Object Oriented Programming
Programming report
Final Assignment: Graph Editor

Corradini Matteo Berke Atac
S3051390 S3075168

June 15, 2016

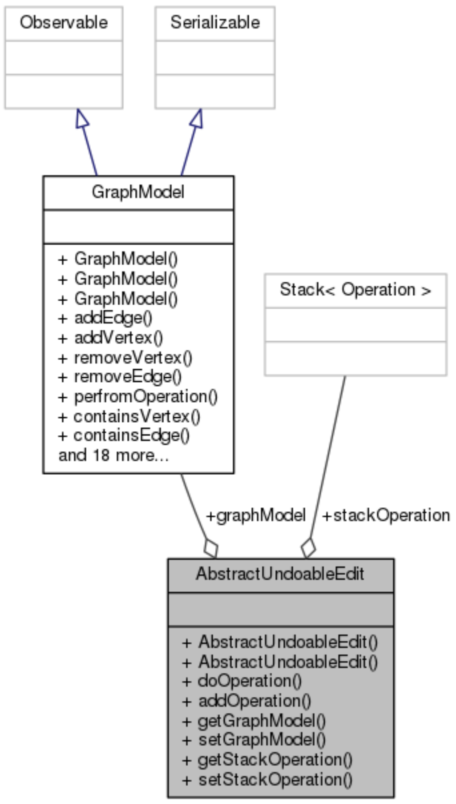
1 Problem description

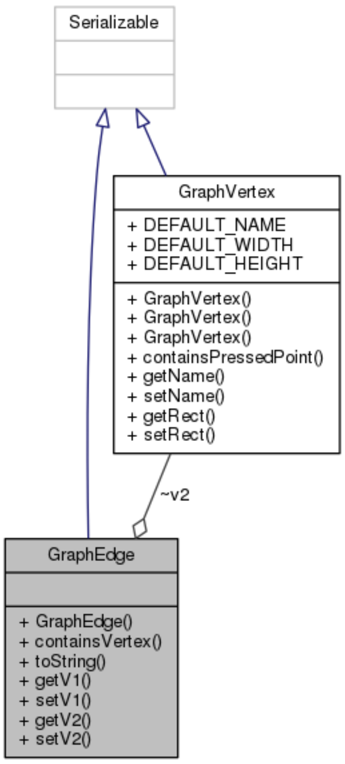
The problem was to create a graphical editor program with which the user would be able to create diagrams. The diagram consists of vertex which vary in their position, shape and names; and the edges in between them where all the elements can be added or removed by the user. We had to implement an user 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.).

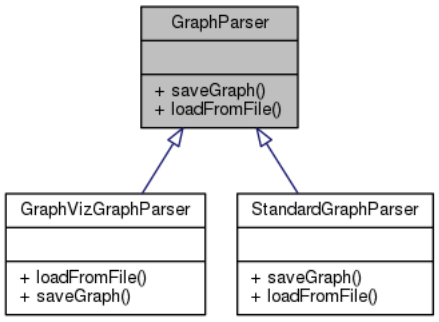
2 Problem analysis

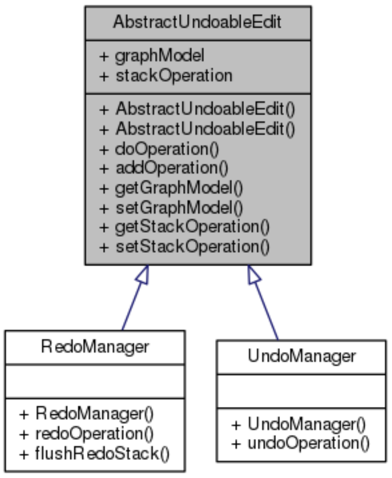
The structure of `GraphModel` was clear for the beginning: a collections of `GraphVertex` and another one of `GraphEdge`, and methods to provide these information with different requests.
More difficult was to implement an `UndoManager` and a `RedoManager`. We can think about the operation as group of 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: addiction 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


Observable
Serializable
GraphModel
+ GraphModel()
+ GraphModel()
+ GraphModel()
+ addEdge()
+ addVertex()
+ removeVertex()
+ removeEdge()
+ perfromOperation()
+ containsVertex()
+ containsEdge()
and 18 more...
Stack< Operation >
+graphModel
+stackOperation
AbstractUndoableEdit
+ AbstractUndoableEdit()
+ AbstractUndoableEdit()
+ doOperation()
+ addOperation()
+ getGraphModel()
+ setGraphModel()
+ getStackOperation()
+ setStackOperation()


Serializable
GraphVertex
+ DEFAULT_NAME
+ DEFAULT_WIDTH
+ DEFAULT_HEIGHT
+ GraphVertex()
+ GraphVertex()
+ GraphVertex()
+ containsPressedPoint()
+ getName()
+ setName()
+ getRect()
+ setRect()
~v2
GraphEdge
+ GraphEdge()
+ containsVertex()
+ toString()
+ getV1()
+ setV1()
+ getV2()
+ setV2()





`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 `tranient`, 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).

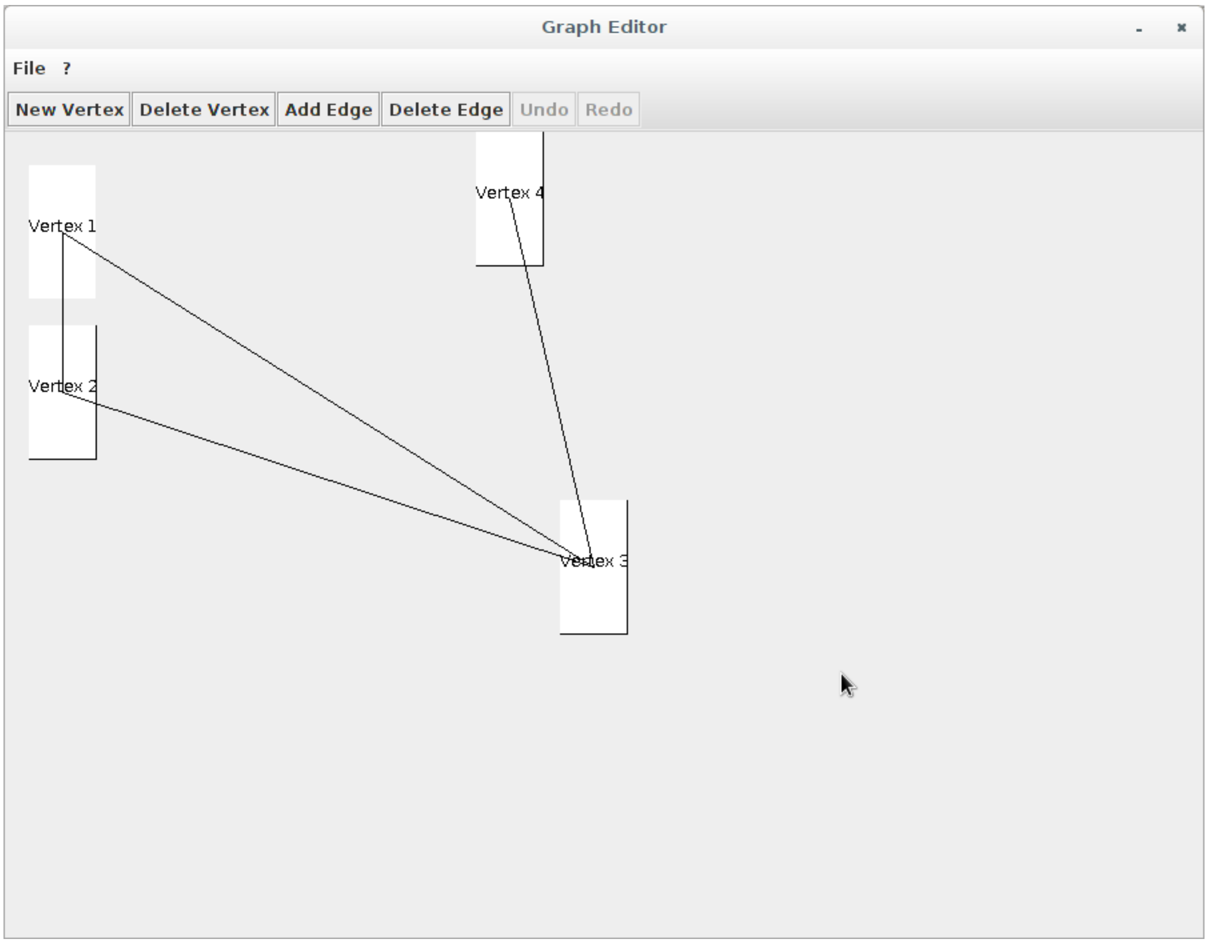
In `GraphFrame.java` class we created the user interface with all the required features such as the menubar and the toolbar using the Swing library. We created methods that provide the event routines for the keyboard shortcut and the click of the button. Furthermore, it also managed the menu, that provides 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 Oberserver of `GraphModel`, and it is responsible to provide the update of the data structure.

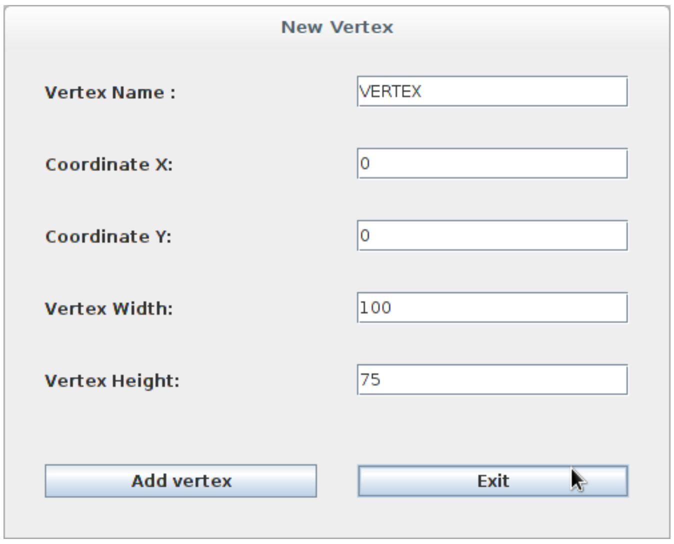
`SelectionController.java` class extends the classes `MouseListener` and `MouseMotionListener`. By overloading the methods, it is possible to track mouse movements on the panel and perform actions accordingly. Double clicking to change the name of a vertex, dragging a vertex to change its position and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

4 Evaluation of the program

We obtain the following windows.



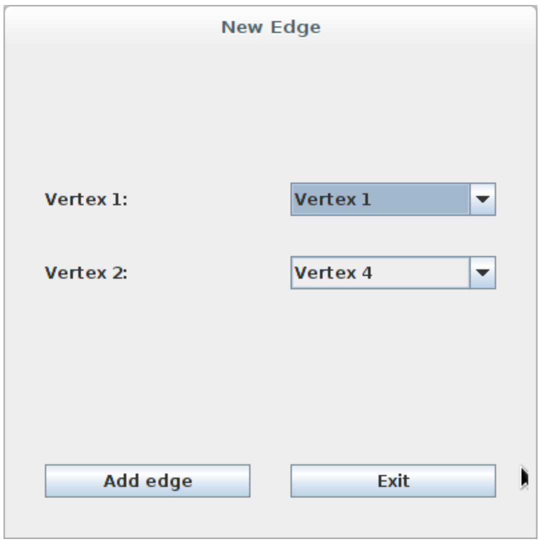
This is the main window that a user comes across when the program is run. The application opens up and adds the vertex and edges according to the file containing the data about them. It contains a menubar where operations on File, Edit and Window can be performed. Furthermore it supports a toolbar where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



New Vertex

Vertex Name :	VERTEX
Coordinate X:	0
Coordinate Y:	0
Vertex Width:	100
Vertex Height:	75

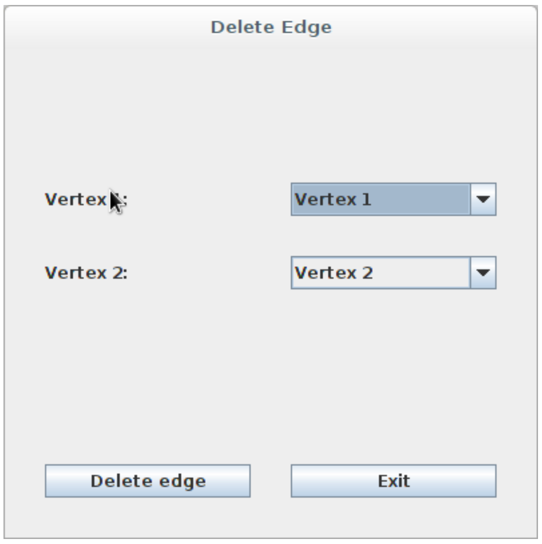
This is the window where it is possible to add a new vertex. The user enters the name of the vertex, upper-left point in regards to x and y coordinates, the width and the height of the vertex. Clicking the ”Add Vertex” button adds the vertex to the diagram.



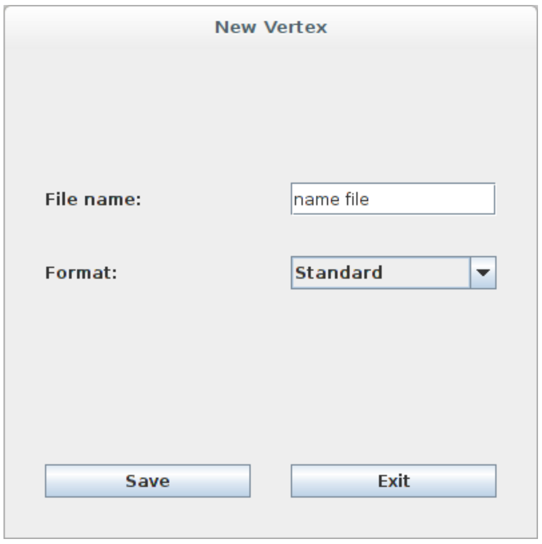
New Edge

Vertex 1:	Vertex 1
Vertex 2:	Vertex 4

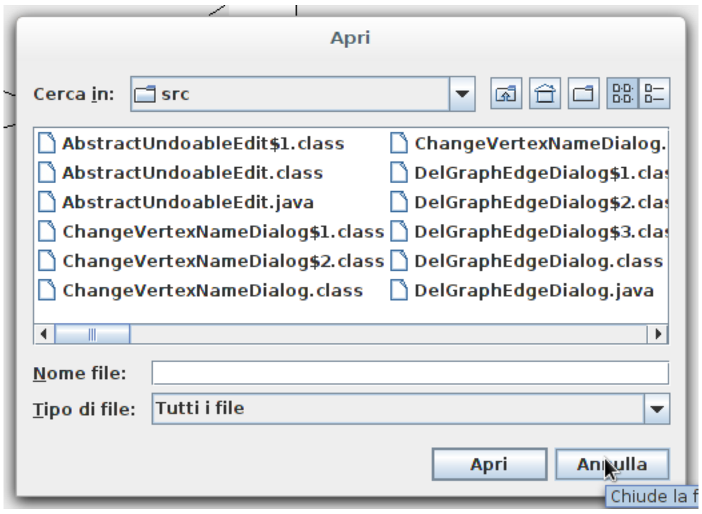
This is the window where it is possible to add a new edge between two vertices. Available vertices that are not connected with the first one can be chosen from the dropdown menu, and a new edge between the two chosen vertices will be created. If the operation can not be performed an error dialog will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



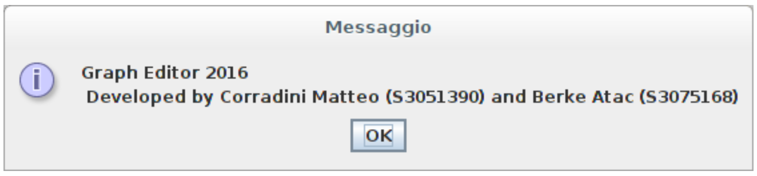
This is the window where it is possible to delete an edge between two vertices. The dropdown menu shows the first vertex, and the second dropdown menu is updated when the first one is selected. Furthermore the program is able to know when there are not edge.



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.

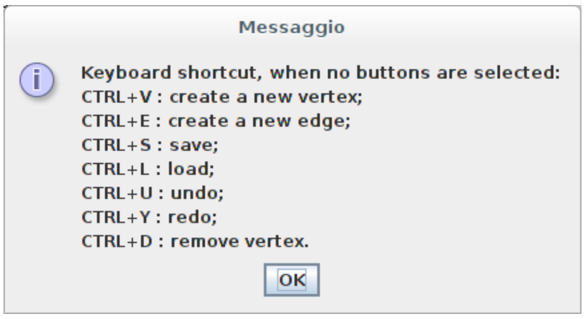


This is the window where it is possible to load a graph from a file, choosing a file from the filesystem.



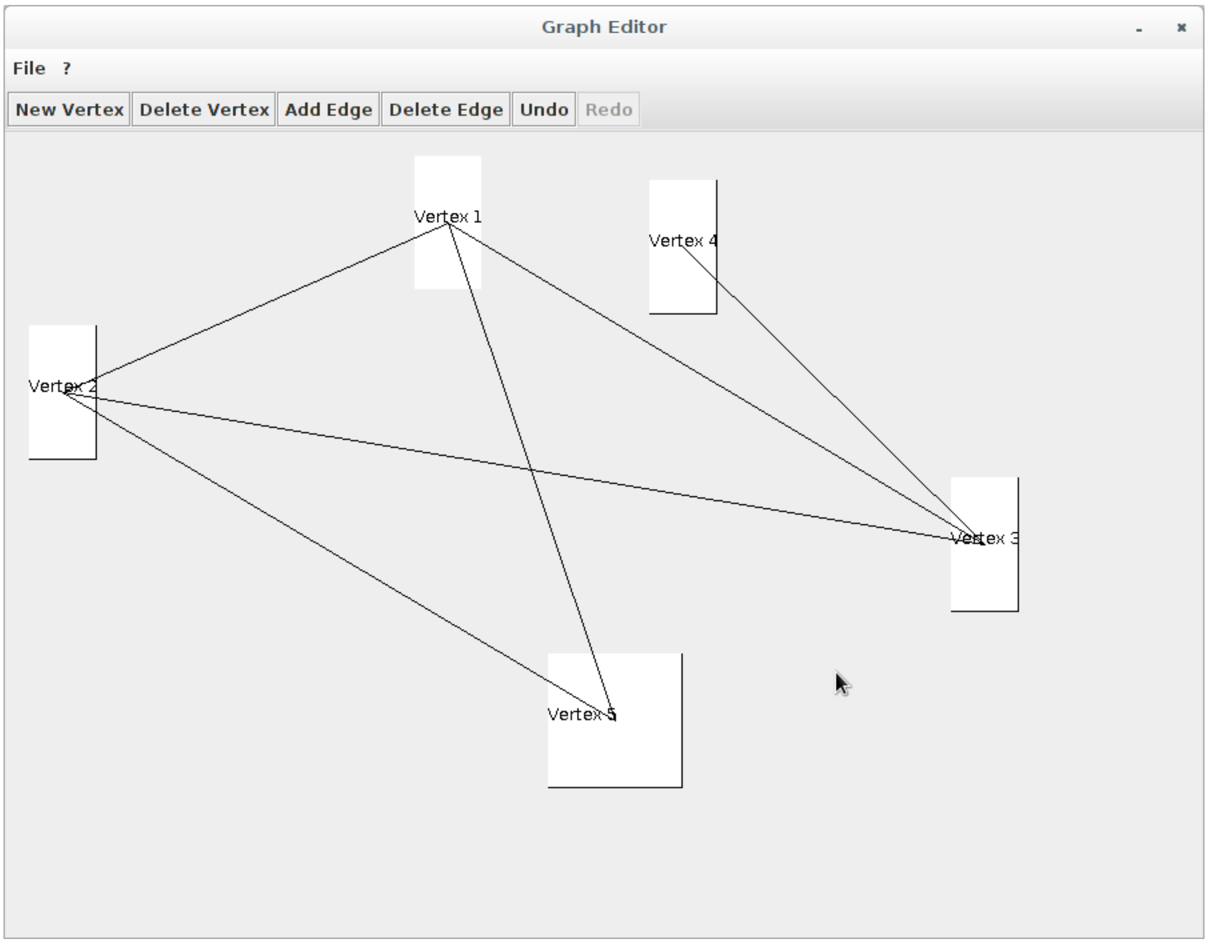
Graph Editor 2016
Developed by Corradini Matteo (S3051390) and Berke Atac (S3075168)

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



Keyboard shortcut, when no buttons are selected:
CTRL+V : create a new vertex;
CTRL+E : create a new edge;
CTRL+S : save;
CTRL+L : load;
CTRL+U : undo;
CTRL+Y : redo;
CTRL+D : remove vertex.

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`

```java
import java.util.Stack;

public abstract class AbstractUndoableEdit {

    public GraphModel graphModel;
    public Stack<Operation> stackOperation;

    public AbstractUndoableEdit(){
        this.graphModel = null;
        this.stackOperation = new Stack<Operation>();
    }

    public AbstractUndoableEdit(GraphModel graphModel){
        this.graphModel = graphModel;
        this.stackOperation = new Stack<Operation>();
    }

    public Operation doOperation(Operation op){
        switch (op.getOperation()){
            case ADD_VERTEX:{
                /*Remove added vertex*/
                this.getGraphModel().removeVertex(op.getVertex());
                op.setOperation(Operation.OperationType.REMOVE_VERTEX);
                break;
            }
            case REMOVE_VERTEX:{
                /*Add removed vertex*/
                this.getGraphModel().addVertex(op.getVertex());
                for (GraphEdge egde : op.getEdges()){
                    this.getGraphModel().addEdge(egde);
                }
                op.setOperation(Operation.OperationType.ADD_VERTEX);
                break;
            }
            case ADD_EDGE:{
                /*Remove added edge*/
                this.getGraphModel().removeEdge(op.getEdges().get(0));
                op.setOperation(Operation.OperationType.REMOVE_EDGE);
                break;
            }
            case REMOVE_EDGE:{
                /*Add removed e*/
                this.getGraphModel().addEdge(op.getEdges().get(0));
                op.setOperation(Operation.OperationType.ADD_EDGE);
                break;
            }
        }

        return op;
    }

    public void addOperation(Operation op) {
        // TODO Auto-generated method stub
        this.getStackOperation().push(op);
    }

    /*AUTOgenerate setters and getters*/
    public GraphModel getGraphModel() {
        return graphModel;
    }
    public void setGraphModel(GraphModel graphModel) {
        this.graphModel = graphModel;
    }
    public Stack<Operation> getStackOperation() {
        return stackOperation;
    }
    public void setStackOperation(Stack<Operation> stackOperation) {
        this.stackOperation = stackOperation;
    }

}
```

Listing 2: ChangeVertexNameDialog

```java
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;


public class ChangeVertexNameDialog extends JDialog{
    final int WIDTH_WINDOW = 500;
    final int HEIGHT_WINDOW = 400;

    private JPanel centerPanel;
    private JPanel southPanel;

    private JTextField txtName;

    private JButton btnSubmit;
    private JButton btnExit;

    private GraphModel graph;
    private GraphVertex vertex;
    private GraphPanel panel;

    private String oldName;

    public ChangeVertexNameDialog(GraphModel _graph, GraphVertex _vertex,
        GraphPanel _panel){
        this.panel = _panel;
        this.graph = _graph;
        this.vertex = _vertex;
        this.oldName = _vertex.getName();

        this.setWindow();
        this.setPanel();
        this.drawFields();

        this.btnSubmit = new JButton("Change␣Name");
        this.btnSubmit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String nameVertex = txtName.getText().equals("") ? GraphVertex
                    .DEFAULT_NAME : txtName.getText();
                if (!ChangeVertexNameDialog.this.graph.containsVertex(
                    nameVertex))
                    ChangeVertexNameDialog.this.vertex.setName(nameVertex);
                else{
                    String error = "Name␣already␣used!";
                    JOptionPane.showMessageDialog(null, error);
                }
                ChangeVertexNameDialog.this.dispose();
                panel.repaint();
            }
        });

        this.btnExit = new JButton("Exit");
        this.btnExit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                ChangeVertexNameDialog.this.dispose();
            }
        });

        this.southPanel.add(btnSubmit);
        this.southPanel.add(btnExit);
        this.add(centerPanel, BorderLayout.CENTER);
        this.add(southPanel, BorderLayout.SOUTH);
    }


    private void clearTextField(){
        this.txtName.setText(this.oldName);
    }

    private void setPanel(){
        this.centerPanel = new JPanel();
        this.centerPanel.setLayout(new GridLayout(5,2,30,30));
        this.centerPanel.setBorder(new EmptyBorder(20, 30, 20, 30));
        this.southPanel = new JPanel();
        this.southPanel.setLayout(new GridLayout(0,2,30,30));
        this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
    }

    private void setWindow(){
        this.setTitle("New␣Vertex");
        this.setResizable(false);
        this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
        this.getContentPane().setLayout(new BorderLayout());
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
    }

    private void drawFields(){
        JLabel lblName = new JLabel("Change␣Vertex␣Name␣:");
        this.centerPanel.add(lblName);
        this.txtName = new JTextField();
        this.centerPanel.add(this.txtName);

        this.clearTextField();
    }
}
```

Listing 3: DelGraphEdgeDialog

```java
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.ArrayList;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class DelGraphEdgeDialog extends JDialog {
    final int WIDTH_WINDOW = 400;
    final int HEIGHT_WINDOW = 400;

    private JPanel centerPanel;
    private JPanel southPanel;

    private JComboBox<String> cmbV1;
    private JComboBox<String> cmbV2;

    private JButton btnSubmit;
    private JButton btnExit;

    private GraphModel graph;

    public DelGraphEdgeDialog(GraphModel _graph){
        this.graph = _graph;
        this.setWindow();
        this.setPanel();
        this.drawFields();

        this.btnSubmit = new JButton("Delete␣edge");
        this.btnSubmit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                DelGraphEdgeDialog.this.removeEdge();
                DelGraphEdgeDialog.this.clearTextField();
            }
        });

        this.btnExit = new JButton("Exit");
        this.btnExit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                DelGraphEdgeDialog.this.dispose();
            }
        });

        this.cmbV1.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                if(e.getStateChange() == ItemEvent.SELECTED) {
                    DelGraphEdgeDialog.this.updateComboV2(DelGraphEdgeDialog.
                        this.cmbV1.getSelectedItem().toString());
                }
            }
        });

        this.southPanel.add(btnSubmit);
        this.southPanel.add(btnExit);
        this.add(centerPanel, BorderLayout.CENTER);
        this.add(southPanel, BorderLayout.SOUTH);
    }

    private void removeEdge(){
        String nameV1 = this.cmbV1.getSelectedItem().toString();
        if (nameV1.equals("")){
            String error = "First␣vertex␣is␣not␣seleceted!";
            JOptionPane.showMessageDialog(null, error);
            return;
        }
        String nameV2 = this.cmbV2.getSelectedItem() == null ? "" : this.cmbV2
            .getSelectedItem().toString();

        if (!nameV2.equals("")){
            GraphVertex v1 = graph.getVertexOfName(nameV1);
            GraphVertex v2 = graph.getVertexOfName(nameV2);
            graph.perfromOperation(new Operation(Operation.OperationType.
                REMOVE_EDGE,
                                                 this.graph.getEdges().get(this
                                                     .graph.
                                                     getIndexEdgeOfVertexes(v1,
                                                     v2))));
        }else{
            //Generate string error
            String error = "Second␣vertex␣is␣not␣seleceted!";
            JOptionPane.showMessageDialog(null, error);
        }
    }

    private void updateComboV2(String nameV1){
        this.cmbV2.setEnabled(true);
        GraphVertex v1 = graph.getVertexOfName(nameV1);
        DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
            String>();
        ArrayList<GraphVertex> adjVertexes = graph.getAdjVertexes(v1);
        for (GraphVertex vertex : graph.getVertexes()){
            if (adjVertexes.contains(vertex))
                vertexes.addElement(vertex.getName());
        }
        if (vertexes.getSize() == 0)
            this.cmbV2.setEnabled(false);
        this.cmbV2.setModel(vertexes);
    }

    private void clearTextField(){
        DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
            String>();
        for (GraphVertex vertex : graph.getVertexes()){
            if (this.graph.getAdjVertexes(vertex).size() != 0)
                vertexes.addElement(vertex.getName());
        }

        if (vertexes.getSize() == 0){
            vertexes.addElement("There␣are␣not␣edges");
            this.cmbV1.setModel(vertexes);
            this.cmbV2.setModel(new DefaultComboBoxModel<String>());
            this.btnSubmit.setEnabled(false);
            return;
        }

        this.cmbV1.setModel(vertexes);
        this.cmbV1.setSelectedIndex(0);
        this.updateComboV2(this.cmbV1.getSelectedItem().toString());
    }

    private void setPanel(){
        this.centerPanel = new JPanel();
        this.centerPanel.setLayout(new GridLayout(2,2,30,30));
        this.centerPanel.setBorder(new EmptyBorder(100, 30, 100, 30));
        this.southPanel = new JPanel();
        this.southPanel.setLayout(new GridLayout(0,2,30,30));
        this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
    }

    private void setWindow(){
        this.setTitle("Delete␣Edge");
        this.setResizable(false);
        this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
        this.getContentPane().setLayout(new BorderLayout());
```

```java
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
    }

    private void drawFields(){
        JLabel lblVertex1 = new JLabel("Vertex␣1:");
        this.centerPanel.add(lblVertex1);
        this.cmbV1 = new JComboBox<String>();
        this.centerPanel.add(this.cmbV1);

        JLabel lblVertex2 = new JLabel("Vertex␣2:");
        this.centerPanel.add(lblVertex2);
        this.cmbV2 = new JComboBox<String>();
        this.centerPanel.add(this.cmbV2);
        this.clearTextField();
    }
}
```

Listing 4: `GraphEdge`

```java
import java.io.Serializable;

public class GraphEdge implements Serializable{

    private GraphVertex v1, v2;

    public GraphEdge(GraphVertex v1, GraphVertex v2){
        this.v1 = v1;
        this.v2 = v2;
    }

    public boolean containsVertex(GraphVertex vertex){
        if (v1.getName().equals(vertex.getName()) ||
            v2.getName().equals(vertex.getName())) return true;

        return false;
    }

    @Override
    public String toString(){
        return this.getV1().getName()+"--"+this.getV2().getName();
    }

    /*AUTOgenerate setters and getters*/
    public GraphVertex getV1() {
        return v1;
    }
    public void setV1(GraphVertex v1) {
        this.v1 = v1;
    }
    public GraphVertex getV2() {
        return v2;
    }
    public void setV2(GraphVertex v2) {
        this.v2 = v2;
    }

}
```

Listing 5: `GraphEdit`

```java
import java.io.IOException;
import java.util.Observable;
import java.util.Observer;

public class GraphEdit{

    public static void main(String[] argv){
        GraphModel graph = null;
        GraphFrame graphFrame;
        try {
            if (argv.length == 1){
                String[] file = argv[argv.length-1].split("\\.");

                if (file[1].equals("txt"))
                    graph = new GraphModel(argv[argv.length-1], new
                        StandardGraphParser());
                if (file[1].equals("dot"))
                    graph = new GraphModel(argv[argv.length-1], new
                        GraphVizGraphParser());
                if (file[1].equals("obj")){
                    graph = new GraphModel();
                    graph.deSerializeGraph(argv[argv.length-1]);
                }
            }else
                graph = new GraphModel();

            graphFrame = new GraphFrame(graph);
            graphFrame.setVisible(true);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            graph = null;
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

Listing 6: `GraphFrame`

```java
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.io.IOException;
import java.util.Observable;
import java.util.Observer;

import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class GraphFrame extends JFrame{

    public static final int WINDOW_WIDTH = 900;
    public static final int WINDOW_HEIGHT = 700;
    private static final String TITLE_NAME = "Graph␣Editor";

    private JPanel northPanel;
    private JPanel centerPanel;

    private JMenuBar menuBar;
    private JMenu menuFile, menuInfo;
    private JMenuItem mnSave, mnLoad, mnCredits, mnTips;
    private JToolBar tb;
    private JButton btnNewVertex, btnDelNode, btnNewEdge, btnDelEdge, undo,
        redo;
    private GraphPanel graphPanel;

    private GraphModel model;

    public GraphFrame(GraphModel _graph){
        this.setModel(_graph);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.setWindowProperites();
        this.createMenu();
        this.createToolbar();
        this.addPanel();

        this.btnNewVertex.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphFrame.this.addVertex();
            }
        });
        this.btnDelNode.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphVertex vertexToDelete = GraphFrame.this.model.
                    getSelectedVertex();
                if (vertexToDelete != null)
                    GraphFrame.this.model.perfromOperation(new Operation(
                        Operation.OperationType.REMOVE_VERTEX,
                                                    vertexToDelete));
                else{
                    String error = "Select␣a␣vertex!";
                    JOptionPane.showMessageDialog(null, error);
                }
                GraphFrame.this.undo.setEnabled(true);
                GraphFrame.this.redo.setEnabled(false);
                GraphFrame.this.graphPanel.repaint();
            }
        });

        this.btnDelEdge.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //Remove edge
                GraphFrame.this.deleteEdge();
                GraphFrame.this.graphPanel.repaint();
            }});
        this.btnNewEdge.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphFrame.this.addEdge();
                GraphFrame.this.graphPanel.repaint();
            }
        });
        this.undo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphFrame.this.model.getUndoManager().undoOperation();
                GraphFrame.this.redo.setEnabled(true);
                if (GraphFrame.this.model.getUndoManager().stackOperation.isEmpty())
                    GraphFrame.this.undo.setEnabled(false);
            }
        });
        this.redo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphFrame.this.model.getRedoManager().redoOperation();
                GraphFrame.this.undo.setEnabled(true);
                if (GraphFrame.this.model.getRedoManager().stackOperation.isEmpty())
                    GraphFrame.this.redo.setEnabled(false);
            }
        });

        this.mnSave.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphFrame.this.saveGraph();
            }
        });

        this.mnLoad.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GraphFrame.this.loadGraph();
            }
        });

        this.mnCredits.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String msg = "Graph␣Editor␣2016␣\n␣Developed␣by␣Corradini␣Matteo␣(S3051390)␣and␣"
                                + "Berke␣Atac␣(S3075168)";
                JOptionPane.showMessageDialog(null, msg);
            }
        });
        this.mnTips.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String msg = "Keyboard␣shortcut,␣when␣no␣buttons␣are␣selected:␣\n"
                            + "CTRL+V␣:␣create␣a␣new␣vertex;\n"
                            + "CTRL+E␣:␣create␣a␣new␣edge;\n"
                            + "CTRL+S␣:␣save;\n"
                            + "CTRL+L␣:␣load;\n"
                            + "CTRL+U␣:␣undo;\n"
                            + "CTRL+Y␣:␣redo;\n"
                            + "CTRL+D␣:␣remove␣vertex.\n";
                JOptionPane.showMessageDialog(null, msg);
            }
        });

        this.graphPanel.addKeyListener(new KeyListener() {
            @Override
            public void keyPressed(KeyEvent e) {
                if ((e.getKeyCode() == KeyEvent.VK_V) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    GraphFrame.this.addVertex();
                    return;
                }
                if ((e.getKeyCode() == KeyEvent.VK_E) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    GraphFrame.this.addEdge();
                    return;
                }
                if ((e.getKeyCode() == KeyEvent.VK_S) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    GraphFrame.this.saveGraph();
                    return;
                }
                if ((e.getKeyCode() == KeyEvent.VK_L) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    GraphFrame.this.loadGraph();
                    return;
                }
                if ((e.getKeyCode() == KeyEvent.VK_U) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    if (!GraphFrame.this.model.getUndoManager().getStackOperation().isEmpty()){
                        GraphFrame.this.redo.setEnabled(true);
                        if (GraphFrame.this.model.getUndoManager().stackOperation.isEmpty())
                            GraphFrame.this.undo.setEnabled(false);
                    }
                    return;
                }
                if ((e.getKeyCode() == KeyEvent.VK_Y) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    if (!GraphFrame.this.model.getRedoManager().getStackOperation().isEmpty()){
                        GraphFrame.this.model.getRedoManager().redoOperation();
                        GraphFrame.this.model.getUndoManager().undoOperation()
```

```
                            ;
                        GraphFrame.this.undo.setEnabled(true);
                        if (GraphFrame.this.model.getRedoManager().stackOperation.isEmpty())
                            GraphFrame.this.redo.setEnabled(false);
                    }
                }
                if ((e.getKeyCode() == KeyEvent.VK_D) && ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0)) {
                    GraphVertex vertexToDelete = GraphFrame.this.model.getSelectedVertex();
                    if (vertexToDelete != null){
                        GraphFrame.this.model.perfromOperation(new Operation(Operation.OperationType.REMOVE_VERTEX,
                                                                                vertexToDelete));
                        GraphFrame.this.undo.setEnabled(true);
                        GraphFrame.this.redo.setEnabled(false);
                        GraphFrame.this.graphPanel.repaint();
                    }
                    return;
                }
            }
            @Override
            public void keyReleased(KeyEvent e) {
                // TODO Auto-generated method stub
            }
            @Override
            public void keyTyped(KeyEvent e) {
                // TODO Auto-generated method stub
            }
        });

        this.drawWholeFrame();
    }

    private void setWindowProperites(){
        this.setTitle(TITLE_NAME);
        this.setResizable(false);
        this.setSize(new Dimension(WINDOW_WIDTH, WINDOW_HEIGHT));
        this.getContentPane().setLayout(new BorderLayout());
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this.getSize().height/2);
    }

    private void createMenu(){
        this.menuBar = new JMenuBar();

        //add first headings
        this.menuFile = new JMenu("File");
        this.menuInfo = new JMenu("?");

        this.mnSave = new JMenuItem("Save Graph");
        this.mnLoad = new JMenuItem("Load Graph");
        this.mnCredits = new JMenuItem("Credits");
        this.mnTips = new JMenuItem("Tips");

        this.menuFile.add(this.mnSave);
        this.menuFile.add(this.mnLoad);

        this.menuInfo.add(this.mnCredits);
        this.menuInfo.add(this.mnTips);

        this.menuBar.add(menuFile);
        this.menuBar.add(menuInfo);
    }

    private void addPanel(){
        this.graphPanel = new GraphPanel(this.model);
        this.northPanel = new JPanel (new GridLayout(2, 0, 0, 0));
        this.centerPanel = new JPanel (new BorderLayout());

        this.northPanel.add(this.menuBar);
        this.northPanel.add(this.tb);
        this.centerPanel.add(graphPanel);
    }

    private void createToolbar(){
        this.tb = new JToolBar();
        tb.setFloatable(false);

        this.btnNewVertex = new JButton("New␣Vertex");
        this.btnDelNode = new JButton("Delete␣Vertex");
        this.btnNewEdge = new JButton("Add␣Edge");
        this.btnDelEdge = new JButton("Delete␣Edge");
        this.undo = new JButton("Undo");
        this.redo = new JButton("Redo");

        this.getUndo().setEnabled(false);
        this.getRedo().setEnabled(false);
        this.btnDelEdge.setEnabled(this.model.getEdges().size() > 0);

        this.getTb().add(this.getBtnNewVertex());
        this.getTb().add(this.getBtnDelNode());
        this.getTb().add(this.getBtnNewEdge());
        this.getTb().add(this.getBtnDelEdge());
        this.getTb().add(this.getUndo());
        this.getTb().add(this.getRedo());
    }

    private void drawWholeFrame(){
        this.add(this.northPanel, BorderLayout.NORTH);
        this.add(this.centerPanel, BorderLayout.CENTER);
    }

    private void saveGraph(){
        SaveGraphDialog saveDialog = new SaveGraphDialog(this.getModel());
        saveDialog.setVisible(true);
    }

    private void loadGraph(){
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setCurrentDirectory(new File(System.getProperty("user.dir"
            )));
        int result = fileChooser.showOpenDialog(this);
```

```java
        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            String[] nameFile = selectedFile.getName().split("\\.");
            try {
                boolean fileCorrect = false;
                if (nameFile[1].equals("txt")){
                    this.model = new GraphModel(selectedFile.getAbsolutePath(), new StandardGraphParser());
                    fileCorrect = true;
                }

                if (nameFile[1].equals("dot")){
                    this.model = new GraphModel(selectedFile.getAbsolutePath(), new GraphVizGraphParser());
                    fileCorrect = true;
                }

                if (nameFile[1].equals("obj")){
                    this.model.deSerializeGraph(selectedFile.getAbsolutePath());
                    fileCorrect = true;
                }

                if (!fileCorrect){
                    String error = "Select␣.txt,␣.dot␣or␣.obj␣file!";
                    JOptionPane.showMessageDialog(null, error);
                    return;
                }
                this.graphPanel.setModel(this.model);
                this.graphPanel.repaint();
            }catch (IOException e) {
                // TODO Auto-generated catch block
                String error = "Problems␣to␣open␣the␣file!";
                JOptionPane.showMessageDialog(null, error);
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                String error = "Parsing␣file␣failed!";
                JOptionPane.showMessageDialog(null, error);
                e.printStackTrace();
            }
        }
    }

    private void addVertex(){
        NewGraphVertexDialog vertexDialog = new NewGraphVertexDialog(this.getModel());
        vertexDialog.setVisible(true);
        vertexDialog.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosed(WindowEvent e) {
                GraphFrame.this.undo.setEnabled(true);
                GraphFrame.this.redo.setEnabled(false);
            }
        });
    }

    private void addEdge(){
        NewGraphEdgeDialog edgeDialog = new NewGraphEdgeDialog(this.getModel());
        edgeDialog.setVisible(true);
        edgeDialog.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosed(WindowEvent e) {
                GraphFrame.this.undo.setEnabled(true);
                GraphFrame.this.redo.setEnabled(false);
                if (GraphFrame.this.model.getEdges().size() > 0)
                    GraphFrame.this.btnDelEdge.setEnabled(true);
            }
        });
    }

    private void deleteEdge(){
        DelGraphEdgeDialog edgeDialog = new DelGraphEdgeDialog(this.getModel());
        edgeDialog.setVisible(true);
        edgeDialog.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosed(WindowEvent e) {
                GraphFrame.this.undo.setEnabled(true);
                GraphFrame.this.redo.setEnabled(false);
                if (GraphFrame.this.model.getEdges().size() == 0)
                    GraphFrame.this.btnDelEdge.setEnabled(false);
            }
        });
    }

    public JMenu getMenuFile() {
        return menuFile;
    }
    public void setMenuFile(JMenu menuFile) {
        this.menuFile = menuFile;
    }
    public JToolBar getTb() {
        return tb;
    }
    public void setTb(JToolBar tb) {
        this.tb = tb;
    }
    public JButton getBtnNewVertex() {
        return btnNewVertex;
    }
    public void setBtnNewVertex(JButton btnNewVertex) {
        this.btnNewVertex = btnNewVertex;
    }
    public JButton getBtnDelNode() {
        return btnDelNode;
    }
    public void setBtnDelNode(JButton btnDelNode) {
        this.btnDelNode = btnDelNode;
    }
    public JButton getBtnNewEdge() {
        return btnNewEdge;
    }
    public void setBtnNewEdge(JButton btnNewEdge) {
        this.btnNewEdge = btnNewEdge;
    }
    public JButton getUndo() {
        return undo;
    }
    public void setUndo(JButton undo) {
        this.undo = undo;
    }
    public JButton getRedo() {
        return redo;
    }
    public void setRedo(JButton redo) {
        this.redo = redo;
    }
    public GraphModel getModel() {
        return model;
    }
    public void setModel(GraphModel model) {
        this.model = model;
    }
    public JButton getBtnDelEdge() {
        return btnDelEdge;
    }
    public void setBtnDelEdge(JButton btnDelEdge) {
        this.btnDelEdge = btnDelEdge;
    }
}
```

Listing 7: `GraphModel`

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Observable;

public class GraphModel extends Observable implements Serializable{

    private ArrayList<GraphVertex> vertexes;
    private ArrayList<GraphEdge> edges;
    transient private UndoManager undoManager;
    transient private RedoManager redoManager;

    transient private GraphVertex selectedVertex;

    public GraphModel(){
        this.vertexes = new ArrayList<GraphVertex>();
        this.edges = new ArrayList<GraphEdge>();
        this.undoManager = new UndoManager(this);
        this.redoManager = new RedoManager(this);
    }

    public GraphModel(GraphModel _graph){
        this.vertexes = _graph.getVertexes();
        this.edges = _graph.getEdges();
        this.undoManager = _graph.getUndoManager();
```

```java
        this.redoManager = _graph.getRedoManager();
    }

    public GraphModel(String nameFile, GraphParser parser) throws IOException{
        this();
        parser.loadFromFile(nameFile, this);
    }

    public void addEdge(GraphEdge edge){
        this.getEdges().add(edge);
        this.sendNotificationToObs("Add␣edge,␣from:" + edge.getV1().getName() + "␣to:"
                                                     + edge.getV2().getName());
    }

    public void addVertex(GraphVertex vertex){
        this.getVertexes().add(vertex);
        this.sendNotificationToObs("Add␣vertex,␣name␣:" + vertex.getName());
    }

    public void removeVertex(GraphVertex vertex){
        String vertexName = vertex.getName();
        for (Iterator<GraphEdge> it = this.getEdges().iterator(); it.hasNext(); ) {
            GraphEdge edgeToRemove = it.next();
            if (edgeToRemove.containsVertex(vertex)){
                String v1Name = edgeToRemove.getV1().getName(), v2Name = edgeToRemove.getV2().getName();
                it.remove();
                this.sendNotificationToObs("Remove␣edge,␣from:" + v1Name + "␣to:"
                                                             + v2Name);
            }
        }
        this.getVertexes().remove(vertex);
        this.sendNotificationToObs("Remove␣vertex,␣name␣:" + vertexName);
    }

    public void removeEdge(GraphEdge edge){
        String v1Name = edge.getV1().getName(), v2Name  = edge.getV2().getName();
        this.getEdges().remove(edge);
        this.sendNotificationToObs("Remove␣edge,␣from:" + v1Name + "␣to:"
                                                     + v2Name);
    }

    public void perfromOperation(Operation op){
        // TODO Auto-generated method stub
        switch (op.getOperation()){
            case ADD_VERTEX:{
                this.addVertex(op.getVertex());
                break;
            }
            case REMOVE_VERTEX:{
                for (GraphEdge edge : this.getEdges()){
                    if (edge.containsVertex(op.getVertex()))
                        op.getEdges().add(edge);
                }
                this.removeVertex(op.getVertex());
                break;
            }
            case ADD_EDGE:{
                this.addEdge(op.getEdges().get(0));
                break;
            }
            case REMOVE_EDGE:{
                this.removeEdge(op.getEdges().get(0));
                break;
            }
        }
        this.getUndoManager().addOperation(op);
        if (!this.getRedoManager().stackOperation.isEmpty())
            this.getRedoManager().flushRedoStack();
    }

    public boolean containsVertex(String vertexName){
        for (GraphVertex vertex : this.getVertexes()){
            if (vertex.getName().equals(vertexName))
                return true;
        }
        return false;
    }

    public boolean containsEdge(GraphVertex v1, GraphVertex v2){
        for (GraphEdge edges : this.getEdges()){
            String v1Name = edges.getV1().getName(), v2Name = edges.getV2().
                getName();
            if (v1Name.equals(v1.getName()) && v2Name.equals(v2.getName()))
                return true;
        }
        return false;
    }

    public int getIndexOfVertex(GraphVertex vertex){
        for (int i = 0; i < this.getVertexes().size(); i++){
            if (this.getVertexes().get(i).getName().equals(vertex.getName()))
                return i;
        }
        return -1;
    }

    public GraphVertex getVertexOfName(String vertexName){
        for (GraphVertex vertex : this.getVertexes()){
            if (vertex.getName().equals(vertexName))
                return vertex;
        }
        return null;
    }

    public ArrayList<GraphVertex> getAdjVertexes(GraphVertex vertex){
        ArrayList<GraphVertex> adj = new ArrayList<GraphVertex>();
        for (GraphEdge edge : this.getEdges()){
            if (edge.containsVertex(vertex)){
                if (edge.getV1().getName().equals(vertex.getName()))
                    adj.add(edge.getV2());
                else
                    adj.add(edge.getV1());
            }
        }
        return adj;
    }

    public void saveGraph(String nameFile, GraphParser parser) throws IOException{
        parser.saveGraph(nameFile, this);
    }

    public void serializeGraph(String nameFile) throws IOException{
        FileOutputStream fileOut = new FileOutputStream(nameFile);
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(this);
        out.close();
        fileOut.close();
    }

    public void deSerializeGraph(String nameFile) throws IOException, ClassNotFoundException{
         FileInputStream fileIn = new FileInputStream(nameFile);
         ObjectInputStream in = new ObjectInputStream(fileIn);
         GraphModel _graph = new GraphModel((GraphModel) in.readObject());
         this.setEdges(_graph.getEdges());
         this.setVertexes(_graph.getVertexes());
         in.close();
         fileIn.close();
    }

    public int getIndexEdgeOfVertexes(GraphVertex v1, GraphVertex v2){
        for (GraphEdge edge : this.getEdges()){
            if (edge.containsVertex(v1) && edge.containsVertex(v2))
                return this.getEdges().indexOf(edge);
        }
        return -1;
    }

    public void sendNotificationToObs(String message){
        setChanged();
        this.notifyObservers(message);
    }

    /*AUTOgenerate setters and getters*/
    public ArrayList<GraphVertex> getVertexes() {
        return vertexes;
    }
    public void setVertexes(ArrayList<GraphVertex> vertexes) {
        this.vertexes = vertexes;
    }
    public ArrayList<GraphEdge> getEdges() {
        return edges;
    }
    public void setEdges(ArrayList<GraphEdge> edges) {
        this.edges = edges;
    }
    public UndoManager getUndoManager() {
        return undoManager;
    }
    public void setUndoManager(UndoManager undoManager) {
        this.undoManager = undoManager;
    }
    public RedoManager getRedoManager() {
        return redoManager;
    }
    public void setRedoManager(RedoManager redoManager) {
        this.redoManager = redoManager;
    }
    public void setSelectedVertex(GraphVertex selected){
        this.selectedVertex = selected;
    }
    public GraphVertex getSelectedVertex(){
        return selectedVertex;
    }
}
```

Listing 8: GraphPanel

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JPanel;
import javax.swing.UIManager;

public class GraphPanel extends JPanel implements Observer {

    private GraphModel graph;
    private SelectionController selCon;

    public GraphPanel(GraphModel _graph){
        selCon = new SelectionController(_graph, this);
        this.setModel(_graph);

        this.addMouseListener(selCon);
        this.addMouseMotionListener(selCon);
    }

    @Override
    public void update(Observable o, Object arg) {
        // TODO Auto-generated method stub
        this.repaint();
        System.out.println("update:␣" + arg);
    }

    public void paintComponent(Graphics g){
        if (g == null) return;
        this.clear(g);
        this.paintVertexes(g);
        this.paintEdge(g);
    }

    private void paintVertexes(Graphics g){
        if (this.graph == null) return;

        for (GraphVertex vertex : this.graph.getVertexes()){

            int x = vertex.getRect().x, y = vertex.getRect().y, width = vertex
                .getRect().width;
            int height = vertex.getRect().height;
            g.drawRect(x, y, width, height);
            g.setColor(Color.white);
            g.fillRect(x, y, width, height);
            g.setColor(Color.black);
            g.drawString(vertex.getName(), vertex.getRect().x,
                         vertex.getRect().y + vertex.getRect().height/2);
        }
    }

    private void clear(Graphics g){
        g.setColor(UIManager.getColor("Panel.background"));
        g.fillRect(0, 0, getWidth(), getHeight());
    }

    private void paintEdge(Graphics g){
        g.setColor(Color.black);
        for(GraphEdge edge : this.graph.getEdges()){
            Rectangle rect1 = edge.getV1().getRect();
            Rectangle rect2 = edge.getV2().getRect();
            g.drawLine(rect1.x + (rect1.width/2) , rect1.y + (rect1.height/2),
                    rect2.x + (rect2.width/2), rect2.y + (rect2.height/2) );
        }

    }

    public void paintRed(GraphVertex vertex){
        Graphics g = this.getGraphics();
        int x = vertex.getRect().x, y = vertex.getRect().y, width = vertex.
            getRect().width;
        int height = vertex.getRect().height;
        g.drawRect(x, y, width, height);
        g.setColor(Color.red);
        g.fillRect(x, y, width, height);
        g.setColor(Color.black);
        g.drawString(vertex.getName(), vertex.getRect().x,
                     vertex.getRect().y + vertex.getRect().height/2);
    }

    public void repaint(){
        this.requestFocus();
        this.paintComponent(this.getGraphics());
    }

    public void setModel(GraphModel _graph){
        this.graph = _graph;
        this.selCon.setModel(_graph);
        this.graph.addObserver(this);
    }
}
```

Listing 9: GraphParser

```
import java.io.IOException;
```

```java

public abstract class GraphParser {
    public abstract void saveGraph(String nameFile, GraphModel graph) throws
        IOException;
    public abstract void loadFromFile(String nameFile, GraphModel graph)
        throws IOException;
}
```

Listing 10: GraphVertex

```java
import java.awt.Point;
import java.awt.Rectangle;
import java.io.Serializable;

public class GraphVertex implements Serializable{

    /*Name suppose to be unique*/
    private String name;
    private Rectangle rect;

    /*Default constant*/
    public static final String DEFAULT_NAME = "VERTEX";
    public static final int DEFAULT_WIDTH = 100;
    public static final int DEFAULT_HEIGHT = 75;

    public GraphVertex(){
        this.name = DEFAULT_NAME;
        this.rect = new Rectangle(0, 0, DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }

    public GraphVertex(String name){
        this.name = name;
        this.rect = new Rectangle(0, 0, DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }

    public GraphVertex(String name, Rectangle rect){
        this.name = name;
        this.rect = rect;
    }

    public Boolean containsPressedPoint(Point p){
        if (p.getX() > this.getRect().x && p.getX() < (this.getRect().x + this
            .getRect().getWidth())
                && p.getY() > this.getRect().y && p.getY() < (this.getRect().y
                    + this.getRect().getHeight())){
            return true;
        }
        return false;
    }

    /*AUTOgenerate setters and getters*/
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Rectangle getRect() {
        return rect;
    }
    public void setRect(Rectangle rect) {
        this.rect = rect;
    }
}
```

Listing 11: `GraphVizGraphParser`

```java
import java.awt.Rectangle;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class GraphVizGraphParser extends GraphParser {

    @Override
    public void loadFromFile(String nameFile, GraphModel graph) throws
        IOException {
        // TODO Auto-generated method stub
        BufferedReader loadFile = new BufferedReader(new FileReader(nameFile))
            ;
        String line = loadFile.readLine();
        while (!(line = loadFile.readLine()).equals("}")){
            String[] vertexes = line.split("␣--␣");
            String v1Name = vertexes[0];
            v1Name = v1Name.replace("\t", "");

            if (v1Name.contains(";")){
                addVertex(vertexes[0].replace(";", ""), graph);
                continue;
            }

            GraphVertex v1 = graph.getVertexOfName(addVertex(vertexes[0].
                replace("\t", ""), graph));
            for (int i = 1; i < vertexes.length; i++){
                GraphVertex v2 = graph.getVertexOfName(addVertex(vertexes[i].
                    replace(";", "").replace("\t", ""), graph));
                graph.addEdge(new GraphEdge(v1, v2));
            }

        }
        loadFile.close();
    }

    @Override
    public void saveGraph(String nameFile, GraphModel graph) throws
        IOException {
        // TODO Auto-generated method stub
        BufferedWriter graphFile = new BufferedWriter(new FileWriter(nameFile)
            );
        graphFile.write("graph␣GraphModel␣{\n");

        boolean[] edge_visited = new boolean[graph.getEdges().size()];
        boolean[] vertex_visited = new boolean[graph.getVertexes().size()];

        for (int i = 0; i < edge_visited.length; i++)
```

```
            edge_visited[i] = false;
        for (int i = 0; i < vertex_visited.length; i++)
            vertex_visited[i] = false;

        for (GraphVertex vertex : graph.getVertexes()){
            boolean vertexFound = false;
            ArrayList<GraphVertex> adjVertexes = graph.getAdjVertexes(vertex);
            if (adjVertexes.isEmpty()){
                vertex_visited[graph.getIndexOfVertex(vertex)] = true;
                graphFile.write("\t"+
                                vertex.getName()+"␣[x="+ vertex.getRect().x + ",y=" + vertex.getRect().y
                                                    +",width=" + vertex.getRect().width+ ",height="+ vertex.getRect().height
                                                    + "]"+ ";\n");
                continue;
            }
            for (GraphVertex adjVertex : adjVertexes){
                int indexEdge = graph.getIndexEdgeOfVertexes(vertex, adjVertex);
                if (!edge_visited[indexEdge]){
                    if (!vertexFound){
                        graphFile.write("\t"+vertex.getName());
                        if (!vertex_visited[graph.getIndexOfVertex(vertex)]){
                            graphFile.write("␣[x="+ vertex.getRect().x + ",y=" + vertex.getRect().y
                                            +",width=" + vertex.getRect().width+ ",height="+ vertex.getRect().height + "]");
                            vertex_visited[graph.getIndexOfVertex(vertex)] = true;
                        }
                    }
                    vertexFound = true;
                    graphFile.write("␣--␣" + adjVertex.getName());
                    if (!vertex_visited[graph.getIndexOfVertex(adjVertex)]){
                        graphFile.write("␣[x="+ adjVertex.getRect().x + ",y=" + adjVertex.getRect().y
                                    +",width=" + adjVertex.getRect().width + ",height="+ adjVertex.getRect().height
                                    + "]");
                        vertex_visited[graph.getIndexOfVertex(adjVertex)] = true;
                    }
                    edge_visited[indexEdge] = true;
                }
            }
            if (vertexFound) graphFile.write(";\n");
        }

        graphFile.write("}");
        graphFile.flush();
        graphFile.close();
    }

```

```java

    private String addVertex(String vertexString, GraphModel graph){
        if (!vertexString.contains("[")){
            if (!graph.containsVertex(vertexString)) {
                graph.addVertex(new GraphVertex(vertexString));
            }
            return vertexString;
        }
        vertexString = vertexString.replace("]", "");
        String[] vertexInfo = vertexString.split("␣\\[");
        GraphVertex vertexToAdd = new GraphVertex(vertexInfo[0]);

        vertexInfo = vertexInfo[1].split(",");
        String x = "", y = "", width = "", height = "";
        for (int i = 0; i < vertexInfo.length; i++){
            String[] attribute = vertexInfo[i].split("=");
            if (attribute[0].equals("x")){
                x = attribute[1];
                continue;
            }
            if (attribute[0].equals("y")){
                y = attribute[1];
                continue;
            }
            if (attribute[0].equals("width")){
                width = attribute[1];
                continue;
            }
            if (attribute[0].equals("height")){
                height = attribute[1];
                continue;
            }
        }
        vertexToAdd.setRect(new Rectangle(x.equals("") ? 0 : Integer.parseInt(
            x),
                                          y.equals("") ? 0 : Integer.parseInt(
                                              y),
                                          width.equals("") ? GraphVertex.
                                              DEFAULT_WIDTH : Integer.parseInt
                                              (width),
                                          height.equals("") ? GraphVertex.
                                              DEFAULT_HEIGHT : Integer.
                                              parseInt(height))
                            );
        graph.addVertex(vertexToAdd);
        return vertexToAdd.getName();
    }
}
```

Listing 12: NewGraphEdgeDialog

```java
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.ArrayList;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class NewGraphEdgeDialog extends JDialog {
    final int WIDTH_WINDOW = 400;
    final int HEIGHT_WINDOW = 400;

    private JPanel centerPanel;
    private JPanel southPanel;

    private JComboBox<String> cmbV1;
    private JComboBox<String> cmbV2;

    private JButton btnSubmit;
    private JButton btnExit;

    private GraphModel graph;

    public NewGraphEdgeDialog(GraphModel _graph){
        this.graph = _graph;
        this.setWindow();
        this.setPanel();
        this.drawFields();

        this.btnSubmit = new JButton("Add␣edge");
        this.btnSubmit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NewGraphEdgeDialog.this.addEdge();
                NewGraphEdgeDialog.this.clearTextField();
            }
        });

        this.btnExit = new JButton("Exit");
        this.btnExit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NewGraphEdgeDialog.this.dispose();
            }
        });

        this.cmbV1.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                if(e.getStateChange() == ItemEvent.SELECTED) {
                    NewGraphEdgeDialog.this.updateComboV2(NewGraphEdgeDialog.
                        this.cmbV1.getSelectedItem().toString());
                }
            }
        });

        this.southPanel.add(btnSubmit);
        this.southPanel.add(btnExit);
        this.add(centerPanel, BorderLayout.CENTER);
        this.add(southPanel, BorderLayout.SOUTH);
    }

    private void addEdge(){
        String nameV1 = this.cmbV1.getSelectedItem().toString();
        if (nameV1.equals("")){
            String error = "First␣vertex␣is␣not␣seleceted!";
            JOptionPane.showMessageDialog(null, error);
            return;
        }
        String nameV2 = this.cmbV2.getSelectedItem() == null ? "" : this.cmbV2
            .getSelectedItem().toString();

        if (!nameV2.equals("")){
            GraphVertex v1 = graph.getVertexOfName(nameV1);
            GraphVertex v2 = graph.getVertexOfName(nameV2);
            graph.perfromOperation(new Operation(Operation.OperationType.
                ADD_EDGE, new GraphEdge(v1, v2)));
        }else{
            //Generate string error
            String error = "Second␣vertex␣is␣not␣seleceted!";
            JOptionPane.showMessageDialog(null, error);
        }
    }

    private void updateComboV2(String nameV1){
        this.cmbV2.setEnabled(true);
        GraphVertex v1 = graph.getVertexOfName(nameV1);
        DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
            String>();
        ArrayList<GraphVertex> adjVertexes = graph.getAdjVertexes(v1);
        for (GraphVertex vertex : graph.getVertexes()){
            if (!adjVertexes.contains(vertex) && !vertex.getName().equals(
                nameV1))
                vertexes.addElement(vertex.getName());
        }
        if (vertexes.getSize() == 0)
            this.cmbV2.setEnabled(false);
        this.cmbV2.setModel(vertexes);
    }

    private void clearTextField(){
        DefaultComboBoxModel<String> vertexes = new DefaultComboBoxModel<
            String>();
        for (GraphVertex vertex : graph.getVertexes()){
            if (this.graph.getAdjVertexes(vertex).size() != this.graph.
                getVertexes().size() - 1)
                vertexes.addElement(vertex.getName());
        }

        if (vertexes.getSize() == 0){
            vertexes.addElement("The␣graph␣is␣connected");
            this.cmbV1.setModel(vertexes);
            this.cmbV2.setModel(new DefaultComboBoxModel<String>());
            this.btnSubmit.setEnabled(false);
            return;
```

```java
        }

        this.cmbV1.setModel(vertexes);
        this.cmbV1.setSelectedIndex(0);
        this.updateComboV2(this.cmbV1.getSelectedItem().toString());
    }

    private void setPanel(){
        this.centerPanel = new JPanel();
        this.centerPanel.setLayout(new GridLayout(2,2,30,30));
        this.centerPanel.setBorder(new EmptyBorder(100, 30, 100, 30));
        this.southPanel = new JPanel();
        this.southPanel.setLayout(new GridLayout(0,2,30,30));
        this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
    }

    private void setWindow(){
        this.setTitle("New␣Edge");
        this.setResizable(false);
        this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
        this.getContentPane().setLayout(new BorderLayout());
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
    }

    private void drawFields(){
        JLabel lblVertex1 = new JLabel("Vertex␣1:");
        this.centerPanel.add(lblVertex1);
        this.cmbV1 = new JComboBox<String>();
        this.centerPanel.add(this.cmbV1);

        JLabel lblVertex2 = new JLabel("Vertex␣2:");
        this.centerPanel.add(lblVertex2);
        this.cmbV2 = new JComboBox<String>();
        this.centerPanel.add(this.cmbV2);
        this.clearTextField();
    }
}
```

Listing 13: NewGraphVertexDialog

```java
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

public class NewGraphVertexDialog extends JDialog {
    final int WIDTH_WINDOW = 500;
    final int HEIGHT_WINDOW = 400;

    private JPanel centerPanel;
    private JPanel southPanel;

    private JTextField txtName;
    private JTextField txtX;
    private JTextField txtY;
    private JTextField txtWidth;
    private JTextField txtHeight;

    private JButton btnSubmit;
    private JButton btnExit;

    private GraphModel graph;

    public NewGraphVertexDialog(GraphModel _graph){
        this.graph = _graph;
        this.setWindow();
        this.setPanel();
        this.drawFields();

        this.btnSubmit = new JButton("Add␣vertex");
        this.btnSubmit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NewGraphVertexDialog.this.addVertex();
                NewGraphVertexDialog.this.clearTextField();
            }
        });

        this.btnExit = new JButton("Exit");
        this.btnExit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NewGraphVertexDialog.this.dispose();
            }
        });

        this.southPanel.add(btnSubmit);
        this.southPanel.add(btnExit);
        this.add(centerPanel, BorderLayout.CENTER);
        this.add(southPanel, BorderLayout.SOUTH);
    }

    private void addVertex(){
        String name = this.txtName.getText();
        int height = this.txtHeight.getText().equals("")? 0 : Integer.parseInt
            (this.txtHeight.getText());
        int width = this.txtWidth.getText().equals("") ? 0 : Integer.parseInt(
            this.txtWidth.getText());
        int x = this.txtX.getText().equals("") ? 0 : Integer.parseInt(this.
            txtX.getText());
        int y = this.txtY.getText().equals("") ? 0 : Integer.parseInt(this.
            txtY.getText());

        if (y >= 0 && y< GraphFrame.WINDOW_HEIGHT && x >= 0 && x < GraphFrame.
            WINDOW_WIDTH
```

```
                    && width >= 0 && height >= 0 && graph.getVertexOfName(name)
                        == null){
                width = width != 0 ? width : GraphVertex.DEFAULT_WIDTH;
                height = height != 0 ? height : GraphVertex.DEFAULT_HEIGHT;
                name = name.equals("") ? GraphVertex.DEFAULT_NAME : name;
                graph.perfromOperation(new Operation(Operation.OperationType.
                    ADD_VERTEX,
                                                       new GraphVertex(name, new
                                                           Rectangle(x, y, width,
                                                           height))));
            }else{
                String error =  "Invalid:␣" + (x < 0 || x > GraphFrame.
                    WINDOW_WIDTH ? "\nValue␣of␣X␣has␣to␣be␣between␣0␣and␣"
                                + GraphFrame.WINDOW_WIDTH : "") +
                                (y < 0 || y > GraphFrame.WINDOW_HEIGHT ? "\nValue␣
                                    of␣Y␣has␣to␣be␣between␣0␣and␣"
                                + GraphFrame.WINDOW_HEIGHT : "") +
                                (width < 0 ? "\nwidth␣<␣0" : "") +
                                (height < 0 ? "\nheight␣<␣0" : "") +
                                (graph.getIndexOfVertex(graph.getVertexOfName(name
                                    )) < 0 ? "\nname␣already␣used" : "");
                JOptionPane.showMessageDialog(null, error);
            }
        }

    private void clearTextField(){
        this.txtX.setText("0");
        this.txtY.setText("0");
        this.txtName.setText(GraphVertex.DEFAULT_NAME);
        this.txtWidth.setText(""+GraphVertex.DEFAULT_WIDTH);
        this.txtHeight.setText(""+GraphVertex.DEFAULT_HEIGHT);
    }

    private void setPanel(){
        this.centerPanel = new JPanel();
        this.centerPanel.setLayout(new GridLayout(5,2,30,30));
        this.centerPanel.setBorder(new EmptyBorder(20, 30, 20, 30));
        this.southPanel = new JPanel();
        this.southPanel.setLayout(new GridLayout(0,2,30,30));
        this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
    }

    private void setWindow(){
        this.setTitle("New␣Vertex");
        this.setResizable(false);
        this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
        this.getContentPane().setLayout(new BorderLayout());
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
    }

    private void drawFields(){
        JLabel lblName = new JLabel("Vertex␣Name␣:");
        this.centerPanel.add(lblName);
        this.txtName = new JTextField();
        this.centerPanel.add(this.txtName);

        JLabel lblX = new JLabel("Coordinate␣X:");
        this.centerPanel.add(lblX);
        this.txtX = new JTextField();
        this.centerPanel.add(this.txtX);

        JLabel lblY = new JLabel("Coordinate␣Y:");
        this.centerPanel.add(lblY);
        this.txtY = new JTextField();
        this.centerPanel.add(this.txtY);

        JLabel lblWidth = new JLabel("Vertex␣Width:");
        this.centerPanel.add(lblWidth);
        this.txtWidth = new JTextField();
        this.centerPanel.add(this.txtWidth);

        JLabel lblHeight = new JLabel("Vertex␣Height:");
        this.centerPanel.add(lblHeight);
        this.txtHeight = new JTextField();
        this.centerPanel.add(this.txtHeight);

        this.clearTextField();
    }
}
```

Listing 14: Operation

```
import java.util.ArrayList;

public class Operation{

    public enum OperationType {ADD_VERTEX, REMOVE_VERTEX, ADD_EDGE,
        REMOVE_EDGE};

    private OperationType operation;
    private ArrayList<GraphEdge> edges;
    private GraphVertex vertex;

    public Operation(OperationType op, GraphVertex vertex){
        this.operation = op;
        this.edges = new ArrayList<GraphEdge>();
        this.vertex = vertex;
    }

    public Operation(OperationType op, GraphEdge edge){
        this.operation = op;
        this.vertex = null;
        this.edges = new ArrayList<GraphEdge>();
        this.getEdges().add(edge);
    }

    public OperationType getOperation() {
        return operation;
    }
    public void setOperation(OperationType operation) {
        this.operation = operation;
    }
    public ArrayList<GraphEdge> getEdges() {
        return edges;
    }
```

```java
    public void setEdges(ArrayList<GraphEdge> edge) {
        this.edges = edge;
    }
    public GraphVertex getVertex() {
        return vertex;
    }
    public void setVertex(GraphVertex vertex) {
        this.vertex = vertex;
    }
}
```

Listing 15: `RedoManager`

```java
import java.io.Serializable;

public class RedoManager extends AbstractUndoableEdit{

    public RedoManager(GraphModel graphModel) {
        super(graphModel);
    }

    public void redoOperation(){
        Operation op = this.getStackOperation().pop();
        this.getGraphModel().getUndoManager().addOperation(this.doOperation(op
            ));
    }

    public void flushRedoStack(){
        this.getStackOperation().removeAllElements();
    }

}
```

Listing 16: `SaveGraphDialog`

```java
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Vector;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

public class SaveGraphDialog extends JDialog {
    final int WIDTH_WINDOW = 400;
    final int HEIGHT_WINDOW = 400;

    private JPanel centerPanel;
    private JPanel southPanel;

    private JComboBox<String> cmbFormat;
    private JTextField txtNameFile;

    private JButton btnSubmit;
    private JButton btnExit;

    private GraphModel graph;

    public SaveGraphDialog(GraphModel _graph){
        this.graph = _graph;
        this.setWindow();
        this.setPanel();
        this.drawFields();

        this.btnSubmit = new JButton("Save");
        this.btnSubmit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                SaveGraphDialog.this.saveGraph();
            }
        });

        this.btnExit = new JButton("Exit");
        this.btnExit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                SaveGraphDialog.this.dispose();
            }
        });

        this.southPanel.add(btnSubmit);
        this.southPanel.add(btnExit);
        this.add(centerPanel, BorderLayout.CENTER);
        this.add(southPanel, BorderLayout.SOUTH);
    }

    private void saveGraph(){
        String format = this.cmbFormat.getSelectedItem().toString();
        String nameFile = this.txtNameFile.getText();
        if (!nameFile.equals("")){
            try {
                if (format.equals("Standard")){
                    nameFile += ".txt";
                    this.graph.saveGraph(nameFile, new StandardGraphParser());
                }
                if (format.equals("GraphViz")){
                    nameFile += ".dot";
                    this.graph.saveGraph(nameFile, new GraphVizGraphParser());
                }
                if (format.equals("Serialized")){
                    nameFile += ".obj";
                    this.graph.serializeGraph(nameFile);
                }
                String msg = "Graph␣is␣saved!";
                JOptionPane.showMessageDialog(null, msg);
                this.dispose();
            }catch (IOException e) {
                    String error = "Unable␣to␣save␣the␣graph!";
                    JOptionPane.showMessageDialog(null, error);
                    e.printStackTrace();
                }
        }
        else{
            String error = "File␣name␣is␣empty!";
            JOptionPane.showMessageDialog(null, error);
        }
    }

    private void setPanel(){
        this.centerPanel = new JPanel();
        this.centerPanel.setLayout(new GridLayout(2,2,30,30));
        this.centerPanel.setBorder(new EmptyBorder(100, 30, 100, 30));
        this.southPanel = new JPanel();
        this.southPanel.setLayout(new GridLayout(0,2,30,30));
        this.southPanel.setBorder(new EmptyBorder(30, 30, 30, 30));
    }

    private void setWindow(){
        this.setTitle("New␣Vertex");
        this.setResizable(false);
        this.setSize(WIDTH_WINDOW, HEIGHT_WINDOW);
        this.getContentPane().setLayout(new BorderLayout());
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-this
            .getSize().height/2);
    }

    private void drawFields(){
        JLabel lblVertex1 = new JLabel("File␣name:");
        this.centerPanel.add(lblVertex1);
        this.txtNameFile = new JTextField("name␣file");
        this.centerPanel.add(this.txtNameFile);

        JLabel lblVertex2 = new JLabel("Format:");
        this.centerPanel.add(lblVertex2);
        this.cmbFormat = new JComboBox<String>();
        DefaultComboBoxModel<String> formats = new DefaultComboBoxModel<String
            >();
        formats.addElement("Standard");
        formats.addElement("GraphViz");
        formats.addElement("Serialized");
        this.cmbFormat.setModel(formats);
        this.centerPanel.add(this.cmbFormat);
    }
}
```

Listing 17: `SelectionController`

```java
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import javax.swing.JOptionPane;

public class SelectionController implements MouseListener, MouseMotionListener
    {

    private GraphVertex selectedVertex;
    private GraphVertex draggedVertex;
    private GraphModel model;
    private GraphPanel panel;

    private boolean moving = false;

    SelectionController(GraphModel _model, GraphPanel _panel){
        this.model = _model;
        this.panel = _panel;
    }

    public int coordX(MouseEvent e) {
        return e.getX();
    }

    public int coordY(MouseEvent e) {
        return e.getY();
    }

    @Override
    public void mouseClicked(MouseEvent e){
        if (e.getClickCount() == 2) {
            ChangeVertexNameDialog nameDialog = new ChangeVertexNameDialog(
                this.model, this.selectedVertex, this.panel);
            nameDialog.setVisible(true);
            return;
          }

        Point pressed = e.getPoint();
        for (GraphVertex vertex : this.model.getVertexes()){
            if (vertex.containsPressedPoint(pressed))
                if (this.selectedVertex == null || !this.selectedVertex.equals
                    (vertex)){
                    this.selectedVertex = vertex;
                    this.model.setSelectedVertex(this.selectedVertex);
                    this.panel.repaint();
                    this.panel.paintRed(this.selectedVertex);
            }
            else{
                this.selectedVertex = null;
                this.model.setSelectedVertex(this.selectedVertex);
                this.panel.repaint();
            }
        }
    }

    @Override
    public void mousePressed(MouseEvent e) {
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        Point pressed = e.getPoint();
        double x = pressed.getX();
        double y = pressed.getY();
        for (GraphVertex vertex : this.model.getVertexes()){
            if (vertex.containsPressedPoint(pressed)){
                if (this.selectedVertex != null && !this.selectedVertex.equals(vertex))
                    continue;

                this.setMoving(true);
                this.draggedVertex = vertex;
                this.selectedVertex = vertex;
                this.model.setSelectedVertex(this.selectedVertex);

                this.draggedVertex.getRect().x = (int) ((int) x - (this.draggedVertex.getRect().getWidth()/2));
                this.draggedVertex.getRect().y = (int) ((int) y - (this.draggedVertex.getRect().getHeight()/2));
                this.panel.repaint();
                this.panel.paintRed(this.selectedVertex);
                return;
            }
        }
        if (this.isMoving()){
            this.setMoving(false);
            String error = "Too␣fast!␣The␣mouse␣lost␣the␣reference␣to␣the␣vertex.";
            JOptionPane.showMessageDialog(null, error);
        }
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseEntered(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseExited(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseReleased(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    public boolean isMoving() {
        return moving;
    }
    public void setMoving(boolean moving) {
        this.moving = moving;
    }

    public void setModel(GraphModel _graph){
        this.model = _graph;
    }

}
```

Listing 18: `StandardGraphParser`

```java
import java.awt.Rectangle;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class StandardGraphParser extends GraphParser {

    @Override
    public void saveGraph(String nameFile, GraphModel graph) throws IOException{
        // TODO Auto-generated method stub
        BufferedWriter graphFile = new BufferedWriter(new FileWriter(nameFile));
        graphFile.write(graph.getVertexes().size()+"␣"+graph.getEdges().size()+"\n");
        for (GraphVertex vertex : graph.getVertexes()){
            graphFile.write((int) vertex.getRect().getX() + "␣" + (int) vertex.getRect().getY() + "␣" +
                                (int) vertex.getRect().getHeight() + "␣" + (int) vertex.getRect().getWidth() + "␣" +
                                vertex.getName() + "\n");
        }
        for (GraphEdge edge : graph.getEdges()){
            graphFile.write(graph.getIndexOfVertex(edge.getV1()) + "␣" + graph.getIndexOfVertex(edge.getV2())+"\n");
        }
        graphFile.flush();
        graphFile.close();
    }

    @Override
    public void loadFromFile(String nameFile, GraphModel graph) throws IOException{
        // TODO Auto-generated method stub
        BufferedReader loadFile = new BufferedReader(new FileReader(nameFile));
        String line = loadFile.readLine();
        String[] infoLine = line.split("␣");
        int sizeVertexes = Integer.parseInt(infoLine[0]);;
        int sizeEdges = Integer.parseInt(infoLine[1]);

        for (int i = 0; i < sizeVertexes; i++){
            line = loadFile.readLine();
            infoLine = line.split("␣");
            String nameVertex = "";
            for (int j = 4; j < infoLine.length; j++){
                if (j != 4) nameVertex += "␣";
                nameVertex += infoLine[j];
            }
            graph.addVertex(new GraphVertex(nameVertex,
                                           new Rectangle(Integer.parseInt(
                                               infoLine[0]),
                                                          Integer.parseInt(
                                                              infoLine[1]),
                                                          Integer.parseInt(
                                                              infoLine[3]),
                                                          Integer.parseInt(
                                                              infoLine[2])))
            );
        }
        for (int i = 0; i < sizeEdges; i++){
            line = loadFile.readLine();
            infoLine = line.split("␣");
            graph.addEdge(new GraphEdge(graph.getVertexes().get(Integer.
                parseInt(infoLine[0])),
                                        graph.getVertexes().get(Integer.
                                            parseInt(infoLine[1]))));
        }
        loadFile.close();
    }

}
```

Listing 19: UndoManager

```
import java.io.Serializable;

public class UndoManager extends AbstractUndoableEdit {

    public UndoManager(GraphModel graphModel) {
        super(graphModel);
    }

    public void undoOperation(){
        Operation op = this.getStackOperation().pop();
        this.getGraphModel().getRedoManager().addOperation(this.doOperation(op
            ));
    }

}
```