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

ОТЧЕТ

по учебной практике

Тема: Алгоритм Борувки

Студент гр. 9303 _____ Лойконен М.Р.

Студент гр. 9303 _____ Микулик Д.П.

Студент гр. 9303 _____ Халилов Ш.А.

Руководитель _____ Ефремов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Лойконен М.Р., гр. 9303

Студент Микулик Д.П., гр. 9303

Студент Халилов Ш.А., гр. 9303

Тема работы: Алгоритм Борувки

Задание на практику:

Разработка визуализатора алгоритма Борувки на Java.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Сроки проведения практики: 01.07.2021-14.07.2021

Дата сдачи отчета: 14.07.2021

Студент _____ Лойконен М.Р.

Студент _____ Микулик Д.П.

Студент _____ Халилов Ш.А.

Руководитель _____ Ефремов М. А.

АННОТАЦИЯ

Цель учебной практики заключается в разработке GUI-приложения, которое позволяет создать граф и визуализировать алгоритм Борувки для нахождения минимального остовного дерева в созданном графе.

Разработка программы происходит на языке Java командой из 3-х человек. Роли каждого из членов команды распределены в соответствии с имеющимися задачами. Сдача и демонстрация проекта происходит согласно плану разработки.

SUMMARY

The purpose of the training practice is to develop a GUI application that allows you to create a graph and visualize the Boruvka algorithm for finding the minimum spanning tree in the created graph.

The program is developed in Java by a team of 3 people. The roles of each of the team members are distributed in accordance with the existing tasks. Delivery and demonstration of the project takes place according to the development plan.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практической работы является разработка графического приложения, выполняющего визуализацию работы алгоритма Борувки нахождения минимального остовного дерева графа.

При разработке приложения планируется реализация графического интерфейса для задания графа (напрямую, используя функционал приложения или из файла, где перечислены вершины и ребра графа). Также важной частью приложения является пошаговая визуализация алгоритма Борувки.

Разработка осуществляется на языке Java с использованием фреймворка Swing командой из трёх человек. В команде распределяются обязанности – разработка алгоритма, разработка интерфейса и визуализации, сборка и тестирование программы.

1. СПЕЦИФИКАЦИЯ ПРОГРАММЫ

1.1 ПРОТОТИП ИНТЕРФЕЙСА И USE-CASE ДИАГРАММА

Опишем изначальные требования к программе:

1. Программа представляет собой визуализацию алгоритма Борувки нахождения наименьшего остовного дерева.
2. Граф, на котором выполняется алгоритм, можно загружать из файла, а также создавать или модифицировать в самой программе.
3. Предполагается пошаговая визуализация алгоритма с возможностью отката на предыдущий шаг.
4. При визуализации алгоритма отображается дополнительная информация (например, разные компоненты связности окрашиваются в разные цвета).

USE-CASE диаграмма:

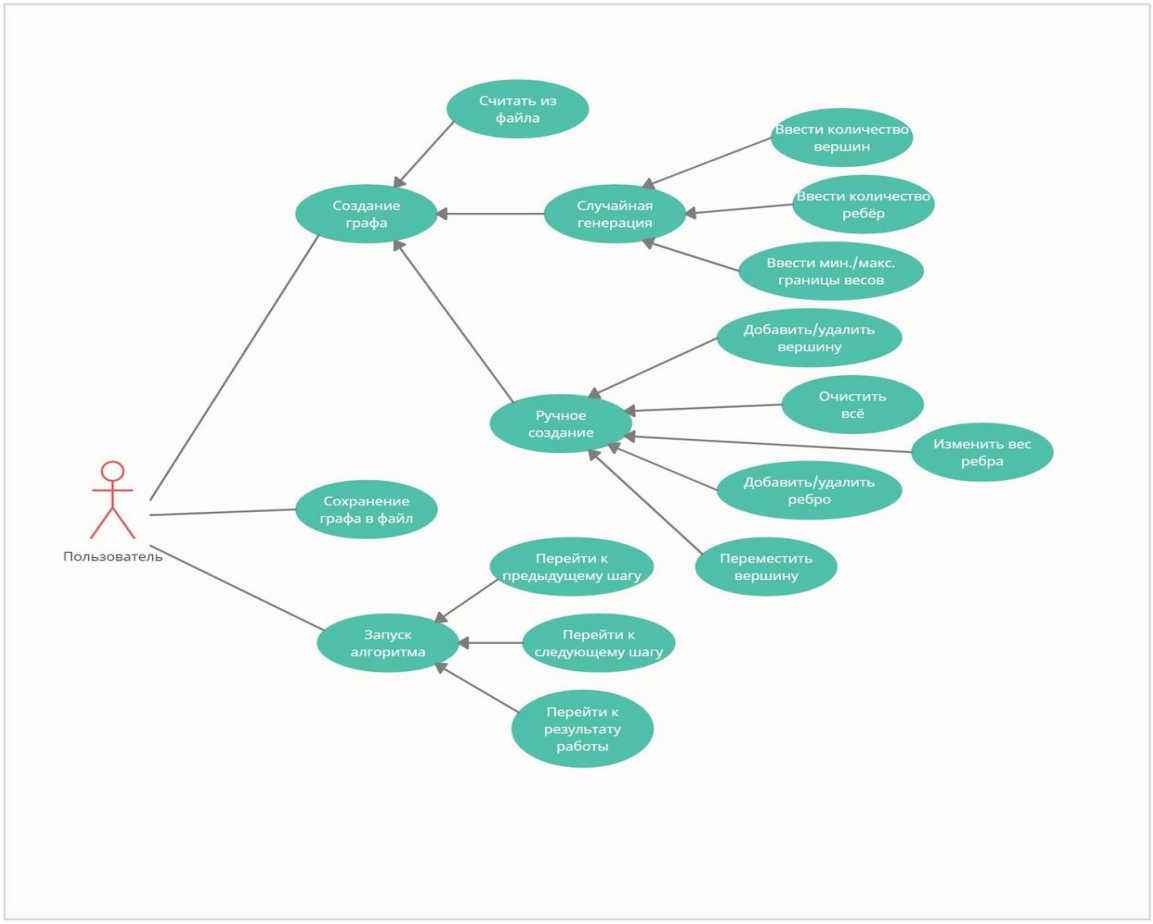

Рисунок 1 - USE-CASE диаграмма

Для реализации требований к функционалу, описанных в USE-CASE диаграмме, в прототип были внесены следующие элементы GUI:

- Три нижние кнопки на боковой панели инструментов позволяют реализовать функционал запуска алгоритма, получения результата работы алгоритма, а так же откат на предыдущий или следующие шаги работы алгоритма.
- Для возможности сохранения и загрузки графа в/из файла была добавлена панель меню, с соответствующим активным элементом «Файл».
- Генерация случайного графа также происходит при помощи вызова соответствующей команды из элемента «Файл» на панели меню.
- Для ручного создания графа служат три верхние кнопки боковой панели инструментов, которые позволяют добавить вершину или ребро, либо же очистить все.
- Созданы две сцены для визуализации графа и алгоритма.

Прототип интерфейса программы:

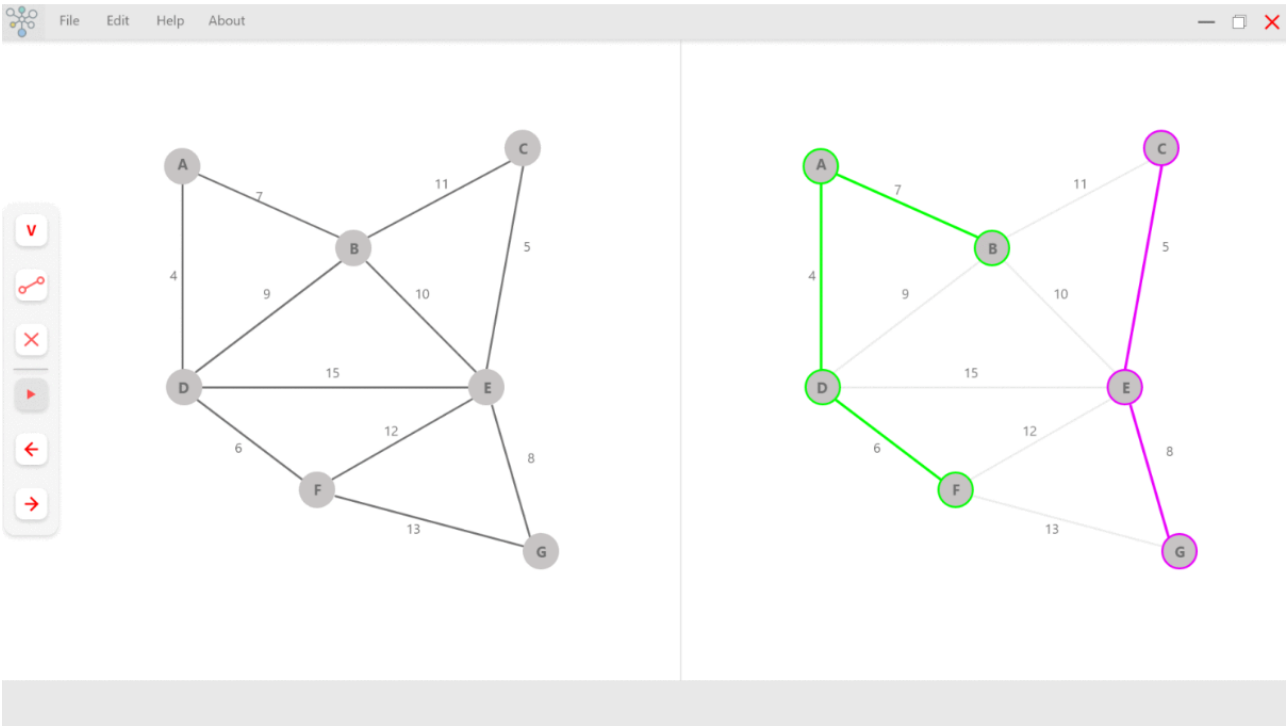

Рисунок 2- Прототип интерфейса программы

2. ОРГАНИЗАЦИЯ РАБОТЫ

2.1 План разработки

Задачи работы: изучение основ Java, создание приложения с графическим интерфейсом на Java (используя библиотеку Swing) для визуализации работы алгоритма, сборка (используя Maven) и тестирование программы.

В соответствии с имеющимися задачами был составлен план разработки приложения:

1. К 02.07.2021 изучить основы Java для дальнейшей работы над проектом, создать репозиторий проекта.
2. К 03.07.2021 обсудить спецификацию программы (представить прототип окна GUI приложения, представление USE-CASE диаграммы).
3. К 06.07.2021 разработать интерфейс приложения, начать реализацию базового функционала приложения (реализация алгоритма Борувки), окончательно проработать архитектуру приложения, построить UML-диаграмму классов.
4. К 08.07.2021 реализовать базовый функционал приложения, связать реализацию функционала (базового, т.е. реализация и визуализация алгоритма Борувки) и интерфейса.
5. К 10.07.2021 расширить имеющийся функционал приложения (например, реализовать разные способы задания графа), убрать критичные ошибки и баги.
6. К 12.07.2021 произвести тестирование и отладку программы, при необходимости этапу тестирования может предшествовать рефакторинг имеющегося кода. Сдача проекта.

2.2 Распределение ролей в команде

Обязанности между участниками команды распределены следующим образом:

Лойконен М.Р. – реализация алгоритма Борувки, тестирование программы.

Микулик Д.П. – реализация классов, связующих интерфейс и бизнес-логику (в данном случае сам алгоритм Борувки), написание отчета.

Халилов Ш.А. – реализация графического интерфейса, сборка программы, реализация прототипа графического интерфейса.

3. ОПИСАНИЕ КОДА ПРОГРАММЫ

3.1 Описание алгоритма и используемых структур данных

Для понимания реализованных классов, реализующих логику приложения, требуется добавить некоторые теоретические сведения об алгоритме Борувки, а также описать подробнее классы, которые обеспечивают хранение графа.

Алгоритм Борувки - алгоритм поиска минимального остовного дерева во взвешенном неориентированном связном графе.

Алгоритм состоит из нескольких шагов:

1. Изначально каждая вершина графа G.— тривиальное дерево, а ребра не принадлежат никакому дереву.
2. Для каждого дерева T найдем минимальное инцидентное ему ребро. Добавим все такие ребра.
3. Повторяем шаг 2 пока в графе не останется только одно дерево T.

Для хранения графа были реализованы три класса: `Graph`, `Edge`, `Node`. Здесь `Graph` – основной класс, который представляет собой граф. Имеет следующие поля:

`ArrayList<Node> Vertices` – массив вершин графа.

`ArrayList<Edge> Edges` – массив ребер графа (каждое ребро имеет ссылки на стартовую и конечную вершины в массиве Vertices).

`Int CountVertices` – количество вершин графа.

`Int CountEdges` – количество ребер графа.

Реализованные методы данного класса представляют функционал для добавление и удаления вершин и ребер, а так же для получения состояния полей графа. Также был переопределен метод `clone()` для корректной создании копии графа.

Класс `Node` предназначен для хранения вершины графа. Имеет следующие поля:

`String name` – имя текущей вершины.

`Int component` – номер компоненты связности, к которой принадлежит текущая вершина.

`Int x` – координата x вершины на холсте.

`Int y` – координата y вершины на холсте.

Реализованные методы позволяют получать и задавать состояния имеющихся полей у класса вершины. Также переопределены методы `equals()`, `hashcode()` (для того, чтобы имелась возможность сравнения двух вершин) и `clone()`, для возможности создания копии вершины (для корректного сохранения состояние графа).

Класс `Edge` предназначен для хранения ребра графа. Имеет следующие поля:

`Node start` – хранит ссылку на стартовую вершину у ребра.

`Node end` – хранит ссылку на конечную вершину у ребра.

`Int weight` – хранит вес ребра.

Реализованные методы позволяют получать состояния полей класса ребра, а также изменять вес имеющегося ребра. Были переопределены методы

clone() для создания корректной копии ребра, методы equals() и hashcode() для корректного сравнения двух ребер а также метод toString() для вывода осмысленной информации о ребре.

3.2 Описание основных классов для реализации логики приложения

Для реализации функционала приложения были реализованы следующие классы:

- Graph, Edge, Node – классы, для хранения графа.
- Algorithm – общий интерфейс алгоритмов поиска МОД, BoruvkaAlg – класс, реализующий интерфейс Algorithm, МОД находится при помощи алгоритма Борувки. Основной метод, который позволяет сделать шаг алгоритма: algorithmStep().
- Facade – класс, обобщающий имеющуюся логику (граф, алгоритм, классы сохранения и команды).
- CareTaker, AlgorithmMemento – набор классов для реализации хранения промежуточных состояний при работе алгоритма (реализуют паттерн «Снимок»)
- Command, LoadCommand, GenerateCommand, VisualizeCommand – классы-команды, которые будут исполняться для загрузки, генерации и визуализации алгоритма Борувки.

Для реализации выбранной архитектуры приложения использовались следующие паттерны:

- Снимок – для реализации функционала сохранения шагов алгоритма. Реализован с помощью классов: AlgorithmMemento (является вложенным классом для BoruvkaAlg), который сохраняет состояние алгоритма, CareTaker, который предоставляет методы backup() и

`undo()` для сохранения промежуточного состояния алгоритма и отката к предыдущему состоянию.

- Команда – для реализации основных команд, не связанных с ручным созданием графа, а именно: команда по загрузке графа из файла (класс `LoadCommand`), генерации графа (класс `GenerateCommand`), запуску визуализации алгоритма (класс `VisualizeCommand`).
- Фасад – для обобщения всех классов, которые реализуют логику (классов графа, алгоритма, команды и сохранения состояния).

Ниже для конкретного понимания реализованных классов и их взаимосвязи представлена UML диаграмма классов.

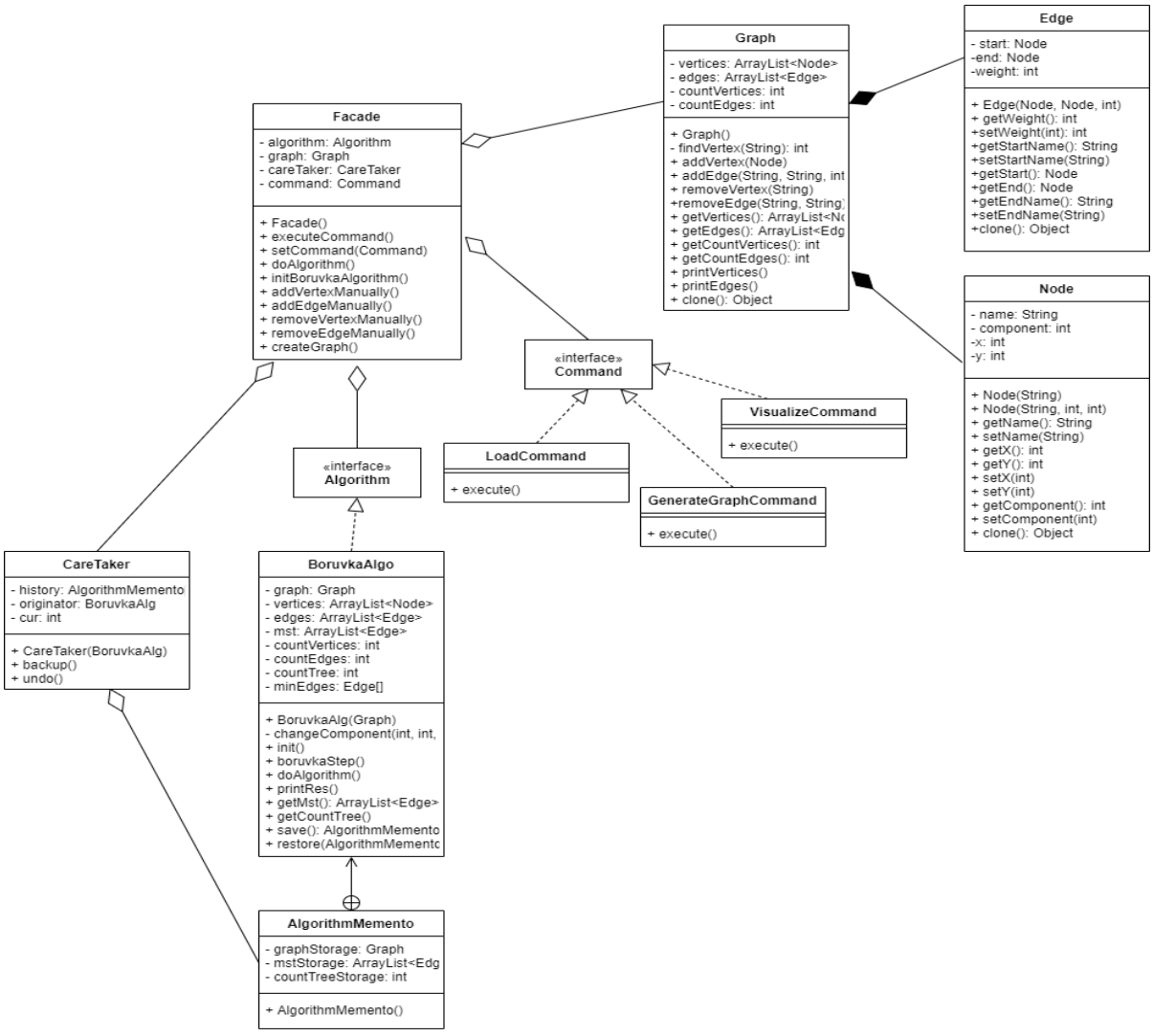

Рисунок 3 - UML диаграмма классов

3.3 Описание реализации пошагового исполнения алгоритма

Для реализации пошаговой работы алгоритма, помимо паттерна «Снимок» , который был несколько видоизменен: вместо стека хранится массив истории сохранений, для возможности получения доступа к элементу по произвольному индексу. Классический же вариант паттерна этого не подразумевает, поэтому также возникла потребность в добавлении следующих методов:

- `stepNextMemento()` – для получения следующего (а не предыдущего) состояния алгоритма.
- `stepFirstMemento()` – для получения самого первого состояния алгоритма.
- `stepLastMemento()` – для получения последнего состояния алгоритма.

В качестве элементов интерфейса были реализованы следующие кнопки на левой панели инструментов (перечислены в порядке «сверху вниз»):

- Кнопка «назад» – откат на предыдущий шаг.
- Кнопка «к результату» – откат на самый последний шаг.
- Кнопка «к началу» – переход в самое начальное состояние работы алгоритма.
- Кнопка «вперед» – переход на следующее состояние работы алгоритма.

Также для корректной работы алгоритма был добавлен режим редактирования (самая верхняя кнопка на панели инструментов), которая позволяет переключать режимы редактирования графа и запуска алгоритма. Такой подход позволяет избежать неоднозначности, т.к. классы для логических операций над графом и визуализации являются различными.

4. ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для проверки правильности реализованной логики работы алгоритма был реализован набор тестов, который представлен в таблице ниже. Здесь под графом G подразумевается следующий граф: граф-треугольник, с тремя вершинами (a, b, c), ребрами (a, b) с весом 3, (a, c) с весом 1, (b, c) с весом 4.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	Граф G	true	Проверка метода поиска вершины b в графе (findExistingVertex())
2	Граф G	true	Проверка того, что в графе нет вершины g (findExistingVertex())
3	Граф G	IllegalArgumentException	Проверка генерации исключения при проверке наличия null вершины в графе.
4	Граф G	true	Проверка добавления вершины k в граф (addVertex())
5	Граф G	IllegalArgumentException	Проверка добавления null в качестве вершины графа
6	Граф G	true	Проверка добавления нового ребра (addEdge())
7	Граф G	Без изменений	Проверка добавления уже существующего ребра (addEdge())
8	Граф G	IllegalArgumentException	Добавление ребра с вершинами null
9	Граф G	IllegalArgumentException	Добавление ребра с нулевым весом
10	Граф G	true	Удаление существующей вершины (removeVertex())
11	Граф G	IllegalArgumentException	Удаление несуществующей вершины (removeVertex())
12	Граф G	IllegalArgumentException	Удаление null- вершины (removeVertex())
13	Граф G	true	Удаление существующего ребра (removeEdge())

14	Граф G	IllegalArgumentException	Удаление несуществующего ребра (removeEdge())
15	Граф G	IllegalArgumentException	Удаление null-ребра (removeEdge())
16	Граф G	true	Удаление ребра-петли (removeEdge())
17	Граф G	true	Получение списка вершин (GetVertices())
18	Граф G	true	Получение списка ребер (GetEdges())
19	Граф G	true	Получение числа вершин (GetCountVertices())
20	Граф G	True (число вершин уменьшилось на 1)	Получение числа вершин после удаления вершины (GetCountVertices())
21	Граф G	True (число вершин не изменилось)	Получение числа вершин после удаления ребра (GetCountVertices())
22	Граф G	true	Получение числа ребер (GetCountEdges())
23	Граф G	True (число ребер не иззменилось)	Получение числа ребер после удаления вершины (GetCountEdges())
24	Граф G	True (число ребер уменьшилось на 1)	Получение числа ребер после удаления ребра (GetCountEdges())
25	Граф G	True (получено корректное МОД)	Проверка работы алгоритма (doAlrorithm())

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие задачи:

- Были распределены роли участников команды, определена цель и план работы. Для большего понимания основ Java всеми участниками команды был пройден соответствующий электронный курс. Также было налажено взаимодействие членов команды (используя репозиторий GitHub). Однако стоит отметить, что во время ведения проекта, из-за неучтенных обстоятельств (малого опыта работы в команде), команда испытывала трудности в синхронизации работы.
- Была создана и согласована спецификация программы, подготовлен эскиз окна приложения а так же USE-CASE и UML диаграммы.
- Было создано графическое приложение, удовлетворяющее большинству требований, которые были оговорены на этапе согласования спецификации программы.
- Были исправлены критические баги при работе приложения, реализован дополнительный функционал (покраска компонент связности в разные цвета).
- Было проведено тестирование программы (а именно логики), для проверки правильности реализации логики приложения.
- Стоит отметить, что не всегда получалось выполнять все требования в срок, ввиду ограниченности (сроков), не всегда удачной коммуникации членов команды и отсутствия опыта использования Java ранее.

Обобщив все вышесказанное, можно сказать, что была достигнута цель учебной практики: разработка графического приложения, выполняющего

визуализацию работы алгоритма Борувки нахождения минимального остовного дерева графа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шилдт, Г., 2015. Java8. Полное руководство. Издательский дом "Вильямс", Москва.
2. Эккель, Б., 2018. Философия Java. Издательство "Питер".
3. Хорстман, К., 2019. Java. Библиотека профессионала, том 1. ООО "Диалектика".
4. Блох, Дж., 2014. Java. Эффективное программирование. Издательство "Лори".

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: Main.java

```
package com.practice.Gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.HashMap;

import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JToolBar;
import javax.swing.SwingConstants;
import javax.swing.border.LineBorder;

import com.practice.Graph.Graph;
import com.practice.Graph.Node;
import com.practice.Utilts.Facade;
import com.practice.Utilts.GenerateCommand;
import com.practice.Utilts.LoadCommand;
import com.practice.Utilts.LoadGraphManuallyCommand;


public class Main extends JFrame {

    static Boolean isRedactorModeOn = true;
    private Scene leftPanel;
    private Scene rightPanel;
    private JLabel statusLabel;
    private final String[] alphabet = new String[]
    {
        "A", "B", "C", "D", "E", "F", "G",
        "H", "I", "J", "K", "L", "M", "N",
        "O", "P", "Q", "R", "S", "T", "U",
        "V", "W", "X", "Y", "Z"
    };
```

```
    private int counter, s_count;
    private int offset = 65;
    private JButton addVertex_btn , addRib_btn , delete_btn, clear_btn,
back_btn, runStop_btn, first_btn, forward_btn, redactor_btn,
generate_btn;
    //static int countVertices, countEdges, min, max;
    static boolean isGenerationOk = false;
    enum Option {
        NONE,
        CREATE,
        CONNECT,
        DELETE,
        CLEAR,
        RUN,
        STOP,
        NEXT,
        PREV,
        FIRST
    }
    private Facade facade;
    public static Option currentOption = Option.NONE;

    public static void main(String[] args) {
        new Main();
    }

    public Main() {
        InitUI();
        this.pack();
        this.setVisible(true);
        facade = new Facade();
        /*Graph g = new Graph();
        g.addEdge("a", "b", 1);
        g.addEdge("b", "c", 4);
        g.addEdge("c", "d", 1);
        g.addEdge("a", "e", 3);
        g.addEdge("e", "f", 1);
        g.addEdge("d", "f", 2);
        BoruvkaAlg alg = new BoruvkaAlg(g);
        alg.doAlgorithm();*/
        //facade.setCommand(new GenerateCommand(3, 3, 4, 10));
        //facade.createGraph();
        //String str = "ABD";
        //System.out.println((int)(str.charAt(1)));

    }


    private void InitUI() {

        // получить размер экрана
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        int W = (int)screenSize.getWidth(); // получить ширину экрана
        int H = (int)screenSize.getHeight(); // получить высоту экрана
```

```
        setMinimumSize(new Dimension(W/2, H/2));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout( mainPanel, BoxLayout.X_AXIS));
        mainPanel.add(createLeftPanel());
        mainPanel.add(createRightPanel());

        add( mainPanel );
        add(createToolBar(), BorderLayout.WEST);
        add(createStatusPanel(), BorderLayout.SOUTH);
        setJMenuBar(createMenuBar());
    }



    public void updateStatus(String msg) {
        String offset = "        ";
        statusLabel.setText(offset+msg);
    }

    public String getName() {
        if(counter > 25) {
            counter = 0;
            s_count++;
        }
        if( s_count > 0 ) {
            return alphabet[counter++]+s_count;
        }
        return  alphabet[counter++];
    }


    private Scene createLeftPanel() {
        leftPanel = new Scene();
        leftPanel.addMouseListener( new MouseAdapter() {
            @Override
            public void mouseClicked( MouseEvent mouseEvent ) {
                if(currentOption == Option.CREATE) {
                    // add after full
                    String name = getName();
                    Vertex vertex = new Vertex(name, mouseEvent.getX(),
mouseEvent.getY(), Color.lightGray );
                    Vertex clone =  new Vertex(name, mouseEvent.getX(),
mouseEvent.getY(), Color.lightGray );

                    vertex.addMouseListener( new MouseAdapter() {

                        @Override
                        public void mouseClicked( MouseEvent e ) {
                            super.mouseClicked( e );

                            if( currentOption == Main.Option.DELETE ) {
```

```
                                ArrayList<Rib> ribs =
leftPanel.getRibList();
                                for(int i = ribs.size()-1; i >= 0; i--) {
                                    if( ribs.get(i).isConnect( vertex ) )
{
                                        ribs.remove(i);
                                    }
                                }
                                leftPanel.remove(vertex);
                                rightPanel.remove(clone);
                                leftPanel.revalidate();
                                leftPanel.repaint();
                                rightPanel.revalidate();
                                rightPanel.repaint();
                            }

                            if (e.getClickCount() == 2 && currentOption
!= Option.CONNECT && isRedactorModeOn) {

                                String answer = "";
                                String msg = "напишите новое имя
вершины";

                                int optionPane =
JOptionPane.QUESTION_MESSAGE;

                                for( int isCurrectName = 0; isCurrectName
< 1;  ) {

                                    answer =
JOptionPane.showInputDialog(null, msg, "изменить имя вершины",
optionPane);
                                    if(answer == null) {
                                            return;
                                    }
                                    else if(answer.length() > 2 ) {
                                        msg = "длина имени вершины не
должна превышать 2";
                                        optionPane =
JOptionPane.WARNING_MESSAGE;
                                        continue;
                                    }else if(answer.length() == 0 || "
".equals(answer) || "  ".equals(answer)) {
                                        msg = "имя вершины не может быть
пустым";
                                        optionPane =
JOptionPane.WARNING_MESSAGE;
                                        continue;
                                    }
                                    isCurrectName = 1;
                                }
                                //added bug fix for vertice name
replacement

                                HashMap<String, Vertex> dictLeft =
leftPanel.getVerticesDict();
```

```
                                    HashMap<String, Vertex> dictRight =
rightPanel.getVerticesDict();
                                    if (dictLeft.get(answer) == null) {
                                        dictLeft.remove(vertex.getId());
                                        vertex.setName(answer);
                                        dictLeft.put(vertex.getId(), vertex);
                                        dictRight.remove(clone.getId());
                                        clone.setName(answer);
                                        dictRight.put(clone.getId(), clone);
                                        //
                                    }
                                }
                            }

                            @Override
                            public void mousePressed( MouseEvent mouseEvent )
{
                                super.mousePressed( mouseEvent );
                                    vertex.setColour( Color.magenta );
                            }
                            @Override
                            public void mouseReleased( MouseEvent mouseEvent
) {
                                super.mouseReleased( mouseEvent );
                                vertex.setColour( Color.lightGray);
                            }
                        } );

                        clone.addMouseMotionListener(new MouseAdapter() {

                            @Override
                            public void mouseDragged( MouseEvent mouseEvent )
{
                                vertex.setLocation(clone.getLocation());
                                revalidate();
                                repaint();
                            }
                        });
                        vertex.addMouseMotionListener(new MouseAdapter() {
                            @Override
                            public void mouseDragged( MouseEvent mouseEvent )
{
                                clone.setLocation(vertex.getLocation());
                                revalidate();
                                repaint();
                            }
                        });

                        leftPanel.addVertex( vertex );
                        rightPanel.addVertex( clone );
                    }
                }
            } );
            return leftPanel;
        }
```

```
    public int convertNameToIndex(String name){
        if (name.length() >= 1){
            return (((int) name.charAt(0)) - offset);
        }
        return 0;
    }

    public int convertNameToCounter(String name){
        if (name.length() > 1){
            String substring = name.substring(1);
            return ( Integer.parseInt(substring));
        }
        return 0;
    }

    public void setNameCounter(ArrayList<Node> nodes){
        int maxCounter = 0;
        int maxStringCounter = 0;
        for (Node node : nodes){
            int cur = this.convertNameToIndex(node.getName());
            int curStringCounter =
this.convertNameToCounter(node.getName());
            if (cur > maxCounter)
                maxCounter = cur;
            if (curStringCounter > maxStringCounter)
                maxStringCounter = curStringCounter;
        }
        counter = maxCounter + 1;
        s_count = maxStringCounter;
    }

    public void setVerticesWhileLoading(ArrayList<Node> nodes){
        leftPanel.clear();
        rightPanel.clear();
        //updateStatus( "Clear...");
        s_count=counter = 0;
        revalidate();
        repaint();

        for (Node node : nodes){
            Vertex vertex = new Vertex(node.getName(), node.getX(),
node.getY(), Color.lightGray);
            Vertex clone = new Vertex(node.getName(), node.getX(),
node.getY(), Color.lightGray);
            vertex.setLocation(vertex.getLocation());
            clone.setLocation(vertex.getLocation());
            //leftPanel.addMouseListener( new MouseAdapter() {
            //    @Override
            //    public void mouseClicked( MouseEvent mouseEvent ) {
                  //if(currentOption == Option.CREATE) {
                      // add after full
                     // String name = getName();
                     // Vertex vertex = new Vertex(name,
mouseEvent.getX(), mouseEvent.getY(), Color.lightGray );
                     // Vertex clone =  new Vertex(name,
mouseEvent.getX(), mouseEvent.getY(), Color.lightGray );
```

```
            vertex.addMouseListener( new MouseAdapter() {

                @Override
                public void mouseClicked( MouseEvent e ) {
                    super.mouseClicked( e );

                    if( currentOption == Main.Option.DELETE ) {

                        ArrayList<Rib> ribs = leftPanel.getRibList();
                        for(int i = ribs.size()-1; i >= 0; i--) {
                            if( ribs.get(i).isConnect( vertex ) ) {
                                ribs.remove(i);
                            }
                        }
                        leftPanel.remove(vertex);
                        rightPanel.remove(clone);
                        leftPanel.revalidate();
                        leftPanel.repaint();
                        rightPanel.revalidate();
                        rightPanel.repaint();
                    }

                    if (e.getClickCount() == 2 && currentOption !=
Option.CONNECT && isRedactorModeOn) {

                        String answer = "";
                        String msg = "напишите новое имя вершины";

                        int optionPane = JOptionPane.QUESTION_MESSAGE;

                        for( int isCurrectName = 0; isCurrectName < 1;  )
{

                            answer = JOptionPane.showInputDialog(null,
msg, "изменить имя вершины", optionPane);
                            if(answer == null) {
                                return;
                            }
                            else if(answer.length() > 2 ) {
                                msg = "длина имени вершины не должна
превышать 2";
                                optionPane = JOptionPane.WARNING_MESSAGE;
                                continue;
                            }else if(answer.length() == 0 || "
".equals(answer) || "  ".equals(answer)) {
                                msg = "имя вершины не может быть пустым";
                                optionPane = JOptionPane.WARNING_MESSAGE;
                                continue;
                            }
                            isCurrectName = 1;
                        }
                        //added bug fix for vertice name replacement
                        HashMap<String, Vertex> dictLeft =
leftPanel.getVerticesDict();
```

```
                        HashMap<String, Vertex> dictRight =
rightPanel.getVerticesDict();
                        dictLeft.remove(vertex.getId());
                        vertex.setName(answer);
                        dictLeft.put(vertex.getId(), vertex);
                        dictRight.remove(clone.getId());
                        clone.setName(answer);
                        dictRight.put(clone.getId(), clone);
                        //
                    }
                }

                @Override
                public void mousePressed( MouseEvent mouseEvent ) {
                    super.mousePressed( mouseEvent );
                    vertex.setColour( Color.magenta );
                }
                @Override
                public void mouseReleased( MouseEvent mouseEvent ) {
                    super.mouseReleased( mouseEvent );
                    vertex.setColour( Color.lightGray);
                }
            } );

            clone.addMouseMotionListener(new MouseAdapter() {

                @Override
                public void mouseDragged( MouseEvent mouseEvent ) {
                    vertex.setLocation(clone.getLocation());
                    revalidate();
                    repaint();
                }
            });
            vertex.addMouseMotionListener(new MouseAdapter() {
                @Override
                public void mouseDragged( MouseEvent mouseEvent ) {
                    clone.setLocation(vertex.getLocation());
                    revalidate();
                    repaint();
                }
            });

            leftPanel.addVertex( vertex );
            rightPanel.addVertex( clone );
            revalidate();
            repaint();
                    //}

        }
            //} );



        //}
    }
```

```
    private Scene createRightPanel() {
        rightPanel = new Scene();
        rightPanel.setBackground(Color.white);
        rightPanel.setBorder(new LineBorder(new Color(232, 232, 232)));
        return rightPanel;
    }

    private JPanel createStatusPanel() {

        JPanel statusPanel = new JPanel();
        statusPanel.setPreferredSize(new Dimension(getWidth(), 42));
        statusPanel.setLayout(new BoxLayout(statusPanel,
BoxLayout.X_AXIS));
        String offset = "        ";
        statusLabel = new JLabel(offset+"[Redactor mode]");
        statusLabel.setHorizontalAlignment(SwingConstants.LEFT);
        statusPanel.add(statusLabel);

        return statusPanel;
    }


    // создать меню файла
    private JMenu createFileMenu(){
        // Создание выпадающего меню
        JMenu fileMenu = new JMenu("Файл");

        // Пункт меню "Читать из файла"
        JMenuItem read = new JMenuItem("Читать из файла");

        // добавить действие Обработка
        read.addActionListener(e -> {
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
            int option = fileChooser.showOpenDialog(getContentPane());
            if(option == JFileChooser.APPROVE_OPTION){
                File file1 = fileChooser.getSelectedFile();
                statusLabel.setText("Folder Selected: " +
file1.getAbsolutePath() );
                if (file1.getAbsolutePath().contains(".json")) {
                    //facade.setCommand(new
LoadCommand(file1.getAbsolutePath()));
                    //facade.createGraph();
                    LoadCommand loadCommand = new
LoadCommand(file1.getAbsolutePath());
                    Graph graph = loadCommand.execute();
                    graph.printVertices();
                    this.setVerticesWhileLoading(graph.getVertices());
                    this.setNameCounter(graph.getVertices());
                    leftPanel.setRibs(graph.getEdges());
                } else {
                    statusLabel.setText("You have chosen file of
incorrect format");
                }

            }else{
```

```
                statusLabel.setText("Open command canceled");
            }
        });
        /*
        // Пункт меню "Сохранить"
        JMenuItem save = new JMenuItem("Сохранить");

        // добавить обработчик действия для кнопки "Сохранить"
        save.addActionListener( actionEvent -> {
            // ...
        });*/

        // Пункт меню "Сохранить как"
        JMenuItem save_as = new JMenuItem("Сохранить как");

        // добавить обработчик действия для кнопки "Сохранить как"
        save_as.addActionListener( actionEvent -> {

            //String filename = JOptionPane.showInputDialog("Name this
file");
            JFileChooser fileChooser = new JFileChooser();
            //fileChooser.setSelectedFile(new File(filename));
            //fileChooser.showSaveDialog(fileChooser); ???
            //BufferedWriter writer;
            int option = fileChooser.showSaveDialog(null);
            if(option == JFileChooser.APPROVE_OPTION){
                try {
                /* writer = new BufferedWriter(new
FileWriter(fileChooser.getSelectedFile()));
                    writer.close();*/
                    if
(fileChooser.getSelectedFile().getAbsolutePath().contains(".json")) {
                        facade.setCommand(new
LoadGraphManuallyCommand(leftPanel.getRibList()));
                        facade.createGraph();

facade.saveGraph(fileChooser.getSelectedFile().getAbsolutePath());
                        JOptionPane.showMessageDialog(null, "File has
been saved","File Saved",JOptionPane.INFORMATION_MESSAGE);
                    } else {
                        JOptionPane.showMessageDialog(null, "Please,
choose file of .json format!","File Save went
wrong",JOptionPane.INFORMATION_MESSAGE);
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }else{
                statusLabel.setText("Open command canceled");
            }
        });


        // Пункт меню из команды с выходом из программы
        JMenuItem exit = new JMenuItem("Выход");
```

```
        // добавить обработчик действия для кнопки "Выход"
        exit.addActionListener( actionEvent -> System.exit(0) );

        fileMenu.add(read);              // Добавление в меню пункт
"Читать из файла"
        //fileMenu.add(save);              // Добавление в меню пункт
"Сохранить"
        fileMenu.add(save_as);           // Добавление в меню пункт
"Сохранить как"
        fileMenu.addSeparator();         // Добавление разделителя
        fileMenu.add(exit);              // Добавление в меню пункт
"Выход"

        return fileMenu;
    }
    // создать строку меню
    private JMenuBar createMenuBar()
    {
        JMenuBar menuBar = new JMenuBar();
        menuBar.add(createFileMenu());
        menuBar.add(createInfoMenu());

        return menuBar;
    }

    // создать информационное меню
    private JMenu createInfoMenu()
    {
        JMenu infoMenu = new JMenu("Информация");
        JMenuItem doc = new JMenuItem("Документация");
        JMenuItem aboutUs = new JMenuItem("О нас!");
        infoMenu.add(doc);

        doc.addActionListener(actionEvent -> new Documentation() );

        infoMenu.add(aboutUs);

        aboutUs.addActionListener(actionEvent -> new AboutUs() );
        return infoMenu;
    }

    public void ChangeCurrentOption( Option option ) {

        switch( option ) {
            case CONNECT:

                addRib_btn.setBackground( new Color( 250, 100, 100 ));
                addVertex_btn.setBackground(Color.WHITE);
                delete_btn.setBackground(Color.WHITE);
                clear_btn.setBackground(Color.WHITE);
                back_btn.setBackground(Color.WHITE);
                runStop_btn.setBackground(Color.WHITE);
                forward_btn.setBackground(Color.WHITE);
                first_btn.setBackground(Color.WHITE);
```

```
            break;
        case CLEAR:
            addRib_btn.setBackground( Color.WHITE);
            addVertex_btn.setBackground(Color.WHITE);
            delete_btn.setBackground(Color.WHITE);
            clear_btn.setBackground(new Color( 250, 100, 100 ));
            back_btn.setBackground(Color.WHITE);
            runStop_btn.setBackground(Color.WHITE);
            forward_btn.setBackground(Color.WHITE);
            first_btn.setBackground(Color.WHITE);
            break;
        case CREATE:
            addRib_btn.setBackground( Color.WHITE);
            addVertex_btn.setBackground(new Color( 250, 100, 100 ));
            delete_btn.setBackground(Color.WHITE);
            clear_btn.setBackground(Color.WHITE);
            back_btn.setBackground(Color.WHITE);
            runStop_btn.setBackground(Color.WHITE);
            forward_btn.setBackground(Color.WHITE);
            first_btn.setBackground(Color.WHITE);
            break;
        case RUN:
            addRib_btn.setBackground( Color.WHITE );
            runStop_btn.setBackground( new Color( 250, 100, 100 ));
            addVertex_btn.setBackground(Color.WHITE);
            delete_btn.setBackground(Color.WHITE);
            clear_btn.setBackground(Color.WHITE);
            back_btn.setBackground(Color.WHITE);
            forward_btn.setBackground(Color.WHITE);
            first_btn.setBackground(Color.WHITE);
            break;
        case NEXT:
            addRib_btn.setBackground( Color.WHITE );
            addVertex_btn.setBackground(Color.WHITE);
            delete_btn.setBackground(Color.WHITE);
            clear_btn.setBackground(Color.WHITE);
            back_btn.setBackground(Color.WHITE);
            runStop_btn.setBackground(Color.WHITE);
            forward_btn.setBackground(new Color( 250, 100, 100 ));
            first_btn.setBackground(Color.WHITE);
            break;
        case PREV:

            addRib_btn.setBackground( Color.WHITE );
            addVertex_btn.setBackground(Color.WHITE);
            delete_btn.setBackground(Color.WHITE);
            clear_btn.setBackground(Color.WHITE);
            runStop_btn.setBackground(Color.WHITE);
            forward_btn.setBackground(Color.WHITE);
            back_btn.setBackground(new Color( 250, 100, 100 ));
            first_btn.setBackground(Color.WHITE);

            break;
        case STOP:
            addRib_btn.setBackground( Color.WHITE );
            addVertex_btn.setBackground(Color.WHITE);
```

```
                delete_btn.setBackground(Color.WHITE);
                clear_btn.setBackground(Color.WHITE);
                back_btn.setBackground(Color.WHITE);
                runStop_btn.setBackground(Color.WHITE);
                forward_btn.setBackground(Color.WHITE);
                runStop_btn.setBackground(new Color( 250, 100, 100 ));
                first_btn.setBackground(Color.WHITE);

                break;
            case DELETE:
                addRib_btn.setBackground( Color.WHITE );
                addVertex_btn.setBackground(Color.WHITE);
                clear_btn.setBackground(Color.WHITE);
                back_btn.setBackground(Color.WHITE);
                runStop_btn.setBackground(Color.WHITE);
                forward_btn.setBackground(Color.WHITE);
                delete_btn.setBackground(new Color( 250, 100, 100 ));
                first_btn.setBackground(Color.WHITE);

                break;
            case FIRST:
                addRib_btn.setBackground( Color.WHITE );
                addVertex_btn.setBackground(Color.WHITE);
                clear_btn.setBackground(Color.WHITE);
                back_btn.setBackground(Color.WHITE);
                runStop_btn.setBackground(Color.WHITE);
                forward_btn.setBackground(Color.WHITE);
                delete_btn.setBackground(Color.WHITE);
                first_btn.setBackground(new Color( 250, 100, 100 ));
                break;
            default:
                addRib_btn.setBackground( Color.WHITE );
                addVertex_btn.setBackground(Color.WHITE);
                clear_btn.setBackground(Color.WHITE);
                back_btn.setBackground(Color.WHITE);
                runStop_btn.setBackground(Color.WHITE);
                forward_btn.setBackground(Color.WHITE);
                delete_btn.setBackground(Color.WHITE );
                first_btn.setBackground(Color.WHITE);
        }

        currentOption = option;
    }

    // создать панель инструментов
    private JToolBar createToolBar()
    {
        JToolBar toolBar = new JToolBar(1);

        redactor_btn = new JButton(new ImageIcon("resources/edit.png"));
        generate_btn = new JButton(new
ImageIcon("resources/gen_graph.png"));
        addVertex_btn = new JButton(new
ImageIcon("resources/vertex.png"));
        addRib_btn = new JButton(new ImageIcon("resources/rib.png"));
        delete_btn = new JButton(new ImageIcon("resources/delete.png"));
```

```
        clear_btn = new JButton(new ImageIcon("resources/trash.png"));
        back_btn = new JButton(new ImageIcon("resources/back.png"));
        runStop_btn = new JButton(new
ImageIcon("resources/go_result.png"));
        first_btn = new JButton(new
ImageIcon("resources/back_start.png"));
        forward_btn = new JButton(new
ImageIcon("resources/forward.png"));

        forward_btn.setEnabled(false);
        back_btn.setEnabled(false);
        runStop_btn.setEnabled(false);
        first_btn.setEnabled(false);

        addVertex_btn.addActionListener( actionEvent -> {

            if(currentOption == Option.CREATE){
                ChangeCurrentOption(Option.NONE);
                addVertex_btn.setBackground(Color.WHITE);
            }
            else {

                updateStatus( "[Redactor mode] Vertices Adding...");
                ChangeCurrentOption(Option.CREATE);
            }
            // leftPanel.setCurrentOption( currentOption );
        });
        addRib_btn.addActionListener( actionEvent -> {
            if(currentOption == Option.CONNECT){
                ChangeCurrentOption(Option.NONE);

            }
            else {
                updateStatus( "[Redactor mode] Ribs Adding...");
                ChangeCurrentOption(Option.CONNECT);
            }
            //  leftPanel.setCurrentOption( currentOption );
        });
        delete_btn.addActionListener( actionEvent -> {
            if(currentOption == Option.DELETE){
                ChangeCurrentOption(Option.NONE);

            } else {

                updateStatus( "[Redactor mode] Deleting...");
                ChangeCurrentOption(Option.DELETE);
            }
            // leftPanel.setCurrentOption( currentOption );
        } );

        clear_btn.addActionListener( actionEvent -> {
                ChangeCurrentOption(Option.NONE);
                //
                facade = new Facade();
                //
                leftPanel.clear();
```

```
                rightPanel.clear();
                updateStatus( "[Redactor mode] Clear...");
                s_count=counter = 0;

                revalidate();
                repaint();
        });

        back_btn.addActionListener( actionEvent -> {
           if (facade.getGraph() != null){
                facade.prev();
                rightPanel.removeRibs();
                facade.visualizeAlgorithm(rightPanel);
            }
            updateStatus( "[Algorithm mode] Prev...");
            ChangeCurrentOption(Option.PREV);
            ChangeCurrentOption(Option.NONE);
        });

        forward_btn.addActionListener( actionEvent -> {
             if (facade.getGraph() != null) {
                 //facade.setCommand(new
LoadGraphManuallyCommand(leftPanel.getRibList()));
                 //facade.createGraph();
                 //facade.initAlgorithm();
                 //facade.doAlgorithm();

                 facade.next();
                 rightPanel.removeRibs();
                 facade.visualizeAlgorithm(rightPanel);
             }
                updateStatus( "[Algorithm mode] Next...");

                ChangeCurrentOption(Option.NEXT);
                ChangeCurrentOption(Option.NONE);
        });

        runStop_btn.addActionListener( actionEvent -> {
            if( currentOption == Option.RUN || currentOption ==
Option.STOP  ){

                ChangeCurrentOption(Option.NONE);
            }else {
                updateStatus( "[Algorithm mode] Result...");
                ChangeCurrentOption( Option.RUN );

                //Added by Mikulik 09.07.2021
                if (facade.getGraph() != null) {
                    //facade.setCommand(new
LoadGraphManuallyCommand(leftPanel.getRibList()));
                    //facade.createGraph();

                    //facade.initAlgorithm();
                    //facade.doAlgorithm();
                    facade.last();
                    facade.visualizeAlgorithm(rightPanel);
```

```
                }
            }
        });

        first_btn.addActionListener(actionEvent -> {
            if (facade.getGraph() != null) {
                //facade.setCommand(new
LoadGraphManuallyCommand(leftPanel.getRibList()));
                //facade.createGraph();
                //facade.initAlgorithm();
                //facade.doAlgorithm();

                facade.first();
                rightPanel.removeRibs();
                facade.visualizeAlgorithm(rightPanel);
            }
            updateStatus( "[Algorithm mode] First step");
            ChangeCurrentOption(Option.FIRST);
            ChangeCurrentOption(Option.NONE);
        });

        redactor_btn.addActionListener( actionEvent -> {
            if (isRedactorModeOn){
                updateStatus( "[Algorithm mode]");
                addVertex_btn.setEnabled(false);
                addRib_btn.setEnabled(false);
                delete_btn.setEnabled(false);
                clear_btn.setEnabled(false);
                generate_btn.setEnabled(false);
                forward_btn.setEnabled(true);
                back_btn.setEnabled(true);
                runStop_btn.setEnabled(true);
                first_btn.setEnabled(true);

                facade.setCommand(new
LoadGraphManuallyCommand(leftPanel.getRibList()));
                facade.createGraph();
                facade.initAlgorithm();
                facade.doAlgorithm();
                isRedactorModeOn = false;
                redactor_btn.setBackground(Color.WHITE);
            } else {
                updateStatus( "[Redactor mode]");
                addVertex_btn.setEnabled(true);
                addRib_btn.setEnabled(true);
                delete_btn.setEnabled(true);
                clear_btn.setEnabled(true);
                generate_btn.setEnabled(true);
                forward_btn.setEnabled(false);
                back_btn.setEnabled(false);
                runStop_btn.setEnabled(false);
                first_btn.setEnabled(false);
                isRedactorModeOn = true;
                redactor_btn.setBackground(new Color( 250, 100, 100 ));
            }
            // leftPanel.setCurrentOption( currentOption );
```

```
        });


        generate_btn.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed( ActionEvent actionEvent ) {

                try {
                    GenerateGraphDialog gen = new GenerateGraphDialog();
                    gen.getButton().addActionListener( e -> {
                        int countVtx = gen.getAnswerArray()[0];
                        int countRibs = gen.getAnswerArray()[1];
                        int min = gen.getAnswerArray()[2];
                        int max = gen.getAnswerArray()[3];
                        System.out.println(countVtx);
                        GenerateCommand generateCommand = new
GenerateCommand(
                                countVtx,
                                Math.min(countRibs, countVtx * (countVtx
- 1)),
                                min,
                                max
                        );


                        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
                        generateCommand.setBorders((int)
screenSize.getHeight() / 5, (int) screenSize.getWidth() / 5);
                        Graph graph = generateCommand.execute();
                        graph.printVertices();
                        graph.printEdges();
                        setVerticesWhileLoading(graph.getVertices());
                        setNameCounter(graph.getVertices());
                        leftPanel.setRibs(graph.getEdges());
                        gen.dispose();
                    });

                } catch (ParseException e1) {
                    e1.printStackTrace();
                }

            }
        } );

        toolBar.add(redactor_btn);
        toolBar.add(generate_btn);
        toolBar.add(addVertex_btn);
        toolBar.add(addRib_btn);
        toolBar.add(delete_btn);
        toolBar.add(clear_btn);
        toolBar.add(back_btn);
        toolBar.add(runStop_btn);
        toolBar.add(first_btn);
        toolBar.add(forward_btn);
```

```
        return toolBar;
    }

}
```

Имя файла: AboutUs.java

```
package com.practice.Gui;

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

public class AboutUs extends JDialog {

    private void InitUI() {

        setMinimumSize(new Dimension(600, 400));
        String offset = "        ";
        JLabel label = new JLabel(offset+"some text");
        label.setVerticalAlignment(SwingConstants.CENTER);
        label.setHorizontalAlignment(SwingConstants.LEFT);
        label.setBackground(Color.lightGray);
        add(label);
    }

    public AboutUs() {
        InitUI();
        this.pack();
        this.setVisible(true);
    }
}
```

Имя файла: Board.java

```
package com.practice.Gui;

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

public class Board extends JComponent {

     private Dimension size;
     private String name;
     protected Cursor draggingCursor =
Cursor.getPredefinedCursor(Cursor.HAND_CURSOR);

     public Board(  String name, int x, int y) {
          super();
          this.name =name;
          size = new Dimension(30, 30);
          setBounds( x-10, y-10, 30, 30 );
          setBackground(Color.WHITE);
          setPreferredSize(size);
          Font f = new Font("Monospaced", Font.BOLD, 20);
          setFont(f);
```

```
            setCursor( draggingCursor );
      }

      public void setName(String str) {
            name = str;
            repaint();
      }

      @Override
      public void paintComponent(Graphics graphics) {
            super.paintComponent( graphics );
            Graphics2D graphics2d= ( Graphics2D ) graphics;

            FontMetrics fm = graphics2d.getFontMetrics();

            graphics2d.setColor( Color.BLACK );
            int lx = ( getWidth() - fm.stringWidth( name ) )/2;
            int ly = (fm.getAscent() + getHeight())/2 ;

            graphics2d.drawString( name, lx, ly );
      }

      @Override
      public Dimension getPreferredSize() {
            return size;
      }
}
```

Имя файла: Documentation.java

```
package com.practice.Gui;

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

public class Documentation  extends JDialog {

    private void InitUI() {

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.X_AXIS));

        setMinimumSize(new Dimension(600, 400));
        String offset = "        ";
        JLabel label = new JLabel(offset+"some Text");
        label.setVerticalAlignment(SwingConstants.CENTER);
        label.setHorizontalAlignment(SwingConstants.LEFT);
        label.setBackground(Color.lightGray);
        mainPanel.add(label);
        add(mainPanel);

    }
    public Documentation() {
        InitUI();
        this.pack();
        this.setVisible(true);
```

```
    }
}
```

Имя файла: GenerateGraphDialog.java

```
package com.practice.Gui;

import javax.swing.*;
import javax.swing.text.DefaultFormatterFactory;
import javax.swing.text.NumberFormatter;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.text.NumberFormat;
import java.text.ParseException;

public class GenerateGraphDialog extends JDialog {

    private JFormattedTextField countVertex;
    private JFormattedTextField countRibs;
    private JFormattedTextField minWeight;
    private JFormattedTextField maxWeight;
    private JButton sendButton = new JButton("Генерировать");
    private Font font = new Font("Tahoma", Font.BOLD, 16);



    public Integer[] getAnswerArray() {

        Integer[] results = {
                Integer.valueOf(countVertex.getText()),
Integer.valueOf(countRibs.getText()),
                Integer.valueOf(minWeight.getText()),
Integer.valueOf(maxWeight.getText())
            };
        return results;
    }
    public Integer getVerticesCount(){
        return Integer.valueOf(countVertex.getText());
    }

    public Integer getRibsCount(){
        return Integer.valueOf(countRibs.getText());
    }

    public Integer getMinWeight(){
        return Integer.valueOf(minWeight.getText());
    }

    public Integer getMaxWeight(){
        return Integer.valueOf(maxWeight.getText());
    }
```

```
    public JButton getButton() {
        return sendButton;
    }

    public GenerateGraphDialog() throws ParseException {
        int width = 208;
        int boxHeight = 24;
        int labelHeight = 20;
        int marging = 8;
        int Offset = 16;

        setResizable(false);
        setMinimumSize(new Dimension(2*Offset+width, 300));
        setMaximumSize(new Dimension(2*Offset+width, 300));
        setLayout(null);

        NumberFormat format = NumberFormat.getInstance();
        NumberFormat formatVtx = NumberFormat.getInstance();
        NumberFormatter formatter = new NumberFormatter(format);
        NumberFormatter formatterVtx = new NumberFormatter(formatVtx);
        formatter.setValueClass(Integer.class);
        formatter.setMinimum(1);
        formatterVtx.setMaximum( 26 );
        formatterVtx.setMinimum( 1 );
        formatterVtx.setAllowsInvalid( true );
        formatter.setAllowsInvalid(true);

        JLabel lblVertex = new JLabel("Количество вершин:");
        lblVertex.setFont(font);
        lblVertex.setBounds(Offset, Offset, width, labelHeight);
        add(lblVertex);

        formatter.setMaximum(26);
        countVertex = new JFormattedTextField( formatterVtx );
        countVertex.setBounds(Offset, marging + Offset + labelHeight,
width, boxHeight);
        add(countVertex);

        formatter.setMaximum(1000);


        JLabel lblRibs = new JLabel("Количество ребер:");
        lblRibs.setFont(font);
        lblRibs.setBounds(Offset, marging + 2*Offset +
labelHeight+boxHeight, width, labelHeight);
        add(lblRibs);


        NumberFormatter formatterRib = new NumberFormatter(format);
        formatterRib.setValueClass(Integer.class);
        formatterRib.setMinimum(1);


        countRibs = new JFormattedTextField(formatterRib);
```

```
        countRibs.setBounds(Offset,
2*marging+2*Offset+2*labelHeight+boxHeight, width, boxHeight);
        add(countRibs);

        JLabel lblWeight = new JLabel("Диапазон веса:");
        lblWeight.setFont(font);
        lblWeight.setBounds(Offset,  2*marging + 3*Offset
+2*labelHeight+2*boxHeight, width, labelHeight);
        add(lblWeight);

        JLabel lblMinWeight = new JLabel("Мин:");
        lblMinWeight.setFont(new Font("Tahoma", Font.BOLD, 12));
        lblMinWeight.setBounds(Offset,  3*marging +
3*Offset+3*labelHeight+2*boxHeight, 36, labelHeight);
        add(lblMinWeight);


        minWeight = new JFormattedTextField(formatter);
        minWeight.setBounds(Offset+42,  3*marging +
3*Offset+3*labelHeight+2*boxHeight, 48, boxHeight);
        add(minWeight);


        JLabel lblMaxWeight = new JLabel("Макс:");
        lblMaxWeight.setFont(new Font("Tahoma", Font.BOLD, 12));
        lblMaxWeight.setBounds(2*Offset+90, 3*marging +
3*Offset+3*labelHeight+2*boxHeight, 42, labelHeight);
        add(lblMaxWeight);


        maxWeight = new JFormattedTextField(formatter);
        maxWeight.setBounds(2*Offset+90+50, 3*marging +
3*Offset+3*labelHeight+2*boxHeight, 48, boxHeight);
        add(maxWeight);

        sendButton.setBounds(Offset, 3*marging+
4*Offset+3*labelHeight+3*boxHeight, 208, 26);
        add(sendButton);

        addMouseMotionListener( new MouseAdapter() {
                @Override
                public void mouseMoved( MouseEvent e ) {
                sendButton.setEnabled(
                    countVertex.getText().length() > 0  &&
                    countRibs.getText().length() > 0 &&
                    minWeight.getText().length() > 0 &&
                    maxWeight.getText().length() > 0
                );
                }
           } );

        setVisible(true);
    }
```

```
}
```

Имя файла: Rib.java

```
package com.practice.Gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Objects;

public class Rib {

	private Vertex sourceVertex, targetVertex;
	private Integer weight;
	private Board component;
	private Color[] colors = {
		Color.CYAN, Color.BLUE, Color.GREEN, Color.MAGENTA,
Color.ORANGE, Color.PINK, Color.RED, Color.YELLOW
	};

	private Color color = Color.CYAN;

	public Rib() {
		sourceVertex = targetVertex = null;
		component = new Board("0", 0, 0);

		component.addMouseListener(new MouseAdapter(){

			@Override
			public void mouseClicked(MouseEvent e) {

				if (e.getClickCount() == 2 &&
Main.isRedactorModeOn) {
					Integer weight = 0;
					String answer;
					String msg = "напишите вес ребра";

					int optionPane = JOptionPane.QUESTION_MESSAGE;

					for( int isNumber = 0; isNumber < 1;  ) {

						answer =
JOptionPane.showInputDialog(null, msg,
								"Вес ребра", optionPane);
						if(answer == null) {
							return;
						}
						try {
							weight = Integer.parseInt(answer);
							if( weight != 0 ) {
								isNumber = 1;
							}else {
								msg = "вес ребра должен быть
целым числом больше нуля";
```

```
                                        optionPane =
JOptionPane.WARNING_MESSAGE;
                                    }
                                }
                                catch (NumberFormatException err)
                                {
                                    msg = "вес ребра должно быть целое
натуральное число";
                                    optionPane =
JOptionPane.ERROR_MESSAGE;
                                    continue;
                                }
                            }
                            setWeight(weight);
                        }
                    }
            });

    }

    public Color getColor() {
        return color;
    }

    public void setColor(int index) {
        int i = index < colors.length ? index: index %  colors.length;
        this.color = colors[i];
    }

    public Board getComponent() {
        return component;
    }

    public void setComponent(Board component) {
        this.component = component;
    }

    public boolean isConnect(Vertex vertex) {

        return sourceVertex == vertex | targetVertex == vertex;
    }

    public Vertex getSourceVertex() {
        return sourceVertex;
    }

    public Vertex getTargetVertex() {
        return targetVertex;
    }

    public void setSourceVertex( Vertex sourceVertex ) {
        this.sourceVertex = sourceVertex;
    }

    public void setTargetVertex( Vertex targetVertex ) {
```

```
            this.targetVertex = targetVertex;
            int _x = (sourceVertex.getX() + targetVertex.getX())/2;
            int _y = (sourceVertex.getY() + targetVertex.getY())/2;
            component.setLocation( _x, _y );
      }

      public boolean isFull() {
            return sourceVertex != null && targetVertex != null;
      }

      public Integer getWeight() { return weight; }

      public void setWeight( Integer weight ) {

            this.weight = weight;
            component.setName(weight.toString());
      }

      public void setNode( Vertex node ) {
            if( sourceVertex == null ) {

                  sourceVertex = node;
            }
            else {
                  targetVertex = node;
                  int _x = (sourceVertex.getX() + targetVertex.getX())/2;
                  int _y = (sourceVertex.getY() + targetVertex.getY())/2;
                  component.setLocation( _x, _y );
            }
      }

      public Point getCenterPoint( Vertex node ) {
            int  x = node.getX()+node.getWidth()/2;
            int  y = node.getY()+node.getHeight()/2;
            return new Point(x, y);
      }

      public Point[] getLine() {
            component.setLocation(     (sourceVertex.getX() +
targetVertex.getX())/2,
                                                (sourceVertex.getY() +
targetVertex.getY())/2 );
            return new Point[]{ getCenterPoint( sourceVertex ),
getCenterPoint( targetVertex ) };
      }

      @Override
      public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            Rib rib = (Rib) o;
            return
(this.sourceVertex.getId().equals(rib.sourceVertex.getId()) &&
this.targetVertex.getId().equals(rib.targetVertex.getId()) &&
Objects.equals(weight, rib.weight)) |
```

```
      (this.sourceVertex.getId().equals(rib.targetVertex.getId()) &&
this.targetVertex.getId().equals(rib.sourceVertex.getId()) &&
Objects.equals(weight, rib.weight));
      }

      @Override
      public int hashCode() {
            return Objects.hash(sourceVertex.getId(),
targetVertex.getId(), weight);
      }
}
```

Имя файла: Scene.java

```
package com.practice.Gui;

import com.practice.Graph.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.HashMap;

public class Scene extends JPanel {

      private Rib rib;
      private Point from, to;
      private ArrayList<Rib> ribs;
      private HashMap<String, Vertex> verticesDict;

      public ArrayList<Rib> getRibList(){
            return ribs;
      }

      public void addRib( Rib rib ) {
            ribs.add(rib);
            add(rib.getComponent());
            repaint();
      }

      Scene() {
            setOpaque( false );
         setBackground(new Color(240, 240, 240));
            setLayout( null );
            Font f = new Font("Monospaced", Font.BOLD, 20);
            setFont(f);
            ribs = new ArrayList<>();
            verticesDict = new HashMap<>();
            addMouseMotionListener( new MouseAdapter() {
                  @Override
                  public void mouseMoved( MouseEvent e ) {
                        super.mouseMoved( e );
                        to = e.getPoint();
                        validate();
```

```
                    repaint();
                }
            } );
      }


    public void addVertex( Vertex vertex ){

          vertex.addMouseListener( new MouseAdapter() {
               @Override
               public void mouseClicked( MouseEvent e ) {
                    super.mouseClicked( e );
                    if(Main.currentOption == Main.Option.CONNECT) {

                         if( rib == null ) {
                              to = from = vertex.getCenterPoint();
                              rib = new Rib();
                              rib.setSourceVertex( vertex );
                              vertex.setColour( Color.cyan );
                              validate();
                              repaint();

                         } else if( rib.isFull() == false &&
rib.getSourceVertex() != vertex ) {

                              Integer weight = 0;
                              String answer;
                              String msg = "напишите вес ребра";

                              int optionPane =
JOptionPane.QUESTION_MESSAGE;

                              for( int isNumber = 0; isNumber < 1;  )
{

                                   answer =
JOptionPane.showInputDialog(null, msg,
                                             "Вес ребра",
optionPane);
                                   if(answer == null) {


     rib.getSourceVertex().setColour( Color.lightGray );
                                        rib = null;
                                        validate();
                                        repaint();
                                        return;
                                   }
                                   try {
                                        weight =
Integer.parseInt(answer);
                                        if( weight != 0 ) {
                                             isNumber = 1;
                                        }else {
                                             msg = "вес ребра должен
быть целым числом больше нуля";
```

```
                                        optionPane =
JOptionPane.WARNING_MESSAGE;
                                    }
                                }
                                catch (NumberFormatException err)
                                {
                                    msg = "вес ребра должно быть
целое натуральное число";
                                    optionPane =
JOptionPane.ERROR_MESSAGE;
                                    continue;
                                }
                            }

                            to = vertex.getCenterPoint();
                            rib.setTargetVertex( vertex );
                            rib.getSourceVertex().setColour(
Color.lightGray );
                            rib.setWeight( weight );

                            JComponent Jc = rib.getComponent();
                            add( Jc );

                            rib.getSourceVertex().addMouseListener(
new MouseAdapter() {
                                @Override
                                public void mouseClicked(
MouseEvent e ) {
                                    super.mouseClicked( e );
                                    if(Main.currentOption ==
Main.Option.DELETE) {
                                        remove( Jc );
                                    }
                                }
                            } );

                            vertex.addMouseListener( new
MouseAdapter() {
                                @Override
                                public void mouseClicked(
MouseEvent e ) {
                                    super.mouseClicked( e );
                                    if(Main.currentOption ==
Main.Option.DELETE) {
                                        remove( Jc );
                                    }
                                }
                            } );

                            Jc.addMouseListener(new MouseAdapter(){

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

```
                                                  if(Main.currentOption ==
Main.Option.DELETE) {
                                                        for(int i = 0; i <
ribs.size(); i++) {
                                                              if ((ribs.get(i) !=
null && rib != null) && ribs.get(i).isConnect( vertex )&&
ribs.get(i).isConnect( rib.getSourceVertex() ) ){

      ribs.remove(i);
                                                                    break;
                                                              }
                                                              if(
/*ribs.get(i).equals(rib)*/ ribs.get(i).isConnect( vertex ) /*&&
ribs.get(i).isConnect( rib.getTargetVertex() )*/) {

      ribs.remove(i);
                                                                    break;
                                                              }
                                                        }
                                                        remove(Jc);
                                                        repaint();

                                                  }
                                            }
                                       });

                                       ribs.add( rib );
                                       rib = null;
                                       validate();
                                       repaint();
                                  }
                             }
                             else if( Main.currentOption == Main.Option.DELETE )
{


                                  for(int i = ribs.size()-1; i >= 0; i--) {
                                       if( ribs.get(i).isConnect( vertex )&&
ribs.get(i).isConnect( rib.getSourceVertex() ) ) {
                                            ribs.remove(i);
                                       }
                                  }
                                  remove(vertex);
                                  revalidate();
                                  repaint();
                             }
                        }
                        @Override
                        public void mousePressed( MouseEvent e ) {
                             super.mousePressed( e );
                        }
                   } );
                   verticesDict.put(vertex.getId(), vertex);
                   add(vertex);
              validate();
              repaint();
```

```
    }

     public void clear() {
          removeAll();
          ribs.clear();
          revalidate();
          repaint();
     }

     @Override
     public Dimension getPreferredSize() {
          return new Dimension( getParent().getWidth()/2,
getParent().getHeight());
     };

     @Override
     public void paintComponent(Graphics graphics) {
          Graphics2D graphics2D = (Graphics2D ) graphics;

          graphics2D.setStroke( new BasicStroke( 3 ) );
          graphics2D.setColor( Color.cyan );

          if(Main.currentOption == Main.Option.CONNECT && rib != null) {
               graphics2D.drawLine( from.x, from.y, to.x, to.y );
          }
          for( Rib rib: ribs ) {

               graphics2D.setColor( rib.getColor() );
               graphics2D.drawLine(
rib.getLine()[0].x,rib.getLine()[0].y,
rib.getLine()[1].x,rib.getLine()[1].y );
          }

     }

     public HashMap<String, Vertex> getVerticesDict(){
          return verticesDict;
     }
     public void setRibs(ArrayList<Edge> edges){
          this.removeRibs();

          for (Edge edge : edges){
               Rib rib = new Rib();
               rib.setWeight(edge.getWeight());

     rib.setSourceVertex(verticesDict.get(edge.getStartName()));

     System.out.println(verticesDict.get(edge.getStartName()).getId());

     rib.setTargetVertex(verticesDict.get(edge.getEndName()));

     System.out.println(verticesDict.get(edge.getEndName()).getId());
               //rib.setComponent(new
Board(String.valueOf(edge.getWeight()), 0, 0));
               //rib.setWeight(edge.getWeight());
               rib.setColor( edge.getStart().getComponent());
```

```
                this.addRib(rib);
            }
        }



        public void removeRibs(){
            for(int i = ribs.size()-1; i >= 0; i--) {
                Board Jc = ribs.get(i).getComponent();
                remove(Jc);
                repaint();
            }
            ribs.clear();
            revalidate();
            repaint();
        }
}
```

Имя файла: Vertex.java

```
package com.practice.Gui;

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

public class Vertex extends JComponent {

      private Dimension size;
      public Color colour;
      private int x, y;
      private final int radius = 25;
      private String id;
      private volatile int draggedAtX, draggedAtY;

      protected Cursor draggingCursor =
Cursor.getPredefinedCursor(Cursor.HAND_CURSOR);

      public Vertex(  String _id, int _x, int _y, Color _colour ) {
            super();
            id = _id; x = _x; y = _y; colour = _colour;
            size = new Dimension(2*radius, 2*radius);
            setBounds( x-radius, y-radius, 2*radius, 2*radius );
            setPreferredSize(size);
            Font f = new Font("Monospaced", Font.BOLD, radius);
            setFont(f);
            setCursor( draggingCursor );

            addMouseListener( new MouseAdapter() {
                  @Override
                  public void mousePressed( MouseEvent mouseEvent ) {
                        super.mousePressed( mouseEvent );

                        draggedAtX = mouseEvent.getX();
                        draggedAtY = mouseEvent.getY();
```

```
                }
            });


            addMouseMotionListener( new MouseAdapter() {

                @Override
                public void mouseDragged( MouseEvent mouseEvent ) {

                    super.mouseDragged( mouseEvent );
                    Point newLocation = new Point(mouseEvent.getX() -
draggedAtX + getLocation().x, mouseEvent.getY() - draggedAtY +
getLocation().y);
                    setLocation(newLocation);
                }
            } );

      }

      public String getId() {
            return id;
      }

      public void setName(String answer) {
            id = answer;
            repaint();
      }

      public Point getCenterPoint( ) {
            int  x = getX()+getWidth()/2;
            int  y = getY()+getHeight()/2;
            return new Point(x, y);
      }

      public void setColour( Color colour ) {
            this.colour = colour;
            repaint();
      }

      public Color getColour() {
            return colour;
      }

      @Override
      public void paintComponent(Graphics graphics) {
            super.paintComponent( graphics );
            Graphics2D graphics2d= ( Graphics2D ) graphics;

            graphics2d.setColor( colour );
            graphics2d.fillOval( 2, 2, 2 * radius-3, 2 * radius-3);

            graphics2d.setColor(new Color( 0, 159, 153 ));
            FontMetrics fm = graphics2d.getFontMetrics();
            int x = ( getWidth() - fm.stringWidth( id ) ) / 2+2;
            int y = (fm.getAscent() + getHeight())/2-1 ;
            graphics2d.drawString( id, x, y );
```

```
      }

      @Override
      public Dimension getPreferredSize() {
            return size;
      }
}
```

Имя файла: Edge.java

```
package com.practice.Graph;

import java.util.Objects;

public class Edge implements Cloneable{

    private Node start;
    private Node end;
    private int weight;

    public Edge(Node start, Node end, int weight) {
        this.start = start;
        this.end = end;
        this.weight = weight;
    }

    public int getWeight() {
        return weight;
    }

    public void setStartName(String start) {
        this.start.setName(start);
    }

    public String getStartName() {
        return start.getName();
    }

    public Node getStart() {
        return start;
    }

    public Node getEnd() {
        return end;
    }

    public void setEndName(String end) {
        this.end.setName(end);
    }

    public String getEndName() {
        return end.getName();
    }

    @Override
    public String toString() {
```

```
        return getStartName() + "--" + getEndName() + " : " + weight + "
| component - " + start.getComponent() +
                " " + end.getComponent();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null) {
            return false;
        }
        if (o instanceof Edge) {
            if (((start.getName().equals(((Edge) o).getStartName()) &&
                    end.getName().equals(((Edge) o).getEndName())) ||
                    (start.getName().equals(((Edge) o).getEndName()) &&
                            end.getName().equals(((Edge)
o).getStartName()))) &&
                    ((Edge)o).getWeight() == weight) {
                return true;
            }
        }
        return false;
    }

    /*@Override
    public int hashCode() {
        return start.hashCode() + end.hashCode();
    }*/

    @Override
    public int hashCode() {
        return start.hashCode() + end.hashCode() + weight;
    }

    @Override
    public Object clone(){
        Edge edge = new Edge((Node)this.getStart().clone(),
(Node)this.getEnd().clone(), weight);
        return edge;
    }
}
```

Имя файла: Node.java

```
package com.practice.Graph;

public class Node implements Cloneable{

    private String name;
    private int component;
    private int x = 0;
    private int y = 0;

    public Node(String name) {
```

```
        this.name = name;
    }

    public Node(String name, int x, int y) {
        this.name = name;
        this.x = x;
        this.y = y;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setComponent(int newComp) {
        component = newComp;
    }

    public int getComponent() {
        return component;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getX() {
        return x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getY() {
        return y;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null) {
            return false;
        }
        if (o instanceof Node) {
            if (this.getName().equals(((Node) o).getName())) {
                return true;
            }
        }
        return false;
    }
```

```
    @Override
    public int hashCode() {
        return name.hashCode();
    }

    @Override
    public Object clone(){
        Node node = new Node(this.getName(), this.getX(), this.getY());
        node.setComponent(this.getComponent());
        return node;
    }

}
```

Имя файла: Graph.java

```
package com.practice.Graph;

import java.util.ArrayList;
import java.util.Iterator;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Graph implements Cloneable{
    private ArrayList<Node> vertices;
    private ArrayList<Edge> edges;
    private int countVertices;
    private int countEdges;
    private static final Logger logger =
LogManager.getLogger(Graph.class);

    public Graph() {
        vertices = new ArrayList<>();
        edges = new ArrayList<>();
        countEdges = 0;
        countVertices = 0;
    }

    public Graph(ArrayList<Node> vertices, ArrayList<Edge> edges) {
        this.vertices = vertices;
        this.edges = edges;
        countEdges = edges.size();
        countVertices = vertices.size();
    }

    public int findVertex(String u) {
        if (u == null) {
            logger.fatal("Нельзя найти пустую вершину");
            throw new IllegalArgumentException("Нельзя найти пустую
вершину");
        }
        for (Node nodes : vertices) {
            if (nodes.getName().equals(u)) {
                return vertices.indexOf(nodes);
            }
```

```
        }
        return -1;
    }

    public void addVertex(String v) {
        if (v == null) {
            logger.fatal("Нельзя добавить пустую вершину");
            throw new IllegalArgumentException("Нельзя добавить пустую
вершину");
        }
        if (findVertex(v) == -1) {
            vertices.add(new Node(v));
            countVertices++;
        }
        else {
            logger.warn("Вершина с таким именем уже существует");
        }
    }

    public void addEdge(String u, String v, int w) {
        if (u == null || v == null || w <= 0) {
            logger.fatal("Нельзя добавить ребро с пустой вершиной или
весом <= 0");
            throw new IllegalArgumentException("Нельзя добавить ребро с
пустой вершиной или весом <= 0");
        }
        if (u.equals(v)) {
            logger.warn("Начальная и конечная вершины одинаковые");
            return;
        }
        for (Edge edge: edges) {
            if ((edge.getStartName().equals(u) &&
edge.getEndName().equals(v)) ||
                    (edge.getStartName().equals(v) &&
edge.getEndName().equals(u))){
                logger.warn("Такое ребро уже существует");
                return;
            }
        }
        int indexStart = findVertex(u);
        int indexEnd = findVertex(v);
        Node start;
        Node end;
        if (indexStart != -1) {
            start = vertices.get(indexStart);
        }
        else {
            start = new Node(u);
            vertices.add(start);
            countVertices++;
        }
        if (indexEnd != -1) {
            end = vertices.get(indexEnd);
        }
        else {
            end = new Node(v);
```

```
            vertices.add(end);
            countVertices++;
        }
        Edge newEdge = new Edge(start, end, w);
        edges.add(newEdge);
        countEdges++;
    }

    public void removeVertex(String v) {
        if (v == null) {
            logger.fatal("Нельзя удалить пустую вершину");
            throw new IllegalArgumentException("Нельзя удалить пустую
вершину");
        }
        int nodeIndex = findVertex(v);
        if (nodeIndex != -1) {
            vertices.remove(nodeIndex);
            countVertices--;
            Iterator<Edge> edgeIterator = edges.iterator();
            while (edgeIterator.hasNext()) {
                Edge edgeNext = edgeIterator.next();
                if (edgeNext.getStartName().equals(v) ||
edgeNext.getEndName().equals(v)) {
                    edgeIterator.remove();
                    countEdges--;
                }
            }
        }
        else {
            logger.warn("Данной вершины нет в графе");
        }
    }

    public void removeEdge(String start, String end, int w) {
        if (start == null || end == null) {
            logger.fatal("Нельзя удалить ребро с пустой вершиной");
            throw new IllegalArgumentException("Нельзя удалить ребро с
пустой вершиной");
        }
        if (start.equals(end)) {
            logger.warn("Начальная и конечная вершины одинаковые");
            return;
        }
        Edge remEdge = new Edge(new Node(start), new Node(end), w);
        if (edges.remove(remEdge)) {
            countEdges--;
        }
    }

    public ArrayList<Node> getVertices() {
        return vertices;
    }

    public ArrayList<Edge> getEdges() {
        return edges;
    }
```

```
    public int getCountVertices() {
        return countVertices;
    }

    public int getCountEdges() {
        return countEdges;
    }

    public void printVertices() {
        for (Node vertex: vertices) {
            System.out.println(vertex.getName() + " " +
vertex.getComponent() + vertex.getX() + "." + vertex.getY());
        }
    }

    public void printEdges() {
        for (Edge edge: edges) {
            System.out.println(edge.toString());
        }
    }

    public Object clone()
    {
        Graph graph = new Graph((ArrayList<Node>)
this.getVertices().clone(), (ArrayList<Edge>) this.getEdges().clone());
        return graph;
    }

}
```

Имя файла: Algorithm.java

```
package com.practice.Utilts;

import java.util.ArrayList;

public interface Algorithm {
    //void doAlgorithm();
    void algorithmStep();
    void printRes();
}
```

Имя файла: BoruvkaAlg.java

```
package com.practice.Utilts;

import com.practice.Graph.Edge;
import com.practice.Graph.Graph;
import com.practice.Graph.Node;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.ArrayList;
import java.util.Arrays;
```

```
public class BoruvkaAlg implements Algorithm, Cloneable {

    private Graph graph;
    private ArrayList<Edge> mst;
    private int countVertices;
    private int countEdges;
    private int countTree;
    private ArrayList<Node> vertices;
    private ArrayList<Edge> edges;
    private Edge[] minEdges;

    public class AlgorithmMemento {
        private Graph graphStorage;
        private ArrayList<Edge> mstStorage;
        private int countTreeStorage;

        public AlgorithmMemento(){
            this.graphStorage = (Graph) graph.clone();
            ArrayList<Edge> cloneMst = new ArrayList<>(mst.size());
            for (Edge edge : mst){
                cloneMst.add((Edge)edge.clone());
            }
            this.mstStorage = cloneMst;
            this.countTreeStorage = countTree;
        }

    }

    public BoruvkaAlg(Graph graph) {
        mst = new ArrayList<>();
        this.graph = graph;
        init();
    }

    private void init() {
        countVertices = graph.getCountVertices();
        countEdges = graph.getCountEdges();
        vertices = graph.getVertices();
        edges = graph.getEdges();
        minEdges = new Edge[countVertices];
        countTree = countVertices;
        int n = 0;
        for (Node vertex: vertices) {
            vertex.setComponent(n);
            n++;
        }
    }

    @Override
    public void algorithmStep() {
        Arrays.fill(minEdges, null);
        for (Edge edge: edges) {
            int comp1 = edge.getStart().getComponent();
            int comp2 = edge.getEnd().getComponent();
            if (comp1 != comp2) {
```

```
                if (minEdges[comp1] == null ||
edges.get(edges.indexOf(minEdges[comp1])).getWeight() > edge.getWeight())
{
                    minEdges[comp1] = edge;
                }
                if (minEdges[comp2] == null ||
edges.get(edges.indexOf(minEdges[comp2])).getWeight() > edge.getWeight())
{
                    minEdges[comp2] = edge;
                }
            }
        }

        for (Edge minEdge: minEdges) {
            if (minEdge != null) {
                int comp1 = minEdge.getStart().getComponent();
                int comp2 = minEdge.getEnd().getComponent();
                if (comp1 != comp2) {
                    mst.add(minEdge);
                    changeComponent(comp1, comp2, minEdge);
                    countTree--;
                }
            }
        }
    }

    /*private void changeComponent(int compStart, int compEnd, Edge edge)
{
        if (compStart > compEnd) {
            edge.getStart().setComponent(compEnd);
        }
        else {
            edge.getEnd().setComponent(compStart);
        }
    }*/

    private void changeComponent(int compStart, int compEnd, Edge edge) {
        int newComp, oldComp;
        String nextVertex;
        ArrayList<Node> changedVertex = new ArrayList<>();

        if (compStart > compEnd) {
            newComp = compEnd;
            oldComp = compStart;
            nextVertex = edge.getStartName();
            changedVertex.add(edge.getStart());
        }
        else {
            newComp = compStart;
            oldComp = compEnd;
            nextVertex = edge.getEndName();
            changedVertex.add(edge.getEnd());
        }

        for (Edge e: edges) {
```

```
            if ((e.getStartName().equals(nextVertex) && oldComp ==
e.getEnd().getComponent())) {
                changedVertex.add(e.getEnd());
                nextVertex = e.getEndName();
            }
            if ((e.getEndName().equals(nextVertex) && oldComp ==
e.getStart().getComponent())) {
                changedVertex.add(e.getStart());
                nextVertex = e.getStartName();
            }
        }

        for (Node v: changedVertex) {
            v.setComponent(newComp);
        }
    }

    public void doAlgorithm() {
        while (countTree > 1) {
            algorithmStep();
            /*printRes();
            System.out.println("-------------------------");*/
        }
    }

    @Override
    public void printRes() {
        for (Edge edge: mst) {
            System.out.println(edge.toString());
        }
    }

    public ArrayList<Edge> getMst() {
        return mst;
    }

    public int getCountTree(){
        return this.countTree;
    }

    public AlgorithmMemento save(){
        return new AlgorithmMemento();
    }

    public void restore(AlgorithmMemento snap) throws
NullPointerException{
        if (snap.graphStorage == null) {
            Logger logger = LogManager.getLogger(BoruvkaAlg.class);
            logger.fatal("Невозможно восстановить состояние алгоритма");
            throw new NullPointerException();
        }
        this.graph = snap.graphStorage;
        this.mst = snap.mstStorage;
        countVertices = graph.getCountVertices();
        countEdges = graph.getCountEdges();
        vertices = graph.getVertices();
```

```
        edges = graph.getEdges();
        countTree = snap.countTreeStorage;
    }

}
```

Имя файла: CareTaker.java

```
package com.practice.Utilts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.ArrayList;

public class CareTaker {
    private BoruvkaAlg originator;
    private ArrayList<BoruvkaAlg.AlgorithmMemento> history;
    //private int cur = -1;
    //private int startIndex = 0;
    private int prevIndex = -2;
    private int nextIndex = 1;

    public CareTaker(BoruvkaAlg algorithm){
        originator = algorithm;
        history = new ArrayList<>();
    }

    public void backup(){
        history.add(originator.save());
        //cur++;
        prevIndex++;
    }

    public void stepNextMemento(){
        if (nextIndex >= 0 & nextIndex < history.size() && history.size()
> 0){
            BoruvkaAlg.AlgorithmMemento snap = history.get(nextIndex);
            //
            prevIndex = nextIndex - 1;
            //
            nextIndex++;
            try {
                originator.restore(snap);
            } catch (Exception e){
                Logger logger = LogManager.getLogger(CareTaker.class);
                logger.error("Restore went wrong");
            }
        }
    }

    public void undo(){
        if (prevIndex >= 0 & prevIndex < history.size() & history.size()
> 0) {
            BoruvkaAlg.AlgorithmMemento snap = history.get(prevIndex);
            nextIndex = prevIndex + 1;
```

```
            prevIndex--;
            try {
                originator.restore(snap);
            } catch (Exception e){
                Logger logger = LogManager.getLogger(CareTaker.class);
                logger.error("Restore went wrong");
            }
        }
    }

    public void stepLastMemento() {
        BoruvkaAlg.AlgorithmMemento snap = history.get(history.size() -
1);
        nextIndex = history.size();
        prevIndex = history.size() - 2;
        try {
            originator.restore(snap);
        } catch (Exception e){
            Logger logger = LogManager.getLogger(CareTaker.class);
            logger.error("Restore went wrong");
        }
    }

    public void stepFirstMemento() {
        BoruvkaAlg.AlgorithmMemento snap = history.get(0);
        nextIndex = 1;
        prevIndex = -1;
        try {
            originator.restore(snap);
        } catch (Exception e){
            Logger logger = LogManager.getLogger(CareTaker.class);
            logger.error("Restore went wrong");
        }
    }
}
```

Имя файла: Command.java

```
package com.practice.Utilts;

import com.practice.Graph.Graph;
import com.practice.Gui.Rib;

import java.util.ArrayList;

public interface Command {
    Graph execute();
    /*Graph execute(ArrayList<Rib> ribs);
    Graph execute(String filename);*/
}
```

Имя файла: Façade.java

```
package com.practice.Utilts;

import com.google.gson.Gson;
```

```
import com.google.gson.GsonBuilder;
import com.practice.Graph.Edge;
import com.practice.Graph.Graph;
import com.practice.Gui.Rib;
import com.practice.Gui.Scene;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.io.IOException;
import java.nio.file.Path;
import java.util.ArrayList;
import java.nio.file.Files;
import java.util.Collections;


public class Facade {
    private Graph graph = null;
    private BoruvkaAlg algorithm;
    private CareTaker careTaker;
    private Command command;
    private static final Logger logger =
LogManager.getLogger(Facade.class);

    public Facade(){

    }
    /*public Facade(Algorithm algorithm){
        this.algorithm = (BoruvkaAlg) algorithm;
        this.careTaker = new CareTaker(this.algorithm);
    }*/

    public void initAlgorithm() throws NullPointerException{
        //создание алгоритма по существующему графу внутри фасада
        if(graph != null) {
            this.algorithm = new BoruvkaAlg(graph);
            this.careTaker = new CareTaker(algorithm);
        } else {
            logger.fatal("Невозможно инициализировать алгоритм, передан
пустой граф");
            throw new NullPointerException();
        }
    }

    public void setCommand(Command command){
        this.command = command;
    }

    public void doAlgorithm() {
        while( algorithm.getCountTree() > 1 ){
            careTaker.backup();
            algorithm.algorithmStep();
            System.out.println("Mst:");
            algorithm.printRes();
        }
        careTaker.backup();
    }
```

```
    public void debug(){
        System.out.println("Debug Mst:");
        careTaker.undo();
        algorithm.printRes();
    }

    /*public void loadGraphFromFile(String filename){
        this.graph = command.execute(filename);

    }*/

    public void createGraph(){
        this.graph = command.execute();
        graph.printEdges();
        graph.printVertices();
        //здесь вызывается одна из команд создания графа
    }

    public ArrayList<Edge> getMst() {
        return algorithm.getMst();
    }

    public void prev(){
        careTaker.undo();
    }

    public void next() {
        careTaker.stepNextMemento();
    }

    public void first() {
        careTaker.stepFirstMemento();
    }

    public void last() {
        careTaker.stepLastMemento();
    }

    //Debug
    public void setGraph(Graph graph){
        this.graph = graph;
    }

    public void saveGraph(String filename){
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        String graphJson = gson.toJson(this.graph);
        //System.out.println(graphJson);
        try{
            if (!Files.exists(Path.of(filename))) {
                Path saveFile = Files.createFile(Path.of(filename));
                Files.write(saveFile, Collections.singleton(graphJson));
            } else {
                Files.write(Path.of(filename),
Collections.singleton(graphJson));
            }
```

```
        } catch (IOException e){

        }
    }

    public void visualizeAlgorithm(Scene scene){
        scene.setRibs(algorithm.getMst());
    }

    public Graph getGraph(){
        return graph;
    }

}
```

Имя файла: GenerateCommand.java

```
package com.practice.Utilts;
import java.awt.Point;
import java.util.Random;

import com.practice.Graph.Graph;
import com.practice.Graph.Node;
public class GenerateCommand implements Command{
    //Класс отвечает за генерацию случайного графа
    private int countVertices;
    private int countEdges;
    private int min;
    private int max;
    private int H = 500;
    private int W = 500;

    private final String[] alphabet = new String[]
            {
                    "A", "B", "C", "D", "E", "F", "G",
                    "H", "I", "J", "K", "L", "M", "N",
                    "O", "P", "Q", "R", "S", "T", "U",
                    "V", "W", "X", "Y", "Z"
            };

    public GenerateCommand(int countVertices, int countEdges, int min,
int max){
        this.countVertices = countVertices;
        this.countEdges = countEdges;
        this.min = min;
        this.max = max;
    }

    @Override
    public Graph execute(){
        Graph graph = new Graph();

        for (int i = 0; i < countVertices; i++){
            graph.addVertex(alphabet[i]);
        }
```

```
        for (int i = 0; i < graph.getCountVertices(); i++){
             Rand rand = new Rand();
            Node node = graph.getVertices().get(i);
            Point pos = rand.genPoint();
            node.setX(pos.x);
            node.setY(pos.y);


        }
        for (int i = 0; i < countEdges; i++){

            while (graph.getCountEdges() != (i + 1)){
                Rand rand = new Rand();
                int startInd = rand.randInt(0, countVertices-1);
                int endInd = rand.randInt(0, countVertices-1);
                int weight = rand.randInt(min, max);
                graph.addEdge(alphabet[startInd], alphabet[endInd],
weight);
            }
        }
        return graph;
    }


    public void setBorders(int H, int W){
        this.H = H;
        this.W = W;
    }

    class Rand {
        int[][] massive;
        private int row = 15, col = 15;
        Rand() {
            massive = new int[row][col];
            for(int j=0;j< massive.length; j++){
                massive[j] = new int[col];
                for(int l = 0; l < massive.length; l++){
                    massive[j][l] = 0;
                }
            }
        }

        public int randInt(int min, int max){
            return (int)Math.floor(min + Math.random()*( Math.abs( max-
min)));
        }

        public Point genPoint(){

            int x = randInt( 0, col ), y = randInt( 0, col );
            boolean flag = true;

            while(flag) {
                x = randInt( 1, col );
                y = randInt( 1, col );
                if( massive[x][y] != 0 ) {
```

```
                    continue;
                }
                massive[x][y] = 1;
                flag = false;
            }
            return  new Point( x*40, y*40 );
        }
    }
}
```

Имя файла: LoadCommand.java

```
package com.practice.Utilts;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.practice.Graph.Edge;
import com.practice.Graph.Graph;
import com.practice.Graph.Node;
import com.practice.Gui.Rib;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;

public class LoadCommand implements Command{
    //Класс отвечает за загрузку графа из файла

    private String filename;

    public LoadCommand(String filename){
        this.filename = filename;
    }

    /*@Override
    public void execute(){
        return;
    }

    @Override
    public Graph execute(ArrayList<Rib> ribs){
        return new Graph();
    }*/

    @Override
    public Graph execute() {
        Graph graph = null;
        Graph cloneGraph = new Graph();
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        try {
            String graphJson = Files.readString(Path.of(filename));
            graph = gson.fromJson(graphJson, Graph.class);

            ArrayList<Node> nodes = graph.getVertices();
            ArrayList<Edge> edges = graph.getEdges();
```

```
            for (Edge edge : edges){
                cloneGraph.addEdge(edge.getStartName(),
edge.getEndName(), edge.getWeight());
                Edge lastEdge =
cloneGraph.getEdges().get(cloneGraph.getCountEdges() - 1);
                Node start = lastEdge.getStart();
                Node end = lastEdge.getEnd();
                start.setX(edge.getStart().getX());
                start.setY(edge.getStart().getY());
                end.setX(edge.getEnd().getX());
                end.setY(edge.getEnd().getY());
            }
        } catch (IOException e){

        }
        return cloneGraph;
    }
}
```

Имя файла: LoadGraphManuallyCommand.java

```
package com.practice.Utilts;
import com.practice.Graph.Edge;
import com.practice.Graph.Graph;
import com.practice.Graph.Node;
import com.practice.Gui.*;

import java.util.ArrayList;

public class LoadGraphManuallyCommand implements Command{

    private ArrayList<Rib> ribs;

    public LoadGraphManuallyCommand(ArrayList<Rib> ribs){
        this.ribs = ribs;
    }
    @Override
    public Graph execute(){
        Graph graph = new Graph();
        for (Rib rib : ribs){
            graph.addEdge(rib.getSourceVertex().getId(),
rib.getTargetVertex().getId(), rib.getWeight().intValue());
            // experimental
            Edge edge = graph.getEdges().get(graph.getCountEdges() - 1);
            Node start = edge.getStart();
            Node end = edge.getEnd();
            start.setX(rib.getSourceVertex().getX());
            start.setY(rib.getSourceVertex().getY());
            end.setX(rib.getTargetVertex().getX());
            end.setY(rib.getTargetVertex().getY());
            //
        }
        return graph;
    }
}
```