HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
GLOBAL ICT K67

IT3100E

OOP

ELECTRONIC CIRCUIT SIMULATOR

OBJECT-ORIENTED PROGRAMING

PROJECT REPORT

MEMBER LIST

Trinh Thi Thuy Duong	20226034	Duong.TTT226034@sis.hust.edu.vn
Doan Thi Thu Quyen	20226063	Quyen.DTT226063@sis.hust.edu.vn
Duong Phuong Thao	20226001	Thao.DP226001@sis.hust.edu.vn

Lecturer: Tran The Hung

Hà Nội — 2024

Contents

1 Project Assignment

We will inform the tasks each member has completed by listing the work he involved in this project.

Number	Full Name	Student ID
1	Trinh Thi Thuy Duong	20226034
2	Doan Thi Thu Quyen	20226063
3	Duong Phuong Thao	20226001

Task	Responsible Member ID	Percentage Contribution
General Class Diagram	Trinh Thi Thuy Duong	50%
	Duong Phuong Thao	50%
Detail Class Diagram	Trinh Thi Thuy Duong	50%
	Duong Phuong Thao	50%
Use-case Diagram	Doan Thi Thu Quyen	100%
Components Package	Doan Thi Thu Quyen	100%
TableAnalysis Package	Trinh Thi Thuy Duong	100%
Drawcircuit Package	Duong Phuong Thao	100%
Source_fxml Package	Trinh Thi Thuy Duong	50%
	Duong Phuong Thao	50%
Test Package	Doan Thi Thu Quyen	100%
AI Chat Box	Trinh Thi Thuy Duong	50%
	Doan Thi Thu Quyen	25%
	Duong Phuong Thao	25%
Report	Trinh Thi Thuy Duong	33%
	Doan Thi Thu Quyen	33%
	Duong Phuong Thao	33%

2 Project Description

2.1 Mini-Project Overview

In this capstion project, we aim to build this application to helps to design an application that enables users to generate electrical circuits and calculate their components. The application should include functionalities for drawing circuit diagrams, adding and configuring various electrical components (resistors, capacitors, inductors, voltage), and performing real-time calculations for voltage, current, resistance, and other relevant parameters. Additionally, the application must integrate a chatbot feature that leverages the ChatGPT API, allowing users to ask questions and receive detailed explanations and guidance on circuit design and electrical concepts.

Additionally, we also employ GitHub to track our progress and ensure project completion.

2.2 Use-case Diagram

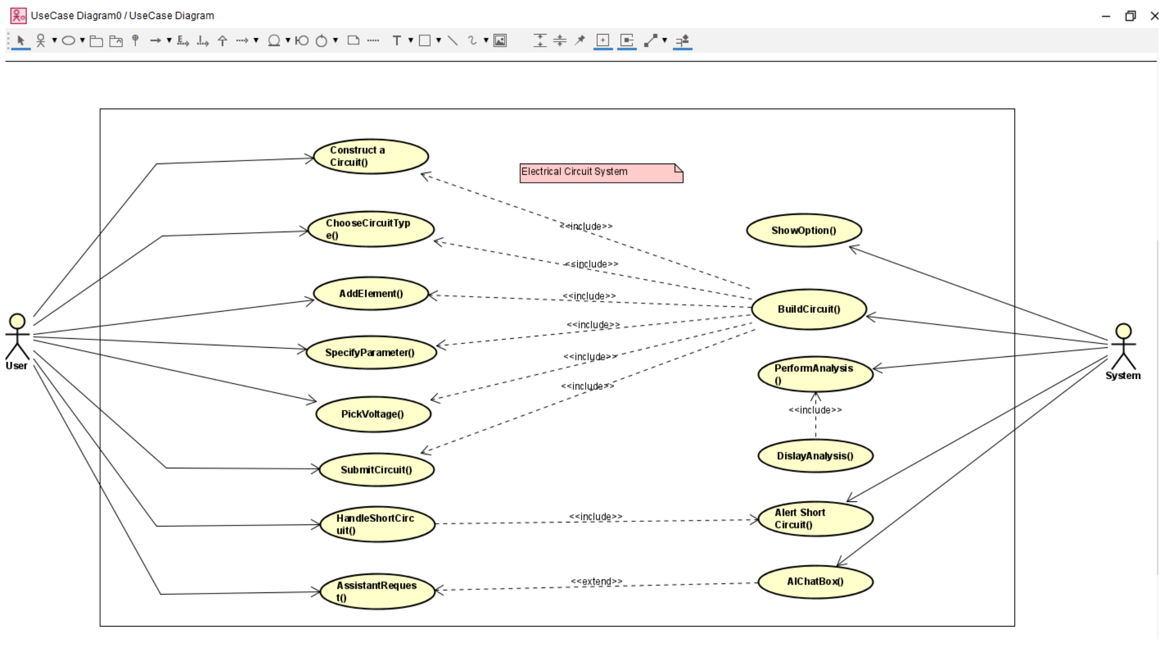


Figure 1: Use-case Diagram

3 Design

3.1 General Class Diagram

Our General Class Diagram consists of 5 main packages: **source-fxml**, **Test**, **Components**, **DrawCircuit** and **TableAnalysis**.

3.2 Package Details

3.2.1 Components package

The Components package includes files for **essential circuit parts** (elements such as resistors, inductors, capacitors, and voltage sources). It also contains the **EleController file**, which has methods for circuit analysis, and the **Complex package**, which handles complex number calculations.

Components.complexNum package
To calculate the parameters U, R, and I for the circuit, complex numbers are used. Therefore,

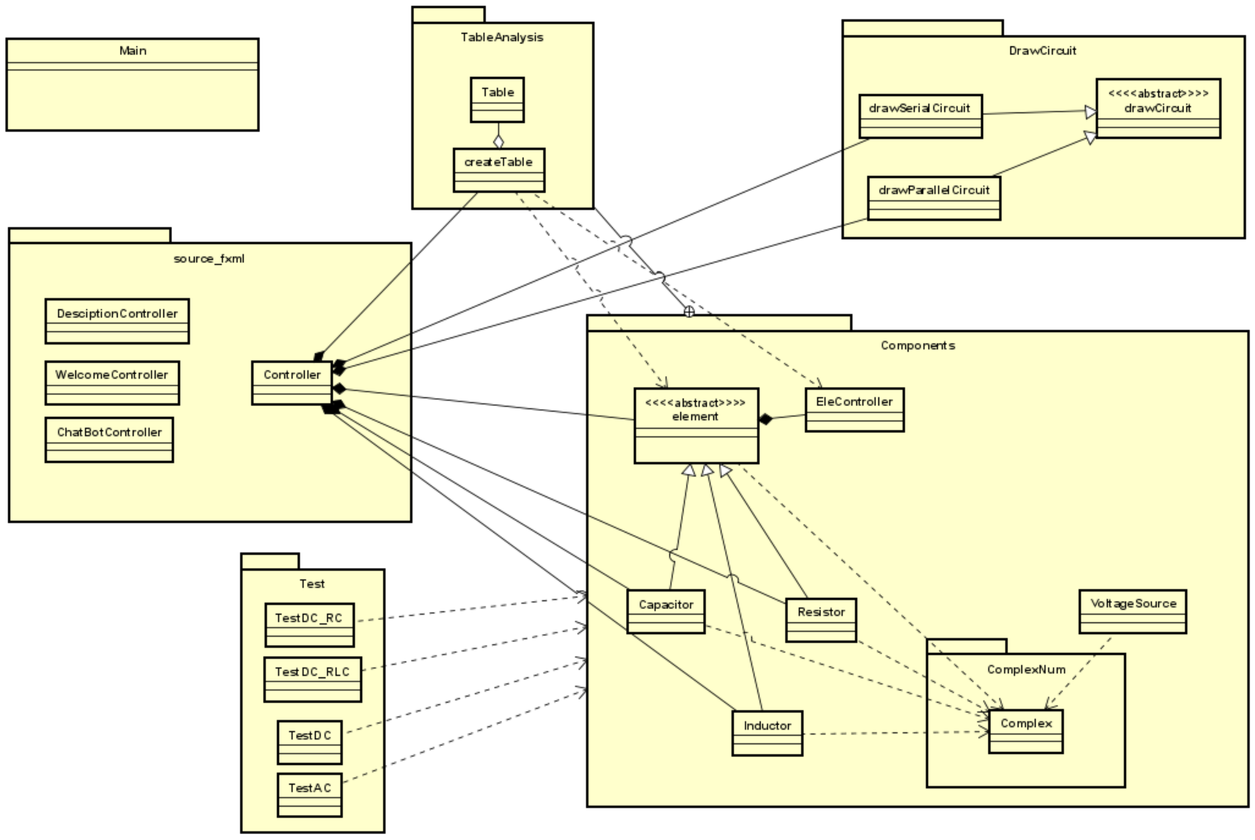


Figure 2: General Class Diagram

we need to implement a complex number package that includes files for handling addition, subtraction, multiplication, and division of complex numbers.

- **Attributes:**
  - **real:** The real value in number
  - **imaginary:** The imaginary value in number. For R and voltage source in DC , this part equals 0.

- **Methods:**
  - **getReal:** get the real value
  - **getImarinary:** get the imarinary value
  - **add:** implement add two Complex
  - **subtract:** implement subtract two Complex
  - **multiply:** implement multiply two Complex
  - **divide:** implement divide two Complex
  - **inverse:** implement inverse Complex

Elements class
The ‘Element‘ class is an abstract class that serves as a base class for components in circuit.

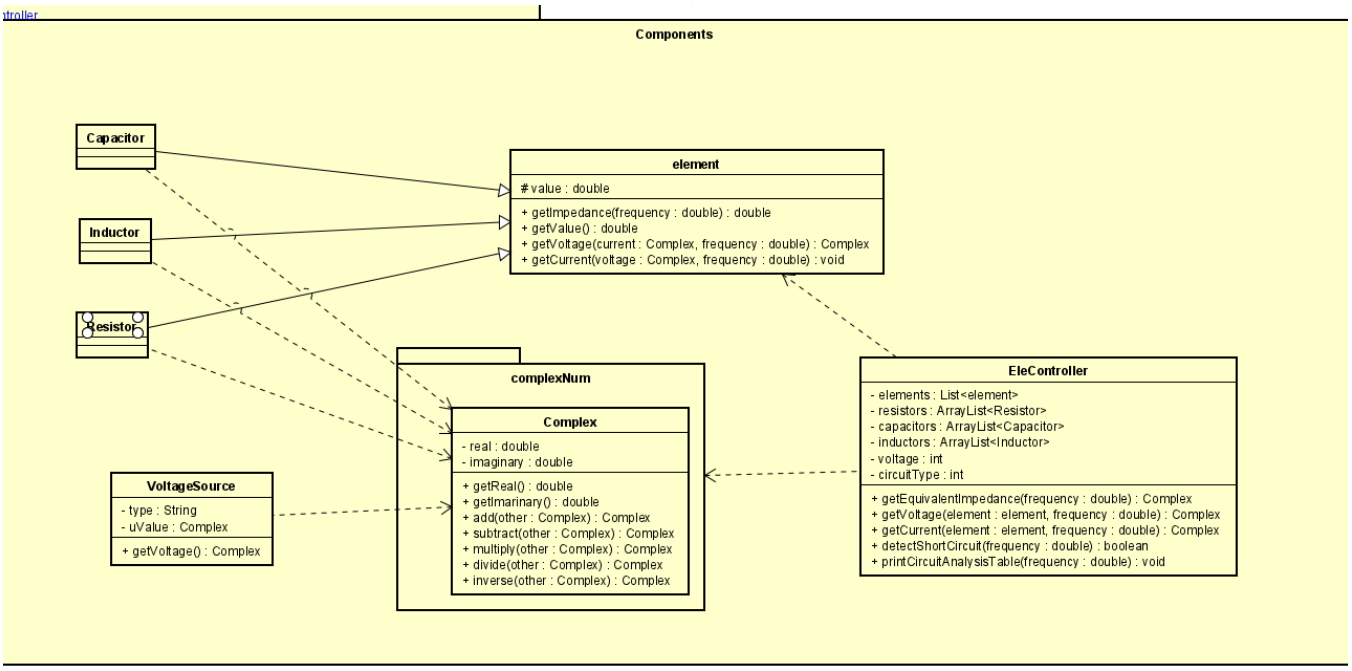

ntroller
Components
Capacitor
Inductor
Resistor
element
# value : double
+ getImpedance(frequency : double) : double
+ getValue() : double
+ getVoltage(current : Complex, frequency : double) : Complex
+ getCurrent(voltage : Complex, frequency : double) : void
complexNum
Complex
- real : double
- imaginary : double
+ getReal() : double
+ getImarinary() : double
+ add(other : Complex) : Complex
+ subtract(other : Complex) : Complex
+ multiply(other : Complex) : Complex
+ divide(other : Complex) : Complex
+ inverse(other : Complex) : Complex
VoltageSource
- type : String
- uValue : Complex
+ getVoltage() : Complex
EleController
- elements : List<element>
- resistors : ArrayList<Resistor>
- capacitors : ArrayList<Capacitor>
- inductors : ArrayList<Inductor>
- voltage : int
- circuitType : int
+ getEquivalentImpedance(frequency : double) : Complex
+ getVoltage(element : element, frequency : double) : Complex
+ getCurrent(element : element, frequency : double) : Complex
+ detectShortCircuit(frequency : double) : boolean
+ printCircuitAnalysisTable(frequency : double) : void

Figure 3: Package **Components** Class Diagram

The ‘Resistor‘, ‘Inductor‘, and ‘Capacitor‘ classes will inherit its properties and methods.

- **Attributes:**
  - **value:** The specific value for each property.
- **Methods:**
  - **getImpedance:** Get the **’Impedance’** for component.
  - **getValue:** Get the **’Value’** for component.
  - **getVoltage:** Get the **’Voltage Electric’** for component.
  - **getCurrent:** Get the **’Current Electric’** for component.

Resistor, Inductor, Capacitor class
The ‘Resistor‘, ‘Inductor‘, and ‘Capacitor‘ classes represent components that can appear in a circuit, inheriting from the ‘Element‘ class.

- **Attributes:**
  - Inherited from element class.
- **Methods:**
  - **super():** call the constructor of the ‘Element‘ class.
  - **getImpedance:** override the ‘getImpedance‘ method in ’Element’ to handle the calculation of impedance specific to each component.

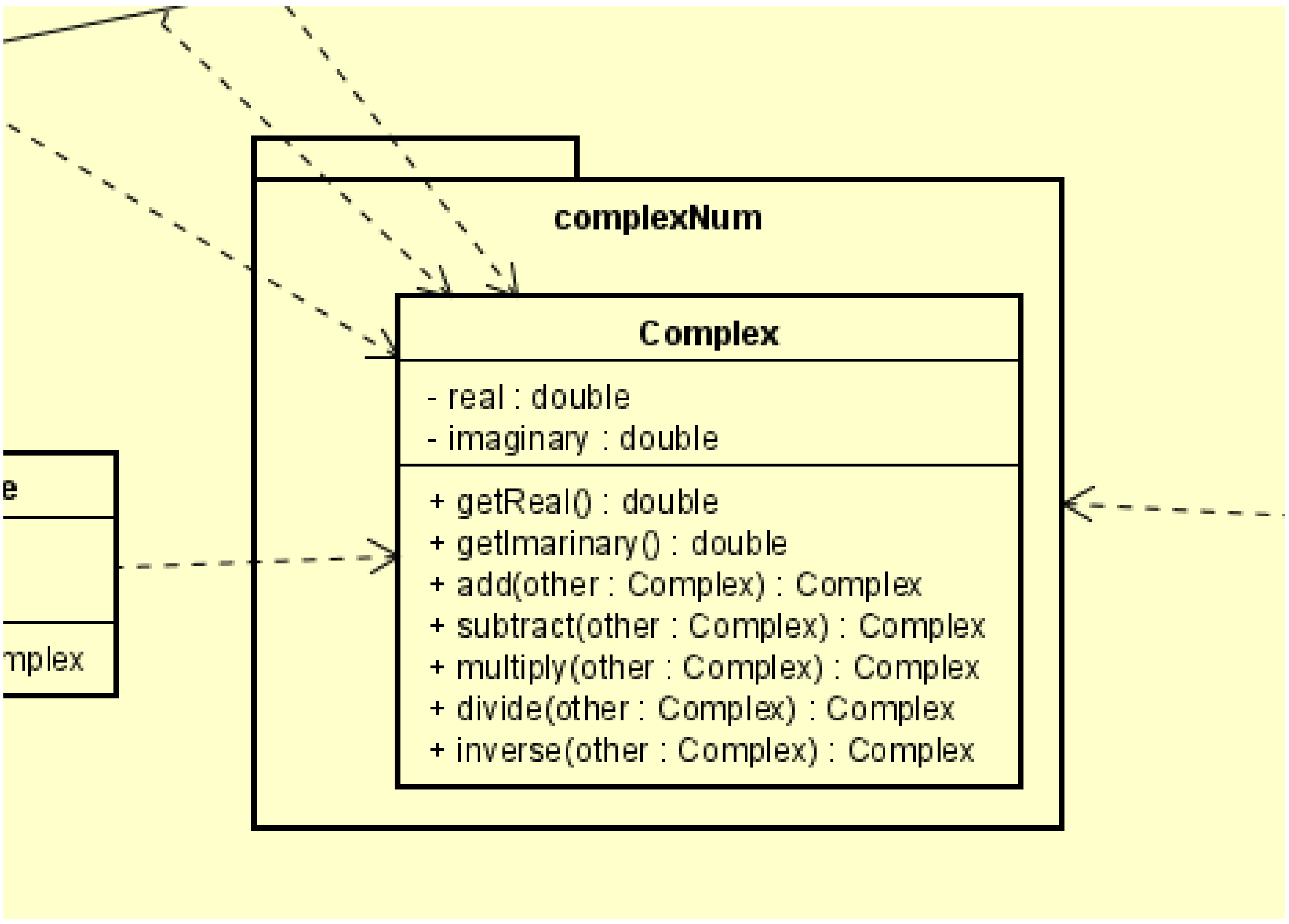

complexNum
Complex
- real : double
- imaginary : double
+ getReal() : double
+ getImarinary() : double
+ add(other : Complex) : Complex
+ subtract(other : Complex) : Complex
+ multiply(other : Complex) : Complex
+ divide(other : Complex) : Complex
+ inverse(other : Complex) : Complex
e
mplex

Figure 4: Package **Components.complexNum** Class Diagram

Figure 6 belowed is the specific code for the Inductor class, implemented as described above:

VoltageSource class The 'VoltageSource' class is used to specify the Type and Value for Voltage in circuit.

- **Attributes:**
  - **type:** AC or DC
  - **uValue:** specific value for Voltage

- **Methods:**
  - **getVoltage:** get the Value for Voltage.

EleController class The 'EleController' class is the main method for handling and calculating values for the circuit. The components are stored in an 'ArrayList' and used in the circuit analysis methods.

- **Attributes:**

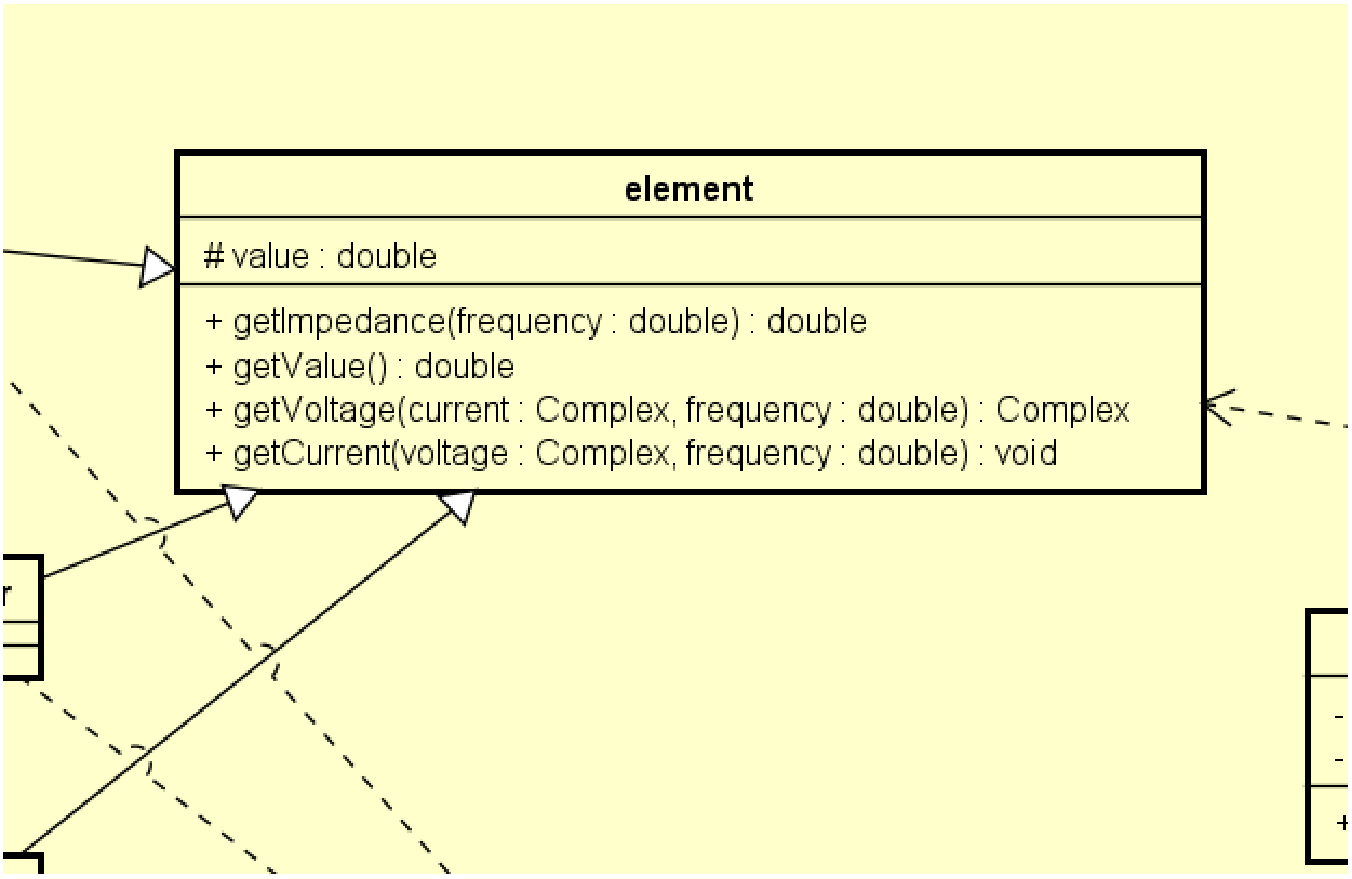
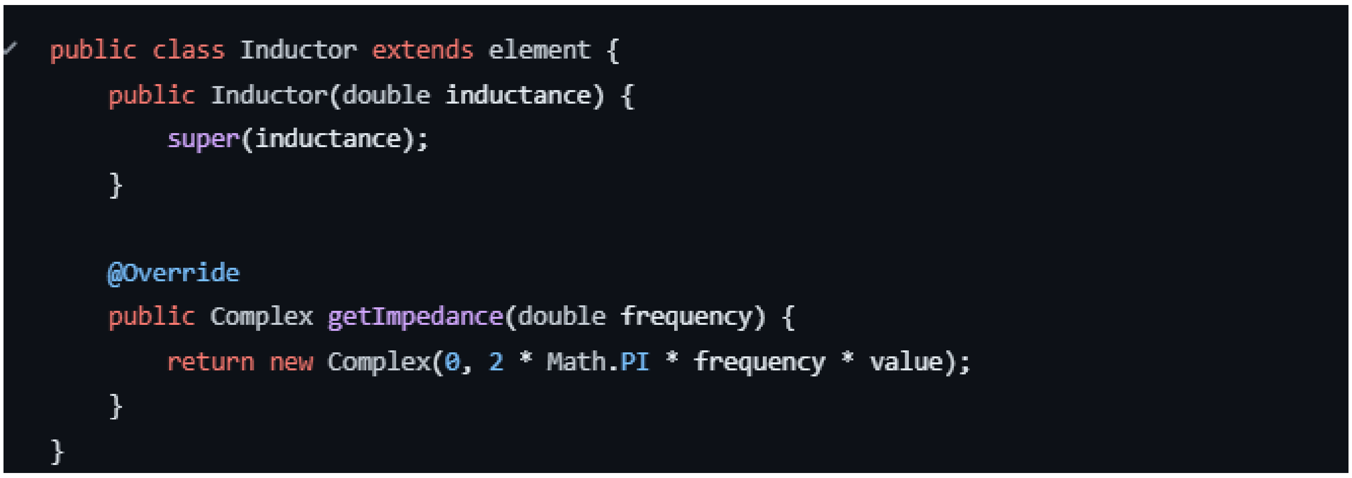

element
# value : double
+ getImpedance(frequency : double) : double
+ getValue() : double
+ getVoltage(current : Complex, frequency : double) : Complex
+ getCurrent(voltage : Complex, frequency : double) : void

Figure 5: Class **element**

- **elements:** represents for all of the components in the circuit
- **resistor:** represents for all Resistor
- **inductor:** represents for all Inductor
- **capacitor:** represents for all Capacitor
- **voltage:** represents for Voltage
- **circuitType:** 1 represents for parallel circuit and 2 represents serial type

• **Methods:**

- **getEquivalentImpedance:** takes the frequency of the circuit as input and calculates the total equivalent resistance *R* of the circuit. It also handles circuit exceptions, such as open circuits.
- **getVoltage:** takes the element to be processed as input and computes the equivalent voltage across it.
- **getCurrent:** takes the element to be processed as input and computes the equivalent current across it.
- **detectShortCircuit**
- **printCircuitAnalysisTable**



```
public class Inductor extends element {
    public Inductor(double inductance) {
        super(inductance);
    }

    @Override
    public Complex getImpedance(double frequency) {
        return new Complex(0, 2 * Math.PI * frequency * value);
    }
}
```

Figure 6: Class **Inductor**

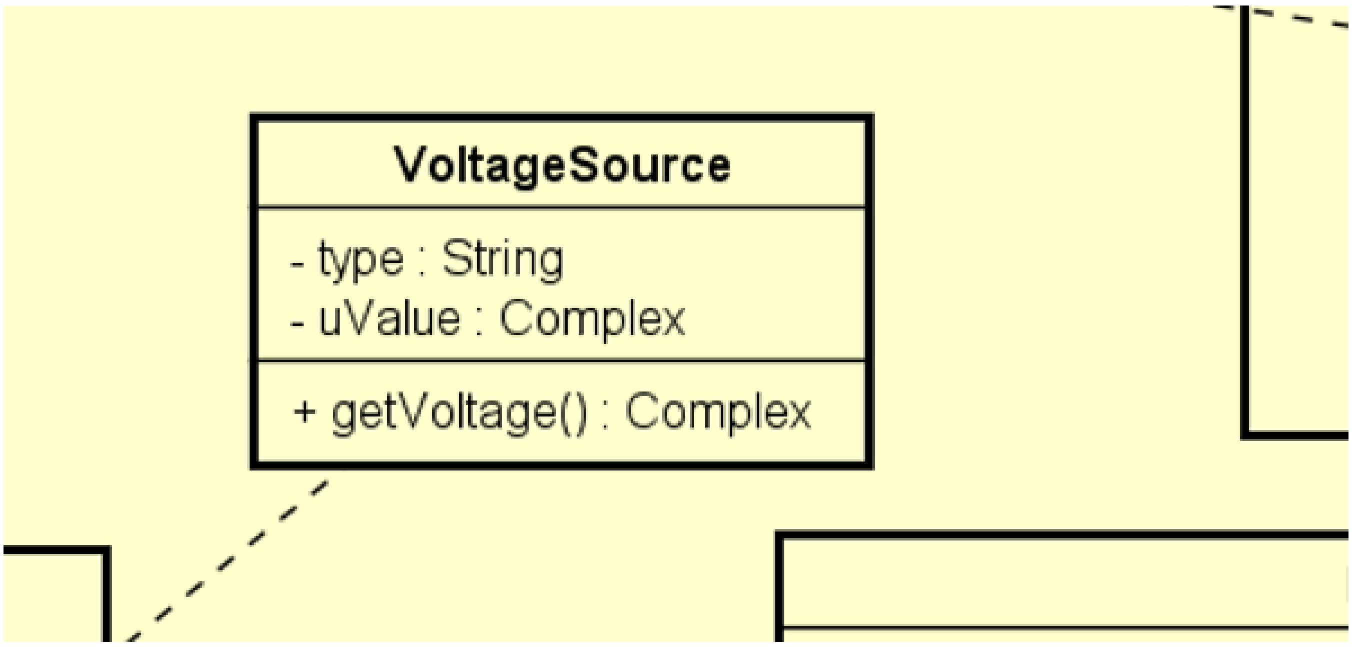


Figure 7: Class **VoltageSource**

3.2.2 Test package

The ‘test‘ package includes files that verify whether the **‘Components‘ package has been correctly implemented**. The test files print results of the equivalent resistance R, voltage U, and current I which all be calculated by EleController. They also demonstrate that **exceptions are handled logically and accurately**.

- **TestAC:** test the analysis when the Circuit Type is AC.
- **TestDC_R:** test the analysis when the Circuit Type is DC and only has Resistor.
- **TestDC_RC:** test the analysis when the Circuit Type is DC and has Resistor and Capacitor.
- **TestDC_RLC:** test the analysis when the Circuit Type is DC and has all components.

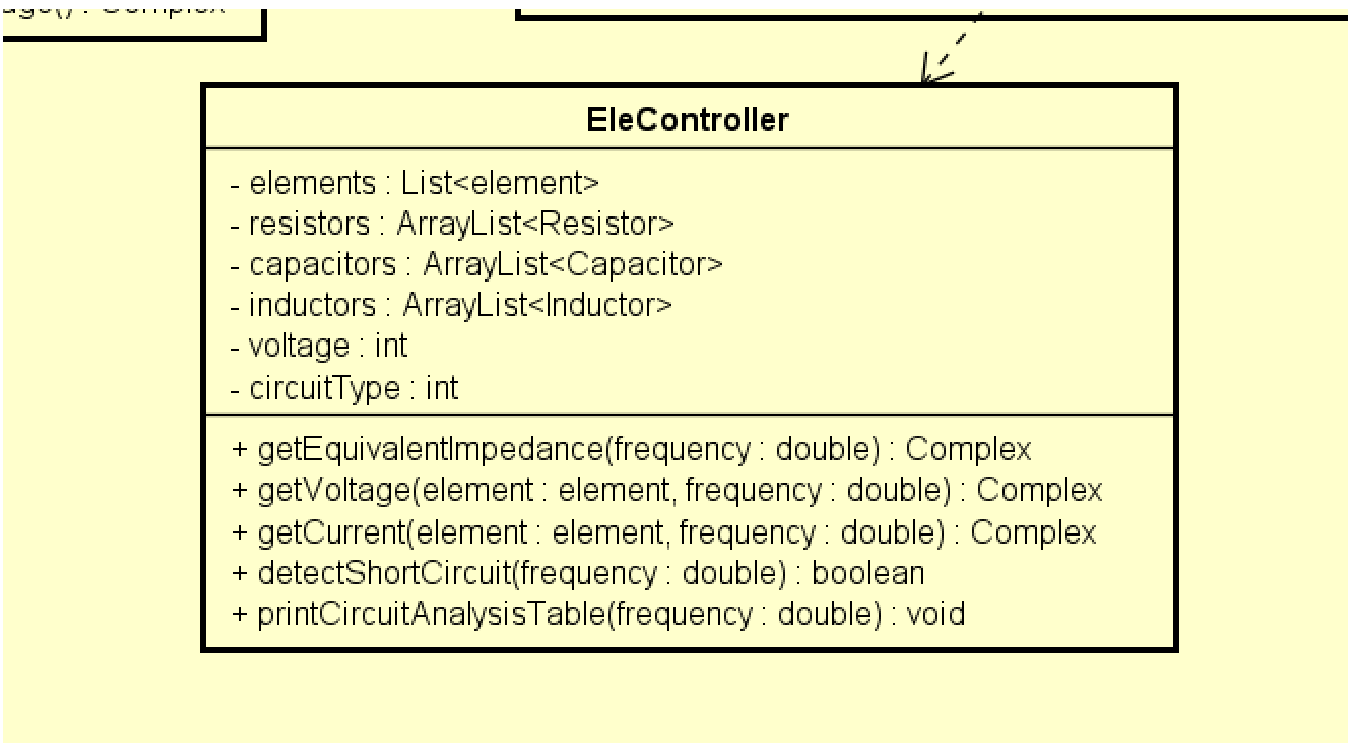

EleController
- elements : List<element>
- resistors : ArrayList<Resistor>
- capacitors : ArrayList<Capacitor>
- inductors : ArrayList<Inductor>
- voltage : int
- circuitType : int
+ getEquivalentImpedance(frequency : double) : Complex
+ getVoltage(element : element, frequency : double) : Complex
+ getCurrent(element : element, frequency : double) : Complex
+ detectShortCircuit(frequency : double) : boolean
+ printCircuitAnalysisTable(frequency : double) : void

Figure 8: Class **EleController**

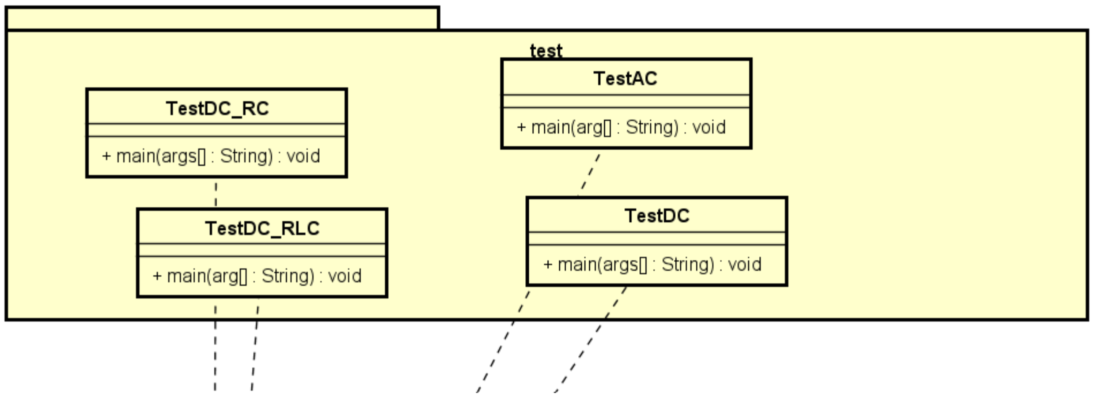
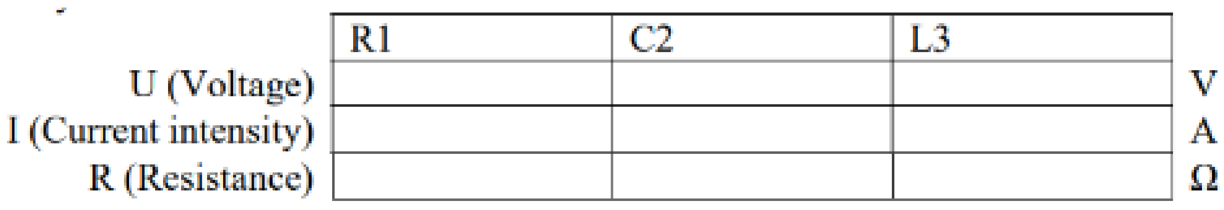

test
TestDC_RC
+ main(args[] : String) : void
TestAC
+ main(arg[] : String) : void
TestDC_RLC
+ main(arg[] : String) : void
TestDC
+ main(args[] : String) : void

Figure 9: Package **test** Class Diagram

This type of circuit is expected as a short circuit.

3.2.3 TableAnalysis package

This package serves as the main package used to generating a electrical circuit analysis table. The analysis table has the following sample:



	R1	C2	L3	
U (Voltage)				V
I (Current intensity)				A
R (Resistance)				Ω

Figure 11: The circuit analysis table's format

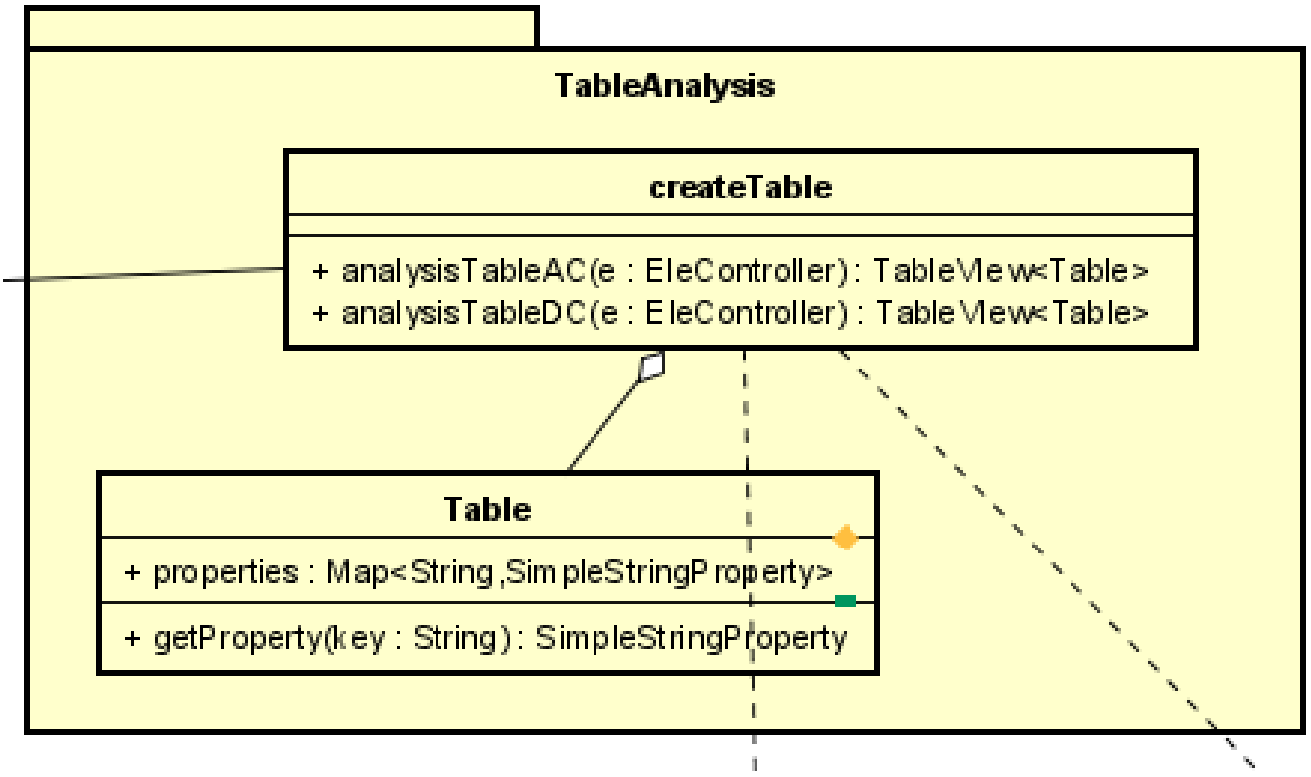

TableAnalysis
createTable
+ analysisTableAC(e : EleController) : TableView<Table>
+ analysisTableDC(e : EleController) : TableView<Table>
Table
+ properties : Map<String,SimpleStringProperty>
+ getProperty(key : String) : SimpleStringProperty

Figure 10: Package **TableAnalysis** Class Diagram

Table This nested class represents a row in the table. Each row has a set of properties (columns) with their respective values.

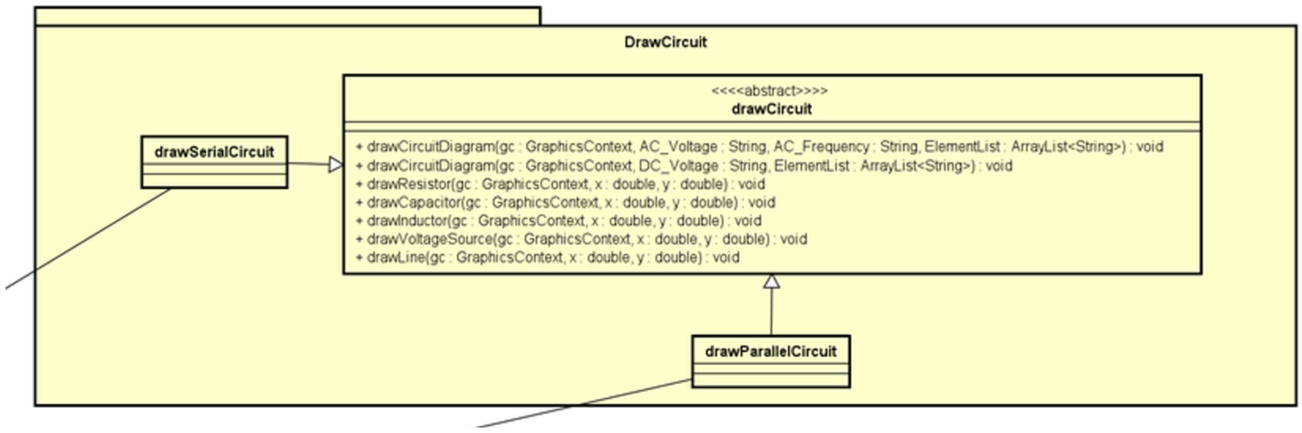
- **Attributes:**
  - **Map<String, SimpleStringProperty> properties = new HashMap<>():** is used to create a map that stores properties of each row in the table. Each entry in this map represents a column in the table, with the key being the column name and the value being a **'SimpleStringProperty'** that holds the data for that column.

- **Methods:**
  - **getProperty():** Get the **'SimpleStringProperty'** associated with the given key.

createTable This class contains nested classes and methods to create and manage a table for displaying circuit analysis data.

- **Methods:**
  - **analysisTableAC(EleController e):** This method creates a TableView for AC circuit analysis.
  - **analysisTableDC(EleController e):** This method creates a TableView for DC circuit analysis.

3.2.4 drawCircuit package



```
+ drawCircuitDiagram(gc : GraphicsContext, AC_Voltage : String, AC_Frequency : String, ElementList : ArrayList<String>) : void
+ drawCircuitDiagram(gc : GraphicsContext, DC_Voltage : String, ElementList : ArrayList<String>) : void
+ drawResistor(gc : GraphicsContext, x : double, y : double) : void
+ drawCapacitor(gc : GraphicsContext, x : double, y : double) : void
+ drawInductor(gc : GraphicsContext, x : double, y : double) : void
+ drawVoltageSource(gc : GraphicsContext, x : double, y : double) : void
+ drawLine(gc : GraphicsContext, x : double, y : double) : void
```

Figure 12: Package **drawCircuit** Class Diagram

The package is designed to create visual representations of electrical circuits. Each method in the `DrawCircuit` class corresponds to a specific element or feature that can be part of a circuit diagram. By implementing these methods in `drawSerialCircuit` and `drawParallelCircuit`, the package can handle both common types of circuit configurations.

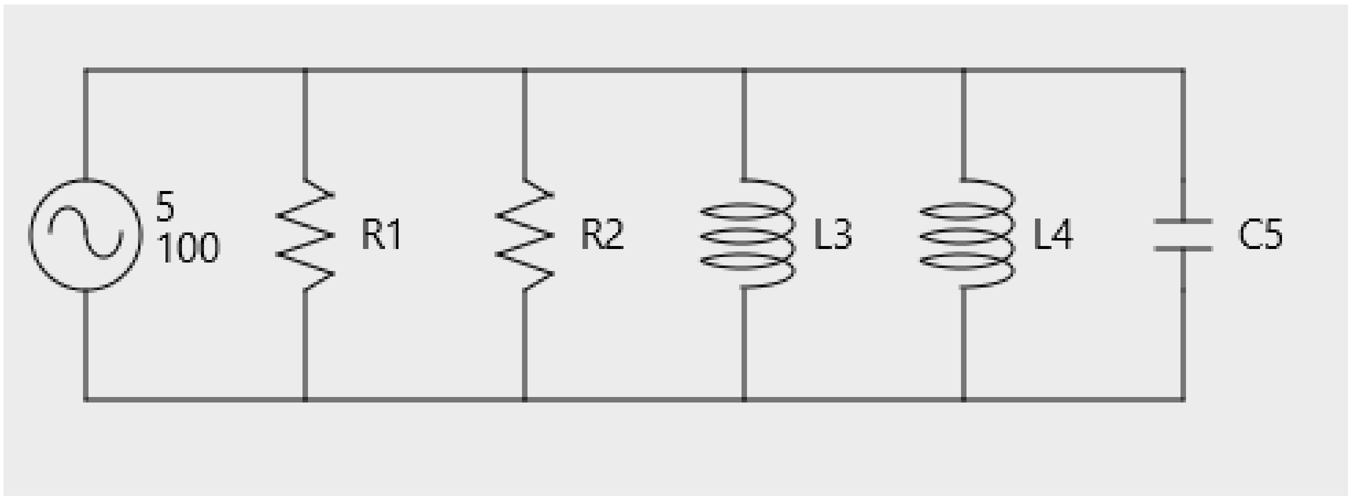


Figure 13: **Paralle Circuit** Diagram

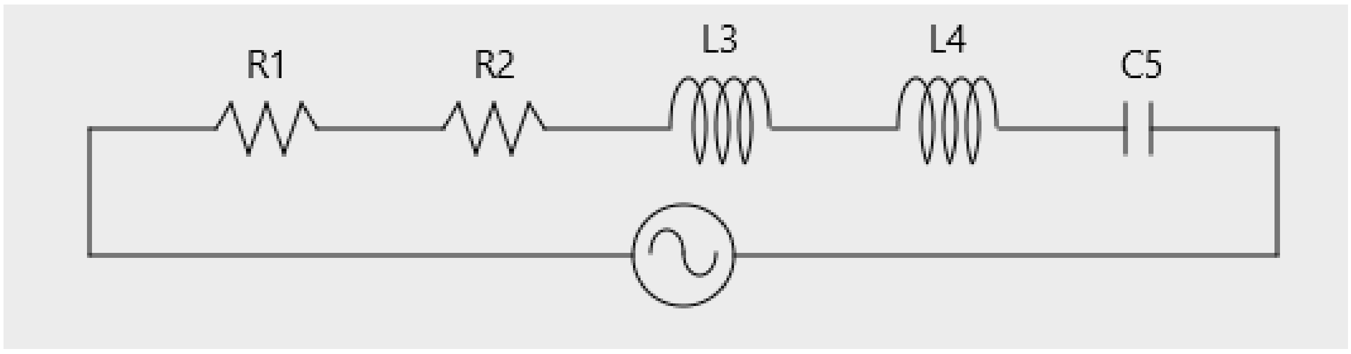


Figure 14: **Serial Circuit** Diagram

drawCircuit is an abstract class containing several abstract methods for drawing various

components of a circuit.
Abstract Methods:

- `drawCircuitDiagram(gc: GraphicsContext, AC_Voltage: String, AC_Frequency: String, ElementList: ArrayList<String>): void`
- `drawCircuitDiagram(gc: GraphicsContext, DC_Voltage: String, ElementList: ArrayList<String>): void`
- `drawResistor(gc: GraphicsContext, x: double, y: double): void`
- `drawCapacitor(gc: GraphicsContext, x: double, y: double): void`
- `drawInductor(gc: GraphicsContext, x: double, y: double): void`
- `drawVoltageSource(gc: GraphicsContext, x: double, y: double): void`
- `drawLine(gc: GraphicsContext, x: double, y: double): void`

These methods are essential for drawing the elements and structure of a circuit diagram, such as resistors, capacitors, inductors, voltage sources, and connecting lines.

drawSerialCircuit class extends the `DrawCircuit` abstract class. It provides concrete implementations for the abstract methods defined in `DrawCircuit`. This class focuses on drawing serial circuits, where components are connected end-to-end.
drawParallelCircuit is similar to `drawSerialCircuit`. This class also extends the `DrawCircuit` abstract class. It provides implementations for drawing parallel circuits, where components are connected alongside each other.

3.2.5 source-fxml package

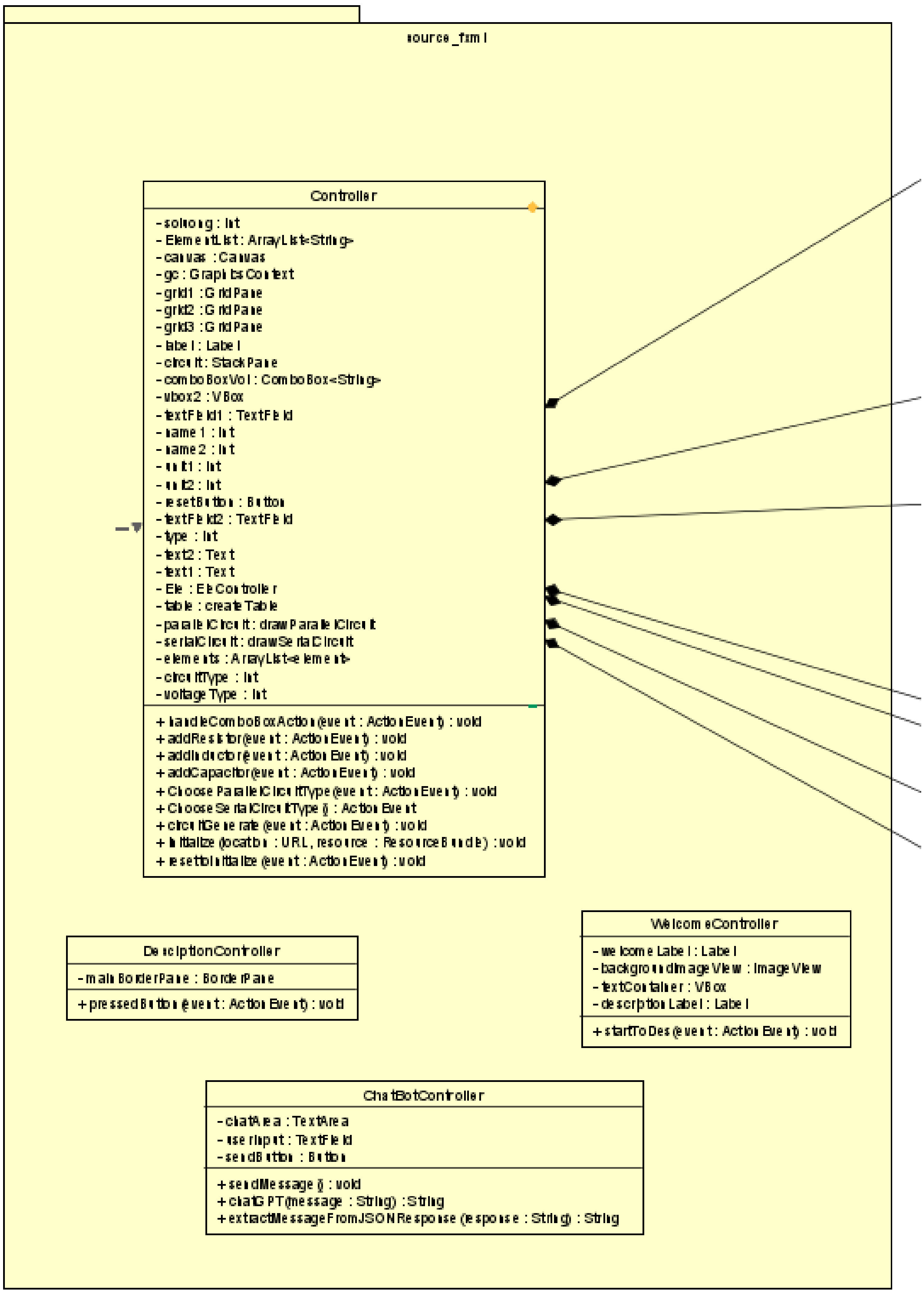

source_fxml
Controller
-solvong : int
-ElementList : ArrayList<String>
-canvas : Canvas
-gc : GraphicsContext
-grid1 : GridPane
-grid2 : GridPane
-grid3 : GridPane
-label : Label
-circuit : StackPane
-comboBoxVol : ComboBox<String>
-vbox2 : VBox
-textField1 : TextField
-name1 : int
-name2 : int
-unit1 : int
-unit2 : int
-resetButton : Button
-textField2 : TextField
-type : int
-text2 : Text
-text1 : Text
-Ele : EleController
-table : createTable
-parallelCircuit : drawParallelCircuit
-serialCircuit : drawSerialCircuit
-elements : ArrayList<element>
-circuitType : int
-voltageType : int
+handleComboBoxAction(event : ActionEvent) : void
+addResistor(event : ActionEvent) : void
+addInductor(event : ActionEvent) : void
+addCapacitor(event : ActionEvent) : void
+ChooseParallelCircuitType(event : ActionEvent) : void
+ChooseSerialCircuitType() : ActionEvent
+circuitGenerate(event : ActionEvent) : void
+initialize(location : URL, resource : ResourceBundle) : void
+resettoInitialize(event : ActionEvent) : void
DescriptionController
-mainBorderPane : BorderPane
+pressedButton(event : ActionEvent) : void
WelcomeController
-welcomeLabel : Label
-backgroundImageView : ImageView
-textContainer : VBox
-descriptionLabel : Label
+startToDes(event : ActionEvent) : void
ChatBotController
-chatArea : TextArea
-userInput : TextField
-sendButton : Button
+sendMessage() : void
+chatGPT(message : String) : String
+extractMessageFromJSONResponse(response : String) : String

Figure 15: Package **source-fxml** Class Diagram

Controller is part of a JavaFX application, responsible for handling user interactions with the GUI. It manages the state and behavior of the application's UI components and orchestrates actions based on user input.

- **Attributes:**
  - **FXML Annotations:** This class links with **sample.fxml** file, so there are fields annotated with @FXML are linked to the components defined in the FXML file, allowing the controller to interact with them directly.
  - **Instance Variables:** These variables hold the state of the application, such as the count of resistors, capacitors, and inductors **(R, C, L)**, the type of voltage **(voltageType)**, the type of circuit **(circuitType)**, and the lists of circuit elements **(CircuitResistor, CircuitInductor, CircuitCapacitor)**.
  - Drawing Circuit Diagrams: These attributes help to build circuit diagrams (**parallelCircuit** and **serialCircuit**).
  - **table:** generating a table of circuit analysis.

- **Methods:**
  - **Initialize()**: is called automatically after the FXML file has been loaded. It sets up the initial state of the UI components.
  - The class has multiple @FXML annotated methods that handle various user actions such as button clicks, combo box selections, and text field inputs... Examples include **'setVoltageType'**, **'resettoInitialize'**, **'circuitGenerate'**, **'openChatbot'**, **'addResistor'**, **'addInductor'**, **'addCapacitor'**, **'ChooseParallelCircuitType'**, and **'ChooseSerialCircuitType'**.
  - **circuitGenerate():** creates and draws circuit diagrams based on the user's input. It distinguishes between AC and DC circuits and calls the appropriate drawing methods (**drawParallelCircuit** and **drawSerialCircuit**).
  - **createTable():** handles both AC and DC analysis tables (analysisTableAC and analysisTableDC).
  - **resettoInitialize ():** resets the UI to its initial state, clearing all inputs and selections.
  - **openChatbot ():** demonstrates how to switch to a new stage with chatbot application.

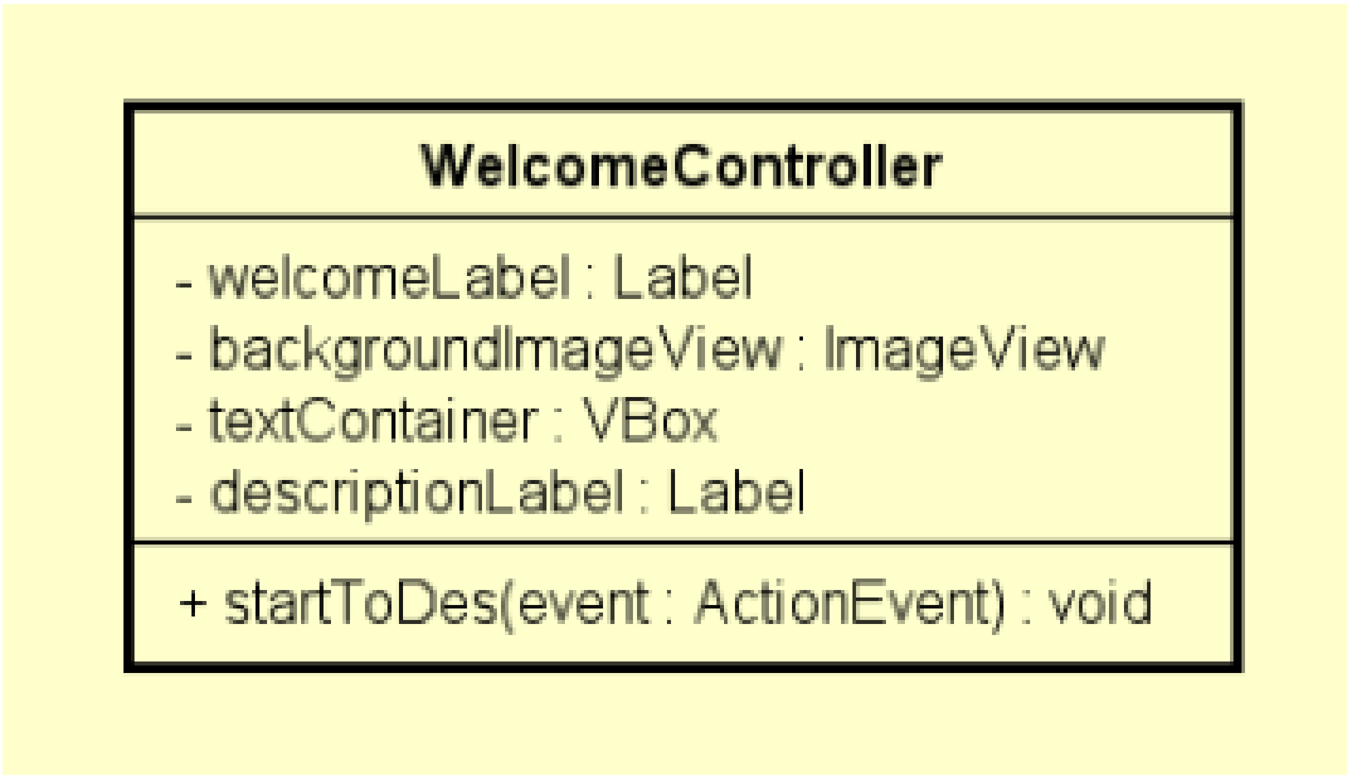

WelcomeController
- welcomeLabel : Label
- backgroundImageView : ImageView
- textContainer : VBox
- descriptionLabel : Label
+ startToDes(event : ActionEvent) : void

Figure 16: **WelcomeController** class

WelcomeController: This class creates a screen that greets the user.

- **Attributes:**
  - **welcomeLabel:** welcome message to the user.
  - **backgroundImageView:** background image in the welcome screen.
  - **textContainer:** a container for holding text elements in a vertical arrangement
  - **descriptionLabel:** additional descriptive text about the application.

- **Methods:**
  - **startToDes(ActionEvent event):** Handles the transition from the welcome screen to the main application interface.

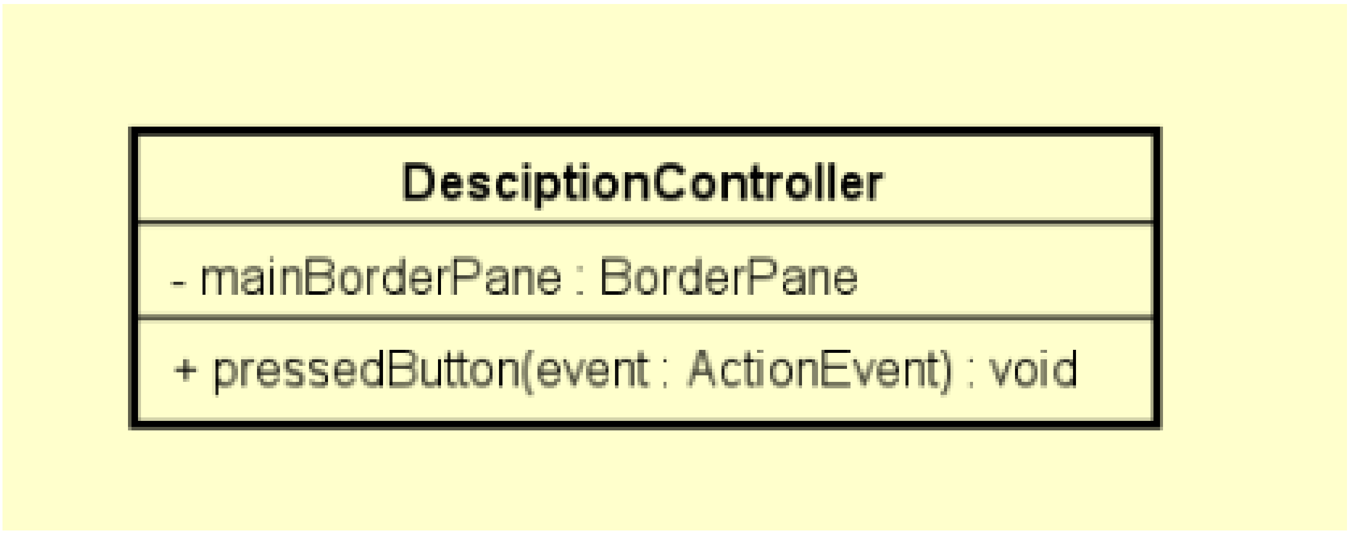

DesciptionController
- mainBorderPane : BorderPane
+ pressedButton(event : ActionEvent) : void

Figure 17: **DescriptionController** class

DescriptionController: This class manages the interactions and transitions for a secondary scene in the JavaFX application. This scene, referred to as the description screen, provides more detailed information before transitioning to the main functionality of the electronic circuit simulatorvisualization.

- **Attributes:**
  - **mainBorderPane:** The main layout container for the description screen

- **Methods:**
  - **pressedButton(ActionEvent event):** Handles the transition from the description screen to the main application interface when a button is pressed.

4 Technology Requirements

4.1 Fundamental technologies

4.1.1 JavaFX

JavaFX is the primary framework utilized for building the graphical user interface (GUI) of the application. It provides a rich set of UI controls, graphics, and media APIs that enable the creation of sophisticated and visually appealing user interfaces. With JavaFX, users can draw circuit diagrams, add and configure various electrical components, and perform real-time calculations seamlessly. The framework's flexibility and comprehensive set of features make it an excellent choice for developing interactive applications.

4.1.2 Object-Oriented Programming (OOP)

The application leverages the principles of Object-Oriented Programming (OOP) to ensure a modular, scalable, and maintainable codebase. By encapsulating data and behavior within classes, OOP promotes reusability and ease of maintenance. Key components of the application, such as circuit elements and user interface elements, are designed as objects, each with specific properties and methods. This approach facilitates the management of complex interactions within the application, making it easier to extend and modify features as needed.

4.1.3 Version Control: Git

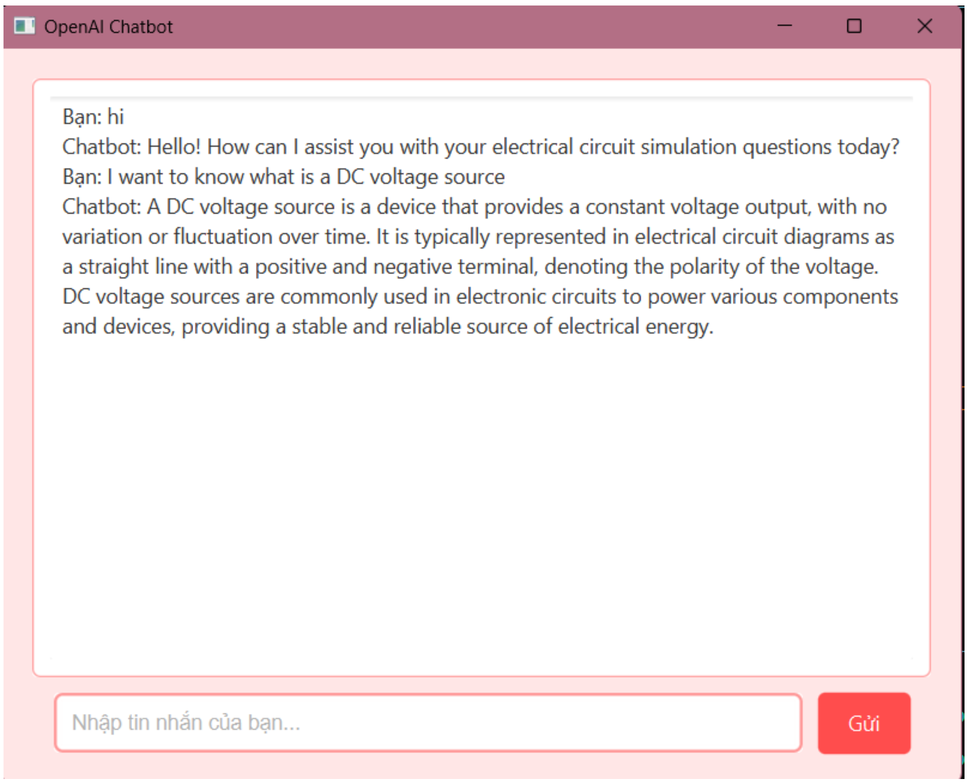
Git is employed as the version control system for managing the application's source code. It allows multiple developers to collaborate efficiently by tracking changes, managing code versions, and resolving conflicts. With Git, the development team can maintain a history of modifications, revert to previous states if necessary, and ensure that the latest updates are consistently integrated. This system is essential for coordinating efforts in a team environment and maintaining the integrity of the project over time.

4.1.4 AI ChatBot: ChatGPT API

The integration of an AI ChatBot, powered by the ChatGPT API, enhances the user experience by providing real-time assistance and guidance. Users can interact with the chatbot to ask questions about circuit design, electrical concepts, and application functionalities. The chatbot offers detailed explanations and solutions, leveraging the advanced natural language processing capabilities of ChatGPT. This feature not only aids users in navigating the application but also serves as an educational tool, helping them understand complex topics more clearly.

4.2 Additional technology - ChatBot using OpenAI's API

In recent years, the deployment of conversational agents, commonly known as chatbots, has seen exponential growth across various industries. One of the forefront technologies enabling this revolution is OpenAI's Assistant API. This API leverages state-of-the-art machine learning models to facilitate natural language processing and understanding, providing robust conversational capabilities.
For specific demand, I recently developed a small chatbot leveraging the ChatGPT API to handle questions related to electrical circuits. This chatbot is designed to assist users by providing accurate and concise answers to their queries on various topics, including circuit components, circuit analysis, and troubleshooting. By integrating the ChatGPT API, I was able to create an interactive and responsive tool that understands and processes natural language queries effectively. The chatbot is now a valuable resource for students and hobbyists looking to deepen their understanding of electrical circuits, offering instant support and enhancing their learning experience.



Bạn: hi
Chatbot: Hello! How can I assist you with your electrical circuit simulation questions today?
Bạn: I want to know what is a DC voltage source
Chatbot: A DC voltage source is a device that provides a constant voltage output, with no variation or fluctuation over time. It is typically represented in electrical circuit diagrams as a straight line with a positive and negative terminal, denoting the polarity of the voltage. DC voltage sources are commonly used in electronic circuits to power various components and devices, providing a stable and reliable source of electrical energy.

Figure 18: OpenAI ChatBot

5 Demo

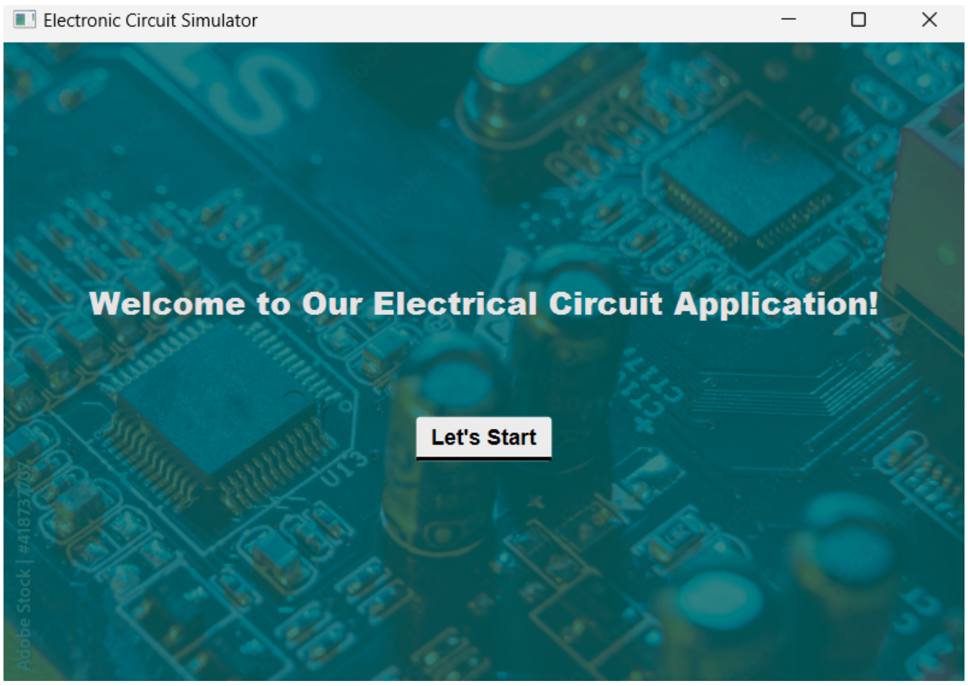
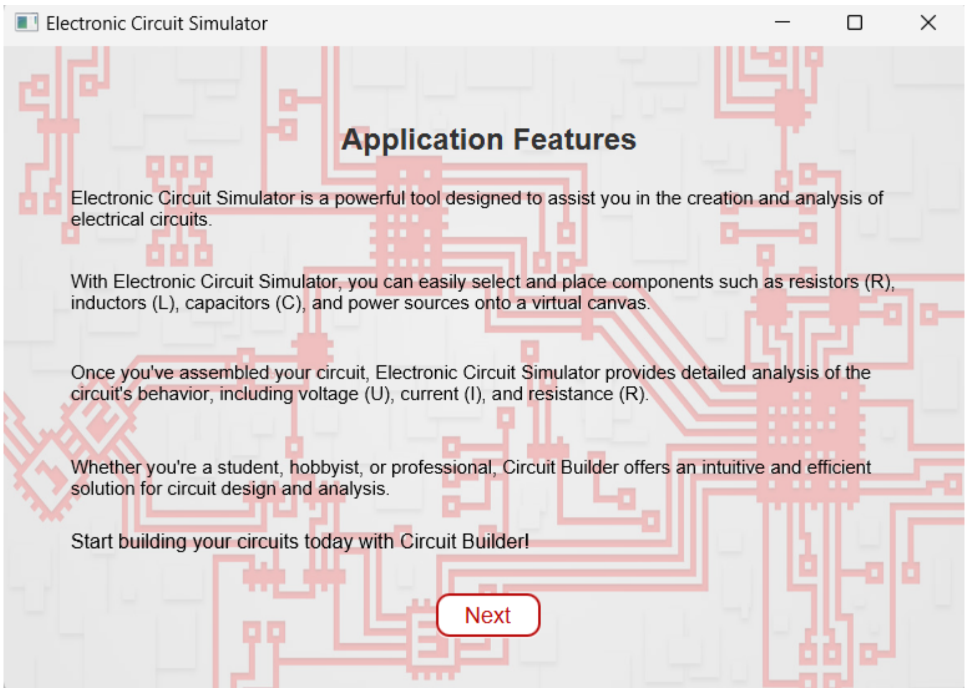

Electronic Circuit Simulator
Welcome to Our Electrical Circuit Application!
Let's Start

Figure 19: Welcome Window



Application Features

Electronic Circuit Simulator is a powerful tool designed to assist you in the creation and analysis of electrical circuits.

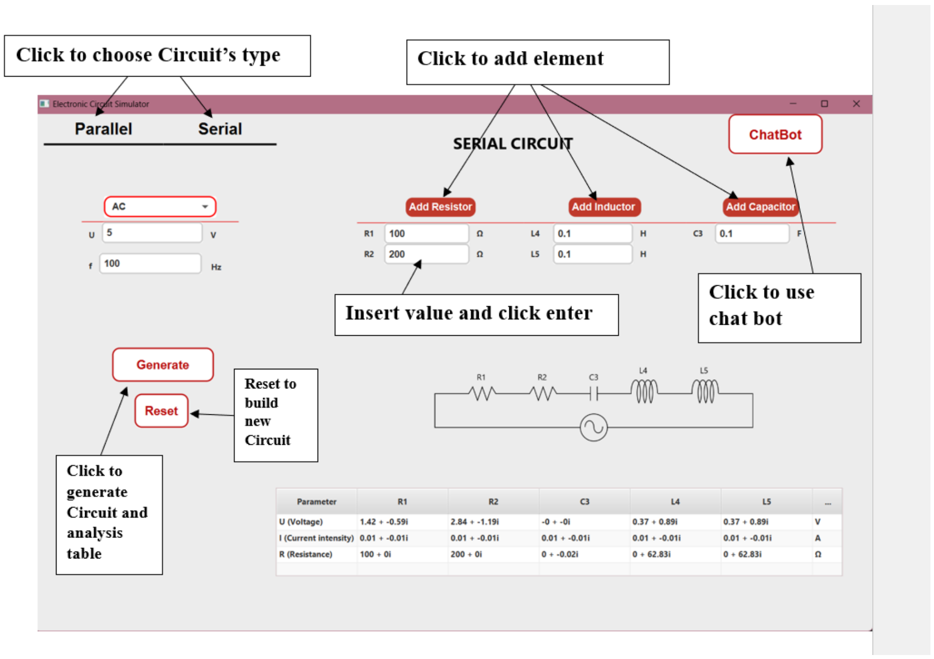
With Electronic Circuit Simulator, you can easily select and place components such as resistors (R), inductors (L), capacitors (C), and power sources onto a virtual canvas.

Once you've assembled your circuit, Electronic Circuit Simulator provides detailed analysis of the circuit's behavior, including voltage (U), current (I), and resistance (R).

Whether you're a student, hobbyist, or professional, Circuit Builder offers an intuitive and efficient solution for circuit design and analysis.

Start building your circuits today with Circuit Builder!

Figure 20: Description Window


Click to choose Circuit's type
Click to add element
Electronic Circuit Simulator
Parallel
Serial
SERIAL CIRCUIT
ChatBot
Add Resistor
Add Inductor
Add Capacitor
Click to use chat bot
Insert value and click enter
Generate
Reset
Reset to build new Circuit
Click to generate Circuit and analysis table
R1
R2
C3
L4
L5

AC

U	5	V
f	100	Hz

R1	100	Ω	L4	0.1	H	C3	0.1	F
R2	200	Ω	L5	0.1	H			

Parameter	R1	R2	C3	L4	L5	...
U (Voltage)	1.42 + -0.59i	2.84 + -1.19i	-0 + -0i	0.37 + 0.89i	0.37 + 0.89i	V
I (Current intensity)	0.01 + -0.01i	0.01 + -0.01i	0.01 + -0.01i	0.01 + -0.01i	0.01 + -0.01i	A
R (Resistance)	100 + 0i	200 + 0i	0 + -0.02i	0 + 62.83i	0 + 62.83i	Ω

Figure 21: OpenAI ChatBot