левая граница отрезка.
     * @param leftBoundColor цвет левой границы отрезка.
     * @param rightBound правая граница отрезка.
     * @param rightBoundColor цвет правой границы отрезка.
     * @param intermediateValue точка на отрезке.
     * @return градиентный цвет, который соответствует точке на
отрезке.
     */
    private static Color getColorGradient(
            double leftBound, Color leftBoundColor,
            double rightBound, Color rightBoundColor,
            double intermediateValue
    ) {
```

```
        if (intermediateValue <= leftBound) {
            return leftBoundColor;
        } else if (intermediateValue >= rightBound) {
            return rightBoundColor;
        }

        double lambda = (intermediateValue - leftBound)
                / (rightBound - intermediateValue);

        double red = (leftBoundColor.getRed() + lambda *
rightBoundColor.getRed())
                / (1 + lambda);
        double green = (leftBoundColor.getGreen() + lambda *
rightBoundColor.getGreen())
                / (1 + lambda);
        double blue = (leftBoundColor.getBlue() + lambda *
rightBoundColor.getBlue())
                / (1 + lambda);

        return new Color((int)red, (int)green, (int)blue);
    }

    /**
     * Получаем единицу геоинформационных данных по ее
     * строковому представлению.
     *
     * @param str строковое представление единицы геоинформационных
данных.
     * @return полученная единица геоинформационных данных.
     */
```

```
    public static GeoinformationDataUnit parse(String str) {
        String[] splitResult = Arrays.stream(str.split("
")).filter(str_ -> !str_.isEmpty()).toArray(String[]::new);

        if (splitResult.length < 3) {
            throw new IllegalArgumentException();
        }

        float polarAngle = Float.parseFloat(splitResult[0]);
        float polarDistance = Float.parseFloat(splitResult[1]);
        float value = Float.parseFloat(splitResult[2]);

        return new GeoinformationDataUnit(polarAngle, polarDistance,
value);
    }
}
```

1.3.5. Файл GeoinformationDataUnitCoordinates.java

```
/*
 * В данном файле содержится реализация класса,
 * который соответствует координатам единицы геоинформационных
 * данных.
 */

package model;

import java.util.Objects;

/**
 * Класс, который соответствует координатам единицы
 * геоинформационных данных.
 *
 * @param polarAngle полярный угол.
 * @param polarDistance полярное расстояние.
 */
public record GeoinformationDataUnitCoordinates(
```

```
        float polarAngle,
        float polarDistance
) {
    /**
     * Выполняем проверку на равенство данного класса и аргумента.
     *
     * @param obj   объект для сравнения.
     * @return true, если равенство истинно. Иначе возвращаем false.
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        } else if (obj == null) {
            return false;
        } else if (getClass() != obj.getClass()) {
            return false;
        }

        GeoinformationDataUnitCoordinates other
                = (GeoinformationDataUnitCoordinates) obj;

        return (polarAngle == other.polarAngle)
                && (polarDistance == other.polarDistance);
    }

    /**
     * Получаем хэш-код.
     *
     * @return упомянутый хэш-код.
     */
    @Override
    public int hashCode() {
        return Objects.hash(polarAngle, polarDistance);
    }
}
```

1.3.6. Файл `HorizonSideType.java`

```
/*
 * В данном файле содержится перечисление с типом полусферы,
 * к которой относятся данные Ovation Prime.
 */

package model;

/**
```

```
 * Перечисление с типом полусферы, к которой относятся
 * данные Ovation prime.
 *
 * @author Иван Шагурин
 */
public enum HorizonSideType {
    /**
     * Северная полусфера.
     */
    NORTH,

    /**
     * Южная полусфера.
     */
    SOUTH
}
```

1.3.7. Файл `Model.java`

```
/*
 * В данном файле содержится реализация модели главного фрейма
 * для паттерна MVC.
 */

package model;

import time.DateAndTimeUtil;
import view.MainFrame;
import view.PointF;

import javax.swing.*;
import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
```

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.List;

/**
 * Модель главного фрейма для паттерна MVC.
 */
public class Model {
    /**
     * Индекс строки, с которой начинается чтение данных о
     * единицах геоинформационных данных.
     */
    private static final int START_INDEX = 5;

    /**
     * Индекс строки, на которой заканчивается (не включительно)
     * чтение данных о единицах геоинформационных данных.
     */
    private static final int END_INDEX = 7684;

    /**
     * Класс с информацией о выбранном типе файлов для визуализации.
     */
    public FileType fileType = new FileType(HorizonSideType.NORTH,
CastType.NOWCAST, EnergyType.TOTAL);
```

```
/**
 * Индекс текущего файла.
 */
private int currentFileIndex = 0;

/**
 * Главный фрейм.
 */
public final MainFrame mainFrame;

/**
 * Список имен файлов с данными Ovation Prime, которые относятся
 * к северной полусфере, прогнозу, общему вкладу
 * авроральных компонент.
 */
private final ArrayList<File> northForecastEnergyFluxFiles
        = new ArrayList<>();

/**
 * Список имен файлов с данными Ovation Prime, которые относятся
 * к северной полусфере, наблюдаемым данным, общему вкладу
 * авроральных компонент.
 */
private final ArrayList<File> northNowcastEnergyFluxFiles
        = new ArrayList<>();

/**
 * Список имен файлов с данными Ovation Prime, которые относятся
```

```
     * к южной полусфере, прогнозу, общему вкладу
     * авроральных компонент.
     */
    private final ArrayList<File> southForecastEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, наблюдаемым данным, общему вкладу
     * авроральных компонент.
     */
    private final ArrayList<File> southNowcastEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, прогнозу, вкладу рассеянного сияния.
     */
    private final ArrayList<File> northForecastDiffuseEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, прогнозу, вкладу ионов.
     */
    private final ArrayList<File> northForecastIonsEnergyFluxFiles
            = new ArrayList<>();

    /**
```

```
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, прогнозу, вкладу моноэнергетических
пиков.
     */
    private final ArrayList<File> northForecastMonoEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, прогнозу, вкладу "broadband" ускорения.
     */
    private final ArrayList<File> northForecastWaveEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, наблюдаемым данным, вкладу рассеянного
сияния.
     */
    private final ArrayList<File> northNowcastDiffuseEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, наблюдаемым данным, вкладу ионов.
     */
    private final ArrayList<File> northNowcastIonsEnergyFluxFiles
            = new ArrayList<>();

    /**
```

```
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, наблюдаемым данным, вкладу
моноэнергетических пиков.
     */
    private final ArrayList<File> northNowcastMonoEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к северной полусфере, наблюдаемым данным, вкладу "broadband"
ускорения.
     */
    private final ArrayList<File> northNowcastWaveEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, прогнозу, вкладу рассеянного сияния.
     */
    private final ArrayList<File> southForecastDiffuseEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, прогнозу, вкладу ионов.
     */
    private final ArrayList<File> southForecastIonsEnergyFluxFiles
            = new ArrayList<>();

    /**
```

```
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, прогнозу, вкладу моноэнергетических пиков.
     */
    private final ArrayList<File> southForecastMonoEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, прогнозу, вкладу "broadband" ускорения.
     */
    private final ArrayList<File> southForecastWaveEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, наблюдаемым данным, вкладу рассеянного
сияния.
     */
    private final ArrayList<File> southNowcastDiffuseEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, наблюдаемым данным, вкладу ионов.
     */
    private final ArrayList<File> southNowcastIonsEnergyFluxFiles
            = new ArrayList<>();

    /**
```

```
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, наблюдаемым данным, вкладу
моноэнергетических пиков.
     */
    private final ArrayList<File> southNowcastMonoEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Список имен файлов с данными Ovation Prime, которые относятся
     * к южной полусфере, наблюдаемым данным, вкладу "broadband"
ускорения.
     */
    private final ArrayList<File> southNowcastWaveEnergyFluxFiles
            = new ArrayList<>();

    /**
     * Выбранный список имен файлов.
     */
    private ArrayList<File> files = new ArrayList<>();

    /**
     * Флажок, который поднят, если на компоненте для рисования
     * нужно отображать информацию о выбранной единице
геоинформационных данных.
     */
    public boolean showInfoFlag = false;

    /**
     * Флажок, который поднят, если на компоненте для рисования
     * нужно отображать границу.
```

```
 */
public boolean showMarginFlag = true;

/**
 * Флажок, который поднят, если на компоненте для рисования
 * нужно отображать тепловую карту.
 */
public boolean showHeatmapFlag = true;

/**
 * Флажок, который поднят, если на компоненте для рисования
 * нужно отображать сглаженную границу.
 */
public boolean smoothMarginFlag = true;

/**
 * Время ожидания в секундах между сеансами автозаргузки данных.
 */
public int waitingTimeInSeconds = 3600;

/**
 * Выбранная папка с данными для визуализации.
 */
public File selectedFolder;

/**
 * Координата x адаптера мыши.
 */
public int mouseAdapterX;
```

```
    /**
     * Уровень границы.
     */
    public float marginLevel = 0.6f;

    /**
     * Координата y адаптера мыши.
     */
    public int mouseAdapterY;

    /**
     * Словарь, ключами которого являются координаты единиц
     * геоинформационных данных, а значениями - соответствующие
     * единицы геоинформационных данных.
     */
    public final HashMap<GeoinformationDataUnitCoordinates,
            GeoinformationDataUnit> geoinformationDataUnits
            = new HashMap<>();

    /**
     * Список ребер границы.
     */
    public final ArrayList<Segment> marginSegments = new
ArrayList<>();

    /**
     * Список отрезков, которые соединяют середины соседних отрезков
границы.
     */
```

```
    public final ArrayList<Segment> middleMarginSegments = new
ArrayList<>();


    /**
     * Конструктор.
     *
     * @param mainFrame главный фрейм.
     */
    public Model(MainFrame mainFrame) {
        this.mainFrame = mainFrame;
    }


    /**
     * Устанавливаем тип фалов для визуализации.
     *
     * @param fileType описанный тип файлов.
     */
    public void setFileType(FileType fileType) {
        this.fileType = fileType;

mainFrame.shownFileTypeTextField.setText(this.fileType.toString());
        selectFiles();
    }


    /**
     * Получаем размер списка файлов для визуализации.
     *
     * @return описанный размер списка.
     */
    public int getFilesSize() {
```

```
        return files.size();
    }

    /**
     * Получаем имя текущего файла для визуализации.
     *
     * @return описанное имя файла.
     */
    public String getCurrentFileName() {
        return files.get(currentFileIndex).getName();
    }

    /**
     * Переходим к файлу с указанным индексом.
     *
     * @param fileIndex упомянутый индекс файла.
     */
    public void gotoCertainFile(int fileIndex) {
        if ((fileIndex < 0) || (fileIndex > files.size() - 1)) {
            return;
        }

        currentFileIndex = fileIndex;
        load();

mainFrame.currentFileNumberTextField.setText(String.valueOf(currentFil
eIndex + 1));
        mainFrame.fileNumberSlider.setValue(currentFileIndex);
```

```
mainFrame.fileNameTextField.setText(files.get(currentFileIndex).getNam
e());
        mainFrame.visualize();
    }

    /**
     * Переходим к следующему файлу.
     */
    public void gotoNextFile() {
        gotoCertainFile(currentFileIndex + 1);
    }

    /**
     * Переходим к предыдущему файлу.
     */
    public void gotoPreviousFile() {
        gotoCertainFile(currentFileIndex - 1);
    }

    /**
     * Очищаем все списки файлов.
     */
    public void clear() {
        northForecastEnergyFluxFiles.clear();
        northNowcastEnergyFluxFiles.clear();
        southForecastEnergyFluxFiles.clear();
        southNowcastEnergyFluxFiles.clear();
        northForecastDiffuseEnergyFluxFiles.clear();
        northForecastIonsEnergyFluxFiles.clear();
```

```
        northForecastMonoEnergyFluxFiles.clear();
        northForecastWaveEnergyFluxFiles.clear();
        northNowcastDiffuseEnergyFluxFiles.clear();
        northNowcastIonsEnergyFluxFiles.clear();
        northNowcastMonoEnergyFluxFiles.clear();
        northNowcastWaveEnergyFluxFiles.clear();
        southForecastDiffuseEnergyFluxFiles.clear();
        southForecastIonsEnergyFluxFiles.clear();
        southForecastMonoEnergyFluxFiles.clear();
        southForecastWaveEnergyFluxFiles.clear();
        southNowcastDiffuseEnergyFluxFiles.clear();
        southNowcastIonsEnergyFluxFiles.clear();
        southNowcastMonoEnergyFluxFiles.clear();
        southNowcastWaveEnergyFluxFiles.clear();
    }

    /**
     * Распределяем файл в соответствующий список.
     *
     * @param file упомянутый файл.
     */
    public void distributeFile(File file) {
        if (file.getName().contains("north")) {
            distributeNorthFile(file);
        } else if (file.getName().contains("south")) {
            distributeSouthFile(file);
        }
    }
```

```
    /**
     * Выбираем список файлов для визуализации.
     */
    public void selectFiles() {
        if (fileType.horizonSideType() == HorizonSideType.NORTH) {
            selectNorthFiles();
        } else if (fileType.horizonSideType() ==
HorizonSideType.SOUTH) {
            selectSouthFiles();
        }


mainFrame.totalFileNumberTextField.setText(String.valueOf(files.size()
));
        mainFrame.fileNumberSlider.setMinimum(0);
        mainFrame.fileNumberSlider.setMaximum(files.size() - 1);
        mainFrame.fileNumberSlider.setValue(0);
        mainFrame.model.gotoCertainFile(0);
    }


    /**
     * Распределяем файл в один из список, соответствующий
     * северной полусфере.
     *
     * @param file упомянутый файл.
     */
    private void distributeNorthFile(File file) {
        if (file.getName().contains("forecast")) {
            distributeNorthForecastFile(file);
        } else if (file.getName().contains("nowcast")) {
```

```
            distributeNorthNowcastFile(file);
        }
    }

    /**
     * Выбираем один из списков файлов, соответствующий
     * северной полусфере.
     */
    private void selectNorthFiles() {
        if (fileType.castType() == CastType.FORECAST) {
            selectNorthForecastFiles();
        } else if (fileType.castType() == CastType.NOWCAST) {
            selectNorthNowcastFiles();
        }
    }

    /**
     * Распределяем файл в один из список, соответствующий
     * южной полусфере.
     *
     * @param file упомянутый файл.
     */
    private void distributeSouthFile(File file) {
        if (file.getName().contains("forecast")) {
            distributeSouthForecastFile(file);
        } else if (file.getName().contains("nowcast")) {
            distributeSouthNowcastFile(file);
        }
    }
```

```
/**
 * Выбираем один из списков файлов, соответствующий
 * южной полусфере.
 */
private void selectSouthFiles() {
    if (fileType.castType() == CastType.FORECAST) {
        selectSouthForecastFiles();
    } else if (fileType.castType() == CastType.NOWCAST) {
        selectSouthNowcastFiles();
    }
}

/**
 * Распределяем файл в один из список, соответствующий
 * северной полусфере, прогнозу.
 *
 * @param file упомянутый файл.
 */
private void distributeNorthForecastFile(File file) {
    if (file.getName().contains("diffuse")) {
        northForecastDiffuseEnergyFluxFiles.add(file);
    } else if (file.getName().contains("ions")) {
        northForecastIonsEnergyFluxFiles.add(file);
    } else if (file.getName().contains("mono")) {
        northForecastMonoEnergyFluxFiles.add(file);
    } else if (file.getName().contains("wave")) {
        northForecastWaveEnergyFluxFiles.add(file);
    } else {
```

```
            northForecastEnergyFluxFiles.add(file);
        }
    }

    /**
     * Выбираем один из списков файлов, соответствующий
     * северной полусфере, прогнозу.
     */
    private void selectNorthForecastFiles() {
        if (fileType.energyType() == EnergyType.DIFFUSE) {
            files = northForecastDiffuseEnergyFluxFiles;
        } else if (fileType.energyType() == EnergyType.IONS) {
            files = northForecastIonsEnergyFluxFiles;
        } else if (fileType.energyType() == EnergyType.MONO) {
            files = northForecastMonoEnergyFluxFiles;
        } else if (fileType.energyType() == EnergyType.WAVE) {
            files = northForecastWaveEnergyFluxFiles;
        } else {
            files = northForecastEnergyFluxFiles;
        }
    }

    /**
     * Распределяем файл в один из список, соответствующий
     * северной полусфере, наблюдаемым данным.
     *
     * @param file упомянутый файл.
     */
    private void distributeNorthNowcastFile(File file) {
```

```
    if (file.getName().contains("diffuse")) {
        northNowcastDiffuseEnergyFluxFiles.add(file);
    } else if (file.getName().contains("ions")) {
        northNowcastIonsEnergyFluxFiles.add(file);
    } else if (file.getName().contains("mono")) {
        northNowcastMonoEnergyFluxFiles.add(file);
    } else if (file.getName().contains("wave")) {
        northNowcastWaveEnergyFluxFiles.add(file);
    } else {
        northNowcastEnergyFluxFiles.add(file);
    }
}

/**
 * Выбираем один из списков файлов, соответствующий
 * северной полусфере, наблюдаемым данным.
 */
private void selectNorthNowcastFiles() {
    if (fileType.energyType() == EnergyType.DIFFUSE) {
        files = northNowcastDiffuseEnergyFluxFiles;
    } else if (fileType.energyType() == EnergyType.IONS) {
        files = northNowcastIonsEnergyFluxFiles;
    } else if (fileType.energyType() == EnergyType.MONO) {
        files = northNowcastMonoEnergyFluxFiles;
    } else if (fileType.energyType() == EnergyType.WAVE) {
        files = northNowcastWaveEnergyFluxFiles;
    } else {
        files = northNowcastEnergyFluxFiles;
    }
```

```
}

/**
 * Распределяем файл в один из список, соответствующий
 * южной полусфере, прогнозу.
 *
 * @param file упомянутый файл.
 */
private void distributeSouthForecastFile(File file) {
    if (file.getName().contains("diffuse")) {
        southForecastDiffuseEnergyFluxFiles.add(file);
    } else if (file.getName().contains("ions")) {
        southForecastIonsEnergyFluxFiles.add(file);
    } else if (file.getName().contains("mono")) {
        southForecastMonoEnergyFluxFiles.add(file);
    } else if (file.getName().contains("wave")) {
        southForecastWaveEnergyFluxFiles.add(file);
    } else {
        southForecastEnergyFluxFiles.add(file);
    }
}

/**
 * Выбираем один из списков файлов, соответствующий
 * южной полусфере, прогнозу.
 */
private void selectSouthForecastFiles() {
    if (fileType.energyType() == EnergyType.DIFFUSE) {
        files = southForecastDiffuseEnergyFluxFiles;
```

```
    } else if (fileType.energyType() == EnergyType.IONS) {
        files = southForecastIonsEnergyFluxFiles;
    } else if (fileType.energyType() == EnergyType.MONO) {
        files = southForecastMonoEnergyFluxFiles;
    } else if (fileType.energyType() == EnergyType.WAVE) {
        files = southForecastWaveEnergyFluxFiles;
    } else {
        files = southForecastEnergyFluxFiles;
    }
}

/**
 * Распределяем файл в один из список, соответствующий
 * южной полусфере, наблюдаемым данным.
 *
 * @param file упомянутый файл.
 */
private void distributeSouthNowcastFile(File file) {
    if (file.getName().contains("diffuse")) {
        southNowcastDiffuseEnergyFluxFiles.add(file);
    } else if (file.getName().contains("ions")) {
        southNowcastIonsEnergyFluxFiles.add(file);
    } else if (file.getName().contains("mono")) {
        southNowcastMonoEnergyFluxFiles.add(file);
    } else if (file.getName().contains("wave")) {
        southNowcastWaveEnergyFluxFiles.add(file);
    } else {
        southNowcastEnergyFluxFiles.add(file);
    }
```

```
    }

    /**
     * Выбираем один из списков файлов, соответствующий
     * южной полусфере, наблюдаемым данным.
     */
    private void selectSouthNowcastFiles() {
        if (fileType.energyType() == EnergyType.DIFFUSE) {
            files = southNowcastDiffuseEnergyFluxFiles;
        } else if (fileType.energyType() == EnergyType.IONS) {
            files = southNowcastIonsEnergyFluxFiles;
        } else if (fileType.energyType() == EnergyType.MONO) {
            files = southNowcastMonoEnergyFluxFiles;
        } else if (fileType.energyType() == EnergyType.WAVE) {
            files = southNowcastWaveEnergyFluxFiles;
        } else {
            files = southNowcastEnergyFluxFiles;
        }
    }

    /**
     * Загружаем список единиц геонформационных данных из файла.
     */
    private void load() {
        geoinformationDataUnits.clear();

        Path filePath
                =
Paths.get(files.get(currentFileIndex).getAbsolutePath());
```

```
        Charset charset = StandardCharsets.UTF_8;
        List<String> lines;

        try {
            lines = Files.readAllLines(filePath, charset);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, "Ошибка при чтении
файла " + filePath);
            return;
        }

        for (int i = START_INDEX; (i < END_INDEX) && (i <
lines.size()); ++i) {
            GeoinformationDataUnit geoinformationDataUnit =
GeoinformationDataUnit.parse(lines.get(i));
            geoinformationDataUnits.put(
                    geoinformationDataUnit.getCoordinates(),
                    geoinformationDataUnit
            );
        }
    }

    /**
     * Получаем значение единицы геоинформационных данных по
     * ее координатам.
     *
     * @param polarDistance полярное расстояние упомянутой
     *                      единицы геоинформационных данных.
     * @param polarAngle полярный угол упомянутой
     *                   единицы геоинформационных данных.
```

```
     * @return значенеи упомянутой единицы геоинформационных данных.
     */
    public double getGeoinformationDataUnitValue(float polarDistance,
float polarAngle) {
        if (geoinformationDataUnits.containsKey(
                new GeoinformationDataUnitCoordinates(polarAngle,
polarDistance))
        ) {
            return geoinformationDataUnits.get(
                    new GeoinformationDataUnitCoordinates(polarAngle,
polarDistance)
            ).getValue();
        }

        return -1;
    }

    /**
     * Переходим к файлу, соответствующему указанным дате и времени.
     *
     * @param calendar календарь с ифнормацией об упомянутых дате и
времени.
     */
    public void gotoFileWithSelectedDateAndTime(Calendar calendar) {
        int indexOfSelectedFile = 0;
        long elapsed = Math.abs(calendar.getTime().getTime()
                -
getCalendarFromFileName(files.get(0).getName()).getTime().getTime());

        for (int i = 1; i < files.size(); ++i) {
```

```
            long currentElapsed =
Math.abs(calendar.getTime().getTime()
                    -
getCalendarFromFileName(files.get(i).getName()).getTime().getTime());

            if (currentElapsed < elapsed) {
                indexOfSelectedFile = i;
                elapsed = currentElapsed;
            }
        }

        gotoCertainFile(indexOfSelectedFile);
    }

    /**
     * Получаем календарь с информацией о дате и времени,
     * соответствующих файлу с указанным именем.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый календарь.
     */
    private Calendar getCalendarFromFileName(String fileName) {
        int year = DateAndTimeUtil.getYearFromFileName(fileName);
        int month = DateAndTimeUtil.getMonthFromFileName(fileName);
        int day = DateAndTimeUtil.getDayFromFileName(fileName);
        int hour = DateAndTimeUtil.getHourFromFileName(fileName);
        int minute = DateAndTimeUtil.getMinuteFromFileName(fileName);

        Calendar calendar = Calendar.getInstance();
```

```
        calendar.set(Calendar.YEAR, year);
        calendar.set(Calendar.MONTH, month);
        calendar.set(Calendar.DAY_OF_MONTH, day);
        calendar.set(Calendar.HOUR_OF_DAY, hour);
        calendar.set(Calendar.MINUTE, minute);

        return calendar;
    }

    /**
     * Находим ребра границы.
     */
    public void findMarginSegments() {
        marginSegments.clear();

        findRadialMarginSegments();
        findRoundMarginSegments();
    }

    /**
     * Находим ребра границы, которые лежат на радиальных линиях
     * координатной решетки.
     */
    public void findRadialMarginSegments() {
        if (mainFrame.model.geoinformationDataUnits.size() == 0) {
            return;
        }
```

```
        for (float polarAngle = 0; polarAngle <= 23.75; polarAngle +=
0.25) {

            for (float polarDistance = 50; polarDistance <= 89.5;
polarDistance += 0.5) {

                GeoinformationDataUnit first

                        = mainFrame.model.geoinformationDataUnits.get(

                        new
GeoinformationDataUnitCoordinates(polarAngle, polarDistance));


                if (first == null) {

                    continue;

                }


                float nextPolarAngle = polarAngle - 0.25f;


                if (nextPolarAngle < 0) {

                    nextPolarAngle = 23.75f;

                }


                GeoinformationDataUnit second

                        = mainFrame.model.geoinformationDataUnits.get(

                        new
GeoinformationDataUnitCoordinates(nextPolarAngle, polarDistance));


                if (second == null) {

                    continue;

                }


                if
(mainFrame.visualizationComponent.marginIsLocated(first.getValue(),
second.getValue())) {
```

```
                    PointF firstPoint =
first.getFirstPoint(mainFrame.visualizationComponent.display.getWidth(
));
                    PointF fourthPoint =
first.getFourthPoint(mainFrame.visualizationComponent.display.getWidth
());


                    marginSegments.add(new Segment(
                            firstPoint,
                            fourthPoint,
                            SegmentOrientation.RADIAL
                    ));
                }
            }
        }
    }


    /**
     * Находим ребра границы, которые приближенно лежат на круговых
     * линиях границы.
     */
    public void findRoundMarginSegments() {
        if (mainFrame.model.geoinformationDataUnits.size() == 0) {
            return;
        }


        for (float polarAngle = 0; polarAngle <= 23.75; polarAngle +=
0.25) {
            for (float polarDistance = 50.5f; polarDistance <= 89.5;
polarDistance += 0.5) {
                GeoinformationDataUnit first
```

```
                    = mainFrame.model.geoinformationDataUnits.get(
                    new
GeoinformationDataUnitCoordinates(polarAngle, polarDistance));

                if (first == null) {
                    continue;
                }

                float nextPolarDistance = polarDistance - 0.5f;

                GeoinformationDataUnit second
                        = mainFrame.model.geoinformationDataUnits.get(
                        new
GeoinformationDataUnitCoordinates(polarAngle, nextPolarDistance));

                if (second == null) {
                    continue;
                }

                if
(mainFrame.visualizationComponent.marginIsLocated(first.getValue(),
second.getValue())) {
                    PointF firstPoint =
first.getFirstPoint(mainFrame.visualizationComponent.display.getWidth(
));
                    PointF secondPoint =
first.getSecondPoint(mainFrame.visualizationComponent.display.getWidth
());

                    marginSegments.add(new Segment(
                            firstPoint,
                            secondPoint,
```

```
                SegmentOrientation.ROUND
            ));
        }
    }
}

/**
 * Связываем соседние отрезки.
 *
 * @param segments список упомянутых отрезков.
 */
private void connectSegments(ArrayList<Segment> segments) {
    for (int i = 0; i < segments.size(); ++i) {
        for (int j = 0; j < segments.size(); ++j) {
            if (i == j) {
                continue;
            }

            if (segments.get(i).isNewNeighbour(segments.get(j))) {

segments.get(i).connectAsNeighbour(segments.get(j));
            }

            if (segments.get(i).hasBothNeighbours()) {
                break;
            }
        }
    }
```

```
    }

    /**
     * Связываем ребра границы.
     */
    public void connectMarginSegments() {
        for (int i = 0; i < marginSegments.size(); ++i) {
            for (int j = 0; j < marginSegments.size(); ++j) {
                if (i == j) {
                    continue;
                }

                if
(marginSegments.get(i).isFirstNeighbour(marginSegments.get(j))) {

marginSegments.get(i).firstEndNeighbours.add(marginSegments.get(j));
                }

                if
(marginSegments.get(i).isSecondNeighbour(marginSegments.get(j))) {

marginSegments.get(i).secondEndNeighbours.add(marginSegments.get(j));
                }
            }
        }

        for (Segment marginSegment : marginSegments) {
            marginSegment.connectWithFirstTrueNeighbour();
            marginSegment.connectWithSecondTrueNeighbour();
        }
```

```
}

/**
 * Находим отрезки, которые соединяют середины соседних
 * ребер границы.
 */
public void findMiddleMarginSegments() {
    middleMarginSegments.clear();

    for (Segment marginSegment : marginSegments) {
        if (marginSegment.getFirstNeighbour() != null) {
            middleMarginSegments.add(new Segment(
                    marginSegment.getMiddle(),
                    marginSegment.getFirstNeighbour().getMiddle()
            ));
        }

        if (marginSegment.getSecondNeighbour() != null) {
            middleMarginSegments.add(new Segment(
                    marginSegment.getMiddle(),
                    marginSegment.getSecondNeighbour().getMiddle()
            ));
        }
    }
}

/**
 * Связываем соседние отрезки, которые соединяют середины
 * соседних ребер границы.
```

```
     */
    public void connectMiddleMarginSegments() {
        connectSegments(middleMarginSegments);
    }

    /**
     * Выполняем подготовку к рисованию сглаженной границы.
     */
    public void prepareToDrawSmoothMargin() {
        findMarginSegments();
        connectMarginSegments();
        findMiddleMarginSegments();
        connectMiddleMarginSegments();
    }
}
```

1.3.8. Файл `Segment.java`

```
/*
 * В данном файле содержится реализация отрезка на плоскости.
 */

package model;

import view.PointF;

import java.util.ArrayList;

/**
```

```
 * Класс реализует отрезок на плоскости.
 */
public class Segment {
    /**
     * Первый конец отрезка.
     */
    private final PointF firstEnd;

    /**
     * Второй конец отрезка.
     */
    private final PointF secondEnd;

    /**
     * Первый соседний отрезок.
     */
    private Segment firstNeighbour;

    /**
     * Второй соседний отрезок.
     */
    private Segment secondNeighbour;

    /**
     * Ориентация отрезка.
     */
    private SegmentOrientation orientation;

    /**
```

```
     * Список соседей по первому концу.
     */
    public ArrayList<Segment> firstEndNeighbours = new ArrayList<>();

    /**
     * Список соседей по второму концу.
     */
    public ArrayList<Segment> secondEndNeighbours = new ArrayList<>();

    /**
     * Конструктор.
     *
     * @param firstEnd первый конец отрезка.
     * @param secondEnd второй конец отрезка.
     */
    public Segment(PointF firstEnd, PointF secondEnd) {
        this.firstEnd = firstEnd;
        this.secondEnd = secondEnd;
    }

    /**
     * Конструктор.
     *
     * @param firstEnd первый конец отрезка.
     * @param secondEnd второй конец отрезка.
     * @param orientation ориентация.
     */
    public Segment(PointF firstEnd, PointF secondEnd,
SegmentOrientation orientation) {
```

```
        this(firstEnd, secondEnd);
        this.orientation = orientation;
    }

    /**
     * Получаем первый конец отрезка.
     *
     * @return упомянутый конец.
     */
    public PointF getFirstEnd() {
        return firstEnd;
    }

    /**
     * Получаем второй конец отрезка.
     *
     * @return упомянутый конец.
     */
    public PointF getSecondEnd() {
        return secondEnd;
    }

    /**
     * Получаем первый соседний отрезок.
     *
     * @return упомянутый отрезок.
     */
    public Segment getFirstNeighbour() {
        return firstNeighbour;
```

```
    }


    /**
     * Получаем второй соседний отрезок.
     *
     * @return упомянутый отрезок.
     */
    public Segment getSecondNeighbour() {
        return secondNeighbour;
    }


    /**
     * Проверяем, что указанный отрезок является новым соседом
     * текущего отрезка.
     *
     * @param segment указанный отрезок.
     * @return true, если упомянутое утверждение истинно. Иначе -
false.
     */
    public boolean isNewNeighbour(Segment segment) {
        if ((firstNeighbour == segment) || (secondNeighbour ==
segment)) {
            return false;
        } else if ((segment.firstNeighbour == this) ||
(segment.secondNeighbour == this)) {
            return false;
        } else {
            return (firstEnd.approximatelyEquals(segment.firstEnd)
                    ||
(firstEnd.approximatelyEquals(segment.secondEnd)
```

```
                    ||
(secondEnd.approximatelyEquals(segment.firstEnd))
                    ||
(secondEnd.approximatelyEquals(segment.secondEnd))));
        }
    }

    /**
     * Проверяем, что указанный отрезок является соседом по первому
концу.
     *
     * @param segment указанный отрезок.
     * @return true, если упомянутое утверждение истинно. Иначе -
false.
     */
    public boolean isFirstNeighbour(Segment segment) {
        if (firstEnd.approximatelyEquals(segment.firstEnd)
                && secondEnd.approximatelyEquals(segment.secondEnd)) {
            return false;
        } else if (firstEnd.approximatelyEquals(segment.secondEnd)
                && secondEnd.approximatelyEquals(segment.firstEnd)) {
            return false;
        }

        return firstEnd.approximatelyEquals(segment.firstEnd)
                || firstEnd.approximatelyEquals(segment.secondEnd);
    }

    /**
     * Проверяем, что указанный отрезок является соседом по второму
концу.
```

```
     *
     * @param segment указанный отрезок.
     * @return true, если упомянутое утверждение истинно. Иначе -
false.
     */
    public boolean isSecondNeighbour(Segment segment) {
        if (firstEnd.approximatelyEquals(segment.firstEnd)
                && secondEnd.approximatelyEquals(segment.secondEnd)) {
            return false;
        } else if (firstEnd.approximatelyEquals(segment.secondEnd)
                && secondEnd.approximatelyEquals(segment.firstEnd)) {
            return false;
        }

        return secondEnd.approximatelyEquals(segment.firstEnd)
                || secondEnd.approximatelyEquals(segment.secondEnd);
    }

    /**
     * Связываем текущий отрезок и указанный в качестве соседей.
     *
     * @param segment указанный отрезок.
     */
    public void connectAsNeighbour(Segment segment) {
        if (firstNeighbour == null) {
            firstNeighbour = segment;
        } else if (secondNeighbour == null) {
            secondNeighbour = segment;
        }
```

```
        if (segment.firstNeighbour == null) {
            segment.firstNeighbour = this;
        } else if (segment.secondNeighbour == null) {
            segment.secondNeighbour = this;
        }
    }

    /**
     * Связываем отрезок с истинным соседом по певому концу.
     * Если по первому концу имеется ровно один сосед, то он
     * считается истинным соседом по первому концу. Если по первому
     * концу имеется более одного соседа, то истинным соседом
считается
     * сосед с той же ориентацией.
     */
    public void connectWithFirstTrueNeighbour() {
        if (firstEndNeighbours.size() == 1) {
            firstNeighbour = firstEndNeighbours.get(0);
        } else {
            Segment firstSameOrientedNeighbour =
getFirstSameOrientedNeighbour();

            if (firstSameOrientedNeighbour != null) {
                firstNeighbour = firstSameOrientedNeighbour;
            }
        }
    }

    /**
```

```
     * Связываем отрезок с истинным соседом по второму концу.
     * Если по второму концу имеется ровно один сосед, то он
     * считается истинным соседом по второму концу. Если по второму
     * концу имеется более одного соседа, то истинным соседом
считается
     * сосед с той же ориентацией.
     */
    public void connectWithSecondTrueNeighbour() {
        if (secondEndNeighbours.size() == 1) {
            secondNeighbour = secondEndNeighbours.get(0);
        } else {
            Segment secondSameOrientedNeighbour =
getSecondSameOrientedNeighbour();

            if (secondSameOrientedNeighbour != null) {
                secondNeighbour = secondSameOrientedNeighbour;
            }
        }
    }

    /**
     * Получаем соседа по первому концу с той же ориентацией.
     *
     * @return упомянтый отрезок.
     */
    private Segment getFirstSameOrientedNeighbour() {
        for (Segment firstEndNeighbour : firstEndNeighbours) {
            if (orientation == firstEndNeighbour.orientation) {
                return firstEndNeighbour;
            }
```

```
        }

        return null;
    }

    /**
     * Получаем соседа по второму концу с той же ориентацией.
     *
     * @return упомянутый отрезок.
     */
    private Segment getSecondSameOrientedNeighbour() {
        for (Segment secondEndNeighbour : secondEndNeighbours) {
            if (orientation == secondEndNeighbour.orientation) {
                return secondEndNeighbour;
            }
        }

        return null;
    }

    /**
     * Получаем середину отрезка.
     *
     * @return середина отрезка.
     */
    public PointF getMiddle() {
        return new PointF(
                (firstEnd.x + secondEnd.x) / 2,
                (firstEnd.y + secondEnd.y) / 2
```

```
        );
    }


    /**
     * Проверяем, что отрезок имеет обоих соседей.
     *
     * @return true, если упомянутое утверждение истинно. Иначе -
false.
     */
    public boolean hasBothNeighbours() {
        return (firstNeighbour != null) && (secondNeighbour != null);
    }


    /**
     * Получаем точку, которая делит данный отрезок в указанном
отношении.
     *
     * @param ratio отношение длины отрезка между первым концом
исходного отрезка
     *              и упомянутой точкой и длины исходного отрезка.
     * @return упомянутая точка.
     */
    public PointF getRatioPoint(float ratio) {
        if (ratio == 0) {
            return firstEnd;
        } else if (ratio == 1) {
            return secondEnd;
        }


        float lambda = ratio / (1 - ratio);
```

```
        float x = (firstEnd.x + lambda * secondEnd.x) / (1 + lambda);
        float y = (firstEnd.y + lambda * secondEnd.y) / (1 + lambda);

        return new PointF(x, y);
    }

    /**
     * Получаем точку пересечения данного отрезка и его первого
соседа.
     *
     * @return упомянутая точка.
     */
    public PointF getIntersectionWithFirstNeighbour() {
        if (firstNeighbour == null) {
            return null;
        } else if
(firstEnd.approximatelyEquals(firstNeighbour.firstEnd)) {
            return firstEnd;
        } else if
(firstEnd.approximatelyEquals(firstNeighbour.secondEnd)) {
            return firstEnd;
        } else if
(secondEnd.approximatelyEquals(firstNeighbour.firstEnd)) {
            return secondEnd;
        } else if
(secondEnd.approximatelyEquals(firstNeighbour.secondEnd)) {
            return secondEnd;
        } else {
            return null;
        }
```

```
    }


    /**
     * Возвращаем точку пересечения данного отрезка и его второго
соседа.
     *
     * @return упомянутая точка.
     */
    public PointF getIntersectionWithSecondNeighbour() {
        if (secondNeighbour == null) {
            return null;
        } else if
(firstEnd.approximatelyEquals(secondNeighbour.firstEnd)) {
            return firstEnd;
        } else if
(firstEnd.approximatelyEquals(secondNeighbour.secondEnd)) {
            return firstEnd;
        } else if
(secondEnd.approximatelyEquals(secondNeighbour.firstEnd)) {
            return secondEnd;
        } else if
(secondEnd.approximatelyEquals(secondNeighbour.secondEnd)) {
            return secondEnd;
        } else {
            return  null;
        }
    }
}
```

1.3.9. Файл SegmentOrientation.java

```
/*
 * В данном файле содержится перечисление с типом
 * ориентации отрезка.
 */

package model;

/**
 * Перечисление с типом ориентации отрезка.
 */
public enum SegmentOrientation {
    /**
     * Радиальная ориентация отрезка.
     */
    RADIAL,

    /**
     * Приближенно круговая ориентация отрезка.
     */
    ROUND
}
```

1.12. Файл DateAndTimeUtil.java

```
/*
 * В данном файле содержится класс с методами
 * для работы с именами файлов и временем.
 */
```

```
package time;


/**
 * Класс для работы с именами файлов и временем.
 *
 * @author Иван Шагурин
 */
public class DateAndTimeUtil {
    /**
     * Получаем год по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый год.
     */
    public static int getYearFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(0, 4));
    }


    /**
     * Получаем месяц по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый месяц.
     */
    public static int getMonthFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(4, 6)) - 1;
    }


    /**
```

```
     * Получаем день по имени файла.
     *
     * @param fileName упомянутре имя файла.
     * @return упомянутый день.
     */
    public static int getDayFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(6, 8));
    }

    /**
     * Получаем час по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый час.
     */
    public static int getHourFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(9, 11));
    }

    /**
     * Получаем минуту по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутая минута.
     */
    public static int getMinuteFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(11, 13));
    }
}
```

1.4. Пакет time

1.4.1. Файл DateAndTimeUtil.java

```
/*
 * В данном файле содержится класс с методами
 * для работы с именами файлов и временем.
 */

package time;

/**
 * Класс для работы с именами файлов и временем.
 *
 * @author Иван Шагурин
 */
public class DateAndTimeUtil {
    /**
     * Получаем год по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый год.
     */
    public static int getYearFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(0, 4));
    }

    /**
     * Получаем месяц по имени файла.
```

```
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый месяц.
     */
    public static int getMonthFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(4, 6)) - 1;
    }

    /**
     * Получаем день по имени файла.
     *
     * @param fileName упомянутре имя файла.
     * @return упомянутый день.
     */
    public static int getDayFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(6, 8));
    }

    /**
     * Получаем час по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутый час.
     */
    public static int getHourFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(9, 11));
    }

    /**
```

```
     * Получаем минуту по имени файла.
     *
     * @param fileName упомянутое имя файла.
     * @return упомянутая минута.
     */
    public static int getMinuteFromFileName(String fileName) {
        return Integer.parseInt(fileName.substring(11, 13));
    }
}
```

1.5. Пакет `view`

1.5.1. Файл `ColorIndicatorComponent.java`

```
/*
 * В данном файле содержится компонента для отображения
 * цветового индикатора.
 */

package view;

import model.GeoinformationDataUnit;

import javax.swing.*;
import java.awt.*;
import java.text.DecimalFormat;

/**
 * Компонента для отображения цветового индикатора.
 *
```

```
 * @author Иван Шагурин
 */
public class ColorIndicatorComponent extends JComponent {
    /**
     * Шрифт для отображения надписей цветового индикатора.
     */
    private static final Font COLOR_INDICATOR_LABELS_FONT
            = new Font("Arial", Font.PLAIN, 12);

    /**
     * Раскрашиваем компоненту.
     *
     * @param graphics the <code>Graphics</code> object to protect
     */
    public void paintComponent(Graphics graphics) {
        Graphics2D graphics2D = (Graphics2D) graphics;

        paintBackground(graphics2D);
        paintColorIndicator(graphics2D);
        adjustGraphics2D(graphics2D);
        paintColorIndicatorLabels(graphics2D);
    }

    /**
     * Настраиваем объект для рисования.
     *
     * @param graphics2D упомянутый объект для рисования.
     */
    private void adjustGraphics2D(Graphics2D graphics2D) {
```

```
graphics2D.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);

        graphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);


graphics2D.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
RenderingHints.VALUE_COLOR_RENDER_QUALITY);

        graphics2D.setRenderingHint(RenderingHints.KEY_DITHERING,
RenderingHints.VALUE_DITHER_ENABLE);


graphics2D.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
RenderingHints.VALUE_FRACTIONALMETRICS_ON);

        graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);

        graphics2D.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);

        graphics2D.setRenderingHint(RenderingHints.KEY_STROKE_CONTROL,
RenderingHints.VALUE_STROKE_PURE);

    }


    /**
     * Раскрашиваем фон.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintBackground(Graphics2D graphics2D) {
        graphics2D.setPaint(Color.WHITE);
        graphics2D.fillRect(0, 0, getWidth(), getHeight());
    }


    /**
     * Раскрашиваем цветовой индикатор.
```

```
     *
     * @param graphics2D объект для рисования.
     */
    private void paintColorIndicator(Graphics2D graphics2D) {
        Rectangle outerRectangle = new Rectangle(
                (getWidth() - 30) / 2,
                (getHeight() - 500) / 2,
                30,
                500
        );

        int totalNumberOfInnerRectangles = 300;

        for (int i = 0; i < totalNumberOfInnerRectangles; ++i) {
            Rectangle innerRectangle = getInnerRectangle(
                    outerRectangle,
                    totalNumberOfInnerRectangles,
                    i
            );

            double ratio = (double) i / totalNumberOfInnerRectangles;
            double value = GeoinformationDataUnit.maxValue * (1 -
ratio);

            Color color = GeoinformationDataUnit.getColor(value);

            graphics2D.setPaint(color);
            graphics2D.fillRect(
                    (int) innerRectangle.getX(),
```

```
                    (int) innerRectangle.getY(),
                    (int) innerRectangle.getWidth(),
                    (int) innerRectangle.getHeight() * 2
            );
        }
    }

    /**
     * Получаем внутренний прямоугольник.
     *
     * @param outerRectangle внешний прямоугольник.
     * @param totalNumberOfRectangles общее число внутренний
прямоугольников.
     * @param indexOfInnerRectangle индекс внутреннего прямогольника.
     * @return упомянутый прямоугольник.
     */
    private Rectangle getInnerRectangle(
            Rectangle outerRectangle,
            int totalNumberOfRectangles,
            int indexOfInnerRectangle
    ) {
        double innerRectangleHeight = outerRectangle.getHeight() /
totalNumberOfRectangles;
        double innerRectangleWidth = outerRectangle.getWidth();
        double innerRectangleX = outerRectangle.getX();
        double innerRectangleY = outerRectangle.y +
innerRectangleHeight * indexOfInnerRectangle;

        return new Rectangle(
                (int) innerRectangleX,
```

```
                (int) innerRectangleY,
                (int) innerRectangleWidth,
                (int) innerRectangleHeight
        );
    }

    /**
     * Раскрашиваем надписи цветового индикатора.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintColorIndicatorLabels(Graphics2D graphics2D) {
        Rectangle outerRectangle = new Rectangle(
                (getWidth() - 30) / 2,
                (getHeight() - 500) / 2,
                30,
                500
        );

        graphics2D.setPaint(Color.BLACK);
        graphics2D.setFont(COLOR_INDICATOR_LABELS_FONT);

        graphics2D.drawString(
                new
DecimalFormat("#0.00").format(GeoinformationDataUnit.maxValue),
                (float) (outerRectangle.x + outerRectangle.getWidth()
* 1.3),
                (float) outerRectangle.y
        );
```

```
        graphics2D.drawString(
                new
DecimalFormat("#0.00").format(GeoinformationDataUnit.maxValue / 2),
                (float) (outerRectangle.x + outerRectangle.getWidth()
* 1.3),
                outerRectangle.y + outerRectangle.height / 2f
        );


        graphics2D.drawString(
                new DecimalFormat("#0.00").format(0),
                (float) (outerRectangle.x + outerRectangle.getWidth()
* 1.3),
                outerRectangle.y + outerRectangle.height
        );
    }
}
```

1.5.2. Файл `CoordinateAdapter.java`

```
/*
 * В данном файле содержится класс для преобразования координат.
 */

package view;

/**
 * Класс для преобразования координат.
 *
 * @author Иван Шагурин
 */
```

```
public class CoordinateAdapter {

    /**
     * Дисплей.
     */
    private Display display;


    /**
     * Координата x в системе отсчета компоненты для рисования.
     */
    private int xInComponent = 0;


    /**
     * Координата y в системе отсчета компоненты для рисования.
     */
    private int yInComponent = 0;


    /**
     * Первый адаптер координат.
     */
    public static CoordinateAdapter firstCoordinateAdapter = new
CoordinateAdapter();


    /**
     * Второй адаптер координат.
     */
    public static CoordinateAdapter secondCoordinateAdapter = new
CoordinateAdapter();


    /**
     * Третий адаптер координат.
```

```
     */
    public static CoordinateAdapter thirdCoordinateAdapter = new
CoordinateAdapter();


    /**
     * Четвертый адаптер координат.
     */
    public static CoordinateAdapter fourthCoordinateAdapter = new
CoordinateAdapter();


    /**
     * Конструктор.
     */
    private CoordinateAdapter() {


    }


    /**
     * Устанавливаем дисплей.
     *
     * @param display упомянутый дисплей.
     */
    public void setDisplay(Display display) {
        this.display = display;
    }


    /**
     * Устанавливаем координаты.
     *
     * @param x координата x.
```

```
     * @param y координата y.
     * @param coordinateSystem система отсчета, в которой заданы
упомянутые координаты.
     */
    public void set(int x, int y, CoordinateSystem coordinateSystem) {
        if (coordinateSystem == CoordinateSystem.COMPONENT) {
            xInComponent = x;
            yInComponent = y;
        } else if (coordinateSystem == CoordinateSystem.DISPLAY) {
            xInComponent = x + display.getOffsetX();
            yInComponent = -y + display.getOffsetY();
        } else {
            xInComponent = 0;
            yInComponent = 0;
        }
    }

    /**
     * Получаем координату x в заданной системе отсчета.
     *
     * @param coordinateSystem упомянутая система отсчета.
     * @return упомянутая координата x.
     */
    public int getX(CoordinateSystem coordinateSystem) {
        if (coordinateSystem == CoordinateSystem.COMPONENT) {
            return xInComponent;
        } else if (coordinateSystem == CoordinateSystem.DISPLAY) {
            return xInComponent - display.getOffsetX();
        } else {
```

```
            return 0;
        }
    }

    /**
     * Получаем координату y в заданной системе отсчета.
     *
     * @param coordinateSystem упомянутая система отсчета.
     * @return упомянутая координата y.
     */
    public int getY(CoordinateSystem coordinateSystem) {
        if (coordinateSystem == CoordinateSystem.COMPONENT) {
            return yInComponent;
        } else if (coordinateSystem == CoordinateSystem.DISPLAY) {
            return -yInComponent + display.getOffsetY();
        } else {
            return 0;
        }
    }
}
```

1.5.3. Файл `CoordinateSystem.java`

```
/*
 * В данном файле содержится перечисление с типом системы отсчета.
 */

package view;
```

```
/**
 * Перечисление с типом системы отсчета.
 */
public enum CoordinateSystem {
    /**
     * Система отсчета относительно компоненты для рисования.
     */
    COMPONENT,
    /**
     * Система отсчета относительно дисплея.
     */
    DISPLAY,
}
```

1.5.4. Файл `Display.java`

```
/*
 * В данном файле содержится дисплей.
 */

package view;

import java.awt.*;

/**
 * Дисплей.
 *
 * @author Иван Шагурин
 */
```

```
public class Display {
    /**
     * Коэффициент сжатия изображения по умолчанию.
     */
    private final static float DEFAULT_COMPRESSION_RATIO = 0.85f;

    /**
     * Коэффициент, который задает позицию изображения по умолчанию.
     */
    private final static float DEFAULT_POSITION_RATIO = 0.5f;

    /**
     * Минимальная ширина.
     */
    private final static int MIN_WIDTH = 495;

    /**
     * Минимальная высота.
     */
    private final static int MIN_HEIGHT = 495;

    /**
     * Максимальная ширина.
     */
    private final static int MAX_WIDTH = 1729;

    /**
     * Максмальная высота.
     */
```

```
private final static int MAX_HEIGHT = 1729;

/**
 * Коэффициент увеличения размеров дисплея при
 * однократной прокрутке колесика мыши.
 */
public final static float INCREASE_SIZE_RATIO = 1.05f;

/**
 * Коэффициент уменьшения размеров дисплея при
 * однократной прокрутке колесика мыши.
 */
public final static float DECREASE_SIZE_RATIO = 1 / 1.05f;

/**
 * Минимальный сдвиг по оси X.
 */
private final static int MIN_OFFSET_X = -890;

/**
 * Максимальный сдвиг по оси X.
 */
private final static int MAX_OFFSET_X = 1680;

/**
 * Минимальный сдвиг по оси Y.
 */
private final static int MIN_OFFSET_Y = -900;
```

```
/**
 * Максимальный сдвиг по оси Y.
 */
private final static int MAX_OFFSET_Y = 1650;

/**
 * Сдвиг по оси X.
 */
private int offsetX;

/**
 * Сдвиг по оси Y.
 */
private int offsetY;

/**
 * Ширина.
 */
private int width;

/**
 * Высота.
 */
private int height;

/**
 * Конструктор.
 *
 * @param visualizationComponent компонента для визуализации.
```

```
     */
    public Display(VisualizationComponent visualizationComponent) {
        adjustOffsetAndSizes(visualizationComponent);
    }

    /**
     * Настраиваем сдвиг и размер.
     *
     * @param visualizationComponent компонента для визуализации.
     */
    public       void       adjustOffsetAndSizes(VisualizationComponent
visualizationComponent) {
        offsetX    =    (int)    (visualizationComponent.getWidth()    *
DEFAULT_POSITION_RATIO);
        offsetY    =    (int)    (visualizationComponent.getHeight()    *
DEFAULT_POSITION_RATIO);

        width = (int) (Math.min(
                visualizationComponent.getWidth(),
                visualizationComponent.getHeight()
        ) * DEFAULT_COMPRESSION_RATIO);
        height = (int) (Math.min(
                visualizationComponent.getWidth(),
                visualizationComponent.getHeight()
        ) * DEFAULT_COMPRESSION_RATIO);
    }

    /**
     * Получаем сдвиг по оси X.
     *
```

```
     * @return упомянутый свдиг.
     */
    public int getOffsetX() {
        return offsetX;
    }

    /**
     * Получаем сдвиг по оси Y.
     *
     * @return упомянутый сдвиг.
     */
    public int getOffsetY() {
        return offsetY;
    }

    /**
     * Получаем ширину.
     *
     * @return упомянутая ширина.
     */
    public int getWidth() {
        return width;
    }

    /**
     * Получаем высоту.
     *
     * @return упомянутая высота.
     */
```

```
public int getHeight() {
    return height;
}

/**
 * Меняем сдвиг на указанные расстояния по оси X и Y.
 *
 * @param deltaX сдвиг по оси X.
 * @param deltaY сдвиг по сои Y.
 */
public void changeOffset(int deltaX, int deltaY) {
    int newOffsetX = offsetX + deltaX;
    int newOffsetY = offsetY + deltaY;

    if (newOffsetX < MIN_OFFSET_X) {
        offsetX = MIN_OFFSET_X;
    } else if (newOffsetX > MAX_OFFSET_Y) {
        offsetX = MAX_OFFSET_X;
    } else {
        offsetX = newOffsetX;
    }

    if (newOffsetY < MIN_OFFSET_Y) {
        offsetY = MIN_OFFSET_Y;
    } else if (newOffsetY > MAX_OFFSET_Y) {
        offsetY = MAX_OFFSET_Y;
    } else {
        offsetY = newOffsetY;
    }
```

```
    }

    /**
     * Проверяем, что можно увеличить размеры.
     *
     * @return true, если размеры можно увеличить. Иначе - false.
     */
    public boolean canIncreaseSizes() {
        return (width * 1.05 <= MAX_WIDTH) && (height * 1.05 <=
MAX_HEIGHT);
    }

    /**
     * Проверяем, что можно уменьшить размеры.
     *
     * @return true, если размеры можно уменьшить. Иначе - false.
     */
    public boolean canDecreaseSizes() {
        return (width / 1.05 >= MIN_WIDTH) && (height / 1.05 >=
MIN_HEIGHT);
    }

    /**
     * Увеличиваем размеры.
     */
    public void increaseSizes() {
        width *= 1.05;
        height *= 1.05;
    }
```

```
    /**
     * Уменьшаем размеры.
     */
    public void decreaseSizes() {
        width /= 1.05;
        height /= 1.05;
    }

    /**
     * Закрашиваем овал.
     *
     * @param x1 координата x левой верхней вершина прямоугольника, в
который вписан овал.
     * @param y1 координаната y левой верхней вершины прямоугольника, в
который вписан овал.
     * @param x2 кординаната x нижней правой вершины прямоугольника, в
который вписан овал.
     * @param y2 координата y нижней правой вершины прямоугольника, в
который вписан овал.
     * @param color цвет овала.
     * @param graphics2D объект для рисования.
     */
    public void fillOval(int x1, int y1,
                         int x2, int y2,
                         Color color, Graphics2D graphics2D) {
        graphics2D.setPaint(color);

        CoordinateAdapter.firstCoordinateAdapter.setDisplay(this);
        CoordinateAdapter.secondCoordinateAdapter.setDisplay(this);
```

```
        CoordinateAdapter.firstCoordinateAdapter.set(x1,            y1,
CoordinateSystem.DISPLAY);

        CoordinateAdapter.secondCoordinateAdapter.set(x2,           y2,
CoordinateSystem.DISPLAY);


        int width

                =
CoordinateAdapter.secondCoordinateAdapter.getX(CoordinateSystem.COMPON
ENT)

                -
CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT);

        int height

                =
CoordinateAdapter.secondCoordinateAdapter.getY(CoordinateSystem.COMPON
ENT)

                -
CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT);


        graphics2D.fillOval(

CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT),

CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT),
                width,
                height
        );
    }


    /**
     * Рисуем овал.
```

```
     *
     * @param x1 координата x левой верхней вершина прямоугольника, в
который вписан овал.
     * @param y1 координаната y левой верхней вершины прямоугольника, в
который вписан овал.
     * @param x2 кординаната x нижней правой вершины прямоугольника, в
который вписан овал.
     * @param y2 координата y нижней правой вершины прямоугольника, в
который вписан овал.
     * @param color цвет овала.
     * @param stroke кисть.
     * @param graphics2D объект для рисования.
     */
    public void drawOval(int x1, int y1,
                         int x2, int y2,
                         Color color,
                         Stroke stroke,
                         Graphics2D graphics2D) {
        graphics2D.setPaint(color);
        graphics2D.setStroke(stroke);

        CoordinateAdapter.firstCoordinateAdapter.setDisplay(this);
        CoordinateAdapter.secondCoordinateAdapter.setDisplay(this);

        CoordinateAdapter.firstCoordinateAdapter.set(x1,           y1,
CoordinateSystem.DISPLAY);
        CoordinateAdapter.secondCoordinateAdapter.set(x2,          y2,
CoordinateSystem.DISPLAY);

        int width
```

```
                =
CoordinateAdapter.secondCoordinateAdapter.getX(CoordinateSystem.COMPON
ENT)

                -
CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT);
        int height

                =
CoordinateAdapter.secondCoordinateAdapter.getY(CoordinateSystem.COMPON
ENT)

                -
CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT);


        graphics2D.drawOval(

CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT),

CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT),
                width,
                height
        );
    }


    /**
     * Рисуем прямую линию.
     *
     * @param x1 координата x начала линии.
     * @param y1 координата y начала линии.
     * @param x2 координата x конца линии.
     * @param y2 координата y конца линии.
```

```
     * @param color увет линии.
     * @param stroke кисть.
     * @param graphics2D объект для рисования.
     */
    public void drawLine(int x1, int y1,
                         int x2, int y2,
                         Color color,
                         Stroke stroke,
                         Graphics2D graphics2D) {
        graphics2D.setPaint(color);
        graphics2D.setStroke(stroke);

        CoordinateAdapter.firstCoordinateAdapter.setDisplay(this);
        CoordinateAdapter.secondCoordinateAdapter.setDisplay(this);

        CoordinateAdapter.firstCoordinateAdapter.set(x1,            y1,
CoordinateSystem.DISPLAY);
        CoordinateAdapter.secondCoordinateAdapter.set(x2,           y2,
CoordinateSystem.DISPLAY);

        graphics2D.drawLine(

CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT),

CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT),

CoordinateAdapter.secondCoordinateAdapter.getX(CoordinateSystem.COMPON
ENT),
```

```
CoordinateAdapter.secondCoordinateAdapter.getY(CoordinateSystem.COMPON
ENT)
        );
    }

    /**
     * Закрашиваем четырехугольник.
     *
     * @param x1 координата x первой вершины четырехугольника.
     * @param y1 координата y первый вершины четырехугольника.
     * @param x2 координата x второй вершины четырехугольника.
     * @param y2 координата y второй вершины четырехугольника.
     * @param x3 координата x третей вершины четырехугольника.
     * @param y3 координата y третей вершины четырехугольника.
     * @param x4 координата x четвертой вершины четырехугольника.
     * @param y4 координата y четвертой вершины четырехугольника.
     * @param color цвет четырехугольника.
     * @param graphics2D объект для рисования.
     */
    public void fillTetragon(
            int x1, int y1,
            int x2, int y2,
            int x3, int y3,
            int x4, int y4,
            Color color,
            Graphics2D graphics2D
    ) {
        graphics2D.setPaint(color);
```

```
        CoordinateAdapter.firstCoordinateAdapter.setDisplay(this);
        CoordinateAdapter.secondCoordinateAdapter.setDisplay(this);
        CoordinateAdapter.thirdCoordinateAdapter.setDisplay(this);
        CoordinateAdapter.fourthCoordinateAdapter.setDisplay(this);

        CoordinateAdapter.firstCoordinateAdapter.set(x1,           y1,
CoordinateSystem.DISPLAY);
        CoordinateAdapter.secondCoordinateAdapter.set(x2,          y2,
CoordinateSystem.DISPLAY);
        CoordinateAdapter.thirdCoordinateAdapter.set(x3,           y3,
CoordinateSystem.DISPLAY);
        CoordinateAdapter.fourthCoordinateAdapter.set(x4,          y4,
CoordinateSystem.DISPLAY);

        graphics2D.fillPolygon(new int[]{
CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT),
CoordinateAdapter.secondCoordinateAdapter.getX(CoordinateSystem.COMPON
ENT),
CoordinateAdapter.thirdCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT),
CoordinateAdapter.fourthCoordinateAdapter.getX(CoordinateSystem.COMPON
ENT)},
                new int[]{
CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT),
CoordinateAdapter.secondCoordinateAdapter.getY(CoordinateSystem.COMPON
ENT),
```

```
CoordinateAdapter.thirdCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT),

CoordinateAdapter.fourthCoordinateAdapter.getY(CoordinateSystem.COMPON
ENT)},
                4);
    }

    /**
     * Пишем строку.
     *
     * @param str текст.
     * @param x координата x.
     * @param y координаната y.
     * @param shiftX сдвиг по оси X.
     * @param shiftY сдвиг по оси Y.
     * @param color цвет строки.
     * @param font шрифт строки.
     * @param graphics2D объект для рисования.
     */
    public void drawString(String str, int x, int y,
                           int shiftX, int shiftY,
                           Color color, Font font,
                           Graphics2D graphics2D) {
        graphics2D.setPaint(color);
        graphics2D.setFont(font);

        CoordinateAdapter.firstCoordinateAdapter.setDisplay(this);
```

```
        CoordinateAdapter.firstCoordinateAdapter.set(x,             y,
CoordinateSystem.DISPLAY);


        graphics2D.drawString(str,

CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.COMPONE
NT) - shiftX,

CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.COMPONE
NT) - shiftY);
    }


    /**
     * Закрашиваем надпись.
     *
     * @param text текст.
     * @param x координата x.
     * @param y координата y.
     * @param labelColor цвет фона.
     * @param textColor цвет текста.
     * @param font шрифт.
     * @param graphics2D объект для рисования.
     */
    public void fillLabel(String text, double x, double y,
                          Color labelColor, Color textColor,
                          Font font,
                          Graphics2D graphics2D) {
        graphics2D.setPaint(labelColor);
        graphics2D.fillRect((int) (x + 10), (int) (y),
                graphics2D.getFontMetrics().stringWidth(text),
                font.getSize());
```

```
        graphics2D.setPaint(textColor);
        graphics2D.setFont(font);
        graphics2D.drawString(text, (float) (x + 10), (float) (y +
13));
    }
}
```

1.5.5. Файл `FileTypeFrame.java`

```
/*
 * В данном файле содержится фрейм для выбора типа файлов
 * для визуализации.
 */

package view;

import model.CastType;
import model.EnergyType;
import model.FileType;
import model.HorizonSideType;

import javax.swing.*;
import java.awt.*;

/**
 * Фрейм для выбора типа файлов для визуализации.
 */
public class FileTypeFrame extends JFrame {
    /**
```

```
     * Главный фрейм.
     */
    private final MainFrame mainFrame;

    /**
     * Флажок для выбора северной полусферы.
     */
    private final JCheckBox northCheckBox = new JCheckBox("north");

    /**
     * Флажок для выбора южной полусферы.
     */
    private final JCheckBox southCheckBox = new JCheckBox("south");

    /**
     * Флажок для выбора прогноза.
     */
    private final JCheckBox forecastCheckBox = new
JCheckBox("forecast");

    /**
     * Флажок для выбора наблюдаемых данных.
     */
    private final JCheckBox nowcastCheckBox = new
JCheckBox("nowcast");

    /**
     * Флажок для выбора вклада рассеянного сияния.
     */
```

```
    private final JCheckBox diffuseCheckBox = new
JCheckBox("diffuse");


    /**
     * Флажок для выбора вклада ионов.
     */
    private final JCheckBox ionsCheckBox = new JCheckBox("ions");


    /**
     * Флажок для выбора вклада моноэнергетических пиков.
     */
    private final JCheckBox monoCheckBox = new JCheckBox("mono");


    /**
     * Флажок для выбора вклада "broadband" ускорения.
     */
    private final JCheckBox waveCheckBox = new JCheckBox("wave");


    /**
     * Флажок для выбора данных об общем вкладе авроральных компонент.
     */
    private final JCheckBox totalCheckBox = new JCheckBox("total");


    /**
     * Кнопка "Применить".
     */
    private final JButton applyButton = new JButton("Применить");


    /**
```

```
     * Конструктор.
     *
     * @param mainFrame главный фрейм.
     */
    public FileTypeFrame(MainFrame mainFrame) {
        this.mainFrame = mainFrame;

        setTitle("Выберите тип отображаемых файлов");
        setSizeAndLocation();
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        marshalContainer();
        setFileType(mainFrame.model.fileType);
        addActionListeners();
    }

    /**
     * Устанавливаем размер и позицию.
     */
    private void setSizeAndLocation() {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension screenSize = toolkit.getScreenSize();

        int screenHeight = screenSize.height - 40;
        int screenWidth = screenSize.width;

        setSize(400, 300);
        setLocation((screenWidth - 400) / 2, (screenHeight - 300) /
2);
    }
```

```
    /**
     * Устанавливаем тип файлов для визуализации.
     *
     * @param fileType упомянутый тип файлов.
     */
    private void setFileType(FileType fileType) {
        northCheckBox.setSelected(fileType.horizonSideType() ==
HorizonSideType.NORTH);
        southCheckBox.setSelected(fileType.horizonSideType() ==
HorizonSideType.SOUTH);


        forecastCheckBox.setSelected(fileType.castType() ==
CastType.FORECAST);
        nowcastCheckBox.setSelected(fileType.castType() ==
CastType.NOWCAST);


        diffuseCheckBox.setSelected(fileType.energyType() ==
EnergyType.DIFFUSE);
        ionsCheckBox.setSelected(fileType.energyType() ==
EnergyType.IONS);
        monoCheckBox.setSelected(fileType.energyType() ==
EnergyType.MONO);
        waveCheckBox.setSelected(fileType.energyType() ==
EnergyType.WAVE);
        totalCheckBox.setSelected(fileType.energyType() ==
EnergyType.TOTAL);
    }


    /**
     * Размещаем элементы GUI на контейре фрейма.
     */
```

```
private void marshalContainer() {
    Container container = getContentPane();
    container.setLayout(null);

    northCheckBox.setBounds(150, 10, 60, 25);
    southCheckBox.setBounds(150, 30, 60, 25);

    forecastCheckBox.setBounds(150, 65, 100, 25);
    nowcastCheckBox.setBounds(150, 85, 100, 25);

    totalCheckBox.setBounds(150, 120, 100, 25);
    diffuseCheckBox.setBounds(150, 140, 100, 25);
    ionsCheckBox.setBounds(150, 160, 100, 25);
    monoCheckBox.setBounds(150, 180, 100, 25);
    waveCheckBox.setBounds(150, 200, 100, 25);

    applyButton.setBounds(150, 230, 100, 25);

    container.add(northCheckBox);
    container.add(southCheckBox);

    container.add(forecastCheckBox);
    container.add(nowcastCheckBox);

    container.add(totalCheckBox);
    container.add(diffuseCheckBox);
    container.add(ionsCheckBox);
    container.add(monoCheckBox);
    container.add(waveCheckBox);
```

```
        container.add(applyButton);
    }

    /**
     * Добавляем элементам GUI обработчики событий.
     */
    private void addActionListeners() {
        northCheckBox.addActionListener(e -> {
            if (northCheckBox.isSelected()) {
                southCheckBox.setSelected(false);
            }
        });

        southCheckBox.addActionListener(e -> {
            if (southCheckBox.isSelected()) {
                northCheckBox.setSelected(false);
            }
        });

        forecastCheckBox.addActionListener(e -> {
            if (forecastCheckBox.isSelected()) {
                nowcastCheckBox.setSelected(false);
            }
        });

        nowcastCheckBox.addActionListener(e -> {
            if (nowcastCheckBox.isSelected()) {
                forecastCheckBox.setSelected(false);
```

```
    }
});

totalCheckBox.addActionListener(e -> {
    if (totalCheckBox.isSelected()) {
        diffuseCheckBox.setSelected(false);
        ionsCheckBox.setSelected(false);
        monoCheckBox.setSelected(false);
        waveCheckBox.setSelected(false);
    }
});

diffuseCheckBox.addActionListener(e -> {
    if (diffuseCheckBox.isSelected()) {
        totalCheckBox.setSelected(false);
        ionsCheckBox.setSelected(false);
        monoCheckBox.setSelected(false);
        waveCheckBox.setSelected(false);
    }
});

ionsCheckBox.addActionListener(e -> {
    totalCheckBox.setSelected(false);
    diffuseCheckBox.setSelected(false);
    monoCheckBox.setSelected(false);
    waveCheckBox.setSelected(false);
});

monoCheckBox.addActionListener(e -> {
```

```
            totalCheckBox.setSelected(false);
            diffuseCheckBox.setSelected(false);
            ionsCheckBox.setSelected(false);
            waveCheckBox.setSelected(false);
        });

        waveCheckBox.addActionListener(e -> {
            totalCheckBox.setSelected(false);
            diffuseCheckBox.setSelected(false);
            ionsCheckBox.setSelected(false);
            monoCheckBox.setSelected(false);
        });

        applyButton.addActionListener(e -> {
            FileType fileType = getFileType();

            if (fileType == null) {
                JOptionPane.showMessageDialog(null, "Ошибка. Один из
параметров не выбран.");
                return;
            }

            mainFrame.model.setFileType(fileType);
            dispose();
        });
    }

    /**
     * Получаем тип файлов для визуализации.
```

```
     *
     * @return упомянутый тип файлов.
     */
    private FileType getFileType() {
        FileType fileType
                = new FileType(getHorizonSideType(),
                getCastType(),
                getEnergyType());

        if ((fileType.horizonSideType() == null)
                || (fileType.castType() == null)
                || (fileType.energyType() == null)) {
            return null;
        } else {
            return fileType;
        }
    }

    /**
     * Получаем тип полусферы.
     *
     * @return упомянутый тип полусферы.
     */
    private HorizonSideType getHorizonSideType() {
        if (northCheckBox.isSelected()) {
            return HorizonSideType.NORTH;
        } else if (southCheckBox.isSelected()) {
            return HorizonSideType.SOUTH;
        } else {
```

```
            return null;
        }
    }

    /**
     * Получаем тип прогноза.
     *
     * @return упомянутый тип прогноза.
     */
    private CastType getCastType() {
        if (forecastCheckBox.isSelected()) {
            return CastType.FORECAST;
        } else if (nowcastCheckBox.isSelected()) {
            return CastType.NOWCAST;
        } else {
            return null;
        }
    }

    /**
     * Выбираем тип энергии.
     *
     * @return упомянутый тип энергии.
     */
    private EnergyType getEnergyType() {
        if (totalCheckBox.isSelected()) {
            return EnergyType.TOTAL;
        } else if (diffuseCheckBox.isSelected()) {
            return EnergyType.DIFFUSE;
```

```
        } else if (ionsCheckBox.isSelected()) {
            return EnergyType.IONS;
        } else if (monoCheckBox.isSelected()) {
            return EnergyType.MONO;
        } else if (waveCheckBox.isSelected()) {
            return EnergyType.WAVE;
        } else {
            return null;
        }
    }
}
```

1.5.6. Файл JDatePickerUtil.java

```
/*
 * В данном файле содержится класс для использования календаря.
 */

package view;

import org.jdatepicker.impl.JDatePanelImpl;
import org.jdatepicker.impl.JDatePickerImpl;
import org.jdatepicker.impl.SqlDateModel;

import javax.swing.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Properties;

/**
 * Класс для использования календаря.
 *
 * @author Иван Шагурин
 */
public class JDatePickerUtil {
    /**
     * Реализация календаря.
     */
    public final JDatePickerImpl datePicker;
```

```
    /**
     * Модель с данными о дате.
     */
    public final SqlDateModel sqlDateModel;

    /**
     * Конструктор.
     */
    public JDatePickerUtil() {
        sqlDateModel = new SqlDateModel();
        Properties p = new Properties();

        sqlDateModel.setSelected(true);

        p.put("text.day", "Day");
        p.put("text.month", "Month");
        p.put("text.year", "Year");

        JDatePanelImpl panel = new JDatePanelImpl(sqlDateModel, p);

        datePicker = new JDatePickerImpl(panel, new
JFormattedTextField.AbstractFormatter() {
            @Override
            public String valueToString(Object value) throws
ParseException {
                if (value != null) {
                    Calendar cal = (Calendar) value;
                    SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd");
                    String strDate = format.format(cal.getTime());

                    return strDate;
                }

                return "";
            }

            @Override
            public Object stringToValue(String text) throws
ParseException {
                return "";
            }
        });
    }
}
```

1.5.7. Файл MainFrame.java

```
/*
 * В данном файле содержится реализация главного фрейма.
 */

package view;

import controller.Controller;
import model.GeoinformationDataUnit;
import model.Model;
import time.DateAndTimeUtil;

import javax.swing.*;
import java.awt.*;
import java.util.Objects;

/**
 * Главный фрейм.
 *
 * @author Иван Шагурин
 */
public class MainFrame extends JFrame {
    /**
     * Заголовок.
     */
    public static final String TITLE = "Ovation Prime Визуализатор
v1.6";

    /**
     * Основной контейнер.
     */
    public final Container container = getContentPane();

    /**
     * Полоса загрузки.
     */
    public final JProgressBar progressBar = new JProgressBar();

    /**
     * Кнопка "Выбрать папку".
     */
    public final JButton selectFolderButton = new JButton("Выбрать
папку");

    /**
     * Кнопка "Папка по умолчанию".
```

```
     */
    public final JButton selectDefaultFolderButton = new
JButton("Папка по умолчанию");

    /**
     * Надпись "Выбранная папка:".
     */
    public final JLabel selectedFolderLabel = new JLabel("Выбранная
папка:");

    /**
     * Текстовое поле для отображения выбранной папки.
     */
    public final JTextField selectedFolderTextField = new
JTextField();

    /**
     * Надпись "Текущий файл:".
     */
    public final JLabel fileNameLabel = new JLabel("Текущий файл:");

    /**
     * Текстовое поле с имененем текущего файла.
     */
    public final JTextField fileNameTextField = new JTextField(40);

    /**
     * Надпись "Дата:".
     */
    public final JLabel dateLabel = new JLabel("Дата:");

    /**
     * Класс для использования календаря.
     */
    public final JDatePickerUtil datePickerUtil = new
JDatePickerUtil();

    /**
     * Надпись "Время:".
     */
    public final JLabel timeLabel = new JLabel("Время:");

    /**
     * Текстовое поле для отображения времени.
     */
    public final JTextField timeTextField = new JTextField(15);
```

```
    /**
     * Кнопка "Выбрать дату и время".
     */
    public final JButton selectDateAndTimeButton = new
JButton("Выбрать дату и время");

    /**
     * Кнопка "Выбрать тип отображаемых файлов".
     */
    public final JButton selectShownFilesTypeButton = new
JButton("Выбрать тип отображаемых файлов");

    /**
     * Текствое поле для отображения выбранного типа файлов.
     */
    public final JTextField shownFileTypeTextField = new
JTextField(40);

    /**
     * Надпись "Предельное значение цветовой шкалы:".
     */
    public final JLabel colorIndicatorLimitLabel = new
JLabel("Предельное значение цветовой шкалы:");

    /**
     * Текстовое поле для отображения предельного значения цветовой
шкалы.
     */
    public final JTextField colorIndicatorLimitTextField = new
JTextField();

    /**
     * Кнопка "Изменить" для смены предельного значения цветовой
шкалы.
     */
    public final JButton changeColorIndicatorLimitButton
            = new JButton("Изменить предельное значение цветовой
шкалы");

    /**
     * Кнопка "Предыдущий файл".
     */
    public final JButton gotoPreviousFileButton = new
JButton("Предыдущий файл");

    /**
     * Кнопка "Следующий файл".
```

```
     */
    public final JButton gotoNextFileButton = new JButton("Следующий
файл");

    /**
     * Надпись "Номер файла:".
     */
    public final JLabel currentFileNumberLabel = new JLabel("Номер
файла:");

    /**
     * Текстовое поле для отображения номера текущего файла.
     */
    public final JTextField currentFileNumberTextField = new
JTextField();

    /**
     * Надпись "из".
     */
    public final JLabel fromLabel = new JLabel("из");

    /**
     * Текствое поле для отображения общего числа файлов выбранного
типа.
     */
    public final JTextField totalFileNumberTextField = new
JTextField();

    /**
     * Кнопка "Перейти к файлу".
     */
    public final JButton gotoFileButton = new JButton("Перейти к
файлу");

    /**
     * Текстовое поле для отображения номера текущего файла.
     */
    public final JTextField fileNumberTextField = new JTextField();

    /**
     * Ползунок для выбора номера файла.
     */
    public final JSlider fileNumberSlider = new JSlider();

    /**
     * Кнопка "Сдвиг и масштаб по умолчанию".
     */
```

```
    public final JButton defaultOffsetAndScaleButton = new
JButton("Сдвиг и масштаб по умолчанию");

    /**
     * Флажок "Отображать тепловую карту".
     */
    public final JCheckBox paintHeatmapCheckBox = new
JCheckBox("Отображать тепловую карту");

    /**
     * Флажок "Отображать границу".
     */
    public final JCheckBox paintMarginCheckBox = new
JCheckBox("Отображать границу");

    /**
     * Флажок "Гладкая граница".
     */
    public final JCheckBox smoothMarginCheckBox = new
JCheckBox("Гладкая граница");

    /**
     * Надпись "Уровень границы:".
     */
    public final JLabel marginLevelLabel = new JLabel("Уровень
границы:");

    /**
     * Текстовое поле для отображения уровня границы.
     */
    public final JTextField marginLevelTextField = new JTextField();

    /**
     * Кнопка "Задать".
     */
    public final JButton specifyMarginLevelButton = new
JButton("Задать");

    /**
     * Флажок "Автозагрузка данных".
     */
    public final JCheckBox automaticDataDownloadCheckBox = new
JCheckBox("Автозагрузка данных");

    /**
     * Надпись "Время ожидания в секундах:".
     */
```

```
    public final JLabel waitingTimeLabel = new JLabel("Время ожидания
в секундах:");

    /**
     * Текстовое поле для отображения времени ожидания между
     * сеансами автозагрузки данных.
     */
    public final JTextField waitingTimeTextField = new JTextField();

    /**
     * Кнопка "Изменить на".
     */
    public final JButton changeWaitingTimeButton = new
JButton("Изменить на");

    /**
     * Текстовое поле для изменения времени ожидания между
     * сеансами автозагрузки данных.
     */
    public final JTextField changeWaitingTimeTextField = new
JTextField();

    /**
     * Компонента для визуализации.
     */
    public final VisualizationComponent visualizationComponent = new
VisualizationComponent(this);

    /**
     * Компонента для отображения цветового индикатора.
     */
    public final ColorIndicatorComponent colorIndicatorComponent = new
ColorIndicatorComponent();

    /**
     * Модель для паттерна MVC.
     */
    public Model model;

    /**
     * Конструктор.
     */
    public MainFrame() {
        setTitle(TITLE);
        setSizeAndLocation();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marshalContainer();
```

```
        trySetLookAndFeel();
        visualizationComponent.initDisplay();

        model = new Model(this);

        Controller controller = new Controller(this, model);
        controller.addEventListeners();

        showFileType();
        visualize();

        automaticDataDownloadCheckBox.setSelected(true);
        controller.automaticDataDownloadCheckBoxListener();
    }

    /**
     * Выполняем попытку выбрать внешний вид элементов GUI.
     */
    private void trySetLookAndFeel() {
        try {
            UIManager.LookAndFeelInfo[] infos =
UIManager.getInstalledLookAndFeels();

            for(var info : infos) {
                if(Objects.equals(info.getName(), "Windows")) {
                    UIManager.setLookAndFeel(info.getClassName());
                    SwingUtilities.updateComponentTreeUI(this);
                }
            }
        } catch (Exception ignored) {
            System.out.println("exception");
        }
    }

    /**
     * Устанавливаем размер и позицию главного фрейма.
     */
    private void setSizeAndLocation() {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension screenSize = toolkit.getScreenSize();

        int screenHeight = screenSize.height;
        int screenWidth = screenSize.width;

        setSize(Math.min(955, screenWidth), Math.min(770, screenHeight
- 40));
```

```
        setLocation((screenWidth - getWidth()) / 2, (screenHeight -
getHeight()) / 2);
    }

    /**
     * Заполняем элеменами GUI главный фрейм.
     */
    private void marshalContainer() {
        container.setLayout(null);

        selectFolderButton.setBounds(10, 10, 120, 25);
        selectDefaultFolderButton.setBounds(135, 10, 150, 25);
        selectedFolderLabel.setBounds(10, 40, 200, 25);
        selectedFolderTextField.setBounds(10, 70, 280, 25);
        fileNameLabel.setBounds(10, 100, 200, 25);
        fileNameTextField.setBounds(10, 130, 280, 25);
        fileNumberSlider.setMinimum(0);
        fileNumberSlider.setMaximum(0);
        fileNumberSlider.setValue(0);
        fileNumberSlider.setBounds(10, 160, 280, 25);
        dateLabel.setBounds(10, 280, 30, 25);
        datePickerUtil.datePicker.setBounds(45, 280, 110, 25);
        timeLabel.setBounds(160, 280, 40, 25);
        timeTextField.setBounds(205, 280, 85, 25);
        selectDateAndTimeButton.setBounds(10, 310, 280, 25);
        shownFileTypeTextField.setBounds(10, 340, 280, 25);
        selectShownFilesTypeButton.setBounds(10, 370, 280, 25);
        colorIndicatorLimitLabel.setBounds(10, 400, 200, 25);
        colorIndicatorLimitTextField.setBounds(205, 400, 85, 25);
        changeColorIndicatorLimitButton.setBounds(10, 430, 280, 25);
        gotoPreviousFileButton.setBounds(10, 190, 137, 25);
        gotoNextFileButton.setBounds(152, 190, 138, 25);
        currentFileNumberLabel.setBounds(10, 220, 70, 25);
        currentFileNumberTextField.setBounds(85, 220, 92, 25);
        fromLabel.setBounds(182, 220, 30, 25);
        totalFileNumberTextField.setBounds(198, 220, 92, 25);
        gotoFileButton.setBounds(10, 250, 182, 25);
        fileNumberTextField.setBounds(198, 250, 92, 25);
        defaultOffsetAndScaleButton.setBounds(10, 460, 280, 25);
        paintHeatmapCheckBox.setBounds(10, 490, 280, 25);
        paintMarginCheckBox.setBounds(10, 520, 280, 25);
        smoothMarginCheckBox.setBounds(10, 550, 280, 25);
        marginLevelLabel.setBounds(10, 580, 100, 25);
        marginLevelTextField.setBounds(115, 580, 70, 25);
        specifyMarginLevelButton.setBounds(190, 580, 100, 25);
        automaticDataDownloadCheckBox.setBounds(10, 610, 280, 25);
        waitingTimeLabel.setBounds(10, 640, 150, 25);
```

```
        waitingTimeTextField.setBounds(155, 640, 135, 25);
        changeWaitingTimeButton.setBounds(10, 670, 140, 25);
        changeWaitingTimeTextField.setBounds(155, 670, 135, 25);
        progressBar.setBounds(10, 710, 280, 10);

        colorIndicatorComponent.setBounds(300, 0, 115, getHeight());
        visualizationComponent.setBounds(415, 0, getWidth() - 415,
getHeight());

        shownFileTypeTextField.setEditable(false);
        currentFileNumberTextField.setEditable(false);
        totalFileNumberTextField.setEditable(false);
        waitingTimeTextField.setEditable(false);

        JPanel panel = new JPanel();
        panel.setLayout(null);
        panel.setBounds(0, 0, 300, getHeight());
        panel.setBackground(Color.LIGHT_GRAY);

        panel.add(selectFolderButton);
        panel.add(selectDefaultFolderButton);
        panel.add(selectedFolderLabel);
        panel.add(selectedFolderTextField);
        panel.add(fileNameLabel);
        panel.add(fileNameTextField);
        panel.add(fileNumberSlider);
        panel.add(dateLabel);
        panel.add(datePickerUtil.datePicker);
        panel.add(timeLabel);
        panel.add(timeTextField);
        panel.add(selectDateAndTimeButton);
        panel.add(shownFileTypeTextField);
        panel.add(selectShownFilesTypeButton);
        panel.add(colorIndicatorLimitLabel);
        panel.add(colorIndicatorLimitTextField);
        panel.add(changeColorIndicatorLimitButton);
        panel.add(gotoPreviousFileButton);
        panel.add(gotoNextFileButton);
        panel.add(currentFileNumberLabel);
        panel.add(currentFileNumberTextField);
        panel.add(fromLabel);
        panel.add(totalFileNumberTextField);
        panel.add(gotoFileButton);
        panel.add(fileNumberTextField);
        panel.add(defaultOffsetAndScaleButton);
        panel.add(paintHeatmapCheckBox);
        panel.add(paintMarginCheckBox);
```

```
        panel.add(smoothMarginCheckBox);
        panel.add(marginLevelLabel);
        panel.add(marginLevelTextField);
        panel.add(specifyMarginLevelButton);
        panel.add(automaticDataDownloadCheckBox);
        panel.add(waitingTimeLabel);
        panel.add(waitingTimeTextField);
        panel.add(changeWaitingTimeButton);
        panel.add(changeWaitingTimeTextField);
        panel.add(progressBar);

        container.add(panel);
        container.add(colorIndicatorComponent);
        container.add(visualizationComponent);

        progressBar.setVisible(false);
    }

    /**
     * Отображаем выбранный тип файлов.
     */
    private void showFileType() {
        shownFileTypeTextField.setText(model.fileType.toString());
    }

    /**
     * Производим визуализацию.
     */
    public void visualize() {
        colorIndicatorLimitTextField

.setText(Float.toString(GeoinformationDataUnit.maxValue));

        if (model.geoinformationDataUnits.size() != 0) {
            int year =
DateAndTimeUtil.getYearFromFileName(model.getCurrentFileName());
            int month =
DateAndTimeUtil.getMonthFromFileName(model.getCurrentFileName());
            int day =
DateAndTimeUtil.getDayFromFileName(model.getCurrentFileName());
            int hour =
DateAndTimeUtil.getHourFromFileName(model.getCurrentFileName());
            int minute =
DateAndTimeUtil.getMinuteFromFileName(model.getCurrentFileName());

            datePickerUtil.sqlDateModel.setDate(year, month, day);
```

```
            timeTextField.setText(String.format("%02d", hour) + ":" +
String.format("%02d", minute));
        }

        paintMarginCheckBox.setSelected(model.showMarginFlag);
        paintHeatmapCheckBox.setSelected(model.showHeatmapFlag);
        smoothMarginCheckBox.setSelected(model.smoothMarginFlag);

marginLevelTextField.setText(Float.toString(model.marginLevel));

waitingTimeTextField.setText(Integer.toString(model.waitingTimeInSecon
ds));

        visualizationComponent.repaint();
        colorIndicatorComponent.repaint();
    }
}
```

1.5.8. Файл `PointF.java`

```
/*
 * В данном файле содержится реализация точки, координаты которой
 * являются числами с плавающей запятой.
 */

package view;

/**
 * Точка, координаты которой являются числа с плавающей запятой.
 *
 * @author Иван Шагурин
 */
public class PointF {
    /**
     * Точность, которая определяет равенство точек.
     */
    public static final float EPS = 0.01f;

    /**
     * Координата x.
     */
    public float x = 0;

    /**
     * Координата y.
     */
    public float y = 0;
```

```
    /**
     * Конструктор.
     *
     * @param x координата x.
     * @param y координата y.
     */
    public PointF(float x, float y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Устанавливаем приблизительное равенство между
     * данной точкой и указанным объектом.
     *
     * @param o указанный объект.
     * @return true, если упомянутое утверждение истинно. Иначе -
false.
     */
    public boolean approximatelyEquals(Object o) {
        if (this == o) {
            return true;
        } else if (getClass() != o.getClass()) {
            return false;
        } else {
            PointF other = (PointF) o;
            return (Math.abs(x - other.x) <= EPS) && (Math.abs(y -
other.y) <= EPS);
        }
    }
}
```

1.5.9. Файл `RadiusVector.java`

```
/*
 * В данном файле содержится реализация радиус-вектора.
 */

package view;

/**
 * Радиус-вектор.
 *
 * @author Иван Шагурин
 */
public class RadiusVector {
```

```
    /**
     * Конец радиус-вектора.
     */
    private PointF arrowhead = new PointF(0, 0);

    /**
     * Получаем координату x конца радиус-вектора.
     *
     * @return упомянутая координата x.
     */
    public float getArrowheadX() {
        return arrowhead.x;
    }

    /**
     * Получаем координату y конца радиус-вектора.
     *
     * @return упомянутая координата y.
     */
    public float getArrowheadY() {
        return arrowhead.y;
    }

    /**
     * Получаем длину радиус-вектора.
     *
     * @return упомянутая длина.
     */
    public double getLength() {
        return Math.sqrt(
                arrowhead.x * arrowhead.x
                        + arrowhead.y * arrowhead.y
        );
    }

    /**
     * Конструктор.
     *
     * @param xFrom координата x начала.
     * @param yFrom координата y начала.
     * @param xTo координата x конца.
     * @param yTo координата y конца.
     */
    public RadiusVector(float xFrom, float yFrom, float xTo, float
yTo) {
        arrowhead = new PointF(xTo - xFrom, yTo - yFrom);
    }
```

```
    /**
     * Применяем поворот к радиус-вектору.
     *
     * @param angle угол поворота.
     */
    public void applyRotation(double angle) {
        float xRotated
                = (float) (arrowhead.x * Math.cos(angle)
                - arrowhead.y * Math.sin(angle));
        float yRotated
                = (float) (arrowhead.x * Math.sin(angle)
                + arrowhead.y * Math.cos(angle));

        arrowhead.x = xRotated;
        arrowhead.y = yRotated;
    }

    /**
     * Применяем гомотетию к радиус-вектору.
     *
     * @param ratio коэффициент гомотетии.
     */
    public void applyHomothety(double ratio) {
        arrowhead.x = (float) (arrowhead.x * ratio);
        arrowhead.y = (float) (arrowhead.y * ratio);
    }
}
```

1.5.10. Файл `VisualizationComponent.java`

```
/*
 * В данном файле содержится реализация компоненты для визуализации.
 */

package view;

import model.GeoinformationDataUnit;
import model.GeoinformationDataUnitCoordinates;
import model.Segment;
```

```
import javax.swing.*;
import java.awt.*;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Collection;


/**
 * Компонента для визуализации.
 */
public class VisualizationComponent extends JComponent {
    /**
     * Цвет толстых линий координатной решетки.
     */
    private static final Color THICK_GRID_LINES_COLOR = new Color(193,
207, 215);


    /**
     * Цвет тонких линий координатной решетки.
     */
    private static final Color THIN_GRID_LINES_COLOR = new Color(120,
120, 120);


    /**
     * Тонкая кисть.
     */
    private static final Stroke THIN_STROKE = new BasicStroke(0.5f,
BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER);


    /**
     * Толстая кисть.
```

```
     */
    private static final Stroke THICK_STROKE = new BasicStroke(1f,
BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER);


    /**
     * Цвет надписей для толстых линий координатной решетки.
     */
    private static final Color THICK_GRID_LINES_LABELS_COLOR = new
Color(0, 0, 0);


    /**
     * Цвет границы.
     */
    public static final Color MARGIN_COLOR = new Color(255, 255, 255);


    /**
     * Кисть для рисования границы.
     */
    public static final Stroke MARGIN_STROKE
            = new BasicStroke(2.5f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER);


    /**
     * Шрифт надписей для толстых линий координатной решетки.
     */
    private static final Font THICK_GRID_LINES_LABELS_FONT = new
Font("Arial", Font.PLAIN, 16);


    /**
     * Шрифт надписей для единиц геоинформационных данных.
```

```
     */
    private static final Font GEOINFORMATION_DATA_UNITS_LABELS_FONT =
new Font("Arial", Font.PLAIN, 16);


    /**
     * Дисплей.
     */
    public Display display;


    /**
     * Главный фрейм.
     */
    private final MainFrame mainFrame;


    /**
     * Конструктор.
     *
     * @param mainFrame главный фрейм.
     */
    public VisualizationComponent(MainFrame mainFrame) {
        this.mainFrame = mainFrame;
    }


    /**
     * Инициализиурем дисплей.
     */
    public void initDisplay() {
        display = new Display(this);
    }
```

```
/**
 * Раскрашиваем компоненту.
 *
 * @param graphics the <code>Graphics</code> object to protect
 */
public void paintComponent(Graphics graphics) {
    Graphics2D graphics2D = (Graphics2D) graphics;
    adjustGraphics2D(graphics2D);

    paintBackground(graphics2D);

    if (mainFrame.model.showHeatmapFlag) {
        paintGeoinformationDataUnits(graphics2D);
    }

    paintThinGridCircles(graphics2D);
    paintThickGridLineCircles(graphics2D);
    paintThinGridLines(graphics2D);
    paintThickGridLines(graphics2D);

    if (mainFrame.model.showMarginFlag) {
        if (mainFrame.model.smoothMarginFlag) {
            mainFrame.model.prepareToDrawSmoothMargin();
            drawSmoothMargin(graphics2D);
        } else {
            paintMargin(graphics2D);
        }
    }
```

```
        paintThickGridLinesLabels(graphics2D);
        paintThickGridCirclesLabels(graphics2D);
        paintLabel(graphics2D);
    }

    /**
     * Настраиваем объект для рисования.
     *
     * @param graphics2D упомянутый объект для рисования.
     */
    private void adjustGraphics2D(Graphics2D graphics2D) {

graphics2D.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
        graphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

graphics2D.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
RenderingHints.VALUE_COLOR_RENDER_QUALITY);
        graphics2D.setRenderingHint(RenderingHints.KEY_DITHERING,
RenderingHints.VALUE_DITHER_ENABLE);

graphics2D.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
RenderingHints.VALUE_FRACTIONALMETRICS_ON);
        graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
        graphics2D.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
        graphics2D.setRenderingHint(RenderingHints.KEY_STROKE_CONTROL,
RenderingHints.VALUE_STROKE_PURE);
    }
```

```
/**
 * Красим фон.
 *
 * @param graphics2D объект для рисования.
 */
private void paintBackground(Graphics2D graphics2D) {
    graphics2D.setPaint(Color.WHITE);
    graphics2D.fill(new Rectangle(0, 0, getWidth(), getHeight()));

    display.fillOval(
            -display.getWidth() / 2,
            display.getHeight() / 2,
            display.getWidth() / 2,
            - display.getHeight() / 2,
            Color.BLACK,
            graphics2D
    );
}

/**
 * Красим тонкие окружности координатной решетки.
 *
 * @param graphics2D объект для рисования.
 */
private void paintThinGridCircles(Graphics2D graphics2D) {
    float stepX = display.getWidth() / 160f;
    float stepY = display.getHeight() / 160f;

    for (int i = 0; i < 80; ++i) {
```

```
            display.drawOval(
                    (int) (-display.getWidth() / 2 + stepX * i),
                    (int) (display.getHeight() / 2 - stepY * i),
                    (int) (display.getWidth() / 2 - stepX * i),
                    (int) (-display.getHeight() / 2 + stepY * i),
                    THIN_GRID_LINES_COLOR,
                    THIN_STROKE,
                    graphics2D
            );
        }
    }

    /**
     * Красим толстые окружности координатной решетки.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintThickGridLineCircles(Graphics2D graphics2D) {
        float stepX = display.getWidth() / 16f;
        float stepY = display.getHeight() / 16f;

        for (int i = 0; i < 8; ++i) {
            display.drawOval(
                    (int) (-display.getWidth() / 2 + stepX * i),
                    (int) (display.getHeight() / 2 - stepY * i),
                    (int) (display.getWidth() / 2 - stepX * i),
                    (int) (-display.getHeight() / 2 + stepY * i),
                    THICK_GRID_LINES_COLOR,
                    THICK_STROKE,
```

```
                    graphics2D
            );
        }
    }

    /**
     * Красим тонкие прямые линии координатной решетки.
     *
     * @param graphics2D объект для рисования.
     */
    public void paintThinGridLines(Graphics2D graphics2D) {
        RadiusVector radiusVector = new RadiusVector(
                0, 0, 0, display.getHeight() / 2f
        );
        double stepAngle = Math.PI / 48;

        for (int i = 0; i < 96; ++i) {
            display.drawLine(
                    0,
                    0,
                    (int) radiusVector.getArrowheadX(),
                    (int) radiusVector.getArrowheadY(),
                    THIN_GRID_LINES_COLOR,
                    THIN_STROKE,
                    graphics2D
            );

            radiusVector.applyRotation(stepAngle);
        }
```

```
}

/**
 * Красим толстые прямые линии координатной решетки.
 *
 * @param graphics2D объект для рисования.
 */
public void paintThickGridLines(Graphics2D graphics2D) {
    RadiusVector radiusVector = new RadiusVector(
            0, 0, 0, display.getHeight() / 2f
    );
    double stepAngle = Math.PI / 12;

    for (int i = 0; i < 24; ++i) {
        display.drawLine(
                0,
                0,
                (int) radiusVector.getArrowheadX(),
                (int) radiusVector.getArrowheadY(),
                THICK_GRID_LINES_COLOR,
                THICK_STROKE,
                graphics2D
        );

        radiusVector.applyRotation(stepAngle);
    }
}

/**
```

```
     * Красим единицы геоинформационных данных.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintGeoinformationDataUnits(Graphics2D graphics2D) {
        try {
            for (GeoinformationDataUnit value :
mainFrame.model.geoinformationDataUnits.values()) {
                paintGeoinformationDataUnit(
                        value,
                        graphics2D
                );
            }
        } catch (Exception ignored) {
        }
    }

    /**
     * Красим единицу геоинформационных данных.
     *
     * @param geoinformationDataUnit упомянутая единица
геоинформационных данных.
     * @param graphics2D объект для рисования.
     */
    private void paintGeoinformationDataUnit(GeoinformationDataUnit
geoinformationDataUnit,
                                             Graphics2D graphics2D) {
        PointF firstPoint =
geoinformationDataUnit.getFirstPoint(display.getWidth());
        PointF secondPoint =
geoinformationDataUnit.getSecondPoint(display.getWidth());
```

```
        PointF thirdPoint =
geoinformationDataUnit.getThirdPoint(display.getWidth());
        PointF fourthPoint =
geoinformationDataUnit.getFourthPoint(display.getWidth());

        float x1 = firstPoint.x;
        float y1 = firstPoint.y;
        float x2 = secondPoint.x;
        float y2 = secondPoint.y;
        float x3 = thirdPoint.x;
        float y3 = thirdPoint.y;
        float x4 = fourthPoint.x;
        float y4 = fourthPoint.y;

        display.fillTetragon((int)x1, (int)y1, (int)x2, (int)y2,
(int)x3, (int)y3, (int)x4, (int)y4,
                geoinformationDataUnit.getColor(), graphics2D);
    }

    /**
     * Красим надписи для толстых прямых линий координатной решетки.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintThickGridLinesLabels(Graphics2D graphics2D) {
        RadiusVector radiusVector = new RadiusVector(
                0, 0, 0, (float) (-display.getHeight() / 2 * 1.05)
        );
        double stepAngle = Math.PI / 4;
        int label = 0;
```

```
        for (int i = 0; i < 8; ++i) {
            display.drawString(
                    String.valueOf(label),
                    (int) (radiusVector.getArrowheadX()),
                    (int) (radiusVector.getArrowheadY()),
                    8, -4,
                    THICK_GRID_LINES_LABELS_COLOR,
                    THICK_GRID_LINES_LABELS_FONT,
                    graphics2D
            );

            radiusVector.applyRotation(stepAngle);
            label += 3;
        }
    }

    /**
     * Красим надписи для толстых окружностей координатной решетки.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintThickGridCirclesLabels(Graphics2D graphics2D) {
        RadiusVector radiusVector = new RadiusVector(
                0, 0, 0, -display.getHeight() / 2f
        );
        radiusVector.applyRotation(Math.PI / 12);
        int label = 50;
        double l0 = (new RadiusVector(
```

```
                0, 0, display.getWidth() / 23f,
                display.getHeight() / 23f
        )).getLength();

        for (int i = 0; i < 7; ++i) {
            display.drawString(
                    String.valueOf(label),
                    (int) radiusVector.getArrowheadX(),
                    (int) radiusVector.getArrowheadY(),
                    0,
                    0,
                    THICK_GRID_LINES_COLOR,
                    THICK_GRID_LINES_LABELS_FONT,
                    graphics2D
            );

            double ratio = (radiusVector.getLength() - l0) /
radiusVector.getLength();
            radiusVector.applyHomothety(ratio);

            label += 5;
        }
    }

    /**
     * Красим надпись.
     *
     * @param graphics2D объект для рисования.
     */
```

```
    private void paintLabel(Graphics2D graphics2D) {
        if (mainFrame.model.geoinformationDataUnits.size() == 0) {
            return;
        } else if (!mainFrame.model.showInfoFlag) {
            return;
        }

        CoordinateAdapter.firstCoordinateAdapter.setDisplay(display);

        CoordinateAdapter.firstCoordinateAdapter
                .set(mainFrame.model.mouseAdapterX,
                        mainFrame.model.mouseAdapterY,
                        CoordinateSystem.COMPONENT);

        int xDisplay =
CoordinateAdapter.firstCoordinateAdapter.getX(CoordinateSystem.DISPLAY
);
        int yDisplay =
CoordinateAdapter.firstCoordinateAdapter.getY(CoordinateSystem.DISPLAY
);

        double standardPolarDistance
                = Math.sqrt(xDisplay * xDisplay + yDisplay *
yDisplay);
        double standardPolarAngle = Math.atan2(yDisplay, xDisplay);

        double modifiedPolarDistance
                = 90 - standardPolarDistance / display.getWidth() *
80;
        double modifiedPolarAngle = standardPolarAngle * 12 / Math.PI
+ 6;
```

```
        if (modifiedPolarAngle < 0) {
            modifiedPolarAngle += 24;
        }

        modifiedPolarDistance =
roundModifiedPolarDistance(modifiedPolarDistance);
        modifiedPolarAngle =
roundModifiedPolarAngle(modifiedPolarAngle);

        if (modifiedPolarDistance < 50 || modifiedPolarDistance >
89.5) {
            return;
        }

        double geoinformationDataUnitValue
                = mainFrame.model.getGeoinformationDataUnitValue(
                (float) modifiedPolarDistance, (float)
modifiedPolarAngle);

        DecimalFormat decimalFormat = new DecimalFormat("#0.00");

        display.fillLabel(
                modifiedPolarAngle + ", "
                + modifiedPolarDistance + ", "
                + decimalFormat.format(geoinformationDataUnitValue),
                mainFrame.model.mouseAdapterX,
                mainFrame.model.mouseAdapterY,
                Color.GRAY,
                Color.WHITE,
```

```
                GEOINFORMATION_DATA_UNITS_LABELS_FONT,
                graphics2D
        );
    }

    /**
     * Рисуем границу.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintMargin(Graphics2D graphics2D) {
        paintRadialMarginLines(graphics2D);
        paintCircleMarginLines(graphics2D);
    }

    /**
     * Рисуем сглаженную границу.
     *
     * @param graphics2D объект для рисования.
     */
    private void drawSmoothMargin(Graphics2D graphics2D) {
        for (Segment middleSegment :
mainFrame.model.middleMarginSegments) {
            if (middleSegment.getFirstNeighbour() != null) {

drawBezierCurve(middleSegment.getFirstNeighbour().getMiddle(),

middleSegment.getIntersectionWithFirstNeighbour(),
                        middleSegment.getMiddle(),
                        MARGIN_COLOR,
```

```
                        MARGIN_STROKE,
                        graphics2D);
            }

            if (middleSegment.getSecondNeighbour() != null) {
                drawBezierCurve(middleSegment.getMiddle(),
middleSegment.getIntersectionWithSecondNeighbour(),
middleSegment.getSecondNeighbour().getMiddle(),
                        MARGIN_COLOR,
                        MARGIN_STROKE,
                        graphics2D);
            }
        }
    }

    /**
     * Рисуем радиальные линии границы.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintRadialMarginLines(Graphics2D graphics2D) {
        if (mainFrame.model.geoinformationDataUnits.size() == 0) {
            return;
        }

        for (float polarAngle = 0; polarAngle <= 23.75; polarAngle +=
0.25) {
```

```
            for (float polarDistance = 50; polarDistance <= 89.5;
polarDistance += 0.5) {
                GeoinformationDataUnit first
                        = mainFrame.model.geoinformationDataUnits.get(
                        new
GeoinformationDataUnitCoordinates(polarAngle, polarDistance));

                if (first == null) {
                    continue;
                }

                float nextPolarAngle = polarAngle - 0.25f;

                if (nextPolarAngle < 0) {
                    nextPolarAngle = 23.75f;
                }

                GeoinformationDataUnit second
                        = mainFrame.model.geoinformationDataUnits.get(
                        new
GeoinformationDataUnitCoordinates(nextPolarAngle, polarDistance));

                if (second == null) {
                    continue;
                }

                if (marginIsLocated(first.getValue(),
second.getValue())) {
                    PointF firstPoint =
first.getFirstPoint(display.getWidth());
```

```
                    PointF fourthPoint =
first.getFourthPoint(display.getWidth());


                    display.drawLine(
                            (int) firstPoint.x,
                            (int) firstPoint.y,
                            (int) fourthPoint.x,
                            (int) fourthPoint.y,
                            MARGIN_COLOR,
                            MARGIN_STROKE,
                            graphics2D
                    );
                }
            }
        }
    }

    /**
     * Рисуем линии границы, которые приближенно являются круговыми.
     *
     * @param graphics2D объект для рисования.
     */
    private void paintCircleMarginLines(Graphics2D graphics2D) {
        if (mainFrame.model.geoinformationDataUnits.size() == 0) {
            return;
        }

        for (float polarAngle = 0; polarAngle <= 23.75; polarAngle +=
0.25) {
```

```
            for (float polarDistance = 50.5f; polarDistance <= 89.5;
polarDistance += 0.5) {

                GeoinformationDataUnit first

                        = mainFrame.model.geoinformationDataUnits.get(

                        new
GeoinformationDataUnitCoordinates(polarAngle, polarDistance));


                if (first == null) {

                    continue;

                }


                float nextPolarDistance = polarDistance - 0.5f;


                GeoinformationDataUnit second

                        = mainFrame.model.geoinformationDataUnits.get(

                        new
GeoinformationDataUnitCoordinates(polarAngle, nextPolarDistance));


                if (second == null) {

                    continue;

                }


                if (marginIsLocated(first.getValue(),
second.getValue())) {

                    PointF firstPoint =
first.getFirstPoint(display.getWidth());

                    PointF secondPoint =
first.getSecondPoint(display.getWidth());


                    display.drawLine(

                            (int) firstPoint.x,
```

```
                        (int) firstPoint.y,
                        (int) secondPoint.x,
                        (int) secondPoint.y,
                        MARGIN_COLOR,
                        MARGIN_STROKE,
                        graphics2D
                    );
                }
            }
        }
    }

    /**
     * Округляем модифицированное полярное расстояние.
     *
     * @param modifiedPolarDistance упомянутое модифицированное
полярное расстояние.
     * @return округленное модицифицированное полярное расстояние.
     */
    private static double roundModifiedPolarDistance(double
modifiedPolarDistance) {
        double delta = modifiedPolarDistance -
Math.floor(modifiedPolarDistance);
        modifiedPolarDistance = Math.floor(modifiedPolarDistance);

        if (delta < 0.5) {
            delta = 0;
        } else {
            delta = 0.5;
        }
```

```
        return modifiedPolarDistance + delta;
    }


    /**
     * Округляем модифицированный полярный угол.
     *
     * @param modifiedPolarAngle упомянутый модифицированный полярный
угол.
     * @return округлыенный модифицированный полярный угол.
     */
    private static double roundModifiedPolarAngle(double
modifiedPolarAngle) {
        double delta = modifiedPolarAngle -
Math.floor(modifiedPolarAngle);
        modifiedPolarAngle = Math.floor(modifiedPolarAngle);


        if (delta < 0.25) {
            delta = 0;
        } else if (delta < 0.5) {
            delta = 0.25;
        } else if (delta < 0.75) {
            delta = 0.5;
        } else {
            delta = 0.75;
        }


        return modifiedPolarAngle + delta;
    }
```

```
    /**
     * Проверяем, что между единицами геоинформационных данных со
значениями
     * <code>firstValue</code> и <code>secondValue</code> пролегает
ребро границы.
     *
     * @param firstValue значение первой единицы геоинформационных
данных.
     * @param secondValue значение второй единицы геоинформационных
данных.
     * @return true, если упомянутое утверждение истинно. Иначе -
false.
     */
    public boolean marginIsLocated(double firstValue, double
secondValue) {
        return (
                (firstValue >= mainFrame.model.marginLevel) &&
(secondValue < mainFrame.model.marginLevel)
                ) || (
                (firstValue < mainFrame.model.marginLevel) &&
(secondValue >= mainFrame.model.marginLevel)
                );
    }


    /**
     * Рисуем кривую Безье по трем точкам.
     *
     * @param firstPoint первая точка для рисования кривой Безье.
     * @param secondPoint вторая точка для рисования кривой Безье.
     * @param thirdPoint третья точка для рисования кривой Безье.
     * @param color цвет кривой Безье.
     * @param stroke кисть для рисования кривой Безье.
```

```
     * @param graphics2D объект для рисования.
     */
    public void drawBezierCurve(PointF firstPoint, PointF secondPoint,
PointF thirdPoint,
                                Color color, Stroke stroke, Graphics2D
graphics2D) {
        Segment firstSegment = new Segment(firstPoint, secondPoint);
        Segment secondSegment = new Segment(secondPoint, thirdPoint);


        PointF previousPoint = null;


        for (float t = 0; t <= 1; t += 0.01) {
            Segment intermediateSegment = new
Segment(firstSegment.getRatioPoint(t),
secondSegment.getRatioPoint(t));
            PointF currentPoint =
intermediateSegment.getRatioPoint(t);


            if (previousPoint != null) {
                display.drawLine(
                        (int) previousPoint.x,
                        (int) previousPoint.y,
                        (int) currentPoint.x,
                        (int) currentPoint.y,
                        color,
                        stroke,
                        graphics2D
                );
            }


            previousPoint = currentPoint;
```

```
            }
        }
    }
```

ПРИЛОЖЕНИЕ 1

Список использованных источников

1) ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2) ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3) ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4) ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5) ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7) ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8) ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9) ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10) ГОСТ 15150-69 Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды. – М.: Изд-во стандартов, 1997.
11) ГОСТ 19.602-78 Правила дублирования, учета и хранения программных документов, выполненных печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
12) ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

RU.17701729.10.03-01 12 01-1

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.10.03-01 12 01-1				
Инв. № подл.	Подп. и	Взам. инв.	Инв. №	Подп. и