HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

*MINI PROJECT REPORT*

**Electrical Circuit Simulator**

Object-Oriented Language and Theory

May 26th, 2021

Team 1:

Phan Nguyên Anh - 20194727

Ngô Quốc Thắng – 20194839

Trần Đức Thắng – 20194842

Lecturer:

Nguyen Thi Thu Trang Ph.D.

Table of Contents

[I. Assignment of members 1](#_Toc73555023)

[1. Phan Anh 2](#_Toc73555024)

[2. Quoc Thang 3](#_Toc73555025)

[3. Duc Thang 5](#_Toc73555026)

[II. Mini-project description 9](#_Toc73555027)

[1. Requirement: 9](#_Toc73555028)

[2. Use case diagram and explanation: 10](#_Toc73555029)

[III. Design 11](#_Toc73555030)

[1. General Class Diagram 11](#_Toc73555031)

[2. Package Class Diagram 12](#_Toc73555032)

[3. Explanation of the design 15](#_Toc73555033)

# Assignment of members

**Members Assignment**

|  |  |  |
| --- | --- | --- |
| **Full name** | **Student ID** | **Assignments** |
| Phan Nguyên Anh | 20194727 | * Use case Diagram * Source code of backend Package * Source code of complex Package * Package Class Diagram for above packages |
| Ngô Quốc Thắng | 20194839 | * General Class Diagram * Use case Diagram * Source code of guiWindow.input Package * Source code of guiWindow.output Package * Source code of circuit Package * Package Class Diagram for above packages * Linking outputs with inputs of modules * Extension on remove components |
| Trần Đức Thắng | 20194842 | * Source code of components Package * Source code of guiWindow.drawcircuit Package * Source code of guiWindow.output Package * Source code of circuit Package * Package Class Diagram for above packages * Extension on draw circuit method to be more   maintainable and customizable. |

***For all remaining documents, we do together.***

Now we will describe in detail what did each of members do.

# Phan Anh

**Package complex**

**Complex.java – Use for defining and computing imaginary number**

Has 2 attributes named **re** and  **im** which represent the real part and imaginary part of a complex number.

**private** **final** **double** re;

**private** **final** **double** im;

The Constructor Method:

**public** Complex(**double** real, **double** imag)

Getters:

**public** **double** re() {

**return** re;

}

**public** **double** im() {

**return** im;

}

Some Instance Methods:

|  |  |
| --- | --- |
| Method name and return Type | Description |
| **public** String toString() | A string shows user the value of a complex number (rounding depends on user, here I set rounding to 3 decimal places) |
| **public** Complex plus(Complex b) | This plus a complex number b  (a + bi) + (c + di) = (a+c) + (b+d)i |
| **public** Complex minus(Complex b) | This minus a complex number b  (a + bi) - (c + di) = (a-c) + (b-d)i |
| **public** Complex times(Complex b) | This times a complex number b  (a + bi) x (c + di) = (a x c – b x d) + (a x d + b x c)i |
| **public** Complex scale(**double** alpha) | This scale alpha  (a + bi ) x alpha = (alpha x a) + (alpha x b)i |
| **public** Complex reciprocal() | 1/ this  1/(a+bi) = (a-bi) / (a+bi)\*(a-bi) = (a-bi)/( a\*a + b\*b) |
| **public** Complex divides(Complex b) | This divides a complex number b = this times b.reciprocal()  This / infinity = 0  This / 0 = infinity |
| **public** **boolean** equals(Object x) | If 2 objects are equal ?  2 objects are equal if this.re = x.re, this.im = x.im  X null -> false  Different class -> false |

**Package backend**

**Calculate.java – Use for calculating the triple-value (R,U,I) for each element**

Has a single attribute named **circuit –** the one that we need to analyze.

**private** Circuit circuit = **new** Circuit();

The Constructor Method:

**public** Calculate(Circuit circuit)

In this Method, I divide into two cases for easier computation:

* if the circuit is serial: Calculate Resistor (R element) for each component -> Equivalent Resistor of circuit -> Calculate I (same I) -> Calculate U
* if the circuit is parallel: Calculate Resistor (R element) for each component -> Equivalent Resistor of circuit -> Calculate U (same U) -> Calculate I

Some Instance Methods:

|  |  |
| --- | --- |
| Method name and return Type | Description |
| **public** **void** CalculateR() | Calculate the R element for each component of the circuit |
| **public** Complex CalculateReq() | Calculate equivalent Resistor of the circuit |
| **public** **void** CalculateI() | Calculate the I element for each object of the circuit |
| **public** **void** CalculateU() | Calculate the U element for each object of the circuit |

# Quoc Thang

In my assignment, all I had done is frontend of the mini project.

I will divide it into 4 parts to explain:

* The main frame.
* The main panel.
* The Circuit panel.
* The Table panel.

In detail,

* The main frame is created firstly, it will contain 2 main panels (2 tabs) parallel and serial for user to choose one of them.
* In each main panel
* I initialize a label SOURCE and a combo box (AC, DC) so user can easily pick 1 of 2 source types for the circuit. If user picks DC, a label and text field VOLTAGE will pop up. Otherwise with AC, 2 labels and text fields for VOLTAGE and FREQUENCY will pop up. And another thing to do in this part is when changing between 2 sources, I have to create a variable to mark that whether the source is picked before or not.
* About 3 buttons to add components (Add Resistor, Add Capacitor and Add Inductor), firstly it has a position right below the text field of Source. After clicking 1 of 3 buttons, a label and text field of corresponding component appears with its name convention. For example, clicking Add Capacitor raises a label C1. And continue until user add maximum 5 components, the application will not allow user to add more component. The most complicate thing in this part is when user choose to add a component, it will generate a label and text field at right that position (position of button) and repaint and create 3 buttons in 3 new positions below the text field. Everything for this part is implemented in BtnAddListener class (inner class of class Panel).
* About 2 remove buttons and submit button, 2 remove buttons is an extension. Function of the 1st remove button (Remove Last Component) is removing the last component that user add into the circuit and function of the remain button (Remove All Components) is removing all the components that user added into the circuit. To implement this, we have to remove the labels and text fields of them and add 3 buttons right the position of labels and text fields at the same time. Otherwise, the button Submit has to collect all information of Source, Components and pass to constructor of Table Panel to show the analysis table. All are implemented in method addActionListener of these 3 buttons.
* The circuit panel is taken the circuit input of main panel and use method DrawCircuit (which will describe in detail in Duc Thang’s part) to draw the circuit. In main panel, I implement a feature that the circuit will update corresponding to what user has done (adding, removing and changing source) which is in BtnPreSubmitListener class (inner class of class main panel).
* After submitting, the analysis table will pop up. To implement this, I create a GUI table in the panel and add into it by column. Each column has 4 rows which are name convention of component, Voltage (U), Intensity current (I) and Z (impedance). Finally, adding 3 labels (3 units of U, I and Z) V, A and Ohm next to the last column. Everything is implemented in class TablePanel. The analysis table only appears after user presses Submit button.

# Duc Thang

a) package circuit:

* Include class Circuit.
* Since a circuit will have 2 types of connection (parallel or serie) and it’s contains a source (can be AC source or DC source) and components (inductor, capacitor, resistor) so I have to create some attribute to define the circuit:
* **Boolean connectType** to define the connection type of the circuit, with value false if the circuit is parallel and true if it connected by serie.
* **ArrayList<RLCcomponent> components** to store components that the circuit contains.
* **Source source** to store the object source of the circuit.
* Because to calculate the required result, we need to know what is the current circuit connect type (serie or parallel) and the source type (AC or DC) so I have to create some getter to get these value:
* **getSourceType():** return true if source type is AC, false if DC.
* **getComponents()**: return an ArrayList contains all the RLC components in the circuit.
* **getConnectType():** return the connect type of the circuit.
* And since we have to add each RLC component into circuit object, so I have to create a method for it to add component:

**addComponent(RLCcomponent rlccomponent)**

* To draw the circuit figure, I implemented a method to draw which is:
* **DrawCircuit(Graphics g)** : But since this method is quite complex and have direct relation to the components package and guiWindows.drawcircuit package so I will explain my approach to draw the circuit in the “draw circuit explanation”.

b) package components:

* In this package, I define classes for all components that a circuit will or may have which include:
* Class **Resistor** inherited from class **RLCcomponent**
* Class **Inductor** inherited from class **RLCcomponent**
* Class **Capacitor** inherited from class **RLCcomponent**
* Class **ACsource** inherited from class **Source**
* Class **DCsource** inherited from class **Source**
* Since these 5 components can be divide into two main type of components which are **Source** component and **RLCcomponent**. However, we do not need object of **Source** and **RLCcomponent** class, so I created them as abstract. The different between them is:
* Class **Source**: have an attribute Frequency
* Class **RLCcomponent**: have attribute U, I, Z (is Complex number to store the required result. This will be explain more clearly in package backend and package complex).
* Also because all component (all components of both RLCcomponent and Source class) has attributes **name** and **spec** (store value of the component) and having methods **drawComponent(Graphics2D g2D, double startX, boolean ConnectType), getComponentImage()** (These method will be explain in “draw circuit explanation”). So, I create a parent abstract class **Component** for the sub class **Source** and **RLCcomponent** to inherit. By design the inheritance tree like this, I can implement drawing method to draw the circuit more logically which will be explain more clearly in the “draw circuit explanation” below.

c) package guiWindows.drawcircuit

* This package contains 2 classes: **CircuitPanel** and **SpecSetting**.
* The purpose of this package is to create a JPanel that draw the circuit figure.
* **CircuitPanel** class: extends JPanel. And because this class inherit JPanel so it has a **paint(Graphics g)** method which to draw on the Panel. So, to draw the circuit figure, I need to override the paint(Graphics g) method and inside this method I will call the **DrawCircuit()** method of class circuit:

circuit.DrawCircuit(g2D);

* **SpecSetting** class: to store parameters which define how the circuit should be display (scale of the figure, width, Height of the container, ….)

* To understand more of how I implement algorithm to draw the circuit figure, read the “draw circuit explanation” title below:

d) draw circuit explanation:

* Graphical user interface, application

  Description automatically generatedTo understand how I draw the circuit, first we need to understand about **paint()** method of Container class. The **paint()** method is an method that user will redefine so that container know what should be draw and how to draw it, this method has a parameter **Graphics g** which controls the Graphics context in which to paint. So if I call an outside method to show how the paint(Graphics g) should paint, I have to pass in the current **g** parameter to the method. In this case:
* Instead of using Graphics type which is quite limited, I used Graphics2D which has more support to draw content onto the container.
* My approach to draw circuit algorithm has basically 3 steps:
* Step 1: calculate the point it should draw the first component depends on which type of the connect type of the circuit (parallel or serie), how many components are there in the circuit, and some other constant such as: how large spacing between components, scale of the figure, …. (all of this constant can be change in the **SpecSetting** class). This calculation purpose is to display the figure at the center of the container.
* Step 2: After calculated the start drawing point (**startX** parameter), It then draw the source image. Also, with the connect wire to connect all of the components.
* Step 3: This is the last step, depending on which connection type of the circuit then the component will have a different way to draw but the main idea is for every RLCcomponent in the circuit, it will call the **drawComponent()** method of that component with the parameter passed in to indicate the point it should draw the component(**startX** parameter), and how it should draw the component (parallel or serie) then after draw the component, startX parameter being recalculated to calculate the point to draw the next component.
* And because Resistor, Capacitor, Inductor have the same way to draw, the only difference between them is the image of the component. So I define how RLC component draw (**drawComponent()** method) in the abstract class **RLCcomponnent**. Inside this method, it’s invoke another method to get the image of the component is drawing (**getComponentImage()** method) and this method is overridden in the sub classes of RLCcomponent since each component has a difference image.
* The same approach to **Source**, **ACsource**, **DCsource** classes.
* This is also the reason I designed the hierarchy inheritance tree of package components like I explained in the package components.
* Also, the way I wrote these codes, I used no fixed number to write code to draw the circuit. Every parameter is calculated relatively to the constant defined in class **SpecSetting** for easier update, changing in the future. For example: if I want the figure scale bigger, I just need to change the scale constant in the class SpecSetting that all, no need to change any code in the draw method itself.

# Mini-project description

# Requirement:

* Frontend
* Create friendly GUI for user to input parameters.
  + - Demonstrate two types of circuits: parallel circuit and serial circuit.
  + User can pick a type of circuit by choosing one or two tabs in navigation bar.
  + User can choose the source type (AC or DC) from a combo box.
  + User can add or remove a component by pressing buttons.
  + User can define the value of components (source or RLC components) using text field.
  + User can start calculating and printout the result table by pressing submit button.
* Can build the electrical circuit corresponding to user’s inputs.
* Can handle exceptions for invalid input.
* Display results table to analyze.
* Can display all the required information.
* Backend
* Accurately calculate results.
* Handle special exception cases.

# Use case diagram and explanation:



Modify Circuit Type: User will choose one of two circuit types (parallel or series)

Modify Source of the circuit: User choose AC or DC Source with its value (Voltage and Frequency for AC or Voltage for DC).

Add Component to the circuit: User can add up to maximum of 5 components

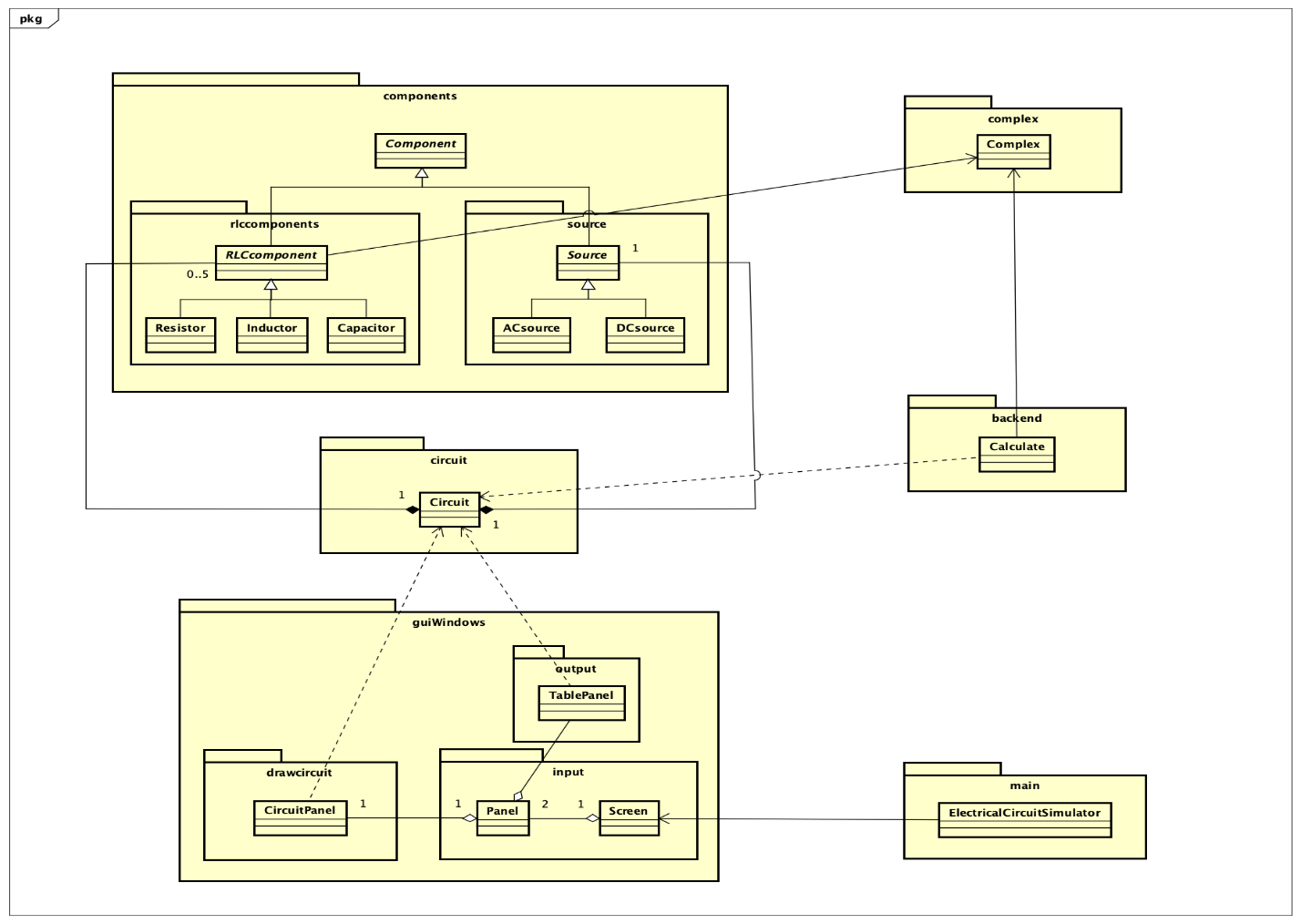
(Resistor, Inductor or Capacitor) with its value.

Remove Component from the circuit: User can remove the last component of the current circuit or remove all of them.

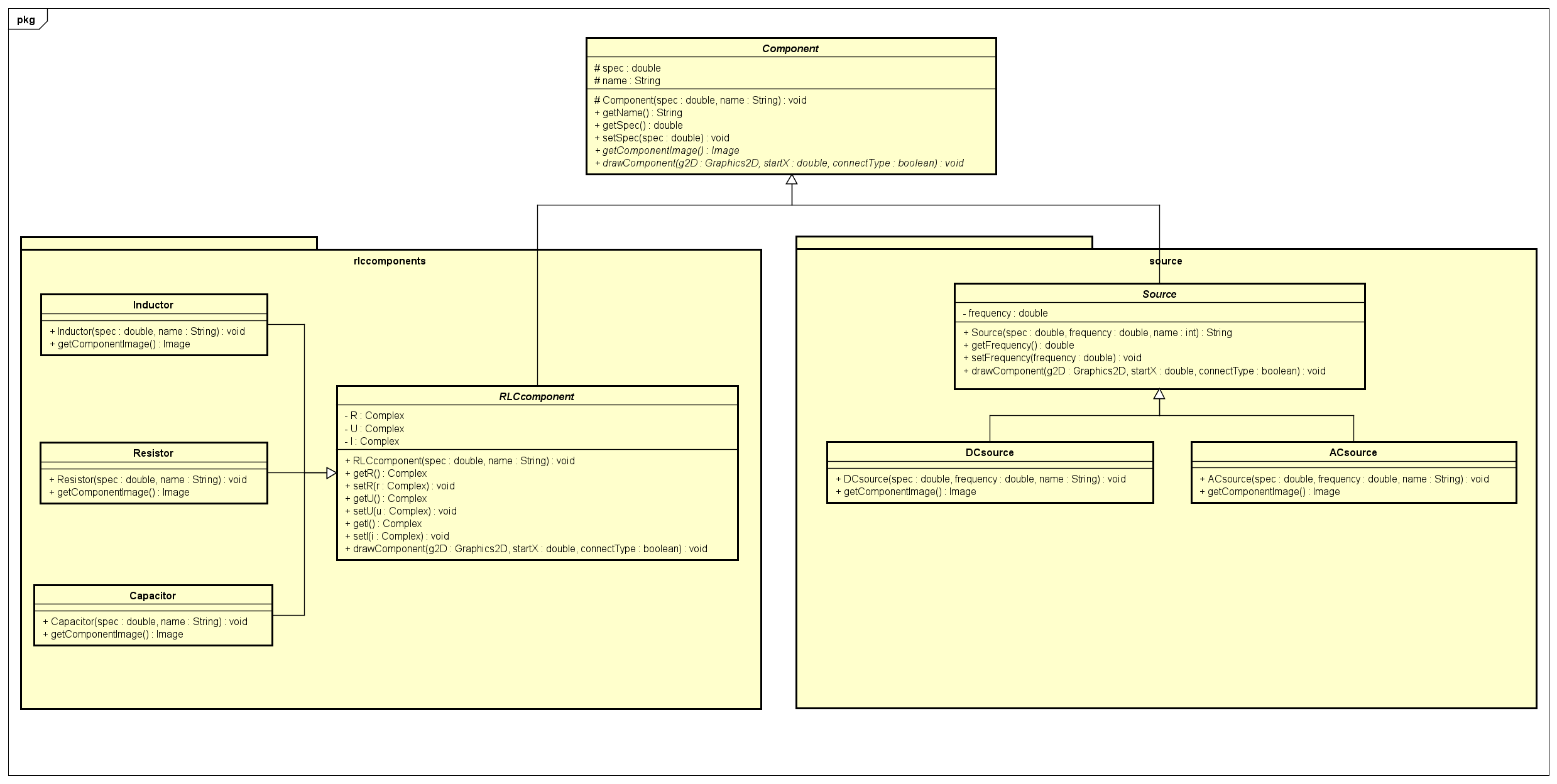
View circuit analysis: User can view the circuit figure corresponding to its components. After pressing submit button, the user can view the results table to analyze.

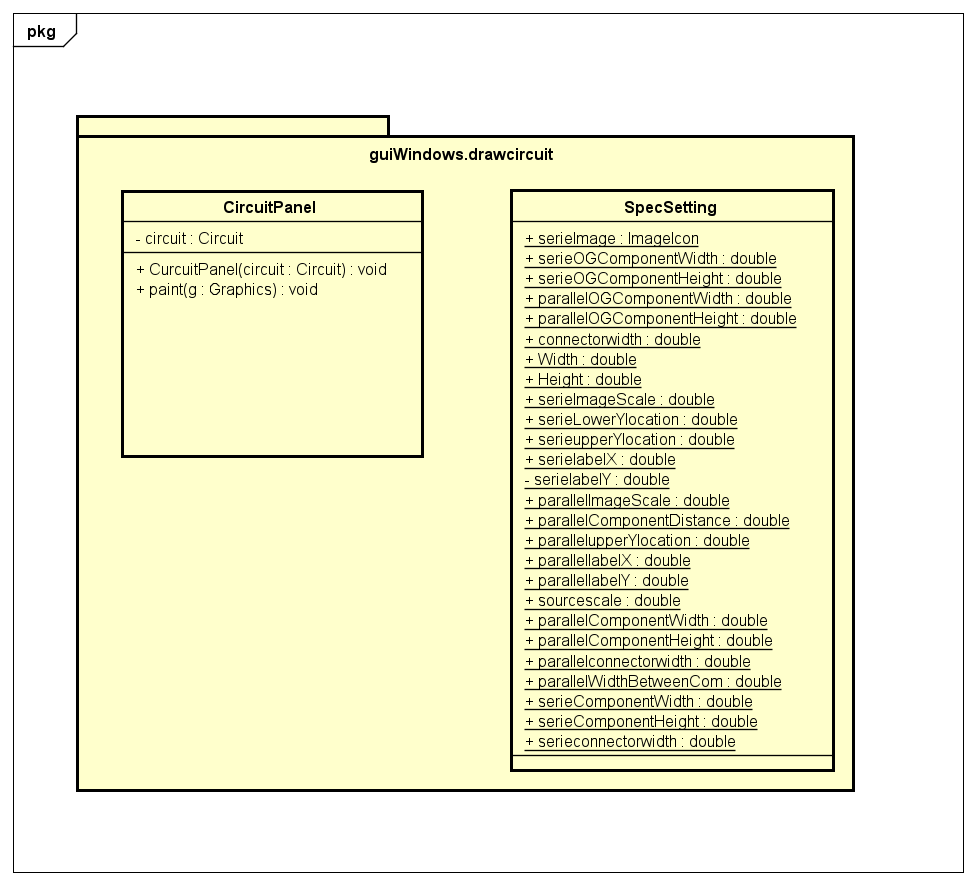
# Design

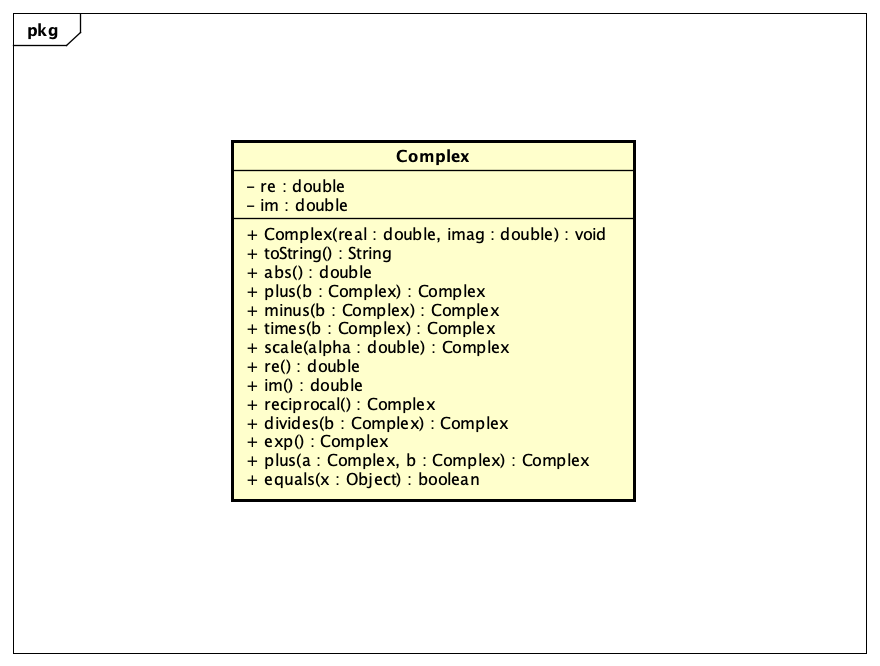
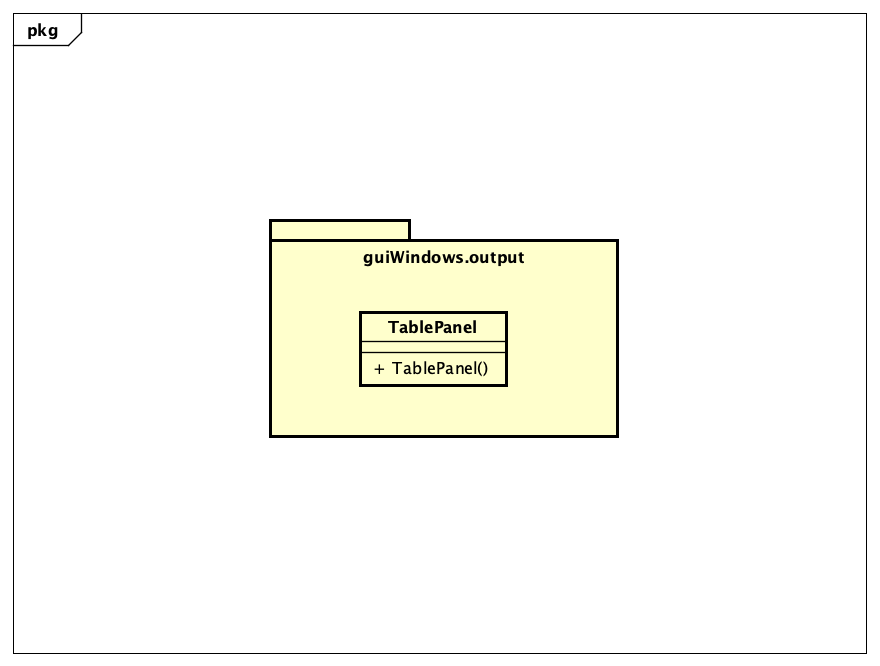
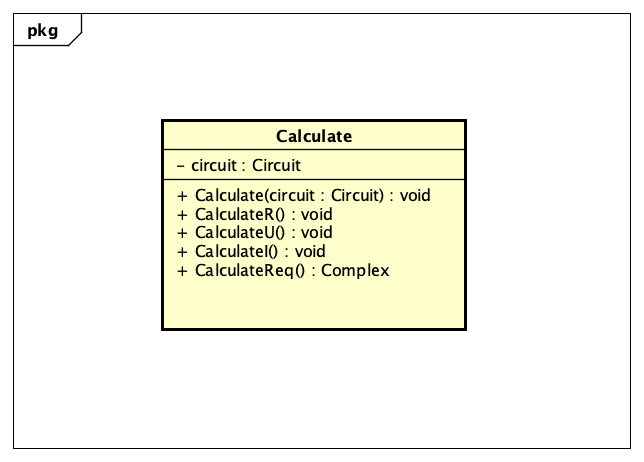
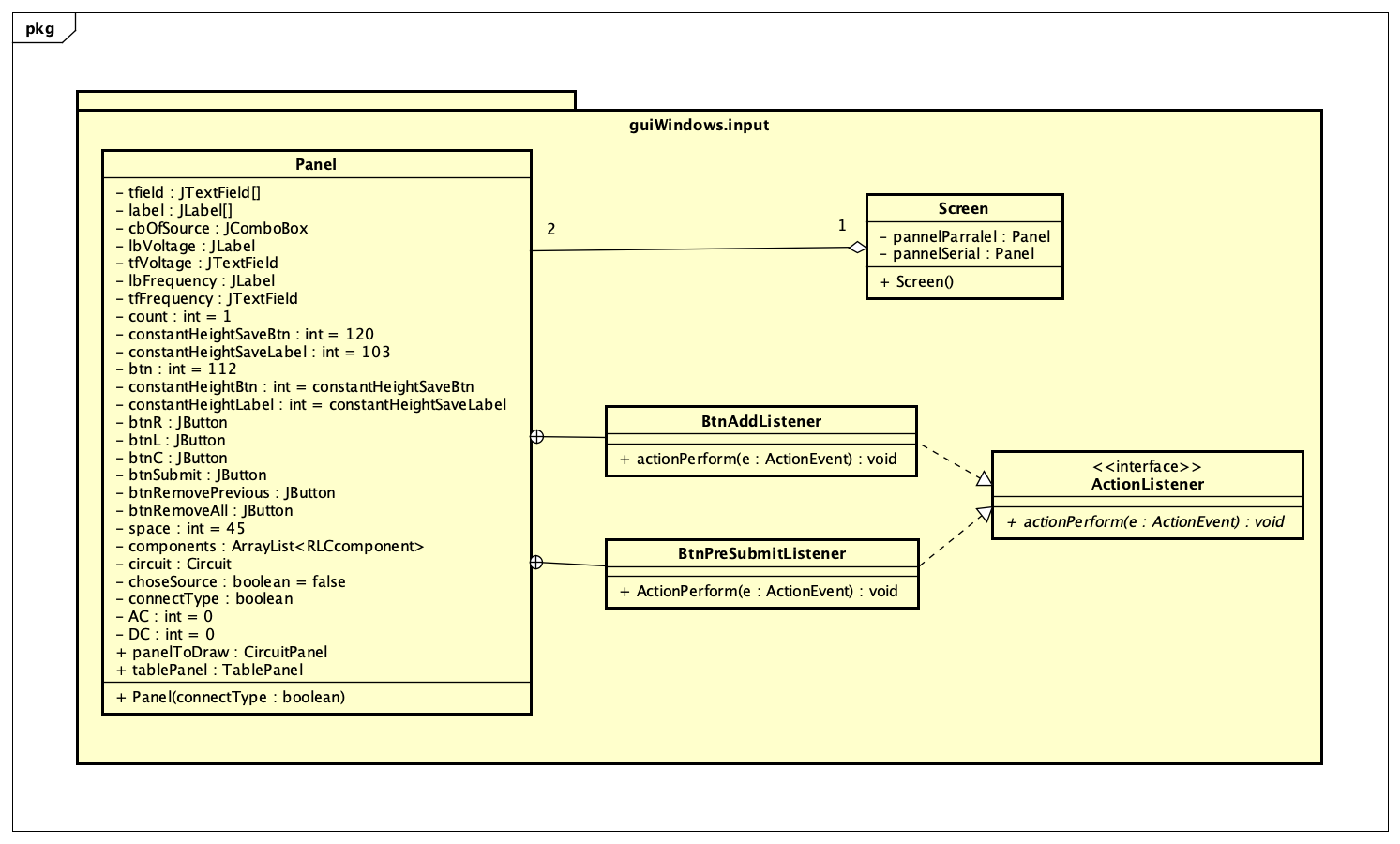
# General Class Diagram

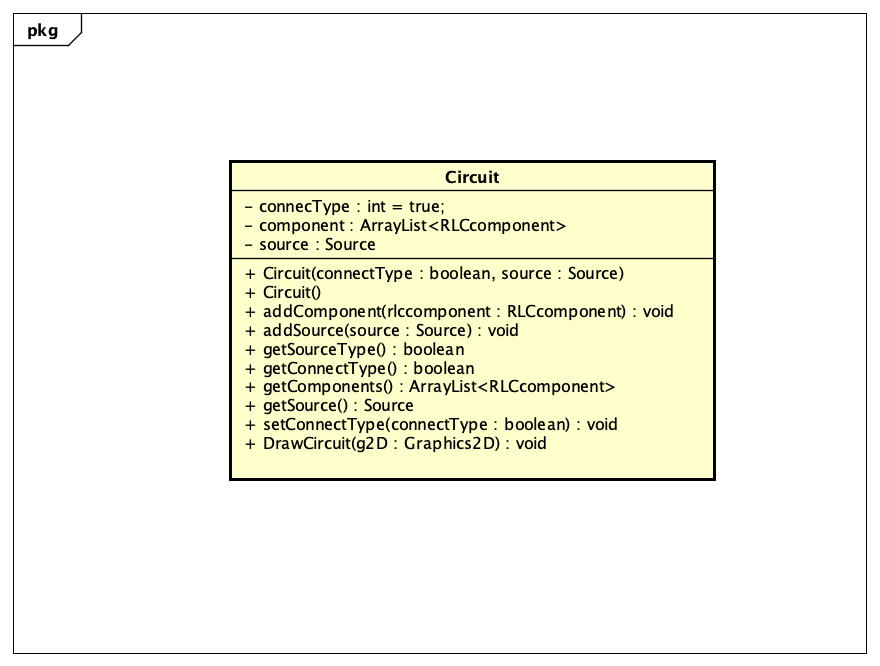


# Package Class Diagram









# Explanation of the design

* General Class Diagram Explanation:

We want to divide the application into 4 main modules.

* Input module: Package guiWindows.input

Purpose: Allow user to input parameters

Input: Circuit parameters

Output: An instance of circuit object

* Draw Circuit module: Package guiWindows.drawcircuit

Purpose: Display the circuit figure

Input: An instance of circuit object

Output: A panel includes the input circuit figure

* Calculate module: Package backend and complex

Purpose: Calculate accurately the results

Input: An instance of circuit object

Output: Calculate the required values in complex form

* Output module: Package guiWindows.ouput

Purpose: Display the analysis table

Input: An instance of circuit object after being calculated

Output: The analysis table

* Package Class Diagram Explaination:

Because the circuit has its components (source and RLC components) so we separate it into 2 **packages** (circuit and components) for easier management.

Approaching components package, we use **inheritance** in OOP to utilize source code and be more logical. Since all components have 2 identical attributes which are name and value and same methods also. Although components class has 2 major subclass which are source and RLC component, we do not need instances of these 3 classes (included components class itself). So, we implemented them as **abstract** class.

When applying 4 modules in the mini-project, we often use **Polymorphism** in OOP to upcasting and down-casting objects (specifically when using components object) to process data.

* Frontend implementation (some important methods):

Draw circuit method: implemented in circuit class and component class. When we call the method, it will call draw method of each components.

Main panel (in input package): divide screen into 3 main areas (input area, table analysis area and circuit figure area).

* Backend implementation (important class/method):

Calculate class: When we create an instance of this class (need to pass in a circuit object to constructor) then this class will calculate the result for the passed in circuit (the result is stored in the attribute of the components in the circuit object).