Signal Flow Graph Drawer.

Problem Statement

An application built to solve any control system using Mason graph representations and to get the total Transfer function, Forward paths, Loops, Combination of non-touching loops and Deltas of the Graph.

Main features

- Drawing nodes freely.
- Ability to move any node anywhere on canvas.
- Ability to change names of nodes after creating it.
- User can draw edges from any node to any other node.
- Ability to stretch and shrink the edges between nodes.
- Modifying the values that each edge.
- Calculating the transfer function.
- Showing the forward paths, loops, combinations of non-touching loops and deltas of the graph.

Data Structure

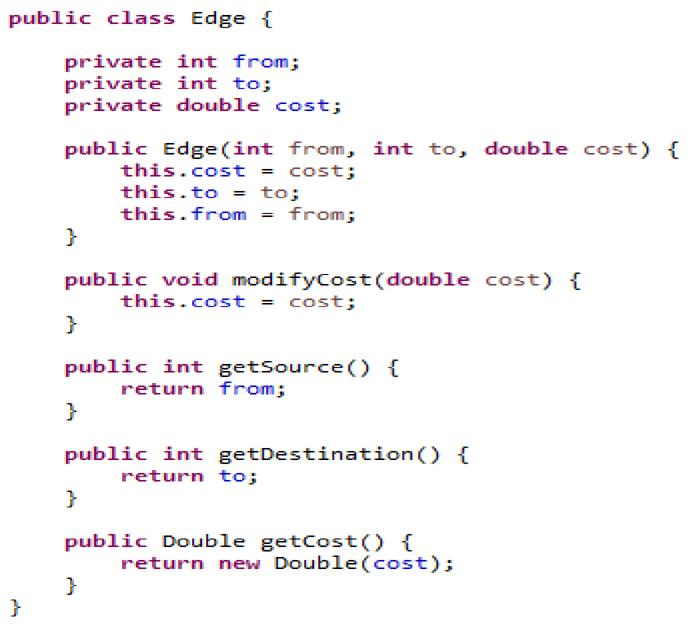
In this application, we used different data-structure, such as:

- Hash Tables.
- Arrays with different dimensions.
- Integers.
- Doubles.
- Strings.

MAIN MODULES

- Edge Module:

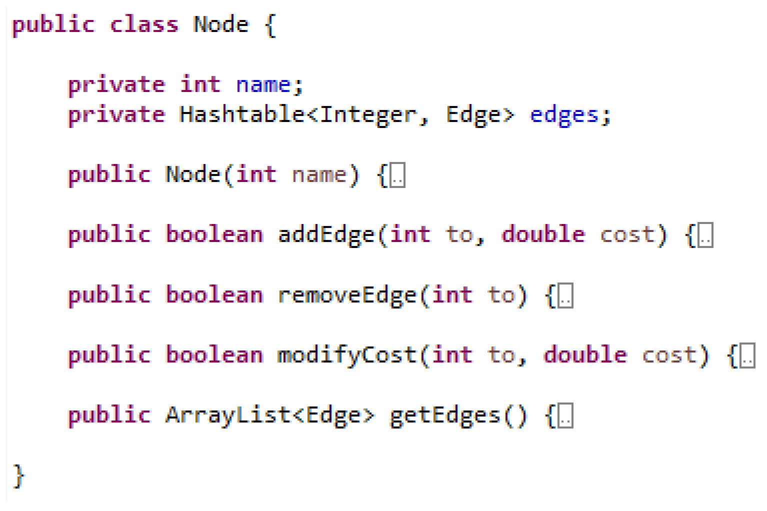
  This module is responsible for saving the date of the edge as it represents the edge in the graph, it contains 3 different variables which are the node from which the edge start and the node to which the edge points to, and the cost of that edge.



```
public class Edge {

    private int from;
    private int to;
    private double cost;

    public Edge(int from, int to, double cost) {
        this.cost = cost;
        this.to = to;
        this.from = from;
    }

    public void modifyCost(double cost) {
        this.cost = cost;
    }

    public int getSource() {
        return from;
    }

    public int getDestination() {
        return to;
    }

    public Double getCost() {
        return new Double(cost);
    }
}
```

- Node Module:

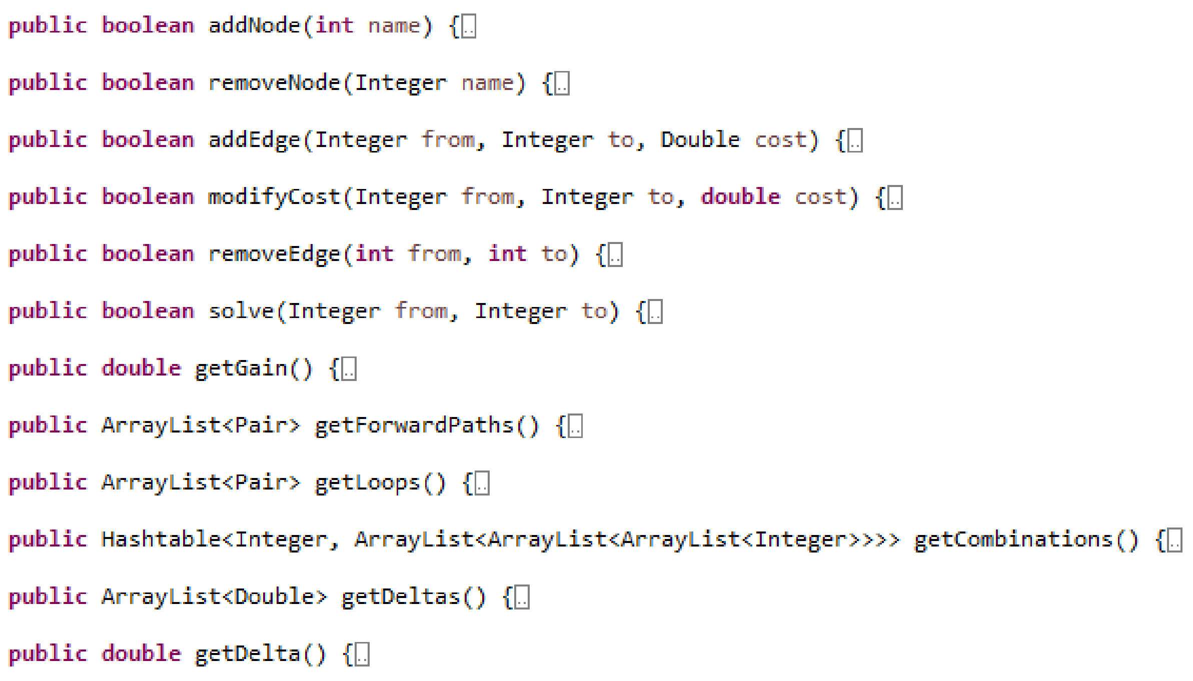
This module is responsible to represent the node in the graph, it contains Hash table of the edges come from the node with key as the target node and the value is the cost, also it contains the name of the node itself.



```java
public class Node {

    private int name;
    private Hashtable<Integer, Edge> edges;

    public Node(int name) {…

    public boolean addEdge(int to, double cost) {…

    public boolean removeEdge(int to) {…

    public boolean modifyCost(int to, double cost) {…

    public ArrayList<Edge> getEdges() {…

}
```

- Graph Module:

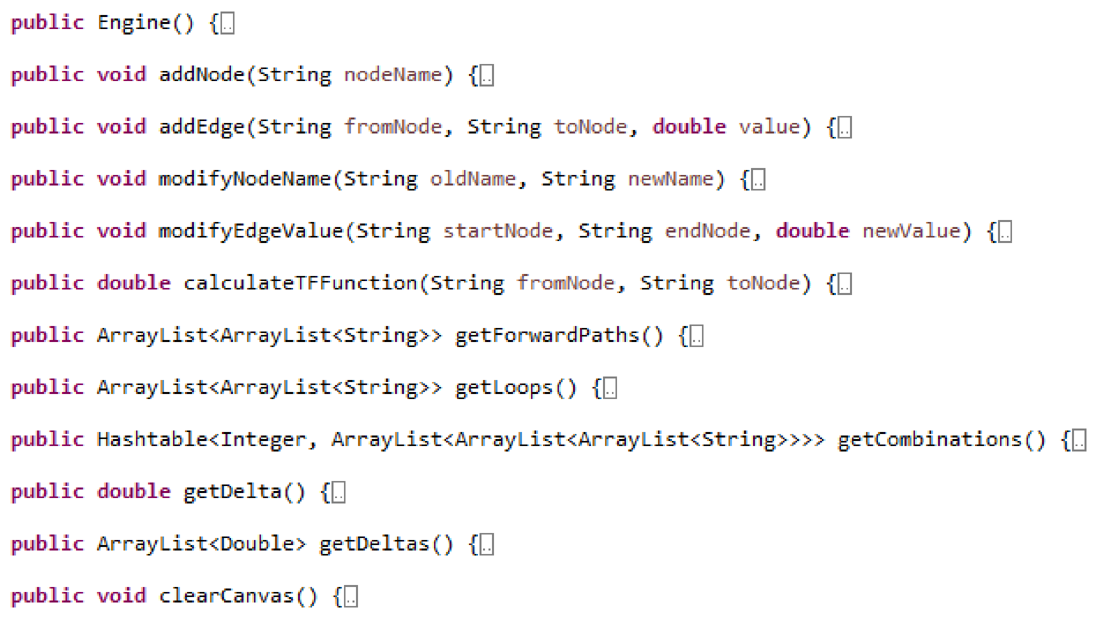
  This module is the one which represent the total graph, it consists of set of nodes, it is also responsible of solving the system and gets the transfer function and the forward paths, loops, combinations of non-touching loops, and deltas.



```java
public boolean addNode(int name) {…

public boolean removeNode(Integer name) {…

public boolean addEdge(Integer from, Integer to, Double cost) {…

public boolean modifyCost(Integer from, Integer to, double cost) {…

public boolean removeEdge(int from, int to) {…

public boolean solve(Integer from, Integer to) {…

public double getGain() {…

public ArrayList<Pair> getForwardPaths() {…

public ArrayList<Pair> getLoops() {…

public Hashtable<Integer, ArrayList<ArrayList<ArrayList<Integer>>>> getCombinations() {…

public ArrayList<Double> getDeltas() {…

public double getDelta() {…
```

- Engine Module:

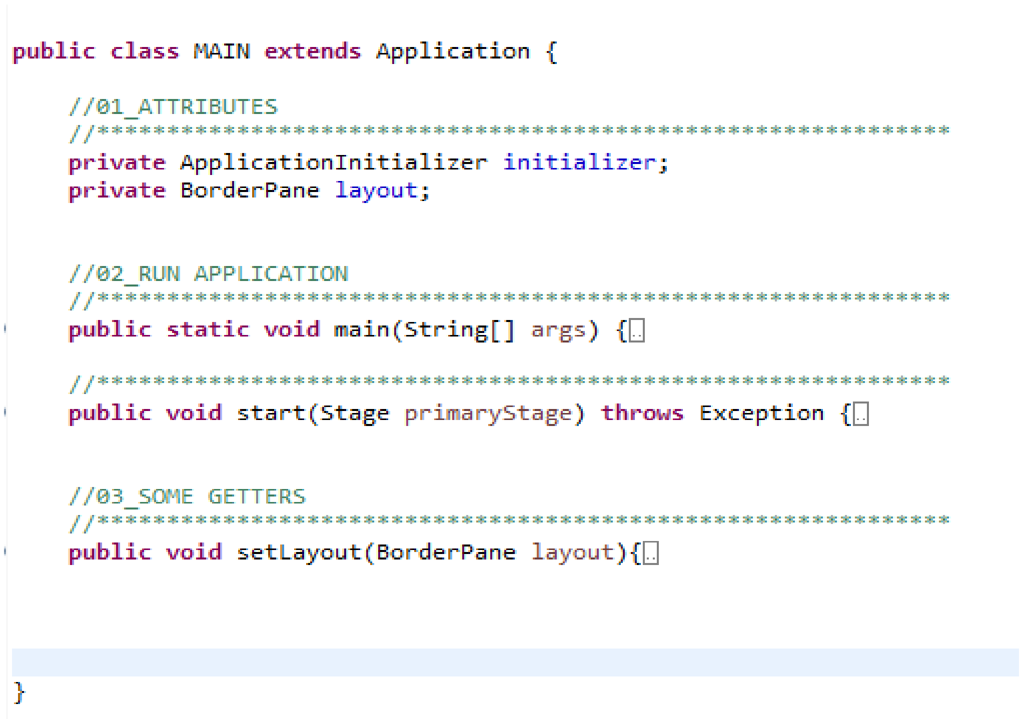
  This is the intermediate module between the GUI and the view and the Graph module, it handles the calls of the view to the graph module and tread with the model.



```java
public Engine() {

public void addNode(String nodeName) {

public void addEdge(String fromNode, String toNode, double value) {

public void modifyNodeName(String oldName, String newName) {

public void modifyEdgeValue(String startNode, String endNode, double newValue) {

public double calculateTFFunction(String fromNode, String toNode) {

public ArrayList<ArrayList<String>> getForwardPaths() {

public ArrayList<ArrayList<String>> getLoops() {

public Hashtable<Integer, ArrayList<ArrayList<ArrayList<String>>>> getCombinations() {

public double getDelta() {

public ArrayList<Double> getDeltas() {

public void clearCanvas() {
```

- Main window Module:

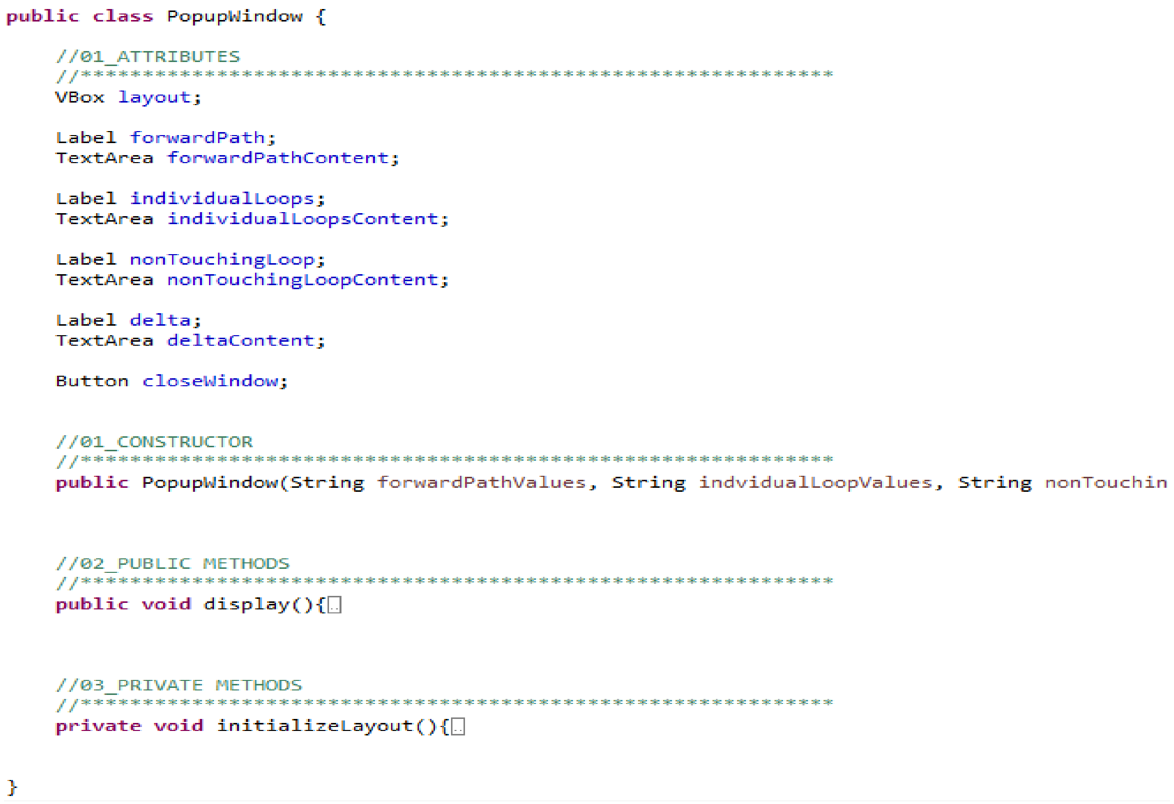
  Used to run the entire application.



```java
public class MAIN extends Application {

    //01_ATTRIBUTES
    //*************************************************************
    private ApplicationInitializer initializer;
    private BorderPane layout;


    //02_RUN APPLICATION
    //*************************************************************
    public static void main(String[] args) {

    //*************************************************************
    public void start(Stage primaryStage) throws Exception {


    //03_SOME GETTERS
    //*************************************************************
    public void setLayout(BorderPane layout){



}
```

- Popup Window Module:

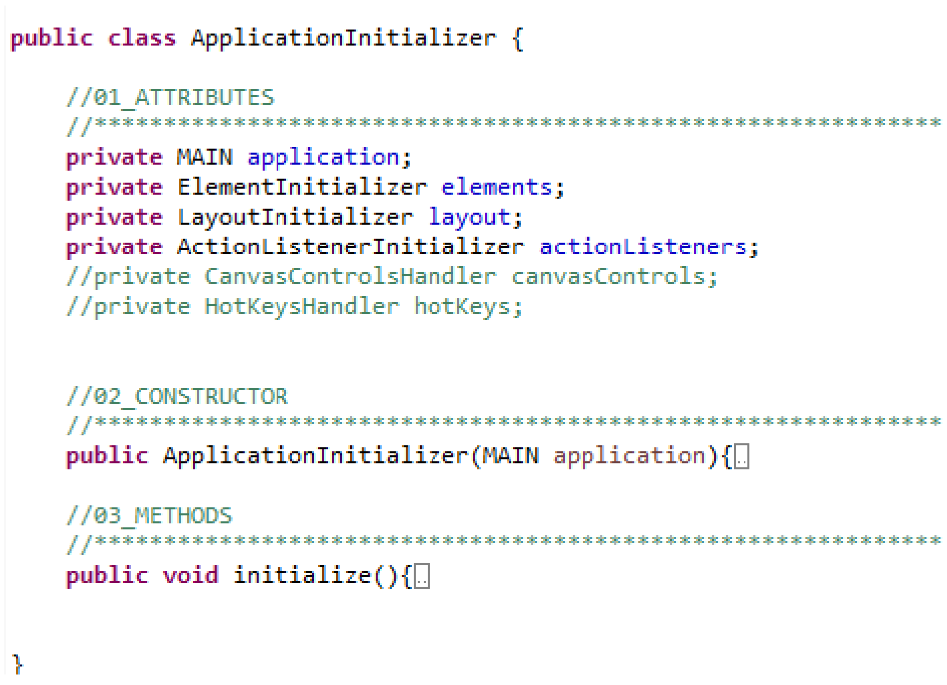
  Used to run the “Control System Details”.



```
public class PopupWindow {

    //01_ATTRIBUTES
    //*************************************************************
    VBox layout;

    Label forwardPath;
    TextArea forwardPathContent;

    Label individualLoops;
    TextArea individualLoopsContent;

    Label nonTouchingLoop;
    TextArea nonTouchingLoopContent;

    Label delta;
    TextArea deltaContent;

    Button closeWindow;


    //01_CONSTRUCTOR
    //*************************************************************
    public PopupWindow(String forwardPathValues, String indvidualLoopValues, String nonTouchin


    //02_PUBLIC METHODS
    //*************************************************************
    public void display(){⋯


    //03_PRIVATE METHODS
    //*************************************************************
    private void initializeLayout(){⋯

}
```

- Window Initializer Module:

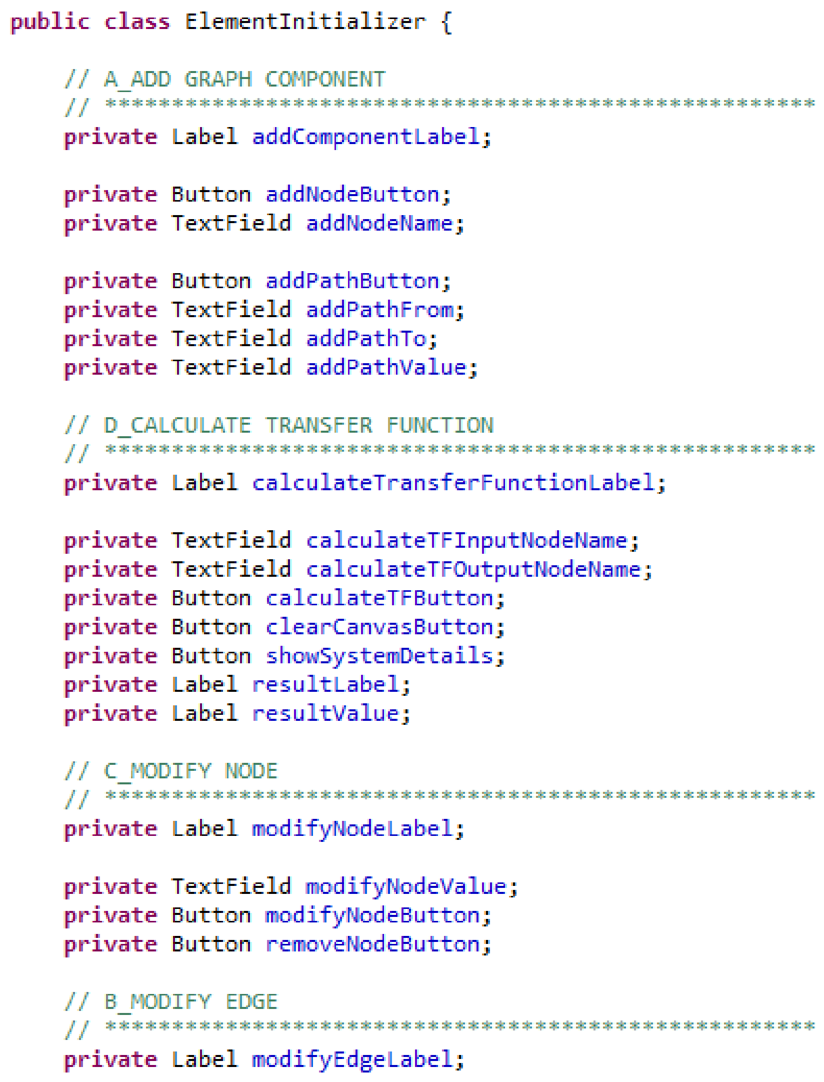
  Is responsible for initializing the entire application (GUI and Back-end modules).



```
public class ApplicationInitializer {

    //01_ATTRIBUTES
    //************************************************************
    private MAIN application;
    private ElementInitializer elements;
    private LayoutInitializer layout;
    private ActionListenerInitializer actionListeners;
    //private CanvasControlsHandler canvasControls;
    //private HotKeysHandler hotKeys;


    //02_CONSTRUCTOR
    //************************************************************
    public ApplicationInitializer(MAIN application){⋯

    //03_METHODS
    //************************************************************
    public void initialize(){⋯


}
```

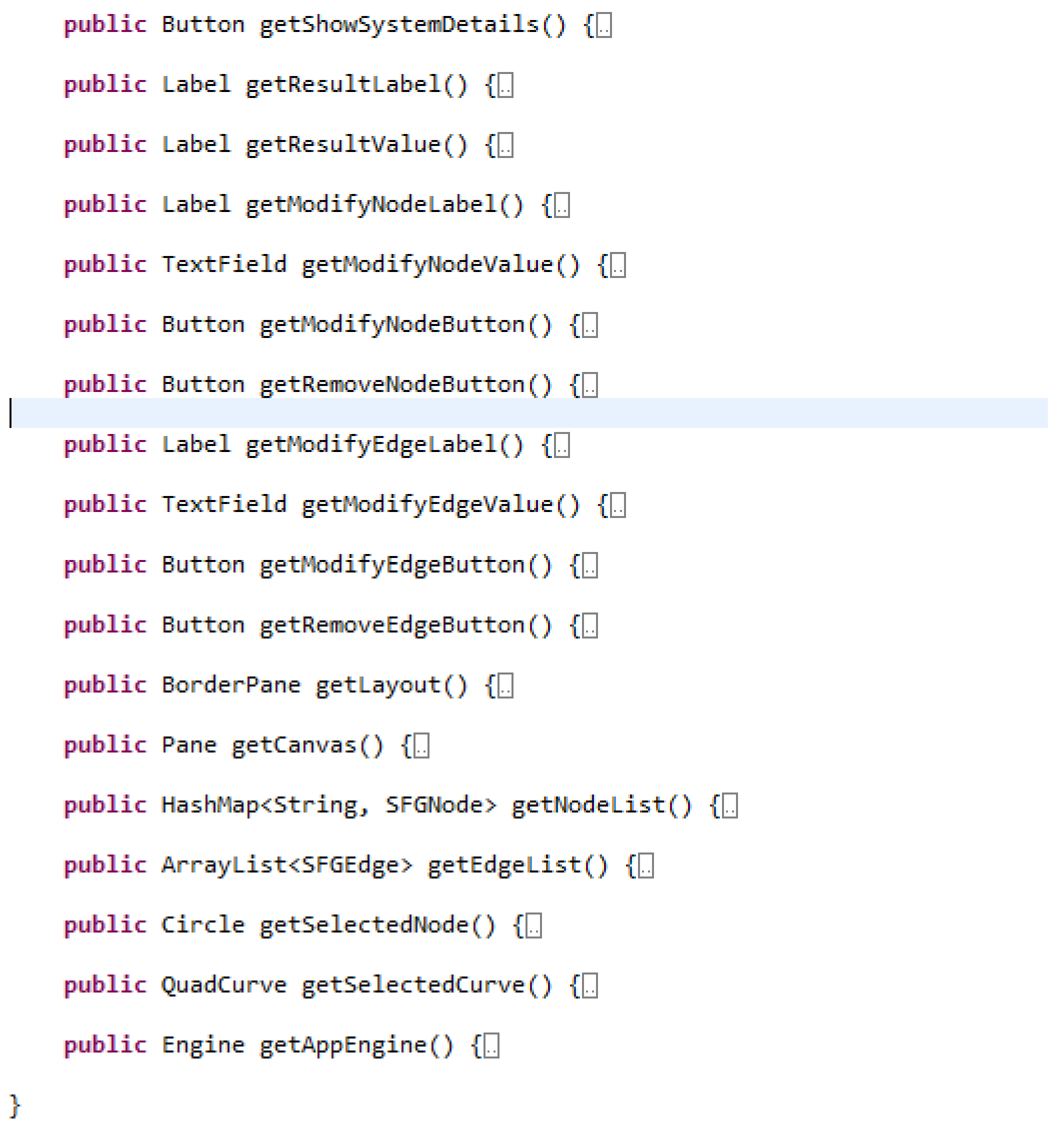
- Element Initializer Module:

  Responsible for initializing GUI elements (Buttons, Canvas, Text Areas).



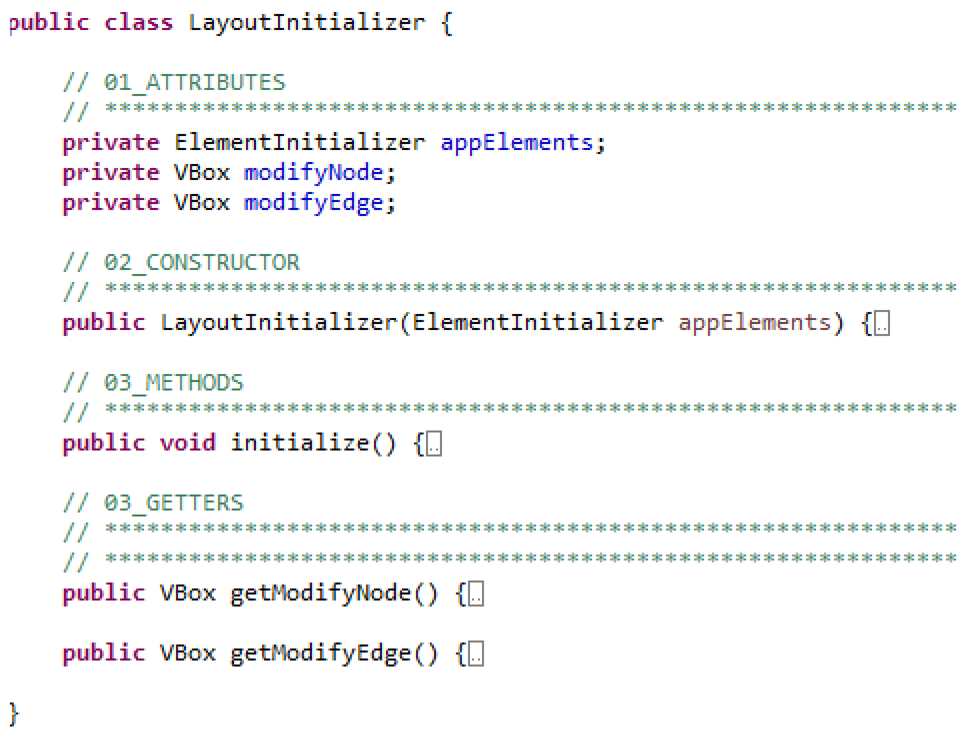
```
public class ElementInitializer {

    // A_ADD GRAPH COMPONENT
    // *****************************************************
    private Label addComponentLabel;

    private Button addNodeButton;
    private TextField addNodeName;

    private Button addPathButton;
    private TextField addPathFrom;
    private TextField addPathTo;
    private TextField addPathValue;

    // D_CALCULATE TRANSFER FUNCTION
    // *****************************************************
    private Label calculateTransferFunctionLabel;

    private TextField calculateTFInputNodeName;
    private TextField calculateTFOutputNodeName;
    private Button calculateTFButton;
    private Button clearCanvasButton;
    private Button showSystemDetails;
    private Label resultLabel;
    private Label resultValue;

    // C_MODIFY NODE
    // *****************************************************
    private Label modifyNodeLabel;

    private TextField modifyNodeValue;
    private Button modifyNodeButton;
    private Button removeNodeButton;

    // B_MODIFY EDGE
    // *****************************************************
    private Label modifyEdgeLabel;
```

```java
// ·······················································
private Label modifyEdgeLabel;

private TextField modifyEdgeValue;
private Button modifyEdgeButton;
private Button removeEdgeButton;

// G_LAYOUT ELEMENTS
// *******************************************************
private BorderPane layout;
private Pane canvas;

private HashMap<String, SFGNode> nodeList;
private ArrayList<SFGEdge> edgeList;
private Circle selectedNode;
private QuadCurve selectedCurve;

private Engine appEngine;

// *************************************************************************
public void initialize() {□

// 03_GETTERS
// ***************************************************************
// ***************************************************************
public Label getAddComponentLabel() {□

public Button getAddNodeButton() {□

public TextField getAddNodeName() {□

public Button getAddPathButton() {□

public TextField getAddPathFrom() {□

public TextField getAddPathTo() {□

public TextField getAddPathValue() {□
public Label getCalculateTransferFunctionLabel() {

public Button getCalculateTFButton() {

public TextField getCalculateTFInputNodeName() {

public TextField getCalculateTFOutputNodeName() {

public Button getClearCanvasButton() {

public Button getShowSystemDetails() {

public Label getResultLabel() {

public Label getResultValue() {

public Label getModifyNodeLabel() {

public TextField getModifyNodeValue() {

public Button getModifyNodeButton() {

public Button getRemoveNodeButton() {

public Label getModifyEdgeLabel() {

public TextField getModifyEdgeValue() {

public Button getModifyEdgeButton() {

public Button getRemoveEdgeButton() {

public BorderPane getLayout() {

public Pane getCanvas() {

public HashMap<String, SFGNode> getNodeList() {

public ArrayList<SFGEdge> getEdgeList() {
```



```
    public Button getShowSystemDetails() {⊡

    public Label getResultLabel() {⊡

    public Label getResultValue() {⊡

    public Label getModifyNodeLabel() {⊡

    public TextField getModifyNodeValue() {⊡

    public Button getModifyNodeButton() {⊡

    public Button getRemoveNodeButton() {⊡

    public Label getModifyEdgeLabel() {⊡

    public TextField getModifyEdgeValue() {⊡

    public Button getModifyEdgeButton() {⊡

    public Button getRemoveEdgeButton() {⊡

    public BorderPane getLayout() {⊡

    public Pane getCanvas() {⊡

    public HashMap<String, SFGNode> getNodeList() {⊡

    public ArrayList<SFGEdge> getEdgeList() {⊡

    public Circle getSelectedNode() {⊡

    public QuadCurve getSelectedCurve() {⊡

    public Engine getAppEngine() {⊡

}
```

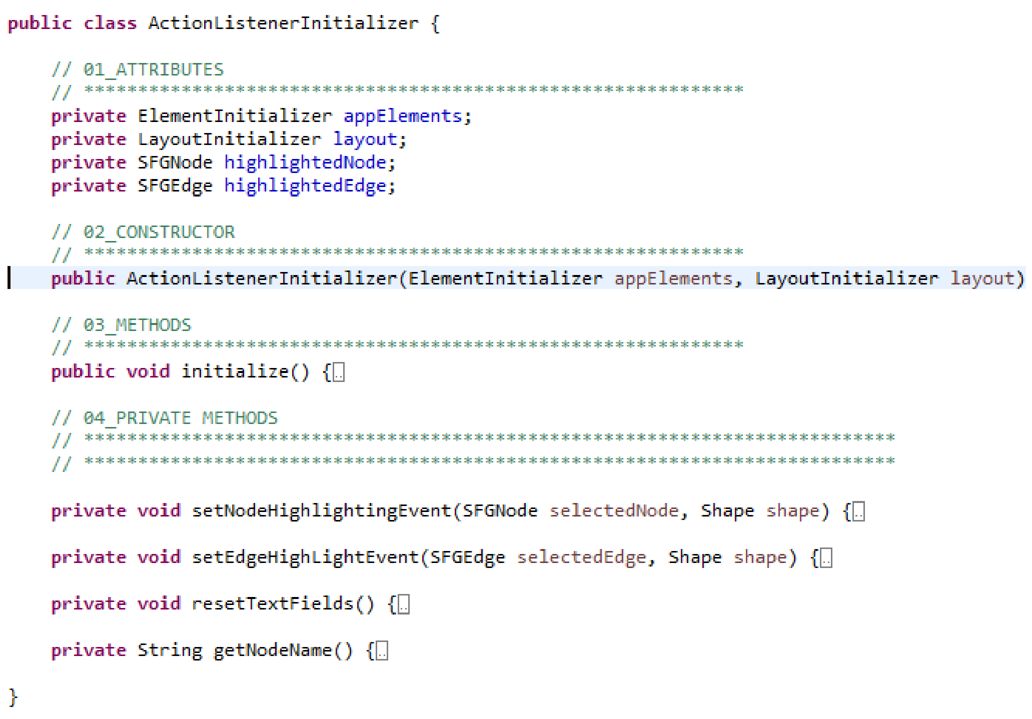
- Layout Initializer Module:
  Responsible for initializing main window layouts (Side Panel, Element positions, Dimensions, Styles, …).



```java
public class LayoutInitializer {

    // 01_ATTRIBUTES
    // *************************************************************
    private ElementInitializer appElements;
    private VBox modifyNode;
    private VBox modifyEdge;

    // 02_CONSTRUCTOR
    // *************************************************************
    public LayoutInitializer(ElementInitializer appElements) {⊡

    // 03_METHODS
    // *************************************************************
    public void initialize() {⊡

    // 03_GETTERS
    // *************************************************************
    // *************************************************************
    public VBox getModifyNode() {⊡

    public VBox getModifyEdge() {⊡

}
```

- ActionListerners Initializer Module:

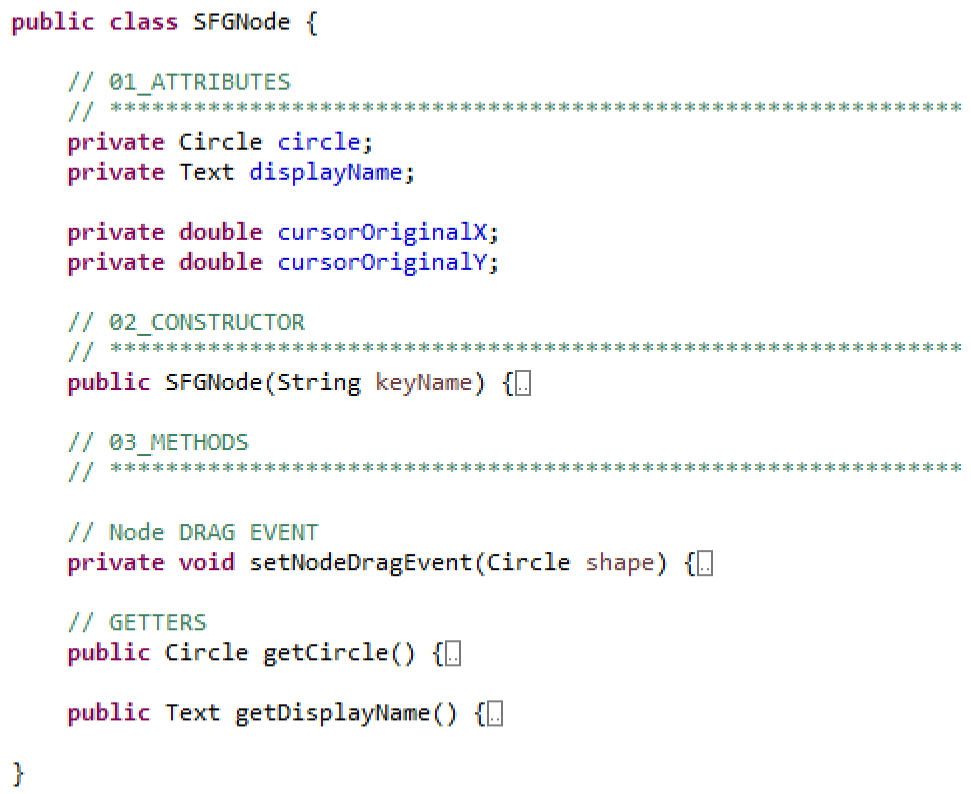
  Responsible for handling user interactions (clicking buttons, adding nodes, adding edges, drag actions, …).



```java
public class ActionListenerInitializer {

    // 01_ATTRIBUTES
    // *************************************************************
    private ElementInitializer appElements;
    private LayoutInitializer layout;
    private SFGNode highlightedNode;
    private SFGEdge highlightedEdge;

    // 02_CONSTRUCTOR
    // *************************************************************
    public ActionListenerInitializer(ElementInitializer appElements, LayoutInitializer layout)

    // 03_METHODS
    // *************************************************************
    public void initialize() {⊡

    // 04_PRIVATE METHODS
    // ***************************************************************************
    // ***************************************************************************

    private void setNodeHighlightingEvent(SFGNode selectedNode, Shape shape) {⊡

    private void setEdgeHighLightEvent(SFGEdge selectedEdge, Shape shape) {⊡

    private void resetTextFields() {⊡

    private String getNodeName() {⊡

}
```

- SFGNode Module:

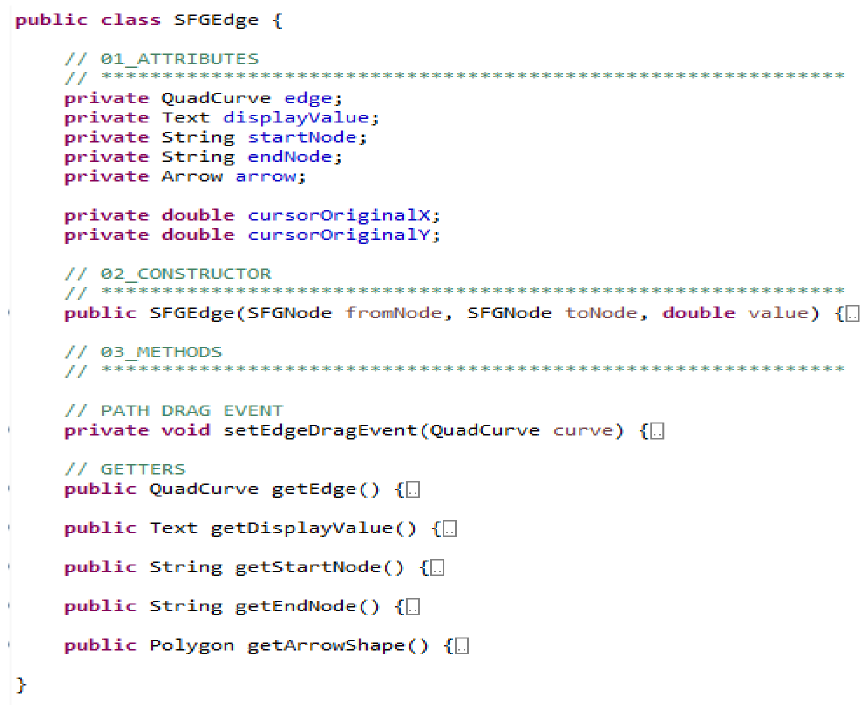
  Responsible for creating Node objects to be represented in the Canvas.



```
public class SFGNode {

    // 01_ATTRIBUTES
    // *************************************************************
    private Circle circle;
    private Text displayName;

    private double cursorOriginalX;
    private double cursorOriginalY;

    // 02_CONSTRUCTOR
    // *************************************************************
    public SFGNode(String keyName) {⊡

    // 03_METHODS
    // *************************************************************

    // Node DRAG EVENT
    private void setNodeDragEvent(Circle shape) {⊡

    // GETTERS
    public Circle getCircle() {⊡

    public Text getDisplayName() {⊡

}
```

- SFGEdge Module:

  Responsible for creating Edge objects to be represented in the Canvas.



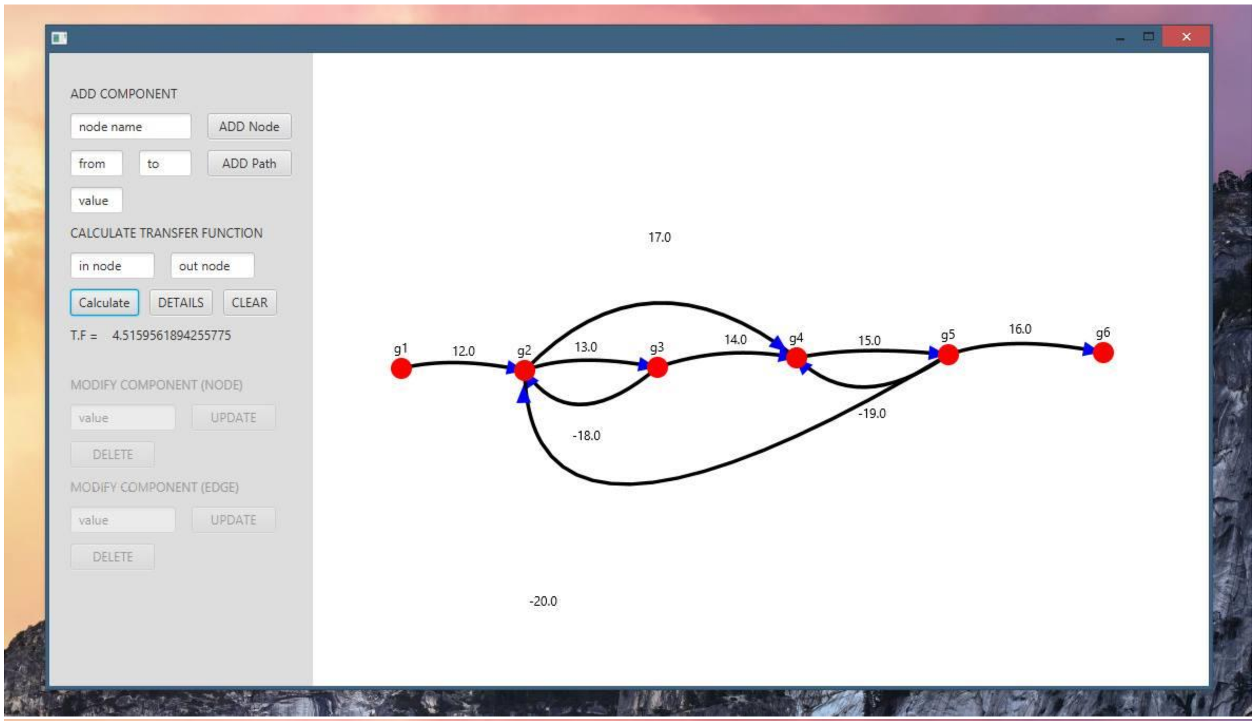
```
public class SFGEdge {

    // 01_ATTRIBUTES
    // *************************************************************
    private QuadCurve edge;
    private Text displayValue;
    private String startNode;
    private String endNode;
    private Arrow arrow;

    private double cursorOriginalX;
    private double cursorOriginalY;

    // 02_CONSTRUCTOR
    // *************************************************************
    public SFGEdge(SFGNode fromNode, SFGNode toNode, double value) {⊡

    // 03_METHODS
    // *************************************************************

    // PATH DRAG EVENT
    private void setEdgeDragEvent(QuadCurve curve) {⊡

    // GETTERS
    public QuadCurve getEdge() {⊡

    public Text getDisplayValue() {⊡

    public String getStartNode() {⊡

    public String getEndNode() {⊡

    public Polygon getArrowShape() {⊡

}
```

ALGORITHM USED

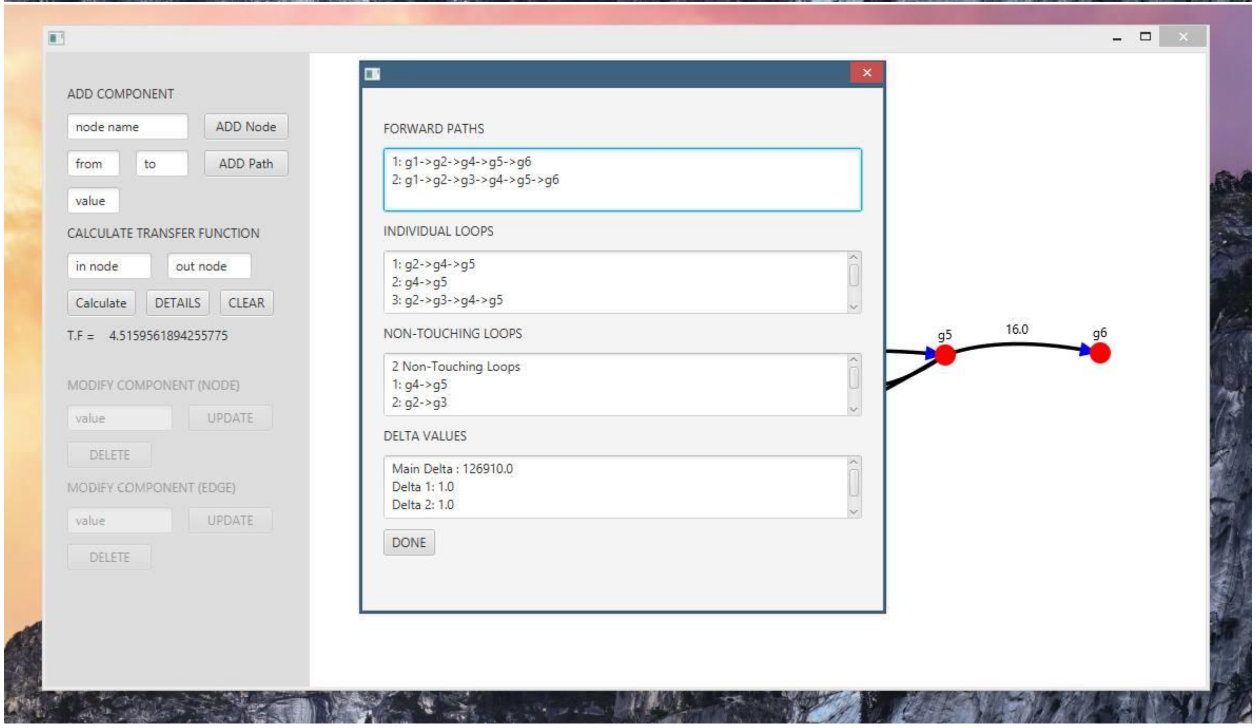
In this application, we used to different algorithms to calculate the forward paths, loops, deltas, overall transfer function. We used:

- Depth First Search: we use this algorithm to calculate the forward paths and the loops.
- Combinatorics: this algorithm used to get the combinations of non-touching loops.

SAMPLE RUNS



```
T.F =   4.5159561894255775
```



```
T.F =   4.5159561894255775

FORWARD PATHS
1: g1->g2->g4->g5->g6
2: g1->g2->g3->g4->g5->g6

INDIVIDUAL LOOPS
1: g2->g4->g5
2: g4->g5
3: g2->g3->g4->g5

NON-TOUCHING LOOPS
2 Non-Touching Loops
1: g4->g5
2: g2->g3

DELTA VALUES
Main Delta : 126910.0
Delta 1: 1.0
Delta 2: 1.0
```

SAMPLE USER GUIDE

There is a Manual for user with the project