HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

LOGIC EXPRESSION NORMALIZER

FINAL PROJECT REPORT

Object Oriented Languge and Theory - Lab

lecturer:

Tran The Hung

group 36:

Lê Thanh Tùng 20226070

Nguyễn Tài Huy 2018 4372

June 23rd, 2024

Table of Contents

[**A.** **Description** 4](#_Toc73810071)

[I. Mini-project requirements 4](#_Toc73810072)

[1. Overview: 4](#_Toc73810073)

[2. Specifications: 4](#_Toc73810074)

[II. Use case diagram 4](#_Toc73810075)

[**B.** **Design idea** 6](#_Toc73810076)

[I. The Model – View – Controller design patterns 6](#_Toc73810077)

[II. General class diagram 8](#_Toc73810078)

[III. Class diagrams and explanation 9](#_Toc73810079)

[1. Model package: 9](#_Toc73810080)

[2. View package: 11](#_Toc73810081)

[3. Controller package: 12](#_Toc73810082)

[4. OOP-Principle: 13](#_Toc73810083)

[5. WorkFlow: 15](#_Toc73810084)

**MEMBERS ASSIGNMENT**

|  |  |  |
| --- | --- | --- |
| **Full name** | **Student ID** | **Assignments** |
| Lê Thanh Tùng  Nguyễn Tài Huy | 20226070  20184372 | * Designed the Model. * Developed the input and output screens. * Created the tables. * Implemented all the classes to build a complete model. * Wrote all the methods in the Controller to streamline the project workflow. * Documents and Slide |

# **Description**

## Mini-project requirements

### Overview:

Create an application to normalize a logic expression using the implementation of the Quine-McCluskey method. The final expression must have the least number of terms. The fewer the number of implicants are, the higher reliability and lower manufacturing cost. This application accepts three-variable or four-variable inputs without don’t care terms.

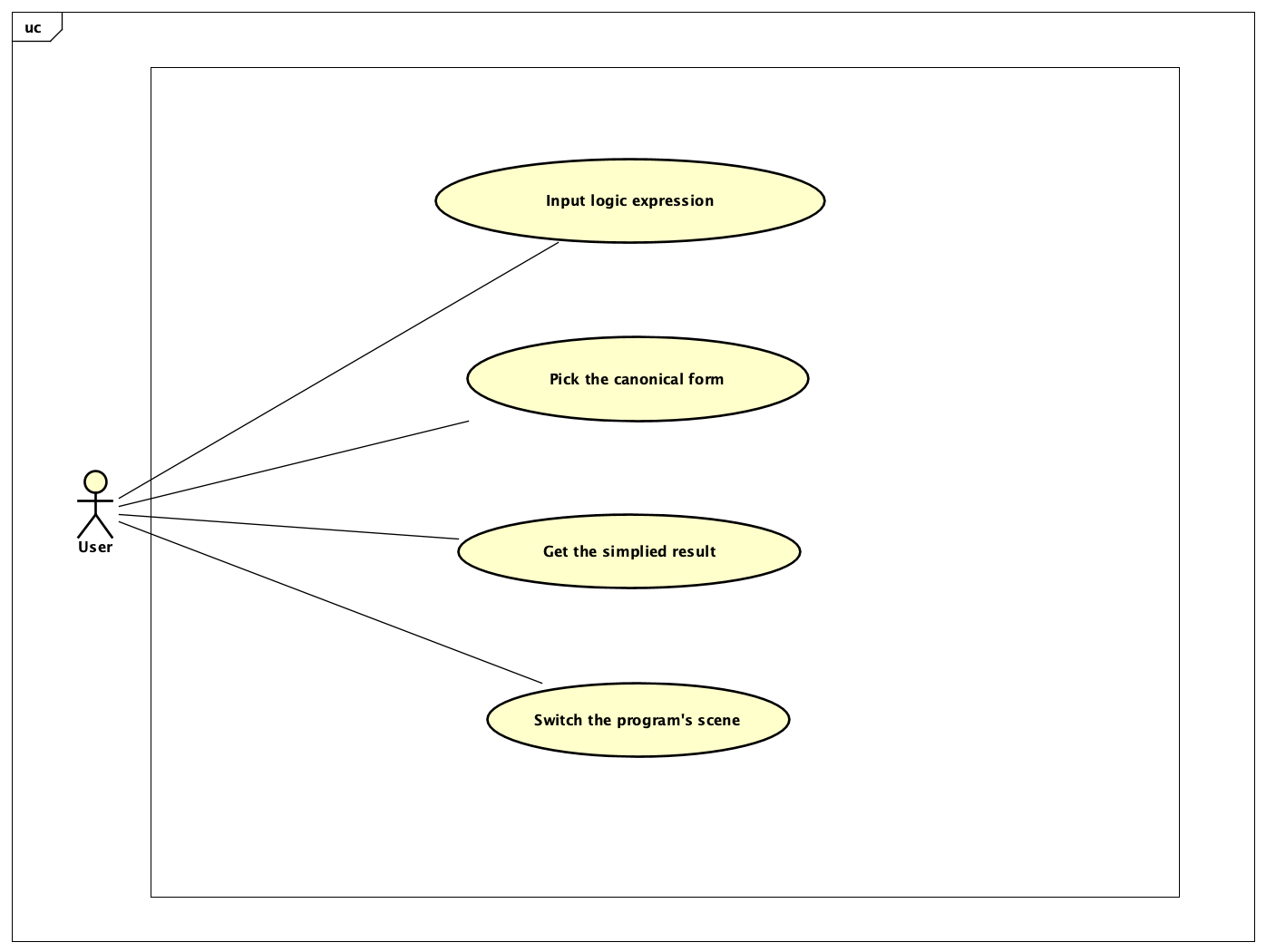
### Specifications:

This application receives a boolean expression from the user through an interface simulating a truth table. The user interface includes:

* Main menu: The user can choose one of the two cases 3 variables or 4 variables expression from the navigation bar.
* Input interface: The user can input a logic expression through a truth table. Don’t care values are not allowed. They can also pick a canonical normal form for the simplified expression – either SOP (Sum of products) or POS (Product of Sums). After the user has finished picking values for the truth table, they can press submit to see the output.
* Output interface: The application will show to the user the intermediate table (contains intermediate columns), the PI table, *the make equation table (contains EPIs and their character equations)* and the final simplified expression.

## Use case diagram

* Get the simplified result including user submits the selected input, and the result is shown up after.
* Pick the canonical form including user select the canonical result form ( in SOP or POS).
* Inputing logic expression includes user choose a sequence of 0s or 1s through the truth table, in which 0 means not chosen and 1 means chosen.
* Switch the program scene including user click the button back or select from the menubar option (3-variable, 4-variable).



*Figure 1. Logic expression normalizer use case diagram*

# **Design idea**

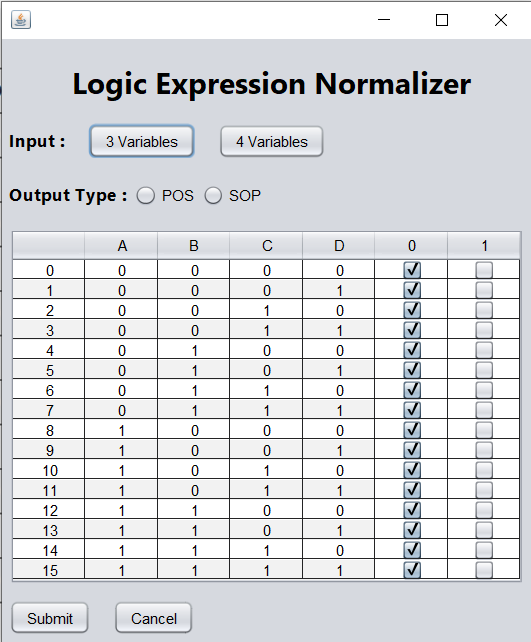
## The Model – View – Controller design patterns

In this mini project, we decided to design our program based on the MVC Pattern or Model – View – Controller Pattern. By this way, the code is organized into different sections with their own purposes and the code will be much easier and cleaner for others to review.

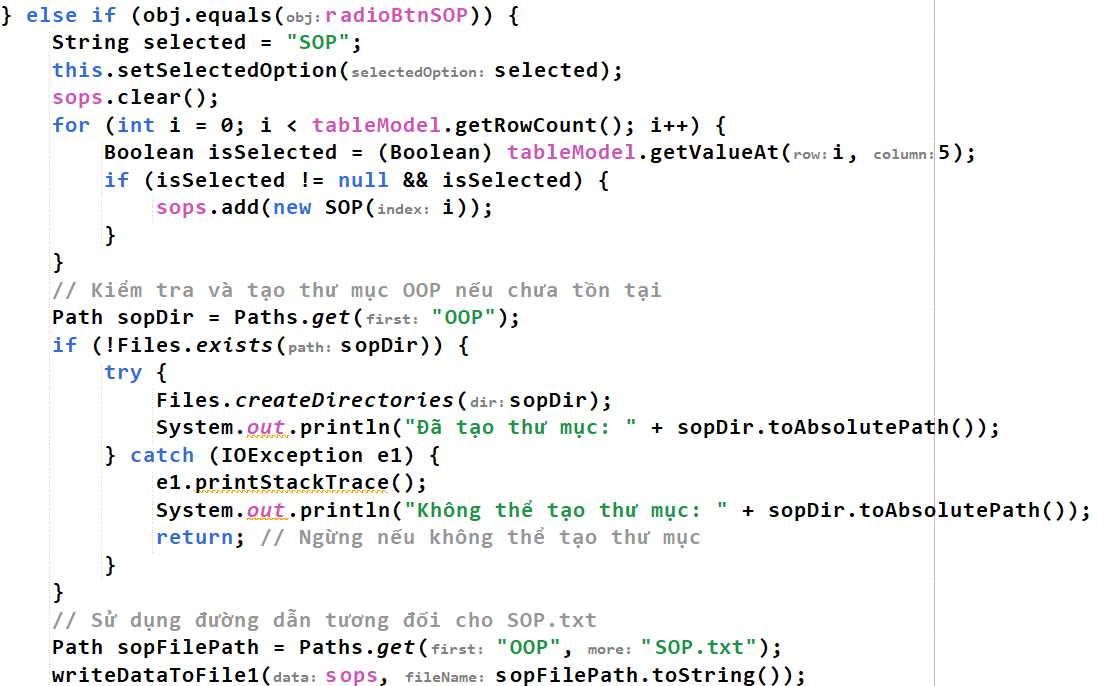
**The Model Package**: This package is responsible for storing raw data and all components utilized in the Quine-McCluskey algorithm.

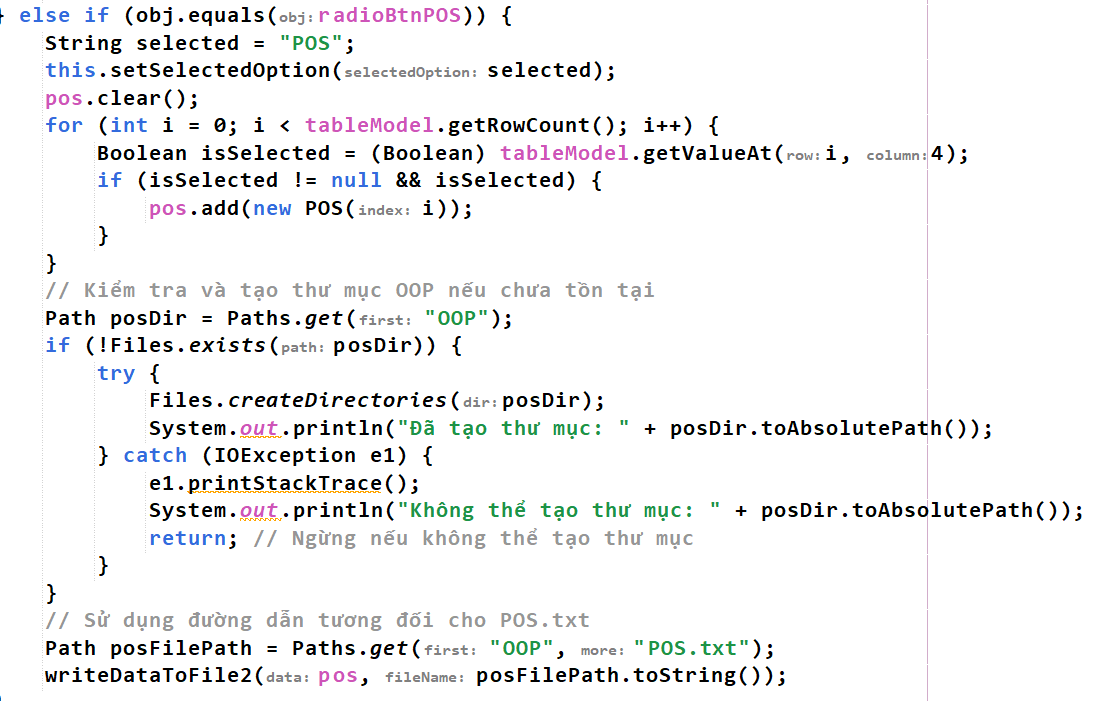
**The View Package**: The program uses Java Swing to display the data entered by the user and to output the results when the user clicks the submit button. Essentially, this package is designed to visualize the data managed by the Model package.

**The Controller Package**: This package serves as a bridge between the Model and View packages. It handles the flow of data from the user input, processes it within the Model, and updates the View accordingly whenever there are changes to the data.

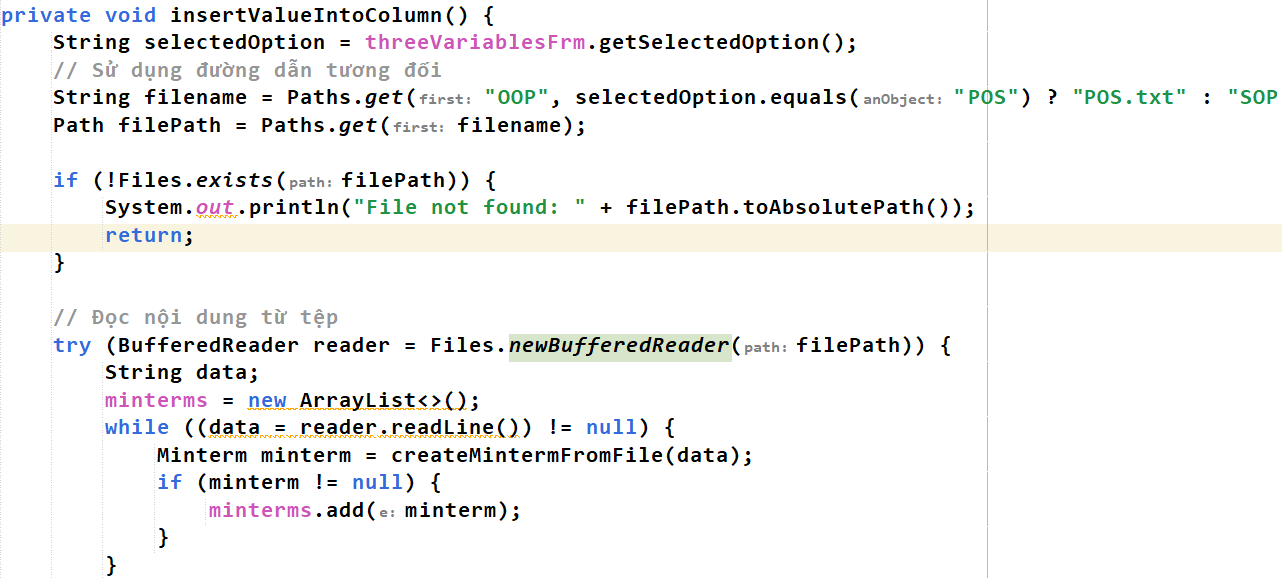


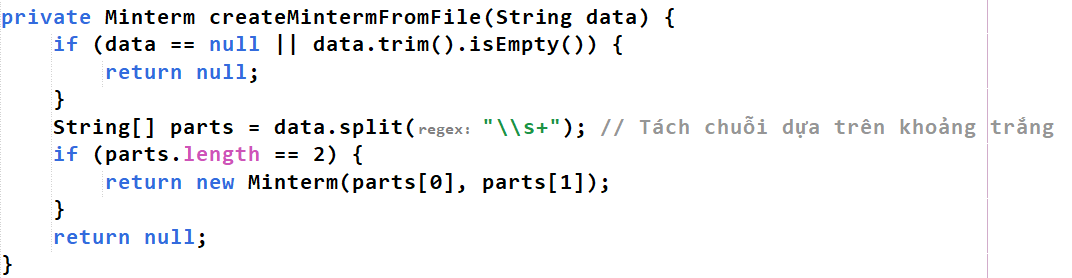
*Figure 2. The connection between Model – View - Controller*





*Figure 3. Extract data from View*

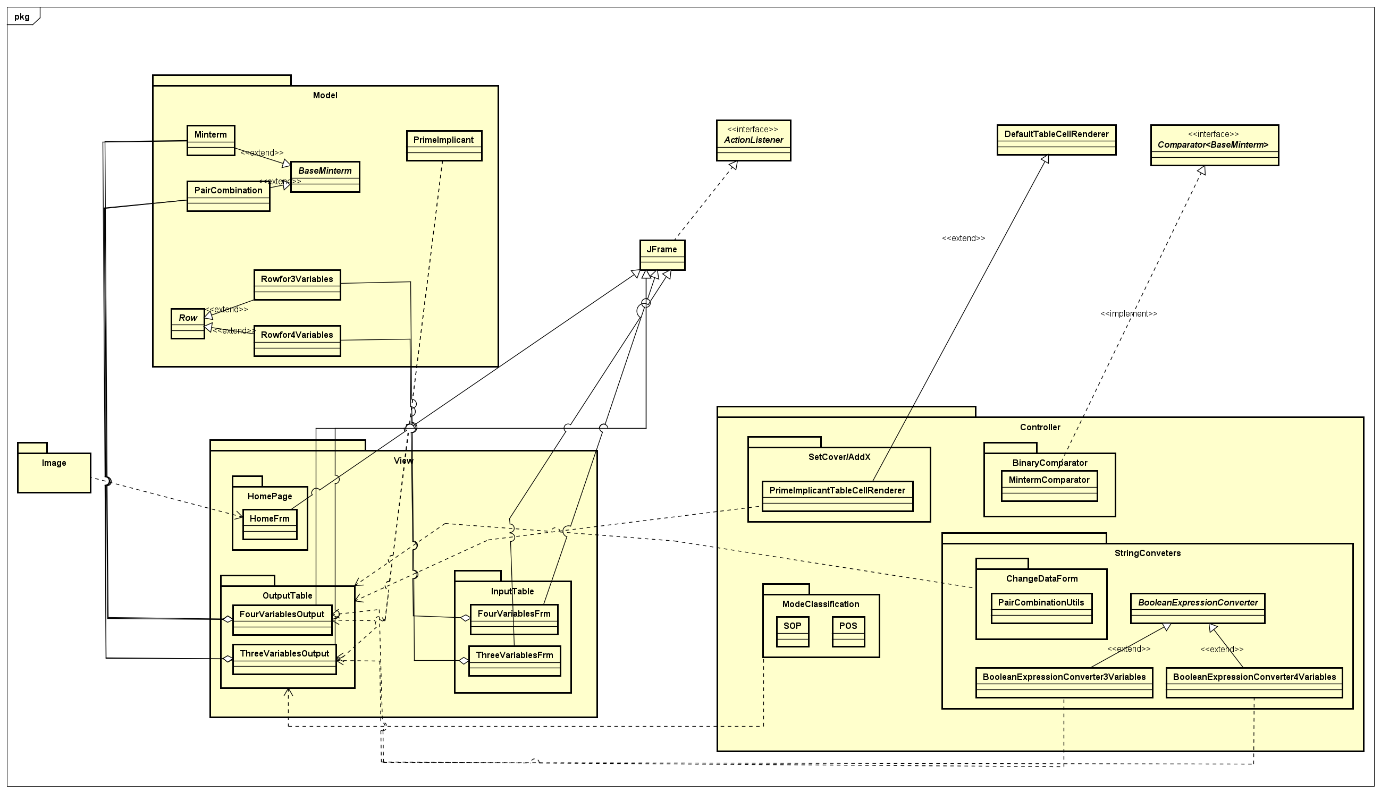
**

**

*Figure 3. Process data and Direct results and trigger View*

*to show data visualization*

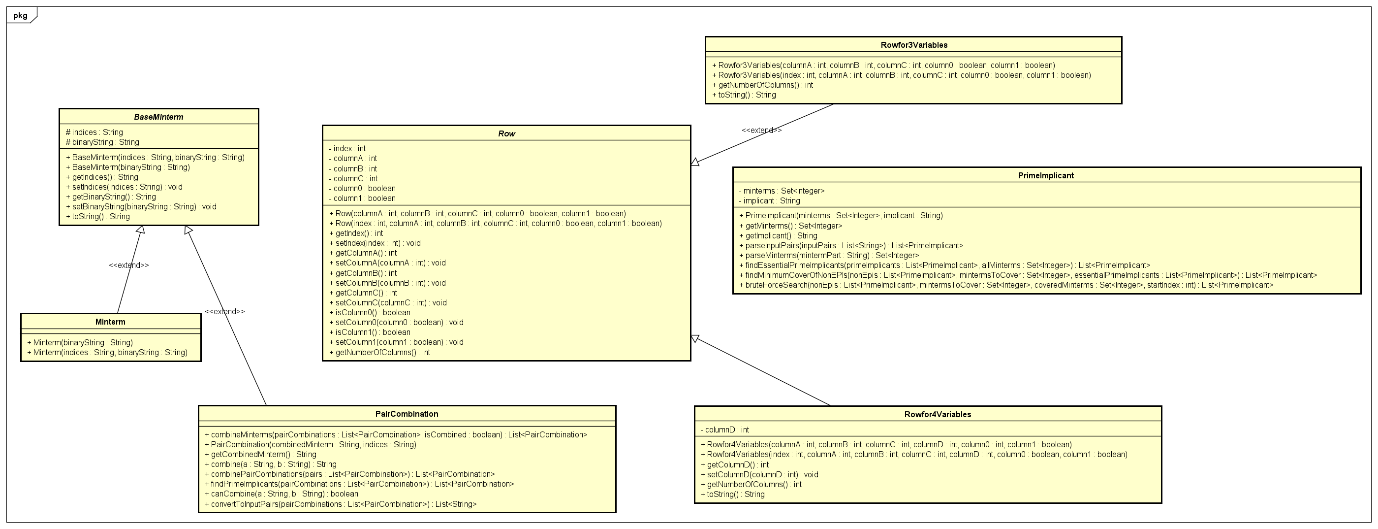
## General class diagram



*Figure 4. Logic expression normalizer class diagram*

## Class diagrams and explanation

### Model package:



*Figure 5. Model package’s class diagram*

This package consists of classes:

* BaseMinterm class: This class serves as an abstract class for prensting minterms (or combinations of minterms) by one algorithm. Here ‘s a detailed breakdown of its components. The attribute “indices” represents for the string representing the indices of the minterms that have been combined. The attributes “binaryString” representing the binary form of the minterm. It also contains the getter and setter method for “indices” and “binaryString”, and also an overrided method “toString()” that returns a string representation of the object, concatenating the indices and binary string, which is useful for debugging and output. This class has the constructors like “BaseMinterm(String indices, String binaryString)” which initializes the object with both indices and a binary string.
* This class is abstract and provides the basic structure for more specific implementations like “Minterm” and “PairCombination”
* Minterm class: The Minterm class extends BaseMinterm and represents a single minterm in the Quine-McCluskey algorithm. It inherits all the properties and methods from BaseMinterm but is tailored to handle individual minterms. While the Minterm class currently doesn't add additional methods, it is used to sort the “binaryString” and after that the results will be inserted into the first column of the IntermediateTable
* PairCombination class: The PairCombination class extends BaseMinterm and is designed to represent combinations of minterms. It plays a critical role in combining minterms to simplify expressions. Its attribute “combinedMinterm” refer to the binaryString which represents the combined form of minterms. Here ‘s a detailed of its methods:
* isCombined(): Checks if the minterm is combined by looking for a space in the indices string.
* combineMinterms(): Static method that attempts to combine a list of PairCombination objects. It identifies pairs of minterms that differ by only one bit and combines them.
* combinePairCombinations(): Static method that further attempts to combine previously combined minterms into larger groups.
* findPrimeImplicants(): Static method that identifies prime implicants from a list of pair combinations. Prime implicants are those that cannot be combined any further.
* canCombine(): Static method to check if two binary strings differ by exactly one bit, which determines if they can be combined.
* convertToInputPairs(): Static method that converts a list of PairCombination objects into a list of strings, suitable for further processing or output.
* The PrimeImplicant class is crucial for managing prime implicants during the simplification process. Prime implicants are essential components in the simplified boolean expression. It has the attributes likes “Set<Integer> minterms”  stores the indices of minterms that this prime implicant covers, “String implicant” represents the implicant in a simplified form. Here is the detailed of the main purpose of its class:
  + PrimeImplicant(Set<Integer> minterms, String implicant): Initializes a PrimeImplicant with a set of minterms and its corresponding implicant string.
  + parseInputPairs(List<String> inputPairs): Firstly, it converts a list of strings representing implicants into a list of PrimeImplicants objects. Each input string is split into minterms and implicant parts, parsed into their respective forms, and then used to create PrimeImplicant objects.
  + parseMinterms(String mintermPart): It converts a string representing minterm indices into a set of integers and removes any parentheses, splits the string by commas, trims spaces, and parses each part into an integer.
* findEssentialPrimeImplicants(List<PrimeImplicant> primeImplicants, Set<Integer> allMinterms):
  + Identifies essential prime implicants from a list of prime implicants.
  + Essential prime implicants are those that cover minterms that no other prime implicant covers. This method builds a map of minterms to the prime implicants that cover them and identifies those minterms covered by exactly one prime implicant.
* findMinimumCoverOfNonEPIs(List<PrimeImplicant> nonEpis, Set<Integer> mintermsToCover, List<PrimeImplicant> essentialPrimeImplicants):
  + Finds the smallest set of non-essential prime implicants (non-EPIs) that covers all remaining minterms not covered by essential prime implicants.
  + It starts with the minterms covered by essential prime implicants and uses a brute force search method to explore combinations of non-essential prime implicants to cover the rest.
* bruteForceSearch(List<PrimeImplicant> nonEpis, Set<Integer> mintermsToCover, Set<Integer> coveredMinterms, int startIndex):
  + A recursive method used in findMinimumCoverOfNonEPIs to explore all possible combinations of non-EPIs to find the smallest set that covers all remaining minterms.
  + It iteratively combines the minterms covered by current non-EPIs and checks if they cover all required minterms.
* The Row class is an abstract base class designed to represent a row in a data table. It is used to insert the value into the ThreeVariablesFrm and FourVariablesFrm and get the value of the minterm by checking the boolean value in the last two columns. It has attributes likes “index”, “columnA”, “columnB”, “columnC” and “column0”, “column1”. This 2 last data type indicates where the row will be selected according to SOP or POS mode that user wants.
* The Rowfor3Variables class extends the Row class and specializes it for the case of three variables. The Rowfor4Variables also extends the Row class, it inherits all Row’s attribute and has one more attribute is “columnD” that makes suitable for the numbers of columns of the table. Both these classes have @override methods toString()

### View package:



*Figure 6. View package’s class diagram*

* In this project I use Jpanel, jLabels, and jButtons to get view and action handling for the users.
* In both of five above class, in its constructor I firstly initializes the form and its components and then set the locations of the window to the center of the screen. I also call addActionListener() to attach action listeners to the buttons.
* Here is a general method I use for 5 classes:

-initComponents() Method:

* + This method sets up the layout and initializes the components of the UI.
  + It defines the layout and properties of the labels, buttons, and panel.
  + It also positions the components using absolute coordinates, providing exact placement within the window.

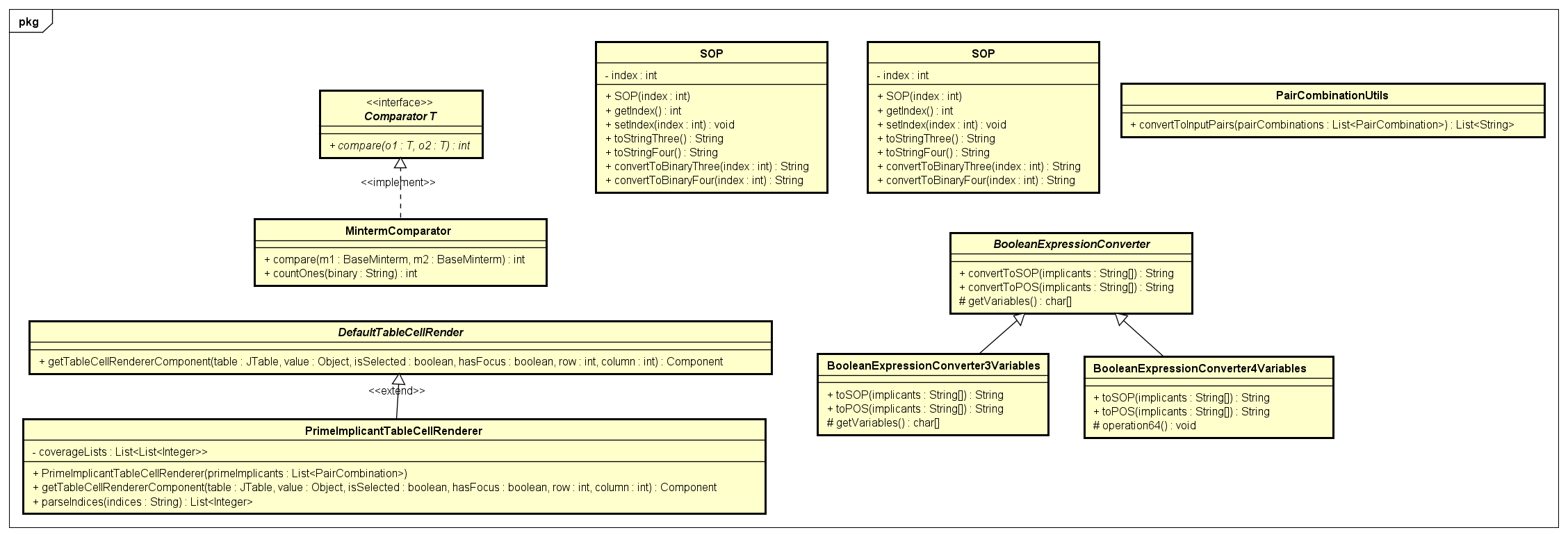
- addActionListener() Method:

* Attaches the current instance (this) as the action listener for each button.
* This ensures that when a button is clicked, the actionPerformed method is called.

- actionPerformed(ActionEvent e) Method:

* This method handles the actions triggered by button clicks:
* In HomeFrm whenever I click the btn3Variables, a new ThreeVariablesFrm is instantiated and displayed. And so does FourVariablesFrm. For the btnTips, a message dialog is shown with a helpful tip.
* For Three/FourVariableFrm class: This class is responsible for displaying the input form where the user can enter a 3-variable Boolean expression and select the output format (Sum of Products (SOP) or Product of Sums (POS)). It manages the input collection and initiates the process of simplification.
* Here is the main method helpers:
  + addActionListener(): Attaches action listeners to the buttons and radio buttons.
  + addButtonGroup(): Groups the radio buttons to allow only one selection at a time.
  + insertValueIntoTable(): Populates the JTable with all combinations of 3 variables.
  + setCellRenderers(): Centers the text in table cells.
  + writeDataToFile1(): Writes SOP data to a file.
  + writeDataToFile2(): Writes POS data to a file.
* For Three/FourVariableOutput class: This class displays the results of the Boolean expression simplification, including intermediate tables and the final simplified expression. It visualizes the intermediate steps of the Quine-McCluskey algorithm.
* Here is the main method helpers:
  + addActionListener(): Attaches action listeners to the buttons.
  + insertValueIntoColumn(): Reads the input data file, processes the minterms, combines them into prime implicants, and populates the tables.
  + setCellRenderers(): Centers the text in the table cells.
  + clearContentsOfFile(): Clears the contents of the specified file after reading.
  + createMintermFromFile(): Parses a minterm from the file data.
  + insertIntoPiTable(): Inserts the Prime Implicants into the PI table.
  + applyPrimeImplicantRenderer(): Applies custom rendering for the PI table to highlight prime implicants.
  + insertValueIntoTable(): Processes the prime implicants to find the essential and non-essential PIs and converts them into the final Boolean expression.

### Controller package:



*Figure 7. Controller package’s class diagram*

* In the controller package I will focus on handle the input.
* PairCombinationUtils class: This utility class is designed to convert a list of PairCombination objects into a list of strings. Each string in the resulting list is formed by concatenating two attributes of each PairCombination object: indices and combinedMinterm, separated by a space. This is useful for tasks that requires processing these combined value as input strings. The return value will be: A list of strings where each string represents a PairCombination object.
* MintermComparator class: This comparator class is used to sort “BaseMinterm” objects based on their binary string representations. It primarily orders the objects by the number of '1' bits in their binary strings, and if two objects have the same number of '1' bits, it then orders them by their binary value. This class is crucial for tasks that involve sorting “BaseMinterm” objects in a specific order required for further processing.
* SOP/POS class: This class represents a Product of Sums (POS) or Sum of Products (SOP) mode classification in Boolean algebra. It encapsulates an index and provides methods to convert the index to its binary representation for both three-variable and four-variable cases.
* BooleanExpressionConverter: This abstract class provides a framework for converting Boolean expressions into Sum of Products (SOP) and Product of Sums (POS) forms. It defines methods to perform these conversions based on the binary representations of the expressions.
* BooleanExpressionConverter3/4Variables: This subclass of BooleanExpressionConverter is specifically tailored for converting Boolean expressions with three variables (A, B, and C) and 4 Variables will use one more variable id D into SOP or POS forms. It implements the abstract method to return the set of variables used in the conversion.

### OOP-Principle in my Java Application:

1. Abstraction:
   * Definition: Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object.
   * Application:
     + Abstract Classes:
       - BaseMinterm: This class abstracts the common properties (indices and binaryString) and behaviors (methods to get and set these properties) of minterms.
       - Row: This class abstracts the concept of a row in the context of variables and Boolean expressions, defining common properties and abstract methods like getNumberOfColumns().
     + BooleanExpressionConverter: This class provides an abstraction for converting Boolean expressions into SOP and POS forms without detailing how variables are handled, which is left to subclasses.
2. Encapsulation:
   * Definition: Encapsulation is the bundling of data and methods that operate on the data within one unit, like a class, and restricting the access to some of the object’s components.
   * Application:
     + Private and Protected Members: Classes like BaseMinterm, Row, and PrimeImplicant encapsulate their data (such as indices, binaryString, minterms, etc.) and provide public methods to access and modify this data. This ensures controlled access and modification of the internal state.
     + Accessors and Mutators: Getter and setter methods (e.g., getIndices(), setIndices(), getMinterms(), setMinterms()) are used in classes like BaseMinterm and PrimeImplicant to control how attributes are accessed and modified.
3. Data Hiding:
   * Definition: Data hiding is a principle where internal object details are hidden from the outside world, exposing only necessary parts through methods.
   * Application:
     + Private Variables: Most of the class fields (e.g., indices, binaryString in BaseMinterm; minterms in PrimeImplicant) are declared private, ensuring that they can’t be accessed directly from outside the class.
     + Controlled Access: Access to these private fields is controlled via public or protected methods, maintaining the integrity of the data.
4. Object Initialization & Usage:
   * Definition: Object initialization refers to creating and setting up an object for use. Usage involves interacting with the object’s methods and properties.
   * Application:
     + Constructors: Each class has constructors to initialize its objects. For example, Minterm and PairCombination use constructors to initialize their properties (indices, binaryString).
     + Factory-like Methods: Methods like PairCombinationUtils.convertToInputPairs are used to create and manipulate collections of objects efficiently.
5. Inheritance:
   * Definition: Inheritance allows a class to inherit fields and methods from another class.
   * Application:
     + Class Hierarchies:
       - Minterm and PairCombination inherit from BaseMinterm, allowing them to share common properties and behaviors while also having additional, specific functionalities.
       - BooleanExpressionConverter3Variables and potentially other converters (not shown but implied) inherit from BooleanExpressionConverter, gaining shared conversion methods and adding specific variable handling.
     + Method Overriding: Subclasses like BooleanExpressionConverter3Variables override methods from BooleanExpressionConverter to provide specific implementations for different sets of variables.
6. Polymorphism:
   * Definition: Polymorphism allows objects of different classes to be treated as objects of a common super class, typically enabling method overriding and dynamic method resolution.
   * Application:
     + Abstract Methods and Overriding: Methods in BooleanExpressionConverter are abstract and implemented differently in its subclasses, allowing different behaviors while treating them as BooleanExpressionConverter type.
     + Dynamic Method Invocation: The use of overridden methods in a polymorphic way, where the actual method that gets called is determined at runtime based on the object’s class (e.g., convertToSOP and convertToPOS in different BooleanExpressionConverter subclasses).
7. Association:
   * Definition: Association is a relationship between two classes that allows one to be associated with the other.
   * Application:
     + HomeFrm and Input Tables: The HomeFrm class has associations with input forms like ThreeVariablesFrm and FourVariablesFrm, which it initializes and interacts with based on user actions.
     + ThreeVariablesOutput and ThreeVariablesFrm: ThreeVariablesOutput has a reference to ThreeVariablesFrm, indicating that it uses this form to read the user's input selection.
8. Aggregation:
   * Definition: Aggregation is a specialized form of association where a class is a collection or container of other classes.
   * Application:
     + Lists of Objects: ThreeVariablesFrm aggregates Rowfor3Variables objects in a list. Similarly, ThreeVariablesOutput holds collections of Minterm and PairCombination objects.
     + Composite Data Structures: Classes like PrimeImplicant aggregate collections of indices (minterms) to represent their relationship in Boolean simplification.
9. Generalization:
   * Definition: Generalization is the process of extracting shared characteristics from two or more classes and combining them into a generalized superclass.
   * Application:
     + Superclass and Subclass Relationships:
       - BaseMinterm generalizes the common properties of minterms, and its subclasses (Minterm, PairCombination) specialize this general concept.
       - BooleanExpressionConverter generalizes the method of converting Boolean expressions, and its subclasses handle specific cases for different numbers of variables.
10. Method Overloading:
    * Definition: Method overloading is the ability to define multiple methods with the same name but different parameter lists.
    * Application:
      + Different Constructors: In classes like Rowfor3Variables and POS, constructors are overloaded to allow different ways of initializing the objects (with varying numbers of parameters).
11. Interface and Abstract Classes:
    * Definition: Interfaces define a contract of methods without implementation, while abstract classes provide a base class with some shared implementation.
    * Application:
      + Abstract Classes: BooleanExpressionConverter and BaseMinterm are abstract classes providing base functionalities and abstract methods to be implemented by subclasses.
      + Potential Interfaces: While not explicitly shown, interfaces could be used for defining behaviors like "Convertible" for objects that can convert to different forms (SOP/POS).

### Workflow:

1. **User Input**:
   * The user interacts with the HomeFrm to select whether they want to work with 3 or 4 variables.
   * This opens either ThreeVariablesFrm or FourVariablesFrm, where the user inputs their boolean expression terms into a table and selects the output type (SOP or POS).
2. **Data Processing**:
   * Upon submission, the form frame (ThreeVariablesFrm or FourVariablesFrm) gathers the user input and writes it to a file (SOP.txt or POS.txt).
   * It then opens the corresponding output frame (ThreeVariablesOutput or FourVariablesOutput).
3. **Minterm Combination**:
   * The output frame reads the minterms from the file, creates Minterm objects, and sorts them.
   * It then uses the PairCombination class to combine minterms, identifying which ones can be combined based on their binary representation.
   * These combinations are further processed to find the prime implicants.
4. **Prime Implicant Identification**:
   * The combined minterms are analyzed to identify the prime implicants using the methods in PairCombination.
   * Prime implicants are those combinations that cover minterms and cannot be combined further.
5. **Final Simplification**:
   * The identified prime implicants are converted into a simplified boolean expression in SOP or POS form using the appropriate converter class (BooleanExpressionConverter3Variables or BooleanExpressionConverter4Variables).
   * The final simplified expression is displayed in the output frame.
6. **User Interaction**:
   * The user can view the intermediate steps and the final simplified expression in the output frame.
   * They have options to convert again or return to the main menu for different operations.