

Object Oriented Programming

Programming report

Final Assignment: Graph Editor

Corradini Matteo Berke Atac
S3051390 S3075168

June 15, 2016

1 Problem description

The problem was to create a graphical editor program with which the user would be able to create diagrams. The diagram consists of vertex which vary in their position, shape and names; and the edges in between them where all the elements can be added or removed by the user. We had to implement an user interface, with the basic features to create a diagram (i.e., add and remove vertex, create edge etc.), and furthermore some extra features (i.e., undo-redo operations, saving the diagram etc.).

2 Problem analysis

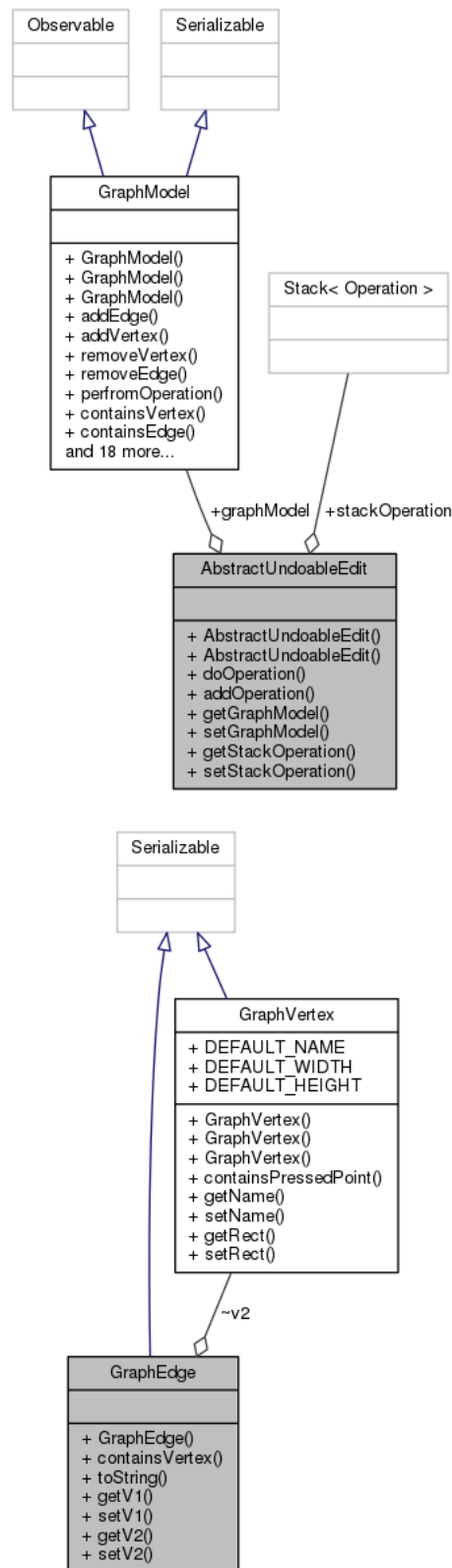
The structure of `GraphModel` was clear for the beginning: a collections of `GraphVertex` and another one of `GraphEdge`, and methods to provide these information with different requests.

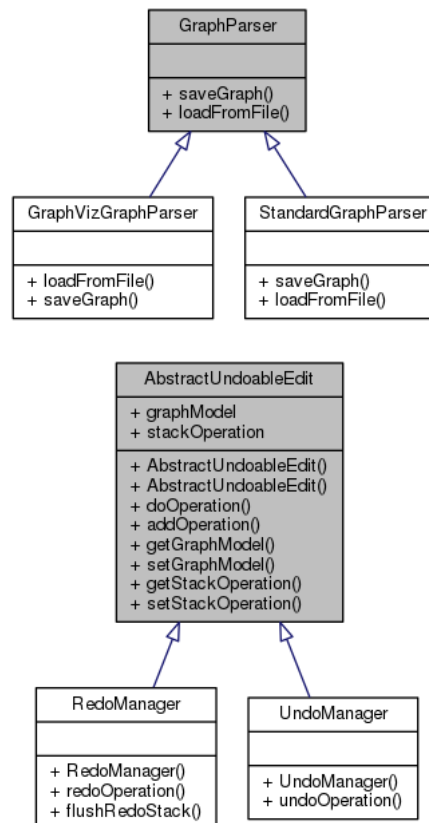
More difficult was to implement an `UndoManager` and a `RedoManager`. We can think about the operation as group of information involved in that operation, that has to be always available in order to insert or remove from the graph, and the type of operation done (in these case we have only 4 types of operation: addition or removal of a vertex of an edge). The manager of these information has to take it from a stack, which someone else provides to fill, and put the opposite operation in the stack of the other manager. The stack helps us to keep these information in the correct order. `RedoManager` needs also to be flush, because it would work only after an undo is done. Thus, `GraphModel` offers a high level, that receives an operation object, and computes it at low level managing the own data structure.

Another problem was to read and to write the graph in a file. We think a general graph parser, and the common features that offers. In this way, we are able to write general code that is working with all the parser that we are implementing. Furthermore, the program has to be able to start with a file already in input or with a blank project.

In order to debug the program and to have an overview of the data structure every time there is an update, we choose to use the observer pattern, where `GraphModel` is the subject and the interface that has to paint the graph is the observer.

3 Program design





GraphModel.java class holds the information about the vertices and edges in the diagram. The methods for the addition and removal of these elements are included in this class, manipulating the arrays they are contained in and providing information to different type of request. It keeps also 2 different object for the operation of undo and redo, which are filled after each operation from the user, and a field to save which vertex is selected in the program. These last information are transient, because they are not structural information of the graph.

GraphVertex.java class is the template for creating vertex objects. It contains the data of the vertex, in the way of storing the name of the vertex and the rectangle object that it has which is later drawn on the panel. The name of the vertex is an ID, it has to be unique in the program.

GraphEdge.java class is the template for creating edge objects. It contains the the data of the edge, which is the two vertices that are connected by the edge.

GraphParser.java is an abstract class containing the abstract methods for saving and loading the graph. It is implemented by GraphVizGraphParser in order to read .dot in GraphViz format (the attribute of the vertices are stored following the standard), and StandardGraphParser in order to read with the provide standard of the assignment. Using the late biding, we are allowed to use the same method in GraphModel.java for both the format.

Operation.java defines the basic operation that is possible to do in the program. It keeps the element involved and the type of the operation done, in order to collect them to be used later from UndoManager.java and RedoManager.java. The operation done and the data structure of these two classes are similar, thus we choose to create an abstract class (AbstractUndoableEdit.java) which provide the common code (methods and field).

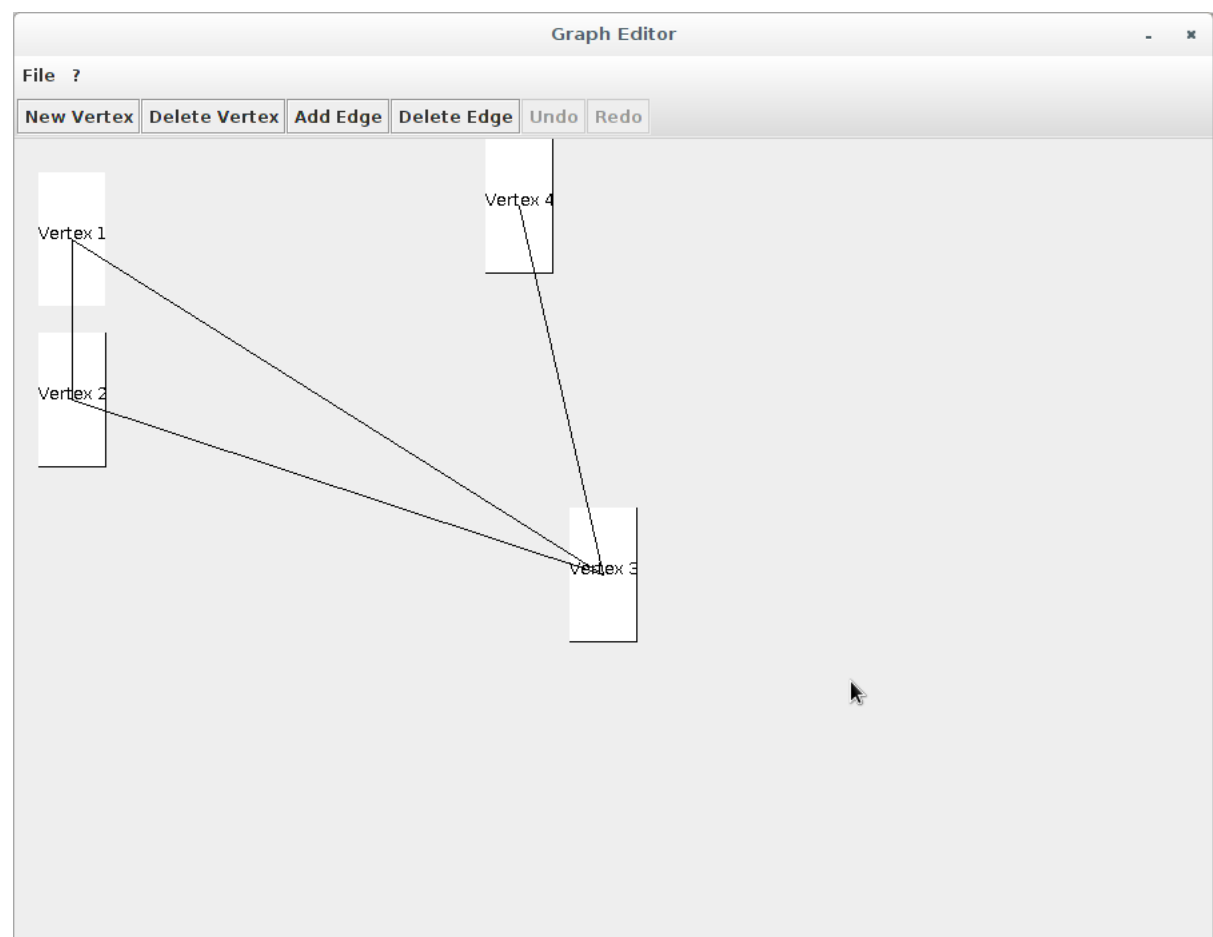
In `GraphFrame.java` class we created the user interface with all the required features such as the menubar and the toolbar using the Swing library. We created methods that provide the event routines for the keyboard shortcut and the click of the button. Furthermore, it also managed the menu, that provides the operation with the filesystem and some basic information about the program.

`GraphPanel.java` class is where the drawing operations are performed. The methods in this class allows the drawing of all vertices and edges, and updates the diagram with each modification done. It is also the Observer of `GraphModel`, and it is responsible to provide the update of the data structure.

`SelectionController.java` class extends the classes `MouseListener` and `MouseMotionListener`. By overloading the methods, it is possible to track mouse movements on the panel and perform actions accordingly. Double clicking to change the name of a vertex, dragging a vertex to change its position and selecting a certain vertex operations are tracked by this class.

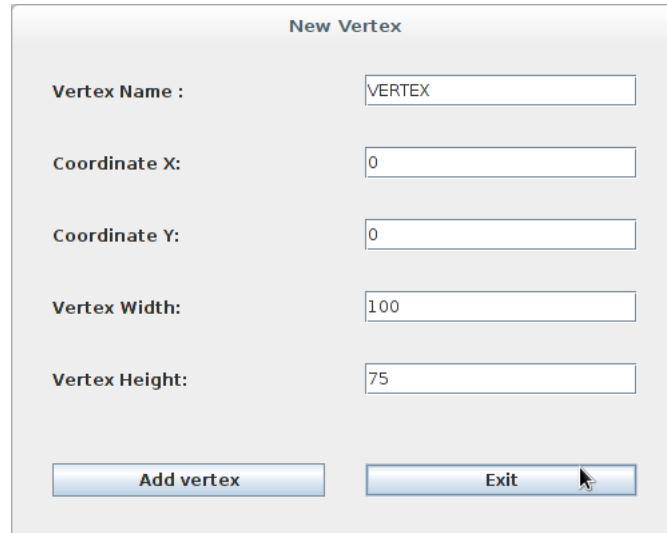
4 Evaluation of the program

We obtain the following windows.



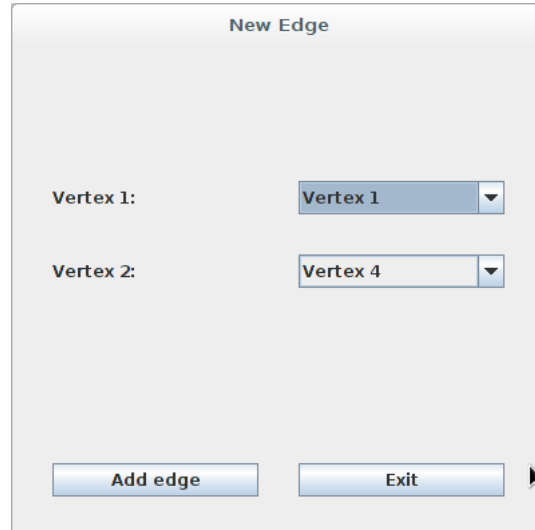
This is the main window that a user comes across when the program is run. The application opens up and adds the vertex and edges according to the file containing the data about them. It contains a menubar where operations on File, Edit and Window can be performed. Furthermore it supports a toolbar where operations

regarding the diagram can be performed such as the addition/removal of vertices and edges, undo and redo operations.



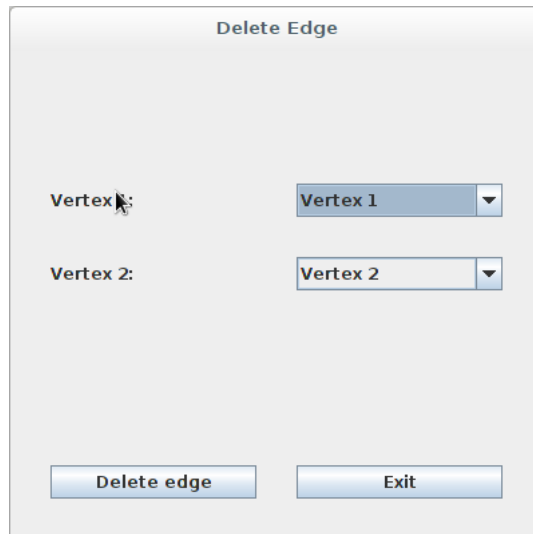
The 'New Vertex' dialog box is a light gray window with a title bar. It contains five input fields arranged vertically. The first field is labeled 'Vertex Name :' and contains the text 'VERTEX'. The second field is labeled 'Coordinate X:' and contains the number '0'. The third field is labeled 'Coordinate Y:' and contains the number '0'. The fourth field is labeled 'Vertex Width:' and contains the number '100'. The fifth field is labeled 'Vertex Height:' and contains the number '75'. At the bottom of the dialog, there are two buttons: 'Add vertex' on the left and 'Exit' on the right. A mouse cursor is pointing at the 'Exit' button.

This is the window where it is possible to add a new vertex. The user enters the name of the vertex, upper-left point in regards to x and y coordinates, the width and the height of the vertex. Clicking the "Add Vertex" button adds the vertex to the diagram.



The 'New Edge' dialog box is a light gray window with a title bar. It contains two dropdown menus arranged vertically. The first dropdown is labeled 'Vertex 1:' and has 'Vertex 1' selected. The second dropdown is labeled 'Vertex 2:' and has 'Vertex 4' selected. At the bottom of the dialog, there are two buttons: 'Add edge' on the left and 'Exit' on the right. A mouse cursor is pointing at the 'Exit' button.

This is the window where it is possible to add a new edge between two vertices. Available vertices that are not connected with the first one can be chosen from the dropdown menu, and a new edge between the two chosen vertices will be created. If the operation can not be performed an error dialog will pop. Furthermore the program is able to know if the graph is connected (all the vertices are connected to each other).

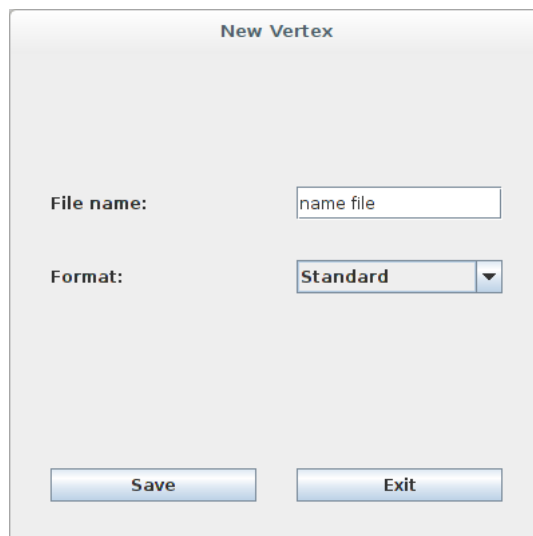
A dialog box titled "Delete Edge" with a light gray background. It contains two labels, "Vertex 1:" and "Vertex 2:", each followed by a dropdown menu. The first dropdown menu is currently showing "Vertex 1" and the second is showing "Vertex 2". At the bottom of the dialog, there are two buttons: "Delete edge" and "Exit".

Vertex 1: Vertex 1

Vertex 2: Vertex 2

Delete edge Exit

This is the window where it is possible to delete an edge between two vertices. The dropdown menu shows the first vertex, and the second dropdown menu is updated when the first one is selected. Furthermore the program is able to know when there are not edge.

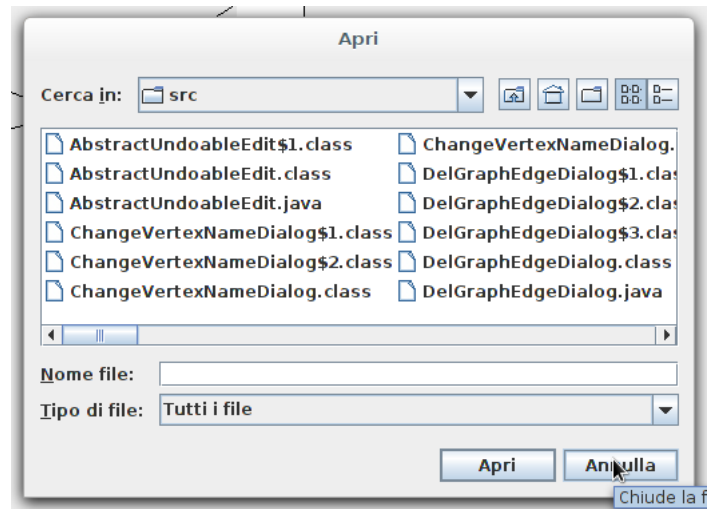
A dialog box titled "New Vertex" with a light gray background. It contains two labels, "File name:" and "Format:", each followed by a text input field or a dropdown menu. The "File name:" field contains the text "name file". The "Format:" dropdown menu is currently showing "Standard". At the bottom of the dialog, there are two buttons: "Save" and "Exit".

File name: name file

Format: Standard

Save Exit

This is the window where it is possible to save the graph. There are three different output format (standard, as required, Dot and serialized) and it is possible to choose the name for the output file. The extension would be add from the program, in according to the output file selected.



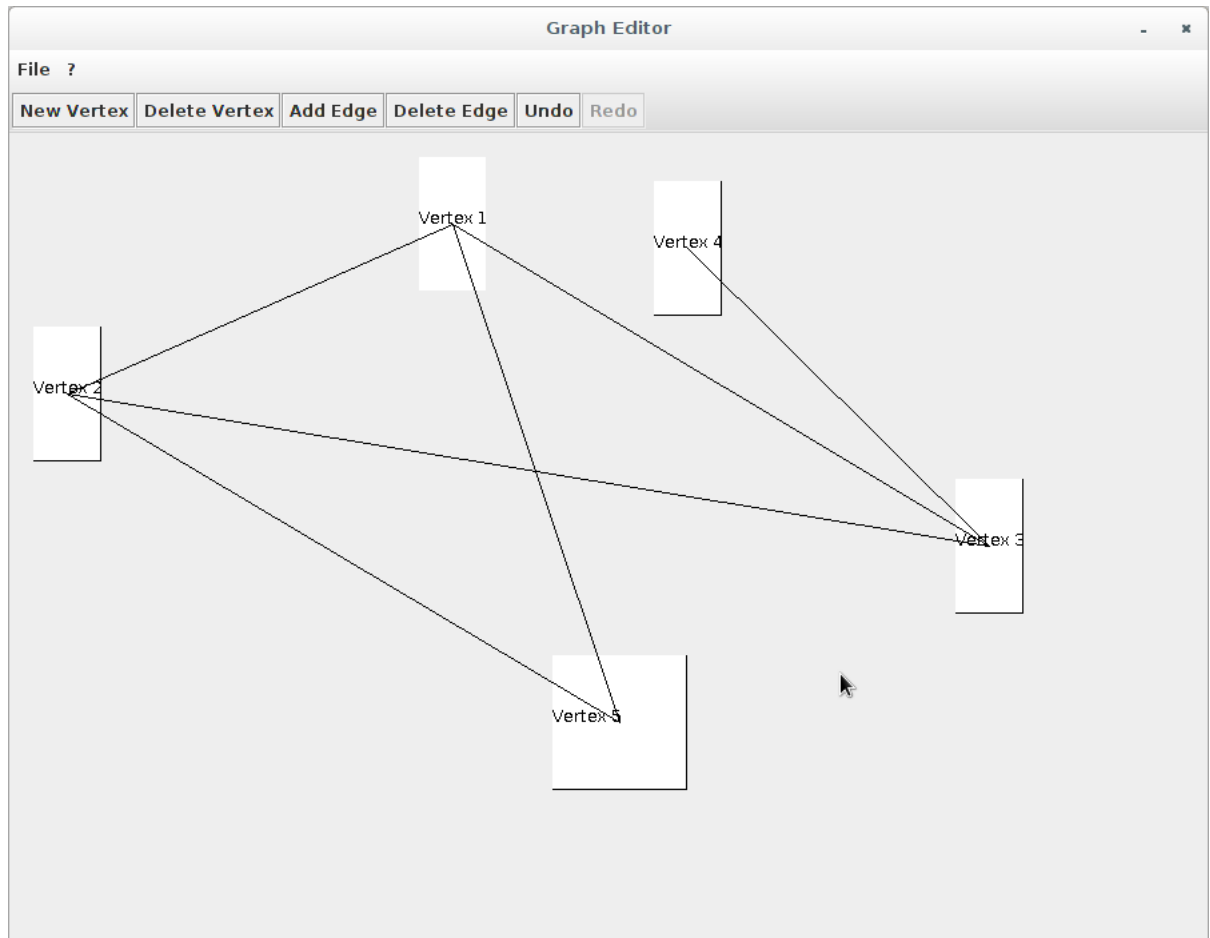
This is the window where it is possible to load a graph from a file, choosing a file from the filesystem.



This is the window where it is showed the credits and the information about the developers.



This is the window where it is showed the tips about the available keyboard shortcut.



This is a representation of how the diagram looks with multiple vertices and edges in between them.

5 Extension of the program

We implemented two different ways to store and restore a graph: the Java serialization and the GraphViz format. The first one was easy to implement and we are sure and safe for the persistence of the data. The second we decided to store the information of the vertices as attributes, following the standard. These information has to appear only once for vertex, to keep the file light and readable. For the same reason, a vertex has to appear less as possible. Thus we choose to do not print one row for each vertex, but print only the necessary row.

We implemented the keyboard shortcut to manage the basic operation. These shortcuts work only when the focus is not on the buttons of the main menu, because it would be useless to use the keyboard shortcut when it is possible to use the buttons with the keyboard.

6 Conclusions

Our program solves the assignment requests implementing features described in the assignment description with some extra features.

With this assignment we learned to use the Java `Graphics` class to create and modify shapes such as lines and rectangles using the `Graphics` class. Furthermore, we learned how to use the `MouseListener` and `MouseMotionListener` classes which allow the user to modify the diagram by using the mouse

events such as clicking, double clicking and dragging the cursor.

We learned to use the abstract class more efficiently, and we are able to confirm the improvement of the code about the readability and the reuse.

We learned to manage files, reading and writing in order to obtain a copy of our data structure, managing the String object. Furthermore, we could notice the efficiency of the Serializer interface of Java, that allows to implement the persistence in few line of code.

The creation of the project required the accumulated knowledge that was gathered through the course, as all concepts were included in the development of the application.

7 Appendix: program text

Listing 1: AbstractUndoableEdit

```
1 import java.util.Stack;
2
3 public abstract class AbstractUndoableEdit {
4
5     public GraphModel graphModel;
6     public Stack<Operation> stackOperation;
7
8     public AbstractUndoableEdit() {
9         this.graphModel = null;
10        this.stackOperation = new Stack<Operation>();
11    }
12
13    public AbstractUndoableEdit(GraphModel graphModel) {
14        this.graphModel = graphModel;
15        this.stackOperation = new Stack<Operation>();
16    }
17
18    public Operation doOperation(Operation op) {
19        switch (op.getOperation()) {
20            case ADD_VERTEX: {
21                /*Remove added vertex*/
22                this.getGraphModel().removeVertex(op.getVertex());
23                op.setOperation(Operation.OperationType.REMOVE_VERTEX);
24                break;
25            }
26            case REMOVE_VERTEX: {
27                /*Add removed vertex*/
28                this.getGraphModel().addVertex(op.getVertex());
29                for (GraphEdge egde : op.getEdges()) {
30                    this.getGraphModel().addEdge(egde);
31                }
32                op.setOperation(Operation.OperationType.ADD_VERTEX);
33                break;
34            }
35            case ADD_EDGE: {
36                /*Remove added edge*/
37                this.getGraphModel().removeEdge(op.getEdges().get(0));
38                op.setOperation(Operation.OperationType.REMOVE_EDGE);
39                break;
40            }
41            case REMOVE_EDGE: {
42                /*Add removed e*/
43                this.getGraphModel().addEdge(op.getEdges().get(0));
```

```

44         op.setOperation(Operation.OperationType.ADD_EDGE);
45         break;
46     }
47 }
48
49     return op;
50 }
51
52     public void addOperation(Operation op) {
53         // TODO Auto-generated method stub
54         this.getStackOperation().push(op);
55     }
56
57     /*AUTOgenerate setters and getters*/
58     public GraphModel getGraphModel() {
59         return graphModel;
60     }
61     public void setGraphModel(GraphModel graphModel) {
62         this.graphModel = graphModel;
63     }
64     public Stack<Operation> getStackOperation() {
65         return stackOperation;
66     }
67     public void setStackOperation(Stack<Operation> stackOperation) {
68         this.stackOperation = stackOperation;
69     }
70 }
71 }

```

Listing 2: ChangeVertexNameDialog

```

1  import java.awt.BorderLayout;
2  import java.awt.Dimension;
3  import java.awt.GridLayout;
4  import java.awt.Rectangle;
5  import java.awt.Toolkit;
6  import java.awt.event.ActionEvent;
7  import java.awt.event.ActionListener;
8
9  import javax.swing.JButton;
10 import javax.swing.JDialog;
11 import javax.swing.JLabel;
12 import javax.swing.JOptionPane;
13 import javax.swing.JPanel;
14 import javax.swing.JTextField;
15 import javax.swing.border.EmptyBorder;
16
17
18 public class ChangeVertexNameDialog extends JDialog{
19     final int WIDTH_WINDOW = 500;
20     final int HEIGHT_WINDOW = 400;
21
22     private JPanel centerPanel;
23     private JPanel southPanel;
24
25     private JTextField txtName;
26
27     private JButton btnSubmit;
28     private JButton btnExit;

```

```

29
30     private GraphModel graph;
31     private GraphVertex vertex;
32     private GraphPanel panel;
33
34     private String oldName;
35
36     public ChangeVertexNameDialog(GraphModel _graph, GraphVertex _vertex,
37         GraphPanel _panel){
38         this.panel = _panel;
39         this.graph = _graph;
40         this.vertex = _vertex;
41         this.oldName = _vertex.getName();
42
43         this.setWindow();
44         this.setPanel();
45         this.drawFields();
46
47         this.btnSubmit = new JButton("Change_Name");
48         this.btnSubmit.addActionListener(new ActionListener() {
49             @Override
50             public void actionPerformed(ActionEvent e) {
51                 String nameVertex = txtName.getText().equals("") ? GraphVertex
52                     .DEFAULT_NAME : txtName.getText();
53                 if (!ChangeVertexNameDialog.this.graph.containsVertex(
54                     nameVertex))
55                     ChangeVertexNameDialog.this.vertex.setName(nameVertex);
56                 else{
57                     String error = "Name_already_used!";
58                     JOptionPane.showMessageDialog(null, error);
59                 }
60                 ChangeVertexNameDialog.this.dispose();
61                 panel.repaint();
62             }
63         });
64
65         this.btnExit = new JButton("Exit");
66         this.btnExit.addActionListener(new ActionListener() {
67             @Override
68             public void actionPerformed(ActionEvent e) {
69                 ChangeVertexNameDialog.this.dispose();
70             }
71         });
72
73         this.southPanel.add(btnSubmit);
74         this.southPanel.add(btnExit);
75         this.add(centerPanel, BorderLayout.CENTER);
76         this.add(southPanel, BorderLayout.SOUTH);
77     }
78
79     private void clearTextField(){
80         this.txtName.setText(this.oldName);
81     }
82
83     private void setPanel(){
84         this.centerPanel = new JPanel();
85         this.centerPanel.setLayout(new GridLayout(5, 2, 30, 30));

```

```

84         this.centerPanel.setBorder(new EmptyBorder(20, 30, 20, 30));
85         this.southPanel = new JPanel();
86         this.southPanel.setLayout(new GridLayout(0,2,30,30));
87         this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
88     }
89
90     private void setWindow(){
91         this.setTitle("New_Vertex");
92         this.setResizable(false);
93         this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
94         this.getContentPane().setLayout(new BorderLayout());
95         Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
96         this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
97     }
98
99     private void drawFields(){
100         JLabel lblName = new JLabel("Change_Vertex_Name:");
101         this.centerPanel.add(lblName);
102         this.txtName = new JTextField();
103         this.centerPanel.add(this.txtName);
104
105         this.clearTextField();
106     }
107 }

```

Listing 3: DelGraphEdgeDialog

```

1  import java.awt.BorderLayout;
2  import java.awt.Dimension;
3  import java.awt.GridLayout;
4  import java.awt.Toolkit;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ItemEvent;
8  import java.awt.event.ItemListener;
9  import java.util.ArrayList;
10 import javax.swing.DefaultComboBoxModel;
11 import javax.swing.JButton;
12 import javax.swing.JComboBox;
13 import javax.swing.JDialog;
14 import javax.swing.JLabel;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17 import javax.swing.border.EmptyBorder;
18
19 public class DelGraphEdgeDialog extends JDialog {
20     final int WIDTH_WINDOW = 400;
21     final int HEIGHT_WINDOW = 400;
22
23     private JPanel centerPanel;
24     private JPanel southPanel;
25
26     private JComboBox<String> cmbV1;
27     private JComboBox<String> cmbV2;
28
29     private JButton btnSubmit;
30     private JButton btnExit;
31 }

```

```

32     private GraphModel graph;
33
34     public DelGraphEdgeDialog(GraphModel _graph){
35         this.graph = _graph;
36         this.setWindow();
37         this.setPanel();
38         this.drawFields();
39
40         this.btnSubmit = new JButton("Delete_edge");
41         this.btnSubmit.addActionListener(new ActionListener() {
42             @Override
43             public void actionPerformed(ActionEvent e) {
44                 DelGraphEdgeDialog.this.removeEdge();
45                 DelGraphEdgeDialog.this.clearTextField();
46             }
47         });
48
49         this.btnExit = new JButton("Exit");
50         this.btnExit.addActionListener(new ActionListener() {
51             @Override
52             public void actionPerformed(ActionEvent e) {
53                 DelGraphEdgeDialog.this.dispose();
54             }
55         });
56
57         this.cmbV1.addItemListener(new ItemListener() {
58             public void itemStateChanged(ItemEvent e){
59                 if(e.getStateChange() == ItemEvent.SELECTED) {
60                     DelGraphEdgeDialog.this.updateComboV2(DelGraphEdgeDialog.
61                         this.cmbV1.getSelectedItem().toString());
62                 }
63             });
64
65         this.southPanel.add(btnSubmit);
66         this.southPanel.add(btnExit);
67         this.add(centerPanel, BorderLayout.CENTER);
68         this.add(southPanel, BorderLayout.SOUTH);
69     }
70
71     private void removeEdge(){
72         String nameV1 = this.cmbV1.getSelectedItem().toString();
73         if (nameV1.equals("")){
74             String error = "First_vertex_is_not_seleceted!";
75             JOptionPane.showMessageDialog(null, error);
76             return;
77         }
78         String nameV2 = this.cmbV2.getSelectedItem() == null ? "" : this.cmbV2
79             .getSelectedItem().toString();
80
81         if (!nameV2.equals("")){
82             GraphVertex v1 = graph.getVertexOfName(nameV1);
83             GraphVertex v2 = graph.getVertexOfName(nameV2);
84             graph.perfromOperation(new Operation(OperationType.
85                 REMOVE_EDGE,
86
87                 this.graph.getEdges().get(this
88                     .graph.
89                     getIndexEdgeOfVertexes(v1,

```

```

85         }else{
86             //Generate string error
87             String error = "Second_vertex_is_not_seleceted!";
88             JOptionPane.showMessageDialog(null, error);
89         }
90     }
91
92     private void updateComboV2(String nameV1){
93         this.cmbV2.setEnabled(true);
94         GraphVertex v1 = graph.getVertexOfName(nameV1);
95         DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
96             String>();
97         ArrayList<GraphVertex> adjVertexes = graph.getAdjVertexes(v1);
98         for (GraphVertex vertex : graph.getVertexes()){
99             if (adjVertexes.contains(vertex))
100                 vertexes.addElement(vertex.getName());
101         }
102         if (vertexes.getSize() == 0)
103             this.cmbV2.setEnabled(false);
104         this.cmbV2.setModel(vertexes);
105     }
106
107     private void clearTextField(){
108         DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
109             String>();
110         for (GraphVertex vertex : graph.getVertexes()){
111             if (this.graph.getAdjVertexes(vertex).size() != 0)
112                 vertexes.addElement(vertex.getName());
113         }
114         if (vertexes.getSize() == 0){
115             vertexes.addElement("There_are_not_edges");
116             this.cmbV1.setModel(vertexes);
117             this.cmbV2.setModel(new DefaultComboBoxModel<String>());
118             this.btnSubmit.setEnabled(false);
119             return;
120         }
121         this.cmbV1.setModel(vertexes);
122         this.cmbV1.setSelectedIndex(0);
123         this.updateComboV2(this.cmbV1.getSelectedItem().toString());
124     }
125
126     private void setPanel(){
127         this.centerPanel = new JPanel();
128         this.centerPanel.setLayout(new GridLayout(2,2,30,30));
129         this.centerPanel.setBorder(new EmptyBorder(100, 30, 100, 30));
130         this.southPanel = new JPanel();
131         this.southPanel.setLayout(new GridLayout(0,2,30,30));
132         this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
133     }
134
135     private void setWindow(){
136         this.setTitle("Delete_Edge");
137         this.setResizable(false);
138         this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
139         this.getContentPane().setLayout(new BorderLayout());

```

```

140         Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
141         this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
142     }
143
144     private void drawFields(){
145         JLabel lblVertex1 = new JLabel("Vertex_1:");
146         this.centerPanel.add(lblVertex1);
147         this.cmbV1 = new JComboBox<String>();
148         this.centerPanel.add(this.cmbV1);
149
150         JLabel lblVertex2 = new JLabel("Vertex_2:");
151         this.centerPanel.add(lblVertex2);
152         this.cmbV2 = new JComboBox<String>();
153         this.centerPanel.add(this.cmbV2);
154         this.clearTextField();
155     }
156 }

```

Listing 4: GraphEdge

```

1  import java.io.Serializable;
2
3  public class GraphEdge implements Serializable{
4
5      private GraphVertex v1, v2;
6
7      public GraphEdge(GraphVertex v1, GraphVertex v2){
8          this.v1 = v1;
9          this.v2 = v2;
10     }
11
12     public boolean containsVertex(GraphVertex vertex){
13         if (v1.getName().equals(vertex.getName()) ||
14             v2.getName().equals(vertex.getName())) return true;
15
16         return false;
17     }
18
19     @Override
20     public String toString(){
21         return this.getV1().getName()+"--"+this.getV2().getName();
22     }
23
24     /*AUTOgenerate setters and getters*/
25     public GraphVertex getV1() {
26         return v1;
27     }
28     public void setV1(GraphVertex v1) {
29         this.v1 = v1;
30     }
31     public GraphVertex getV2() {
32         return v2;
33     }
34     public void setV2(GraphVertex v2) {
35         this.v2 = v2;
36     }
37
38 }

```

Listing 5: GraphEdit

```

1  import java.io.IOException;
2  import java.util.Observable;
3  import java.util.Observer;
4
5  public class GraphEdit{
6
7      public static void main(String[] argv){
8          GraphModel graph = null;
9          GraphFrame graphFrame;
10         try {
11             if (argv.length == 1){
12                 String[] file = argv[argv.length-1].split("\\.");
13
14                 if (file[1].equals("txt"))
15                     graph = new GraphModel(argv[argv.length-1], new
                        StandardGraphParser());
16                 if (file[1].equals("dot"))
17                     graph = new GraphModel(argv[argv.length-1], new
                        GraphVizGraphParser());
18                 if (file[1].equals("obj")){
19                     graph = new GraphModel();
20                     graph.deserializeGraph(argv[argv.length-1]);
21                 }
22             }else
23                 graph = new GraphModel();
24
25             graphFrame = new GraphFrame(graph);
26             graphFrame.setVisible(true);
27         } catch (IOException e) {
28             // TODO Auto-generated catch block
29             graph = null;
30             e.printStackTrace();
31         } catch (ClassNotFoundException e) {
32             // TODO Auto-generated catch block
33             e.printStackTrace();
34         }
35     }
36
37 }

```

Listing 6: GraphFrame

```

1  import java.awt.BorderLayout;
2  import java.awt.Dimension;
3  import java.awt.GridLayout;
4  import java.awt.Toolkit;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.InputEvent;
8  import java.awt.event.KeyEvent;
9  import java.awt.event.KeyListener;
10 import java.awt.event.WindowAdapter;
11 import java.awt.event.WindowEvent;
12 import java.io.File;
13 import java.io.IOException;

```



```

14 import java.util.Observable;
15 import java.util.Observer;
16
17 import javax.swing.*;
18 import javax.swing.border.EmptyBorder;
19
20 public class GraphFrame extends JFrame{
21
22     public static final int WINDOW_WIDTH = 900;
23     public static final int WINDOW_HEIGHT = 700;
24     private static final String TITLE_NAME = "Graph_Editor";
25
26     private JPanel northPanel;
27     private JPanel centerPanel;
28
29     private JMenuBar menuBar;
30     private JMenu menuFile, menuInfo;
31     private JMenuItem mnSave, mnLoad, mnCredits, mnTips;
32     private JToolBar tb;
33     private JButton btnNewVertex, btnDelNode, btnNewEdge, btnDelEdge, undo,
        redo;
34     private GraphPanel graphPanel;
35
36     private GraphModel model;
37
38     public GraphFrame(GraphModel _graph){
39         this.setModel(_graph);
40         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41
42         this.setWindowProperites();
43         this.createMenu();
44         this.createToolBar();
45         this.addPanel();
46
47         this.btnNewVertex.addActionListener(new ActionListener() {
48             @Override
49             public void actionPerformed(ActionEvent e) {
50                 GraphFrame.this.addVertex();
51             }
52         });
53         this.btnDelNode.addActionListener(new ActionListener() {
54             @Override
55             public void actionPerformed(ActionEvent e) {
56                 GraphVertex vertexToDelete = GraphFrame.this.model.
                    getSelectedVertex();
57                 if (vertexToDelete != null)
58                     GraphFrame.this.model.perfromOperation(new Operation(
                        Operation.OperationType.REMOVE_VERTEX,
59                         vertexToDelete));
60                 else{
61                     String error = "Select_a_vertex!";
62                     JOptionPane.showMessageDialog(null, error);
63                 }
64                 GraphFrame.this.undo.setEnabled(true);
65                 GraphFrame.this.redo.setEnabled(false);
66                 GraphFrame.this.graphPanel.repaint();
67             }
68         });

```

```

69
70     this.btnDelEdge.addActionListener(new ActionListener() {
71         @Override
72         public void actionPerformed(ActionEvent e) {
73             //Remove edge
74             GraphFrame.this.deleteEdge();
75             GraphFrame.this.graphPanel.repaint();
76         });
77     this.btnNewEdge.addActionListener(new ActionListener() {
78         @Override
79         public void actionPerformed(ActionEvent e) {
80             GraphFrame.this.addEdge();
81             GraphFrame.this.graphPanel.repaint();
82         }
83     });
84     this.undo.addActionListener(new ActionListener() {
85         @Override
86         public void actionPerformed(ActionEvent e) {
87             GraphFrame.this.model.getUndoManager().undoOperation();
88             GraphFrame.this.redo.setEnabled(true);
89             if (GraphFrame.this.model.getUndoManager().stackOperation.
                isEmpty())
                GraphFrame.this.undo.setEnabled(false);
90         }
91     });
92
93     this.redo.addActionListener(new ActionListener() {
94         @Override
95         public void actionPerformed(ActionEvent e) {
96             GraphFrame.this.model.getRedoManager().redoOperation();
97             GraphFrame.this.undo.setEnabled(true);
98             if (GraphFrame.this.model.getRedoManager().stackOperation.
                isEmpty())
                GraphFrame.this.redo.setEnabled(false);
99         }
100    });
101
102
103     this.mnSave.addActionListener(new ActionListener() {
104         @Override
105         public void actionPerformed(ActionEvent e) {
106             GraphFrame.this.saveGraph();
107         }
108     });
109
110     this.mnLoad.addActionListener(new ActionListener() {
111         @Override
112         public void actionPerformed(ActionEvent e) {
113             GraphFrame.this.loadGraph();
114         }
115     });
116
117     this.mnCredits.addActionListener(new ActionListener() {
118         @Override
119         public void actionPerformed(ActionEvent e) {
120             String msg = "Graph_Editor_2016_\n_Developed_by_Corradini_
                Matteo_(S3051390)_and_"
121                 + "Berke_Atac_(S3075168)";
122             JOptionPane.showMessageDialog(null, msg);
123         }

```

```

124     });
125     this.mnTips.addActionListener(new ActionListener() {
126         @Override
127         public void actionPerformed(ActionEvent e) {
128             String msg = "Keyboard shortcut, when no buttons are selected:
129                 \n"
130                 + "CTRL+V: create a new vertex; \n"
131                 + "CTRL+E: create a new edge; \n"
132                 + "CTRL+S: save; \n"
133                 + "CTRL+L: load; \n"
134                 + "CTRL+U: undo; \n"
135                 + "CTRL+Y: redo; \n"
136                 + "CTRL+D: remove vertex. \n";
137             JOptionPane.showMessageDialog(null, msg);
138         }
139     });
140     this.graphPanel.addKeyListener(new KeyListener() {
141         @Override
142         public void keyPressed(KeyEvent e) {
143             if ((e.getKeyCode() == KeyEvent.VK_V) && ((e.getModifiers() &
144                 KeyEvent.CTRL_MASK) != 0)) {
145                 GraphFrame.this.addVertex();
146                 return;
147             }
148             if ((e.getKeyCode() == KeyEvent.VK_E) && ((e.getModifiers() &
149                 KeyEvent.CTRL_MASK) != 0)) {
150                 GraphFrame.this.addEdge();
151                 return;
152             }
153             if ((e.getKeyCode() == KeyEvent.VK_S) && ((e.getModifiers() &
154                 KeyEvent.CTRL_MASK) != 0)) {
155                 GraphFrame.this.saveGraph();
156                 return;
157             }
158             if ((e.getKeyCode() == KeyEvent.VK_L) && ((e.getModifiers() &
159                 KeyEvent.CTRL_MASK) != 0)) {
160                 GraphFrame.this.loadGraph();
161                 return;
162             }
163             if ((e.getKeyCode() == KeyEvent.VK_U) && ((e.getModifiers() &
164                 KeyEvent.CTRL_MASK) != 0)) {
165                 if (!GraphFrame.this.model.getUndoManager().
166                     getStackOperation().isEmpty()) {
167                     GraphFrame.this.redo.setEnabled(true);
168                     if (GraphFrame.this.model.getUndoManager().
169                         stackOperation.isEmpty())
170                         GraphFrame.this.undo.setEnabled(false);
171                 }
172                 return;
173             }
174             if ((e.getKeyCode() == KeyEvent.VK_Y) && ((e.getModifiers() &
175                 KeyEvent.CTRL_MASK) != 0)) {
176                 if (!GraphFrame.this.model.getRedoManager().
177                     getStackOperation().isEmpty()) {
178                     GraphFrame.this.model.getRedoManager().redoOperation()
179                     ;
180                     GraphFrame.this.model.getUndoManager().undoOperation()

```

```

171         ;
172         GraphFrame.this.undo.setEnabled(true);
173         if (GraphFrame.this.model.getRedoManager().
174             stackOperation.isEmpty())
175             GraphFrame.this.redo.setEnabled(false);
176     }
177     }
178     if ((e.getKeyCode() == KeyEvent.VK_D) && ((e.getModifiers() &
179         KeyEvent.CTRL_MASK) != 0)) {
180         GraphVertex vertexToDelete = GraphFrame.this.model.
181             getSelectedVertex();
182         if (vertexToDelete != null){
183             GraphFrame.this.model.performOperation(new Operation(
184                 Operation.OperationType.REMOVE_VERTEX,
185                 vertexToDelete
186             ));
187             GraphFrame.this.undo.setEnabled(true);
188             GraphFrame.this.redo.setEnabled(false);
189             GraphFrame.this.graphPanel.repaint();
190         }
191         return;
192     }
193     }
194     @Override
195     public void keyReleased(KeyEvent e) {
196         // TODO Auto-generated method stub
197     }
198     @Override
199     public void keyTyped(KeyEvent e) {
200         // TODO Auto-generated method stub
201     }
202     });
203
204     this.drawWholeFrame();
205 }
206
207 private void setWindowProperties(){
208     this.setTitle(TITLE_NAME);
209     this.setResizable(false);
210     this.setSize(new Dimension(WINDOW_WIDTH, WINDOW_HEIGHT));
211     this.getContentPane().setLayout(new BorderLayout());
212     Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
213     this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
214         .getSize().height/2);
215 }
216
217 private void createMenu(){
218     this.menuBar = new JMenuBar();
219
220     //add first headings
221     this.menuFile = new JMenu("File");
222     this.menuInfo = new JMenu("?");
223
224     this.mnSave = new JMenuItem("Save_Graph");
225     this.mnLoad = new JMenuItem("Load_Graph");
226     this.mnCredits = new JMenuItem("Credits");
227     this.mnTips = new JMenuItem("Tips");

```

```

222         this.menuFile.add(this.mnSave);
223         this.menuFile.add(this.mnLoad);
224
225         this.menuInfo.add(this.mnCcredits);
226         this.menuInfo.add(this.mnTips);
227
228         this.menuBar.add(menuFile);
229         this.menuBar.add(menuInfo);
230     }
231
232     private void addPanel() {
233         this.graphPanel = new GraphPanel(this.model);
234         this.northPanel = new JPanel (new GridLayout(2, 0, 0, 0));
235         this.centerPanel = new JPanel (new BorderLayout());
236
237         this.northPanel.add(this.menuBar);
238         this.northPanel.add(this.tb);
239         this.centerPanel.add(graphPanel);
240     }
241
242     private void createToolbar() {
243         this.tb = new JToolBar();
244         tb.setFloatable(false);
245
246         this.btnNewVertex = new JButton("New_Vertex");
247         this.btnDelNode = new JButton("Delete_Vertex");
248         this.btnNewEdge = new JButton("Add_Edge");
249         this.btnDelEdge = new JButton("Delete_Edge");
250         this.undo = new JButton("Undo");
251         this.redo = new JButton("Redo");
252
253         this.getUndo().setEnabled(false);
254         this.getRedo().setEnabled(false);
255         this.btnDelEdge.setEnabled(this.model.getEdges().size() > 0);
256
257         this.getTb().add(this.getBtnNewVertex());
258         this.getTb().add(this.getBtnDelNode());
259         this.getTb().add(this.getBtnNewEdge());
260         this.getTb().add(this.getBtnDelEdge());
261         this.getTb().add(this.getUndo());
262         this.getTb().add(this.getRedo());
263     }
264
265     private void drawWholeFrame() {
266         this.add(this.northPanel, BorderLayout.NORTH);
267         this.add(this.centerPanel, BorderLayout.CENTER);
268     }
269
270     private void saveGraph() {
271         SaveGraphDialog saveDialog = new SaveGraphDialog(this.getModel());
272         saveDialog.setVisible(true);
273     }
274
275     private void loadGraph() {
276         JFileChooser fileChooser = new JFileChooser();
277         fileChooser.setCurrentDirectory(new File(System.getProperty("user.dir"
278             )),);
278         int result = fileChooser.showOpenDialog(this);

```

```

279     if (result == JFileChooser.APPROVE_OPTION) {
280         File selectedFile = fileChooser.getSelectedFile();
281         String[] nameFile = selectedFile.getName().split("\\.");
282         try {
283             boolean fileCorrect = false;
284             if (nameFile[1].equals("txt")) {
285                 this.model = new GraphModel(selectedFile.getAbsolutePath()
286                     , new StandardGraphParser());
287                 fileCorrect = true;
288             }
289             if (nameFile[1].equals("dot")) {
290                 this.model = new GraphModel(selectedFile.getAbsolutePath()
291                     , new GraphVizGraphParser());
292                 fileCorrect = true;
293             }
294             if (nameFile[1].equals("obj")) {
295                 this.model.deSerializeGraph(selectedFile.getAbsolutePath()
296                     );
297                 fileCorrect = true;
298             }
299             if (!fileCorrect) {
300                 String error = "Select_.txt,_.dot_or_.obj_file!";
301                 JOptionPane.showMessageDialog(null, error);
302                 return;
303             }
304             this.graphPanel.setModel(this.model);
305             this.graphPanel.repaint();
306         } catch (IOException e) {
307             // TODO Auto-generated catch block
308             String error = "Problems_to_open_the_file!";
309             JOptionPane.showMessageDialog(null, error);
310             e.printStackTrace();
311         } catch (ClassNotFoundException e) {
312             // TODO Auto-generated catch block
313             String error = "Parsing_file_failed!";
314             JOptionPane.showMessageDialog(null, error);
315             e.printStackTrace();
316         }
317     }
318 }
319
320 private void addVertex() {
321     NewGraphVertexDialog vertexDialog = new NewGraphVertexDialog(this.
322         getModel());
323     vertexDialog.setVisible(true);
324     vertexDialog.addWindowListener(new WindowAdapter() {
325         @Override
326         public void windowClosed(WindowEvent e) {
327             GraphFrame.this.undo.setEnabled(true);
328             GraphFrame.this.redo.setEnabled(false);
329         }
330     });
331 }
332 private void addEdge() {

```

```

333         NewGraphEdgeDialog edgeDialog = new NewGraphEdgeDialog(this.getModel()
334             );
335         edgeDialog.setVisible(true);
336         edgeDialog.addWindowListener(new WindowAdapter() {
337             @Override
338             public void windowClosed(WindowEvent e) {
339                 GraphFrame.this.undo.setEnabled(true);
340                 GraphFrame.this.redo.setEnabled(false);
341                 if (GraphFrame.this.model.getEdges().size() > 0)
342                     GraphFrame.this.btnDelEdge.setEnabled(true);
343             }
344         });
345     }
346     private void deleteEdge() {
347         DelGraphEdgeDialog edgeDialog = new DelGraphEdgeDialog(this.getModel()
348             );
349         edgeDialog.setVisible(true);
350         edgeDialog.addWindowListener(new WindowAdapter() {
351             @Override
352             public void windowClosed(WindowEvent e) {
353                 GraphFrame.this.undo.setEnabled(true);
354                 GraphFrame.this.redo.setEnabled(false);
355                 if (GraphFrame.this.model.getEdges().size() == 0)
356                     GraphFrame.this.btnDelEdge.setEnabled(false);
357             }
358         });
359     }
360     public JMenu getMenuFile() {
361         return menuFile;
362     }
363     public void setMenuFile(JMenu menuFile) {
364         this.menuFile = menuFile;
365     }
366     public JToolBar getTb() {
367         return tb;
368     }
369     public void setTb(JToolBar tb) {
370         this.tb = tb;
371     }
372     public JButton getBtnNewVertex() {
373         return btnNewVertex;
374     }
375     public void setBtnNewVertex(JButton btnNewVertex) {
376         this.btnNewVertex = btnNewVertex;
377     }
378     public JButton getBtnDelNode() {
379         return btnDelNode;
380     }
381     public void setBtnDelNode(JButton btnDelNode) {
382         this.btnDelNode = btnDelNode;
383     }
384     public JButton getBtnNewEdge() {
385         return btnNewEdge;
386     }
387     public void setBtnNewEdge(JButton btnNewEdge) {
388         this.btnNewEdge = btnNewEdge;

```

```

389     }
390     public JButton getUndo() {
391         return undo;
392     }
393     public void setUndo(JButton undo) {
394         this.undo = undo;
395     }
396     public JButton getRedo() {
397         return redo;
398     }
399     public void setRedo(JButton redo) {
400         this.redo = redo;
401     }
402     public GraphModel getModel() {
403         return model;
404     }
405     public void setModel(GraphModel model) {
406         this.model = model;
407     }
408     public JButton getBtnDelEdge() {
409         return btnDelEdge;
410     }
411     public void setBtnDelEdge(JButton btnDelEdge) {
412         this.btnDelEdge = btnDelEdge;
413     }
414 }

```

Listing 7: GraphModel

```

1  import java.io.FileInputStream;
2  import java.io.FileOutputStream;
3  import java.io.IOException;
4  import java.io.ObjectInputStream;
5  import java.io.ObjectOutputStream;
6  import java.io.Serializable;
7  import java.util.ArrayList;
8  import java.util.Iterator;
9  import java.util.Observable;
10
11  public class GraphModel extends Observable implements Serializable{
12
13      private ArrayList<GraphVertex> vertexes;
14      private ArrayList<GraphEdge> edges;
15      transient private UndoManager undoManager;
16      transient private RedoManager redoManager;
17
18      transient private GraphVertex selectedVertex;
19
20      public GraphModel(){
21          this.vertexes = new ArrayList<GraphVertex>();
22          this.edges = new ArrayList<GraphEdge>();
23          this.undoManager = new UndoManager(this);
24          this.redoManager = new RedoManager(this);
25      }
26
27      public GraphModel(GraphModel _graph){
28          this.vertexes = _graph.getVertexes();
29          this.edges = _graph.getEdges();
30          this.undoManager = _graph.getUndoManager();

```



```

31         this.redoManager = _graph.getRedoManager();
32     }
33
34     public GraphModel(String nameFile, GraphParser parser) throws IOException{
35         this();
36         parser.loadFromFile(nameFile, this);
37     }
38
39     public void addEdge(GraphEdge edge){
40         this.getEdges().add(edge);
41         this.sendNotificationToObs("Add_edge,_from:" + edge.getV1().getName()
42             + "_to:"
43                                     + edge.getV2().getName())
44             ;
45     }
46
47     public void addVertex(GraphVertex vertex){
48         this.getVertexes().add(vertex);
49         this.sendNotificationToObs("Add_vertex,_name:" + vertex.getName());
50     }
51
52     public void removeVertex(GraphVertex vertex){
53         String vertexName = vertex.getName();
54         for (Iterator<GraphEdge> it = this.getEdges().iterator(); it.hasNext()
55             ; ) {
56             GraphEdge edgeToRemove = it.next();
57             if (edgeToRemove.containsVertex(vertex)){
58                 String v1Name = edgeToRemove.getV1().getName(), v2Name =
59                     edgeToRemove.getV2().getName();
60                 it.remove();
61                 this.sendNotificationToObs("Remove_edge,_from:" + v1Name + "_
62                     to:"
63                                     + v2Name);
64             }
65         }
66         this.getVertexes().remove(vertex);
67         this.sendNotificationToObs("Remove_vertex,_name:" + vertexName);
68     }
69
70     public void removeEdge(GraphEdge edge){
71         String v1Name = edge.getV1().getName(), v2Name = edge.getV2().getName
72             ();
73         this.getEdges().remove(edge);
74         this.sendNotificationToObs("Remove_edge,_from:" + v1Name + "_to:"
75             + v2Name);
76     }
77
78     public void perfromOperation(Operation op){
79         // TODO Auto-generated method stub
80         switch (op.getOperation()){
81             case ADD_VERTEX:{
82                 this.addVertex(op.getVertex());
83                 break;
84             }
85             case REMOVE_VERTEX:{
86                 for (GraphEdge edge : this.getEdges()){
87                     if (edge.containsVertex(op.getVertex()))
88                         op.getEdges().add(edge);
89                 }
90             }
91         }
92     }

```

```

83         }
84         this.removeVertex(op.getVertex());
85         break;
86     }
87     case ADD_EDGE: {
88         this.addEdge(op.getEdges().get(0));
89         break;
90     }
91     case REMOVE_EDGE: {
92         this.removeEdge(op.getEdges().get(0));
93         break;
94     }
95 }
96 this.getUndoManager().addOperation(op);
97 if (!this.getRedoManager().stackOperation.isEmpty())
98     this.getRedoManager().flushRedoStack();
99 }
100
101 public boolean containsVertex(String vertexName) {
102     for (GraphVertex vertex : this.getVertexes()) {
103         if (vertex.getName().equals(vertexName))
104             return true;
105     }
106     return false;
107 }
108
109 public boolean containsEdge(GraphVertex v1, GraphVertex v2) {
110     for (GraphEdge edges : this.getEdges()) {
111         String v1Name = edges.getV1().getName(), v2Name = edges.getV2().
            getName();
112         if (v1Name.equals(v1.getName()) && v2Name.equals(v2.getName()))
113             return true;
114     }
115     return false;
116 }
117
118 public int getIndexOfVertex(GraphVertex vertex) {
119     for (int i = 0; i < this.getVertexes().size(); i++) {
120         if (this.getVertexes().get(i).getName().equals(vertex.getName()))
121             return i;
122     }
123     return -1;
124 }
125
126 public GraphVertex getVertexOfName(String vertexName) {
127     for (GraphVertex vertex : this.getVertexes()) {
128         if (vertex.getName().equals(vertexName))
129             return vertex;
130     }
131     return null;
132 }
133
134 public ArrayList<GraphVertex> getAdjVertexes(GraphVertex vertex) {
135     ArrayList<GraphVertex> adj = new ArrayList<GraphVertex>();
136     for (GraphEdge edge : this.getEdges()) {
137         if (edge.containsVertex(vertex)) {
138             if (edge.getV1().getName().equals(vertex.getName()))
139                 adj.add(edge.getV2());

```

```

140         else
141             adj.add(edge.getV1());
142     }
143 }
144 return adj;
145 }
146
147 public void saveGraph(String nameFile, GraphParser parser) throws
148     IOException{
149     parser.saveGraph(nameFile, this);
150 }
151
152 public void serializeGraph(String nameFile) throws IOException{
153     FileOutputStream fileOut = new FileOutputStream(nameFile);
154     ObjectOutputStream out = new ObjectOutputStream(fileOut);
155     out.writeObject(this);
156     out.close();
157     fileOut.close();
158 }
159
160 public void deSerializeGraph(String nameFile) throws IOException,
161     ClassNotFoundException{
162     FileInputStream fileIn = new FileInputStream(nameFile);
163     ObjectInputStream in = new ObjectInputStream(fileIn);
164     GraphModel _graph = new GraphModel((GraphModel) in.readObject());
165     this.setEdges(_graph.getEdges());
166     this.setVertexes(_graph.getVertexes());
167     in.close();
168     fileIn.close();
169 }
170
171 public int getIndexEdgeOfVertexes(GraphVertex v1, GraphVertex v2){
172     for (GraphEdge edge : this.getEdges()){
173         if (edge.containsVertex(v1) && edge.containsVertex(v2))
174             return this.getEdges().indexOf(edge);
175     }
176     return -1;
177 }
178
179 public void sendNotificationToObs(String message){
180     setChanged();
181     this.notifyObservers(message);
182 }
183
184 /*AUTOgenerate setters and getters*/
185 public ArrayList<GraphVertex> getVertexes() {
186     return vertexes;
187 }
188 public void setVertexes(ArrayList<GraphVertex> vertexes) {
189     this.vertexes = vertexes;
190 }
191 public ArrayList<GraphEdge> getEdges() {
192     return edges;
193 }
194 public void setEdges(ArrayList<GraphEdge> edges) {
195     this.edges = edges;
196 }
197 public UndoManager getUndoManager() {

```

```

196         return undoManager;
197     }
198     public void setUndoManager(UndoManager undoManager) {
199         this.undoManager = undoManager;
200     }
201     public RedoManager getRedoManager() {
202         return redoManager;
203     }
204     public void setRedoManager(RedoManager redoManager) {
205         this.redoManager = redoManager;
206     }
207     public void setSelectedVertex(GraphVertex selected){
208         this.selectedVertex = selected;
209     }
210     public GraphVertex getSelectedVertex(){
211         return selectedVertex;
212     }
213 }

```

Listing 8: GraphPanel

```

1  import java.awt.Color;
2  import java.awt.Graphics;
3  import java.awt.Rectangle;
4  import java.util.Observable;
5  import java.util.Observer;
6
7  import javax.swing.JPanel;
8  import javax.swing.UIManager;
9
10 public class GraphPanel extends JPanel implements Observer {
11
12     private GraphModel graph;
13     private SelectionController selCon;
14
15     public GraphPanel(GraphModel _graph){
16         selCon = new SelectionController(_graph, this);
17         this.setModel(_graph);
18
19         this.addMouseListener(selCon);
20         this.addMouseMotionListener(selCon);
21     }
22
23     @Override
24     public void update(Observable o, Object arg) {
25         // TODO Auto-generated method stub
26         this.repaint();
27         System.out.println("update:_" + arg);
28     }
29
30     public void paintComponent(Graphics g){
31         if (g == null) return;
32         this.clear(g);
33         this.paintVertexes(g);
34         this.paintEdge(g);
35     }
36
37     private void paintVertexes(Graphics g){
38         if (this.graph == null) return;

```

```

39
40     for (GraphVertex vertex : this.graph.getVertexes()) {
41
42         int x = vertex.getRect().x, y = vertex.getRect().y, width = vertex
            .getRect().width;
43         int height = vertex.getRect().height;
44         g.drawRect(x, y, width, height);
45         g.setColor(Color.white);
46         g.fillRect(x, y, width, height);
47         g.setColor(Color.black);
48         g.drawString(vertex.getName(), vertex.getRect().x,
49                     vertex.getRect().y + vertex.getRect().height/2);
50     }
51 }
52
53 private void clear(Graphics g) {
54     g.setColor(UIManager.getColor("Panel.background"));
55     g.fillRect(0, 0, getWidth(), getHeight());
56 }
57
58 private void paintEdge(Graphics g) {
59     g.setColor(Color.black);
60     for (GraphEdge edge : this.graph.getEdges()) {
61         Rectangle rect1 = edge.getV1().getRect();
62         Rectangle rect2 = edge.getV2().getRect();
63         g.drawLine(rect1.x + (rect1.width/2), rect1.y + (rect1.height/2),
64                   rect2.x + (rect2.width/2), rect2.y + (rect2.height/2));
65     }
66 }
67
68
69 public void paintRed(GraphVertex vertex) {
70     Graphics g = this.getGraphics();
71     int x = vertex.getRect().x, y = vertex.getRect().y, width = vertex.
        getRect().width;
72     int height = vertex.getRect().height;
73     g.drawRect(x, y, width, height);
74     g.setColor(Color.red);
75     g.fillRect(x, y, width, height);
76     g.setColor(Color.black);
77     g.drawString(vertex.getName(), vertex.getRect().x,
78                 vertex.getRect().y + vertex.getRect().height/2);
79 }
80
81 public void repaint() {
82     this.requestFocus();
83     this.paintComponent(this.getGraphics());
84 }
85
86 public void setModel(GraphModel _graph) {
87     this.graph = _graph;
88     this.selCon.setModel(_graph);
89     this.graph.addObserver(this);
90 }
91 }

```

Listing 9: GraphParser

```

1 import java.io.IOException;

```

```

2
3 public abstract class GraphParser {
4     public abstract void saveGraph(String nameFile, GraphModel graph) throws
        IOException;
5     public abstract void loadFromFile(String nameFile, GraphModel graph)
        throws IOException;
6 }

```

Listing 10: GraphVertex

```

1 import java.awt.Point;
2 import java.awt.Rectangle;
3 import java.io.Serializable;
4
5 public class GraphVertex implements Serializable{
6
7     /*Name suppose to be unique*/
8     private String name;
9     private Rectangle rect;
10
11     /*Default constant*/
12     public static final String DEFAULT_NAME = "VERTEX";
13     public static final int DEFAULT_WIDTH = 100;
14     public static final int DEFAULT_HEIGHT = 75;
15
16     public GraphVertex(){
17         this.name = DEFAULT_NAME;
18         this.rect = new Rectangle(0, 0, DEFAULT_WIDTH, DEFAULT_HEIGHT);
19     }
20
21     public GraphVertex(String name){
22         this.name = name;
23         this.rect = new Rectangle(0, 0, DEFAULT_WIDTH, DEFAULT_HEIGHT);
24     }
25
26     public GraphVertex(String name, Rectangle rect){
27         this.name = name;
28         this.rect = rect;
29     }
30
31     public Boolean containsPressedPoint(Point p){
32         if (p.getX() > this.getRect().x && p.getX() < (this.getRect().x + this
            .getRect().getWidth())
33             && p.getY() > this.getRect().y && p.getY() < (this.getRect().y
                + this.getRect().getHeight())){
34             return true;
35         }
36         return false;
37     }
38
39     /*AUTOgenerate setters and getters*/
40     public String getName() {
41         return name;
42     }
43     public void setName(String name) {
44         this.name = name;
45     }
46     public Rectangle getRect() {
47         return rect;

```

```

48     }
49     public void setRect(Rectangle rect) {
50         this.rect = rect;
51     }
52 }

```

Listing 11: GraphVizGraphParser

```

1  import java.awt.Rectangle;
2  import java.io.BufferedReader;
3  import java.io.BufferedWriter;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.util.ArrayList;
8
9  public class GraphVizGraphParser extends GraphParser {
10
11      @Override
12      public void loadFromFile(String nameFile, GraphModel graph) throws
13          IOException {
14          // TODO Auto-generated method stub
15          BufferedReader loadFile = new BufferedReader(new FileReader(nameFile))
16          ;
17          String line = loadFile.readLine();
18          while (!(line = loadFile.readLine()).equals("")){
19              String[] vertexes = line.split("_--_");
20              String v1Name = vertexes[0];
21              v1Name = v1Name.replace("\t", "");
22
23              if (v1Name.contains(";")){
24                  addVertex(vertexes[0].replace(";", ""), graph);
25                  continue;
26              }
27
28              GraphVertex v1 = graph.getVertexOfName(addVertex(vertexes[0].
29                  replace("\t", ""), graph));
30              for (int i = 1; i < vertexes.length; i++){
31                  GraphVertex v2 = graph.getVertexOfName(addVertex(vertexes[i].
32                      replace(";", "").replace("\t", ""), graph));
33                  graph.addEdge(new GraphEdge(v1, v2));
34              }
35          }
36          loadFile.close();
37      }
38
39      @Override
40      public void saveGraph(String nameFile, GraphModel graph) throws
41          IOException {
42          // TODO Auto-generated method stub
43          BufferedWriter graphFile = new BufferedWriter(new FileWriter(nameFile)
44          );
45          graphFile.write("graph_GraphModel_{\n");
46
47          boolean[] edge_visited = new boolean[graph.getEdges().size()];
48          boolean[] vertex_visited = new boolean[graph.getVertexes().size()];
49
50          for (int i = 0; i < edge_visited.length; i++)

```

```

46     edge_visited[i] = false;
47     for (int i = 0; i < vertex_visited.length; i++)
48         vertex_visited[i] = false;
49
50     for (GraphVertex vertex : graph.getVertexes()){
51         boolean vertexFound = false;
52         ArrayList<GraphVertex> adjVertexes = graph.getAdjVertexes(vertex);
53         if (adjVertexes.isEmpty()){
54             vertex_visited[graph.getIndexOfVertex(vertex)] = true;
55             graphFile.write("\t"+
56                 vertex.getName()+"_["+x="+ vertex.getRect().x +
57                     ",y="+ vertex.getRect().y
58                         +",width="+ vertex.
59                             getRect().width+ ",
60                                 height="+ vertex.
61                                     getRect().height
62                                         + "]" + ";\n");
63             continue;
64         }
65         for (GraphVertex adjVertex : adjVertexes){
66             int indexEdge = graph.getIndexEdgeOfVertexes(vertex, adjVertex
67                 );
68             if (!edge_visited[indexEdge]){
69                 if (!vertexFound){
70                     graphFile.write("\t"+vertex.getName());
71                     if (!vertex_visited[graph.getIndexOfVertex(vertex)]){
72                         graphFile.write("_["+x="+ vertex.getRect().x + ",y="
73                             + vertex.getRect().y
74                                 +",width="+ vertex.getRect().
75                                     width+ ",height="+ vertex.
76                                         getRect().height + "]"");
77                         vertex_visited[graph.getIndexOfVertex(vertex)] =
78                             true;
79                     }
80                 }
81                 vertexFound = true;
82                 graphFile.write("_--_" + adjVertex.getName());
83                 if (!vertex_visited[graph.getIndexOfVertex(adjVertex)]){
84                     graphFile.write("_["+x="+ adjVertex.getRect().x + ",y="
85                         + adjVertex.getRect().y
86                             +",width="+ adjVertex.getRect().width
87                                 + ",height="+ adjVertex.getRect().
88                                     height
89                                         + "]"");
90                     vertex_visited[graph.getIndexOfVertex(adjVertex)] =
91                         true;
92                 }
93                 edge_visited[indexEdge] = true;
94             }
95         }
96         if (vertexFound) graphFile.write(";\n");
97     }
98
99     graphFile.write("{}");
100    graphFile.flush();
101    graphFile.close();
102 }

```



```

91
92     private String addVertex(String vertexString, GraphModel graph){
93         if (!vertexString.contains("[")){
94             if (!graph.containsVertex(vertexString)) {
95                 graph.addVertex(new GraphVertex(vertexString));
96             }
97             return vertexString;
98         }
99         vertexString = vertexString.replace("]", "");
100         String[] vertexInfo = vertexString.split("_\\[");
101         GraphVertex vertexToAdd = new GraphVertex(vertexInfo[0]);
102
103         vertexInfo = vertexInfo[1].split(",");
104         String x = "", y = "", width = "", height = "";
105         for (int i = 0; i < vertexInfo.length; i++){
106             String[] attribute = vertexInfo[i].split("=");
107             if (attribute[0].equals("x")){
108                 x = attribute[1];
109                 continue;
110             }
111             if (attribute[0].equals("y")){
112                 y = attribute[1];
113                 continue;
114             }
115             if (attribute[0].equals("width")){
116                 width = attribute[1];
117                 continue;
118             }
119             if (attribute[0].equals("height")){
120                 height = attribute[1];
121                 continue;
122             }
123         }
124         vertexToAdd.setRect(new Rectangle(x.equals("") ? 0 : Integer.parseInt(
125             x),
126             y.equals("") ? 0 : Integer.parseInt(
127                 y),
128             width.equals("") ? GraphVertex.
129                 DEFAULT_WIDTH : Integer.parseInt(
130                     width),
131             height.equals("") ? GraphVertex.
132                 DEFAULT_HEIGHT : Integer.
133                     parseInt(height))
134             );
135         graph.addVertex(vertexToAdd);
136         return vertexToAdd.getName();
137     }
138 }

```

Listing 12: NewGraphEdgeDialog

```

1  import java.awt.BorderLayout;
2  import java.awt.Dimension;
3  import java.awt.GridLayout;
4  import java.awt.Toolkit;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ItemEvent;
8  import java.awt.event.ItemListener;

```

```

9  import java.util.ArrayList;
10 import javax.swing.DefaultComboBoxModel;
11 import javax.swing.JButton;
12 import javax.swing.JComboBox;
13 import javax.swing.JDialog;
14 import javax.swing.JLabel;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17 import javax.swing.border.EmptyBorder;
18
19 public class NewGraphEdgeDialog extends JDialog {
20     final int WIDTH_WINDOW = 400;
21     final int HEIGHT_WINDOW = 400;
22
23     private JPanel centerPanel;
24     private JPanel southPanel;
25
26     private JComboBox<String> cmbV1;
27     private JComboBox<String> cmbV2;
28
29     private JButton btnSubmit;
30     private JButton btnExit;
31
32     private GraphModel graph;
33
34     public NewGraphEdgeDialog(GraphModel _graph) {
35         this.graph = _graph;
36         this.setWindow();
37         this.setPanel();
38         this.drawFields();
39
40         this.btnSubmit = new JButton("Add_edge");
41         this.btnSubmit.addActionListener(new ActionListener() {
42             @Override
43             public void actionPerformed(ActionEvent e) {
44                 NewGraphEdgeDialog.this.addEdge();
45                 NewGraphEdgeDialog.this.clearTextField();
46             }
47         });
48
49         this.btnExit = new JButton("Exit");
50         this.btnExit.addActionListener(new ActionListener() {
51             @Override
52             public void actionPerformed(ActionEvent e) {
53                 NewGraphEdgeDialog.this.dispose();
54             }
55         });
56
57         this.cmbV1.addItemListener(new ItemListener() {
58             public void itemStateChanged(ItemEvent e) {
59                 if (e.getStateChange() == ItemEvent.SELECTED) {
60                     NewGraphEdgeDialog.this.updateComboV2(NewGraphEdgeDialog.
61                         this.cmbV1.getSelectedItem().toString());
62                 }
63             }
64         });
65         this.southPanel.add(btnSubmit);

```

```

66         this.southPanel.add(btnExit);
67         this.add(centerPanel, BorderLayout.CENTER);
68         this.add(southPanel, BorderLayout.SOUTH);
69     }
70
71     private void addEdge() {
72         String nameV1 = this.cmbV1.getSelectedItem().toString();
73         if (nameV1.equals("")) {
74             String error = "First_vertex_is_not_seleceted!";
75             JOptionPane.showMessageDialog(null, error);
76             return;
77         }
78         String nameV2 = this.cmbV2.getSelectedItem() == null ? "" : this.cmbV2
            .getSelectedItem().toString();
79
80         if (!nameV2.equals("")) {
81             GraphVertex v1 = graph.getVertexOfName(nameV1);
82             GraphVertex v2 = graph.getVertexOfName(nameV2);
83             graph.perfromOperation(new Operation(Operation.OperationType.
                ADD_EDGE, new GraphEdge(v1, v2)));
84         } else {
85             //Generate string error
86             String error = "Second_vertex_is_not_seleceted!";
87             JOptionPane.showMessageDialog(null, error);
88         }
89     }
90
91     private void updateComboV2(String nameV1) {
92         this.cmbV2.setEnabled(true);
93         GraphVertex v1 = graph.getVertexOfName(nameV1);
94         DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
            String>();
95         ArrayList<GraphVertex> adjVertexes = graph.getAdjVertexes(v1);
96         for (GraphVertex vertex : graph.getVertexes()) {
97             if (!adjVertexes.contains(vertex) && !vertex.getName().equals(
                nameV1))
98                 vertexes.addElement(vertex.getName());
99         }
100         if (vertexes.getSize() == 0)
101             this.cmbV2.setEnabled(false);
102         this.cmbV2.setModel(vertexes);
103     }
104
105     private void clearTextField() {
106         DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
            String>();
107         for (GraphVertex vertex : graph.getVertexes()) {
108             if (this.graph.getAdjVertexes(vertex).size() != this.graph.
                getVertexes().size() - 1)
109                 vertexes.addElement(vertex.getName());
110         }
111
112         if (vertexes.getSize() == 0) {
113             vertexes.addElement("The_graph_is_connected");
114             this.cmbV1.setModel(vertexes);
115             this.cmbV2.setModel(new DefaultComboBoxModel<String>());
116             this.btnSubmit.setEnabled(false);
117             return;

```

```

118     }
119
120     this.cmbV1.setModel(vertexes);
121     this.cmbV1.setSelectedIndex(0);
122     this.updateComboV2(this.cmbV1.getSelectedItem().toString());
123 }
124
125 private void setPanel() {
126     this.centerPanel = new JPanel();
127     this.centerPanel.setLayout(new GridLayout(2, 2, 30, 30));
128     this.centerPanel.setBorder(new EmptyBorder(100, 30, 100, 30));
129     this.southPanel = new JPanel();
130     this.southPanel.setLayout(new GridLayout(0, 2, 30, 30));
131     this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
132 }
133
134 private void setWindow() {
135     this.setTitle("New_Edge");
136     this.setResizable(false);
137     this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
138     this.getContentPane().setLayout(new BorderLayout());
139     Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
140     this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
        .getSize().height/2);
141 }
142
143 private void drawFields() {
144     JLabel lblVertex1 = new JLabel("Vertex_1:");
145     this.centerPanel.add(lblVertex1);
146     this.cmbV1 = new JComboBox<String>();
147     this.centerPanel.add(this.cmbV1);
148
149     JLabel lblVertex2 = new JLabel("Vertex_2:");
150     this.centerPanel.add(lblVertex2);
151     this.cmbV2 = new JComboBox<String>();
152     this.centerPanel.add(this.cmbV2);
153     this.clearTextField();
154 }
155 }

```

Listing 13: NewGraphVertexDialog

```

1  import java.awt.BorderLayout;
2  import java.awt.Dimension;
3  import java.awt.GridLayout;
4  import java.awt.Rectangle;
5  import java.awt.Toolkit;
6  import java.awt.event.ActionEvent;
7  import java.awt.event.ActionListener;
8  import javax.swing.JButton;
9  import javax.swing.JDialog;
10 import javax.swing.JLabel;
11 import javax.swing.JOptionPane;
12 import javax.swing.JPanel;
13 import javax.swing.JTextField;
14 import javax.swing.border.EmptyBorder;
15
16 public class NewGraphVertexDialog extends JDialog {
17     final int WIDTH_WINDOW = 500;

```

```

18     final int HEIGHT_WINDOW = 400;
19
20     private JPanel centerPanel;
21     private JPanel southPanel;
22
23     private JTextField txtName;
24     private JTextField txtX;
25     private JTextField txtY;
26     private JTextField txtWidth;
27     private JTextField txtHeight;
28
29     private JButton btnSubmit;
30     private JButton btnExit;
31
32     private GraphModel graph;
33
34     public NewGraphVertexDialog(GraphModel _graph){
35         this.graph = _graph;
36         this.setWindow();
37         this.setPanel();
38         this.drawFields();
39
40         this.btnSubmit = new JButton("Add_vertex");
41         this.btnSubmit.addActionListener(new ActionListener() {
42             @Override
43             public void actionPerformed(ActionEvent e) {
44                 NewGraphVertexDialog.this.addVertex();
45                 NewGraphVertexDialog.this.clearTextField();
46             }
47         });
48
49         this.btnExit = new JButton("Exit");
50         this.btnExit.addActionListener(new ActionListener() {
51             @Override
52             public void actionPerformed(ActionEvent e) {
53                 NewGraphVertexDialog.this.dispose();
54             }
55         });
56
57         this.southPanel.add(btnSubmit);
58         this.southPanel.add(btnExit);
59         this.add(centerPanel, BorderLayout.CENTER);
60         this.add(southPanel, BorderLayout.SOUTH);
61     }
62
63     private void addVertex(){
64         String name = this.txtName.getText();
65         int height = this.txtHeight.getText().equals("") ? 0 : Integer.parseInt(
            this.txtHeight.getText());
66         int width = this.txtWidth.getText().equals("") ? 0 : Integer.parseInt(
            this.txtWidth.getText());
67         int x = this.txtX.getText().equals("") ? 0 : Integer.parseInt(this.
            txtX.getText());
68         int y = this.txtY.getText().equals("") ? 0 : Integer.parseInt(this.
            txtY.getText());
69
70         if (y >= 0 && y < GraphFrame.WINDOW_HEIGHT && x >= 0 && x < GraphFrame.
            WINDOW_WIDTH

```

```

71         && width >= 0 && height >= 0 && graph.getVertexOfName(name)
           == null){
72         width = width != 0 ? width : GraphVertex.DEFAULT_WIDTH;
73         height = height != 0 ? height : GraphVertex.DEFAULT_HEIGHT;
74         name = name.equals("") ? GraphVertex.DEFAULT_NAME : name;
75         graph.perfromOperation(new Operation(Operation.OperationType.
           ADD_VERTEX,
76
           new GraphVertex(name, new
               Rectangle(x, y, width,
               height))));
77     }else{
78         String error = "Invalid:_" + (x < 0 || x > GraphFrame.
           WINDOW_WIDTH ? "\nValue_of_X_has_to_be_between_0_and_"
79             + GraphFrame.WINDOW_WIDTH : "") +
80             (y < 0 || y > GraphFrame.WINDOW_HEIGHT ? "\nValue_
           of_Y_has_to_be_between_0_and_"
81             + GraphFrame.WINDOW_HEIGHT : "") +
82             (width < 0 ? "\nwidth_<_0" : "") +
83             (height < 0 ? "\nheight_<_0" : "") +
84             (graph.getIndexOfVertex(graph.getVertexOfName(name)
           )) < 0 ? "\nname_already_used" : "");
85         JOptionPane.showMessageDialog(null, error);
86     }
87 }
88
89 private void clearTextField(){
90     this.txtX.setText("0");
91     this.txtY.setText("0");
92     this.txtName.setText(GraphVertex.DEFAULT_NAME);
93     this.txtWidth.setText(""+GraphVertex.DEFAULT_WIDTH);
94     this.txtHeight.setText(""+GraphVertex.DEFAULT_HEIGHT);
95 }
96
97 private void setPanel(){
98     this.centerPanel = new JPanel();
99     this.centerPanel.setLayout(new GridLayout(5,2,30,30));
100    this.centerPanel.setBorder(new EmptyBorder(20, 30, 20, 30));
101    this.southPanel = new JPanel();
102    this.southPanel.setLayout(new GridLayout(0,2,30,30));
103    this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
104 }
105
106 private void setWindow(){
107     this.setTitle("New_Vertex");
108     this.setResizable(false);
109     this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
110     this.getContentPane().setLayout(new BorderLayout());
111     Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
112     this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
           .getSize().height/2);
113 }
114
115 private void drawFields(){
116     JLabel lblName = new JLabel("Vertex_Name:");
117     this.centerPanel.add(lblName);
118     this.txtName = new JTextField();
119     this.centerPanel.add(this.txtName);
120

```

```

121     JLabel lblX = new JLabel("Coordinate_X:");
122     this.centerPanel.add(lblX);
123     this.txtX = new JTextField();
124     this.centerPanel.add(this.txtX);
125
126     JLabel lblY = new JLabel("Coordinate_Y:");
127     this.centerPanel.add(lblY);
128     this.txtY = new JTextField();
129     this.centerPanel.add(this.txtY);
130
131     JLabel lblWidth = new JLabel("Vertex_Width:");
132     this.centerPanel.add(lblWidth);
133     this.txtWidth = new JTextField();
134     this.centerPanel.add(this.txtWidth);
135
136     JLabel lblHeight = new JLabel("Vertex_Height:");
137     this.centerPanel.add(lblHeight);
138     this.txtHeight = new JTextField();
139     this.centerPanel.add(this.txtHeight);
140
141     this.clearTextField();
142 }
143 }

```

Listing 14: Operation

```

1  import java.util.ArrayList;
2
3  public class Operation{
4
5      public enum OperationType {ADD_VERTEX, REMOVE_VERTEX, ADD_EDGE,
6          REMOVE_EDGE};
7
8      private OperationType operation;
9      private ArrayList<GraphEdge> edges;
10     private GraphVertex vertex;
11
12     public Operation(OperationType op, GraphVertex vertex){
13         this.operation = op;
14         this.edges = new ArrayList<GraphEdge>();
15         this.vertex = vertex;
16     }
17
18     public Operation(OperationType op, GraphEdge edge){
19         this.operation = op;
20         this.vertex = null;
21         this.edges = new ArrayList<GraphEdge>();
22         this.getEdges().add(edge);
23     }
24
25     public OperationType getOperation() {
26         return operation;
27     }
28
29     public void setOperation(OperationType operation) {
30         this.operation = operation;
31     }
32
33     public ArrayList<GraphEdge> getEdges() {
34         return edges;
35     }
36 }

```

```

33     public void setEdges(ArrayList<GraphEdge> edge) {
34         this.edges = edge;
35     }
36     public GraphVertex getVertex() {
37         return vertex;
38     }
39     public void setVertex(GraphVertex vertex) {
40         this.vertex = vertex;
41     }
42 }

```

Listing 15: RedoManager

```

1  import java.io.Serializable;
2
3  public class RedoManager extends AbstractUndoableEdit{
4
5      public RedoManager(GraphModel graphModel) {
6          super(graphModel);
7      }
8
9      public void redoOperation(){
10         Operation op = this.getStackOperation().pop();
11         this.getGraphModel().getUndoManager().addOperation(this.doOperation(op));
12     }
13
14     public void flushRedoStack(){
15         this.getStackOperation().removeAllElements();
16     }
17
18 }

```

Listing 16: SaveGraphDialog

```

1  import java.awt.BorderLayout;
2  import java.awt.Dimension;
3  import java.awt.GridLayout;
4  import java.awt.Rectangle;
5  import java.awt.Toolkit;
6  import java.awt.event.ActionEvent;
7  import java.awt.event.ActionListener;
8  import java.awt.event.ItemEvent;
9  import java.awt.event.ItemListener;
10 import java.io.FileInputStream;
11 import java.io.FileOutputStream;
12 import java.io.IOException;
13 import java.io.ObjectInputStream;
14 import java.io.ObjectOutputStream;
15 import java.util.ArrayList;
16 import java.util.Vector;
17
18 import javax.swing.DefaultComboBoxModel;
19 import javax.swing.JButton;
20 import javax.swing.JComboBox;
21 import javax.swing.JDialog;
22 import javax.swing.JLabel;
23 import javax.swing.JOptionPane;
24 import javax.swing.JPanel;

```



```

25 import javax.swing.JTextField;
26 import javax.swing.border.EmptyBorder;
27
28 public class SaveGraphDialog extends JDialog {
29     final int WIDTH_WINDOW = 400;
30     final int HEIGHT_WINDOW = 400;
31
32     private JPanel centerPanel;
33     private JPanel southPanel;
34
35     private JComboBox<String> cmbFormat;
36     private JTextField txtNameFile;
37
38     private JButton btnSubmit;
39     private JButton btnExit;
40
41     private GraphModel graph;
42
43     public SaveGraphDialog(GraphModel _graph) {
44         this.graph = _graph;
45         this.setWindow();
46         this.setPanel();
47         this.drawFields();
48
49         this.btnSubmit = new JButton("Save");
50         this.btnSubmit.addActionListener(new ActionListener() {
51             @Override
52             public void actionPerformed(ActionEvent e) {
53                 SaveGraphDialog.this.saveGraph();
54             }
55         });
56
57         this.btnExit = new JButton("Exit");
58         this.btnExit.addActionListener(new ActionListener() {
59             @Override
60             public void actionPerformed(ActionEvent e) {
61                 SaveGraphDialog.this.dispose();
62             }
63         });
64
65         this.southPanel.add(btnSubmit);
66         this.southPanel.add(btnExit);
67         this.add(centerPanel, BorderLayout.CENTER);
68         this.add(southPanel, BorderLayout.SOUTH);
69     }
70
71     private void saveGraph() {
72         String format = this.cmbFormat.getSelectedItem().toString();
73         String nameFile = this.txtNameFile.getText();
74         if (!nameFile.equals("")) {
75             try {
76                 if (format.equals("Standard")) {
77                     nameFile += ".txt";
78                     this.graph.saveGraph(nameFile, new StandardGraphParser());
79                 }
80                 if (format.equals("GraphViz")) {
81                     nameFile += ".dot";
82                     this.graph.saveGraph(nameFile, new GraphVizGraphParser());

```

```

83         }
84         if (format.equals("Serialized")){
85             nameFile += ".obj";
86             this.graph.serializeGraph(nameFile);
87         }
88         String msg = "Graph_is_saved!";
89         JOptionPane.showMessageDialog(null, msg);
90         this.dispose();
91     }catch (IOException e) {
92         String error = "Unable_to_save_the_graph!";
93         JOptionPane.showMessageDialog(null, error);
94         e.printStackTrace();
95     }
96 }
97 else{
98     String error = "File_name_is_empty!";
99     JOptionPane.showMessageDialog(null, error);
100 }
101 }
102
103 private void setPanel(){
104     this.centerPanel = new JPanel();
105     this.centerPanel.setLayout(new GridLayout(2,2,30,30));
106     this.centerPanel.setBorder(new EmptyBorder(100, 30, 100, 30));
107     this.southPanel = new JPanel();
108     this.southPanel.setLayout(new GridLayout(0,2,30,30));
109     this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
110 }
111
112 private void setWindow(){
113     this.setTitle("New_Vertex");
114     this.setResizable(false);
115     this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
116     this.getContentPane().setLayout(new BorderLayout());
117     Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
118     this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
        .getSize().height/2);
119 }
120
121 private void drawFields(){
122     JLabel lblVertex1 = new JLabel("File_name:");
123     this.centerPanel.add(lblVertex1);
124     this.txtNameFile = new JTextField("name_file");
125     this.centerPanel.add(this.txtNameFile);
126
127     JLabel lblVertex2 = new JLabel("Format:");
128     this.centerPanel.add(lblVertex2);
129     this.cmbFormat = new JComboBox<String>();
130     DefaultComboBoxModel<String> formats = new DefaultComboBoxModel<String>
        >();
131     formats.addElement("Standard");
132     formats.addElement("GraphViz");
133     formats.addElement("Serialized");
134     this.cmbFormat.setModel(formats);
135     this.centerPanel.add(this.cmbFormat);
136 }
137 }

```

Listing 17: SelectionController

```

1  import java.awt.Point;
2  import java.awt.event.MouseEvent;
3  import java.awt.event.MouseListener;
4  import java.awt.event.MouseMotionListener;
5
6  import javax.swing.JOptionPane;
7
8  public class SelectionController implements MouseListener, MouseMotionListener
9  {
10
11     private GraphVertex selectedVertex;
12     private GraphVertex draggedVertex;
13     private GraphModel model;
14     private GraphPanel panel;
15
16     private boolean moving = false;
17
18     SelectionController(GraphModel _model, GraphPanel _panel){
19         this.model = _model;
20         this.panel = _panel;
21     }
22
23     public int coordX(MouseEvent e) {
24         return e.getX();
25     }
26
27     public int coordY(MouseEvent e) {
28         return e.getY();
29     }
30
31     @Override
32     public void mouseClicked(MouseEvent e){
33         if (e.getClickCount() == 2) {
34             ChangeVertexNameDialog nameDialog = new ChangeVertexNameDialog(
35                 this.model, this.selectedVertex, this.panel);
36             nameDialog.setVisible(true);
37             return;
38         }
39
40         Point pressed = e.getPoint();
41         for (GraphVertex vertex : this.model.getVertexes()){
42             if (vertex.containsPressedPoint(pressed))
43                 if (this.selectedVertex == null || !this.selectedVertex.equals(
44                     vertex)){
45                     this.selectedVertex = vertex;
46                     this.model.setSelectedVertex(this.selectedVertex);
47                     this.panel.repaint();
48                     this.panel.paintRed(this.selectedVertex);
49                 }
50             else{
51                 this.selectedVertex = null;
52                 this.model.setSelectedVertex(this.selectedVertex);
53                 this.panel.repaint();
54             }
55         }
56     }
57 }

```

```

55     @Override
56     public void mousePressed(MouseEvent e) {
57     }
58
59     @Override
60     public void mouseDragged(MouseEvent e) {
61         Point pressed = e.getPoint();
62         double x = pressed.getX();
63         double y = pressed.getY();
64         for (GraphVertex vertex : this.model.getVertexes()) {
65             if (vertex.containsPressedPoint(pressed)) {
66                 if (this.selectedVertex != null && !this.selectedVertex.equals
67                     (vertex))
68                     continue;
69
70                 this.setMoving(true);
71                 this.draggedVertex = vertex;
72                 this.selectedVertex = vertex;
73                 this.model.setSelectedVertex(this.selectedVertex);
74
75                 this.draggedVertex.getRect().x = (int) ((int) x - (this.
76                     draggedVertex.getRect().getWidth()/2));
77                 this.draggedVertex.getRect().y = (int) ((int) y - (this.
78                     draggedVertex.getRect().getHeight()/2));
79                 this.panel.repaint();
80                 this.panel.paintRed(this.selectedVertex);
81                 return;
82             }
83         }
84         if (this.isMoving()) {
85             this.setMoving(false);
86             String error = "Too_fast!_The_mouse_lost_the_reference_to_the_
87                 vertex.";
88             JOptionPane.showMessageDialog(null, error);
89         }
90     }
91
92     @Override
93     public void mouseMoved(MouseEvent e) {
94         // TODO Auto-generated method stub
95     }
96
97     @Override
98     public void mouseEntered(MouseEvent arg0) {
99         // TODO Auto-generated method stub
100     }
101
102     @Override
103     public void mouseExited(MouseEvent arg0) {
104         // TODO Auto-generated method stub
105     }
106
107     @Override
108     public void mouseReleased(MouseEvent arg0) {
109         // TODO Auto-generated method stub
110     }
111
112     public boolean isMoving() {

```

```

109         return moving;
110     }
111     public void setMoving(boolean moving) {
112         this.moving = moving;
113     }
114
115     public void setModel(GraphModel _graph){
116         this.model = _graph;
117     }
118
119 }

```

Listing 18: StandardGraphParser

```

1  import java.awt.Rectangle;
2  import java.io.BufferedReader;
3  import java.io.BufferedWriter;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.io.IOException;
7
8  public class StandardGraphParser extends GraphParser {
9
10     @Override
11     public void saveGraph(String nameFile, GraphModel graph) throws
        IOException{
12         // TODO Auto-generated method stub
13         BufferedWriter graphFile = new BufferedWriter(new FileWriter(nameFile)
14             );
15         graphFile.write(graph.getVertexes().size()+"_"+graph.getEdges().size()
16             +"\n");
17         for (GraphVertex vertex : graph.getVertexes()){
18             graphFile.write((int) vertex.getRect().getX() + "_" + (int) vertex
19                 .getRect().getY() + "_" +
20                 (int) vertex.getRect().getHeight() + "_" + (int)
21                 vertex.getRect().getWidth() + "_" +
22                 vertex.getName() + "\n");
23         }
24         for (GraphEdge edge : graph.getEdges()){
25             graphFile.write(graph.getIndexOfVertex(edge.getV1()) + "_" + graph
26                 .getIndexOfVertex(edge.getV2())+"\n");
27         }
28         graphFile.flush();
29         graphFile.close();
30     }
31
32     @Override
33     public void loadFromFile(String nameFile, GraphModel graph) throws
        IOException{
34         // TODO Auto-generated method stub
35         BufferedReader loadFile = new BufferedReader(new FileReader(nameFile))
36             ;
37         String line = loadFile.readLine();
38         String[] infoLine = line.split("_");
39         int sizeVertexes = Integer.parseInt(infoLine[0]);
40         int sizeEdges = Integer.parseInt(infoLine[1]);
41
42         for (int i = 0; i < sizeVertexes; i++){
43             line = loadFile.readLine();

```

```

38         infoLine = line.split("_");
39         String nameVertex = "";
40         for (int j = 4; j < infoLine.length; j++){
41             if (j != 4) nameVertex += "_";
42             nameVertex += infoLine[j];
43         }
44         graph.addVertex(new GraphVertex(nameVertex,
45                                     new Rectangle(Integer.parseInt(
46                                                         infoLine[0]),
47                                                         Integer.parseInt(
48                                                         infoLine[1]),
49                                                         Integer.parseInt(
50                                                         infoLine[3]),
51                                                         Integer.parseInt(
52                                                         infoLine[2]))))
53         );
54     }
55     for (int i = 0; i < sizeEdges; i++){
56         line = loadFile.readLine();
57         infoLine = line.split("_");
58         graph.addEdge(new GraphEdge(graph.getVertexes().get(Integer.
59                                     parseInt(infoLine[0])),
60                                     graph.getVertexes().get(Integer.
61                                     parseInt(infoLine[1]))));
62     }
63     loadFile.close();
64 }
65 }

```

Listing 19: UndoManager

```

1  import java.io.Serializable;
2
3  public class UndoManager extends AbstractUndoableEdit {
4
5      public UndoManager(GraphModel graphModel) {
6          super(graphModel);
7      }
8
9      public void undoOperation(){
10         Operation op = this.getStackOperation().pop();
11         this.getGraphModel().getRedoManager().addOperation(this.doOperation(op));
12     }
13
14 }

```