Mohammad Reza Baghban Haghighi

378474

Abstract

design decisions and their advatanges and disadvantages for the project

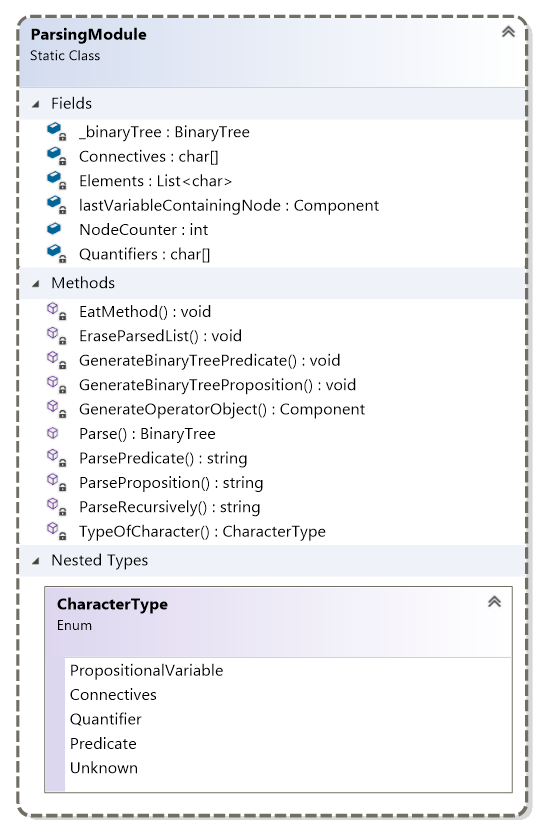
LPP   
 Design Documentation

FHICT English Stream

# **Introduction**

In this document I will try to elaborate on the design decisions that I have taken in the process of designing, implementing, and testing the LPP application.

1. **Using Singleton Design Pattern for Parsing Module**

For designing the parsing module, initially, I was thinking either I need to use common class, singleton class, or a simple static class with some public static method.

First of all, I did not use a common class because I did not find any point of having multiple objects of Parsing Module especially since none of the potential objects does do have their own set of properties which should be distinguished from other objects of that class.

And between choosing the singleton pattern and static common class, since the singleton pattern usually comes with the concept of **immutability** and the fact that one universal object needs to be used within different classes. As we do not have any of these situations there is no need application for singleton pattern.

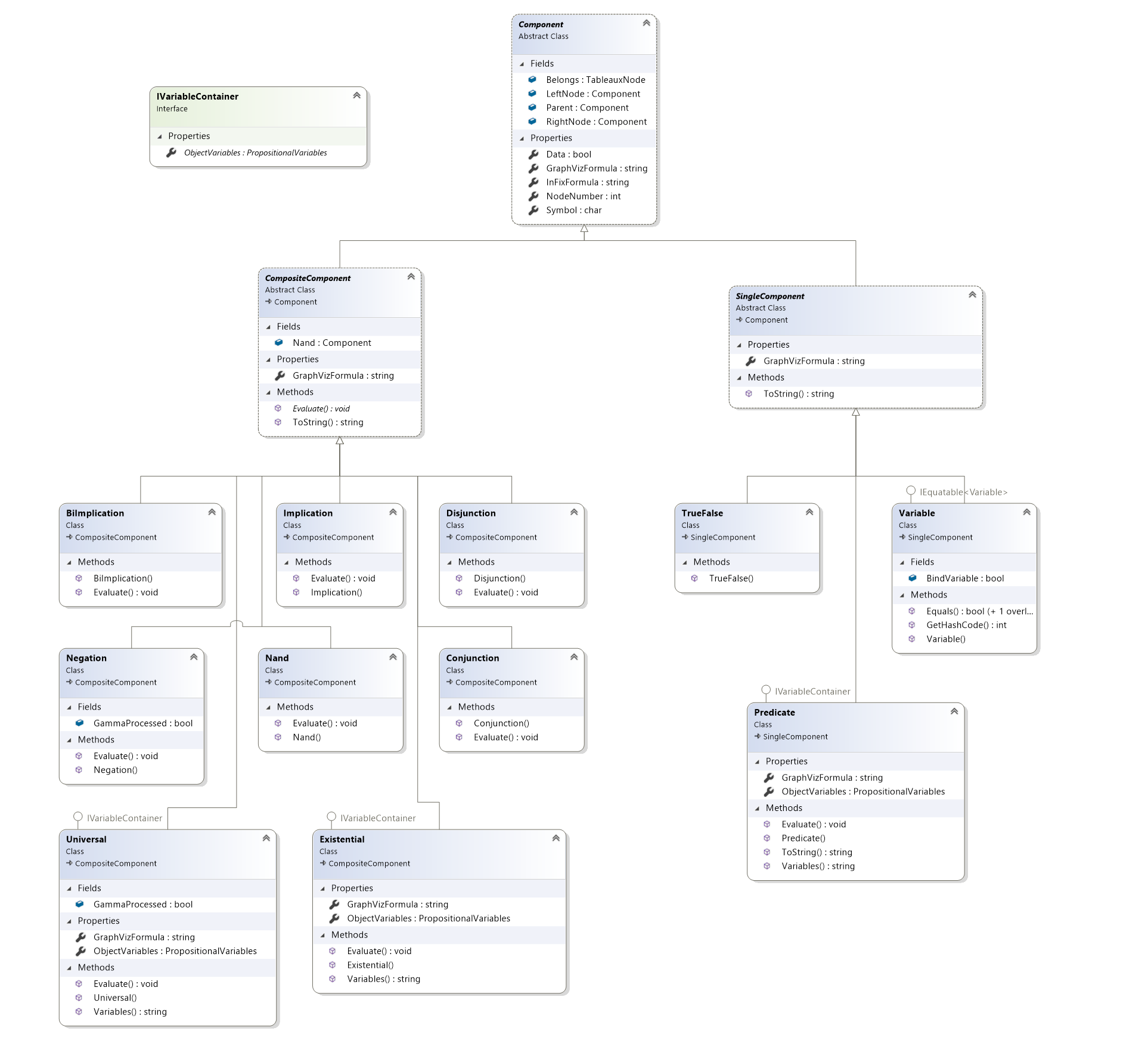
Lastly, since only a few methods of parsing modules are needed to being exposed to other classes, the last paradigm of having a static class is being chosen.

Also for the sake of encapsulation and separation of concerns, all the methods of Parsing Module become private except **Parse**() method as the only gate for interacting with a module where it will do following Operations:

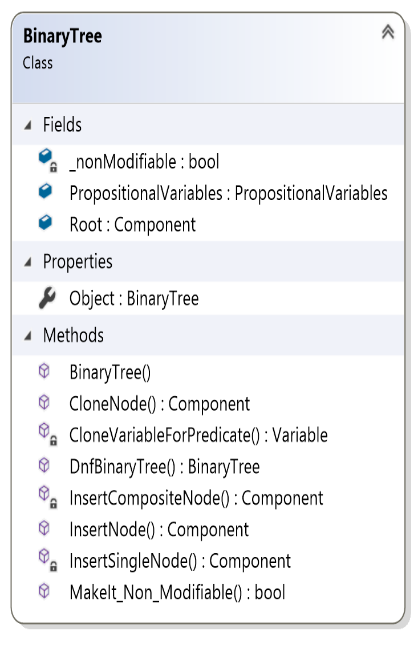
1. Parse the input from prefix notation to extract all members by calling based on if the formula is proposition formula or predicate recursively
2. Generate a binary tree out of given formula by interacting with BinaryTree object bt
3. Return the root of the binary tree to the caller
4. **Using the Composite Design Pattern for Object Structure**

Since we have a binary tree as the data structure of storing the components of our abstract propositions, using the Composite Pattern would be an asset to show the part-whole structure of our binary tree. Especially where for some operations on each level we need to have the result of all its sub-levels so traversing and perform actions can be achieved uniformly.

For the implementation, we have two Composite Component and Single Component for connectives and propositional variables which both inherit from Component class for the sake of being able to use the benefits of **Polymorphism** and [Single Responsibility](https://en.wikipedia.org/wiki/Single-responsibility_principle) and [Liskov Substitution](https://en.wikipedia.org/wiki/Liskov_substitution_principle) concepts of SOLID Principle**.**

****

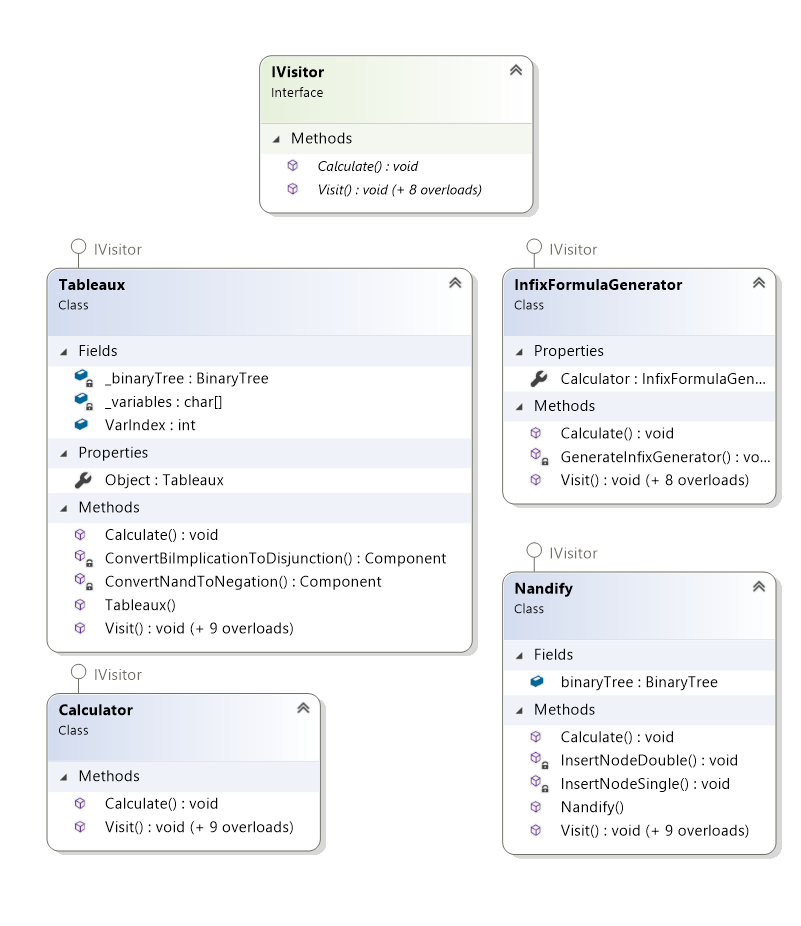
1. **Binary Tree**

****The BinaryTree class is being used to create tree structure from the object structure using InsertNode() method which the entry gate and based on if the new node is being a Composite or Single Component it will call to respective insertion method and will return the root of a binary tree as the result.

In the implementation of the insertion method it has tried to make the binary tree balanced such that on one branch operators like Universal, Existential, and Negation, the sub-nodes be inserted on the left node of the operator.

Also, two **DeepClone** methods are being placed for the sake of Deep Cloning Composite Pattern Object with all their sub-nodes and its structure like variables list in a recursive manner.

1. **Using the Visitor Design Pattern for Functionalities and Operations**

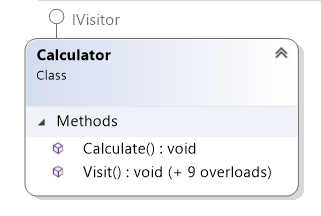
Based on the requirements of the project where different operations (Truth Table calculations, simplification, InfixFormula Generation, Nandification, and Semantic Tableaux) needed to be performed on the composite object structure and binary trees within software gradually, it is good to use Visitor Design Pattern.

In this Design pattern, new functionalities can be added without changing the implementation of pre-existing objects structure that was built using Composite Pattern, and by following this approach the maintainability and extensibility of software design next to its flexibility would be increased considerably in loosely coupled architecture. Also, this design pattern is one of the instances of using [Open/Closed](https://en.wikipedia.org/wiki/Open%E2%80%93closed_principle) concept of [SOLID principle](https://en.wikipedia.org/wiki/SOLID)

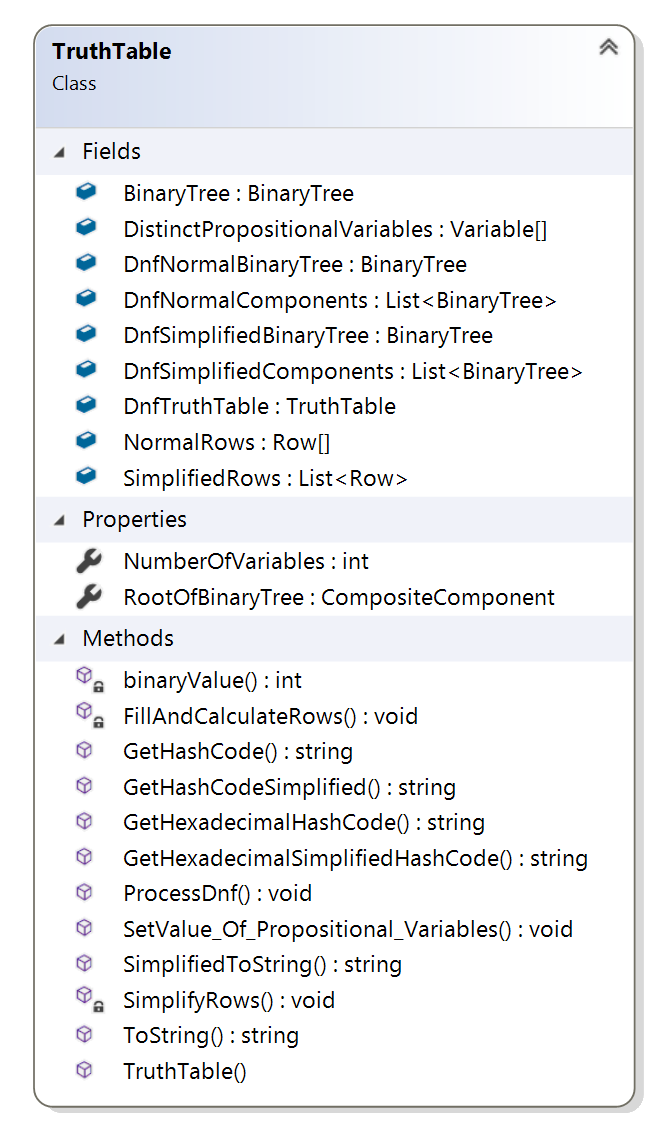
By doing this, instead of adding new methods to objects and made alternation to classes in an object structure, Objects would be passed to individual Visitor classes that implemented visitor to perform desired operations.

In InfixGenerator is a concrete Visitor class that implements IVisitor Interface which would be used for generating the infix formula of the abstract proposition.

# **Assignment 2: Truth table + Hash code**

1. **Calculator Class for Evaluating Abstract Propositions**

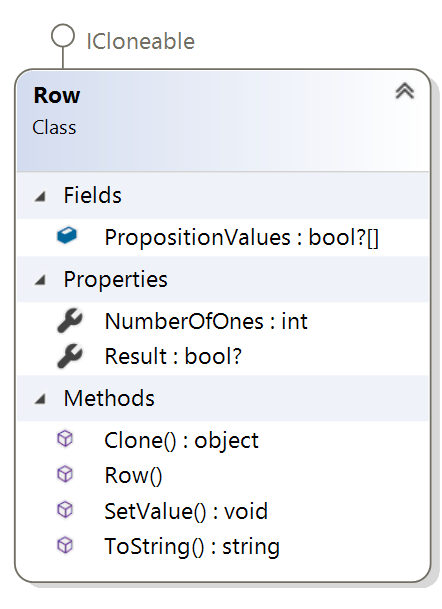
Calculator class is a concrete Visitor class where instead of placing calculation logic in each connective class, we extract those logics and put all of them in this operation class. The calculate method recursively traverse abstract proposition tree and evaluate each node based on its type using Visit() methods. Also, the algorithm for calculation of the Truth Table and its row has been placed in this class as one of the Visit overloads.



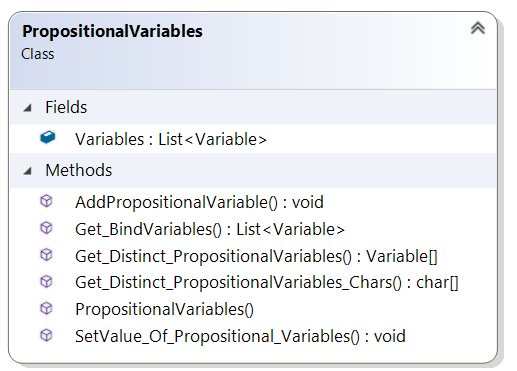
1. **TruthTable | Row | PropositionalVariables**

**TruthTable** class is being designed such that it will have a fixed-sized row as an array and a Fill Rows() method which contains the algorithm of filling truth table row dynamically both based the number of distinct propositions variables within the abstract proposition.

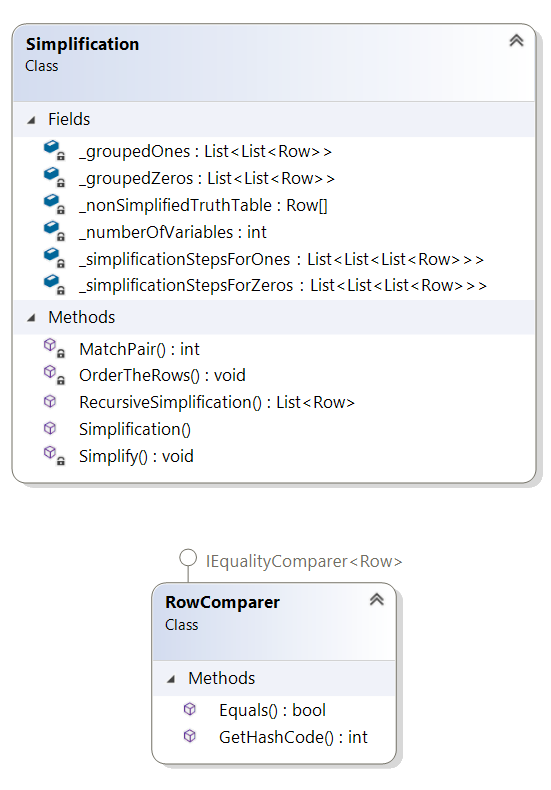
After filling the truth able with row objects it will use a calculator class to calculate associated binary tree based on the values of variables on each truth table row.



In designing **Row** class, **Nullable approachable has been used** for both each proposition variable value and total row value where Null represents this concept that the value of each proposition variable does not cast any effect on final row value which would be used intensively in the simplify assignment

**PropositionVariables** class also represents a data structure to preserve a list of all proposition variables within a truth table and some method to return a specific propositional variable or list of distinct Propositional variables. It is also responsible for changing the value of all same-symbol propositional variables.

# **Assignment 3 & 4: Simplify + Normalize**

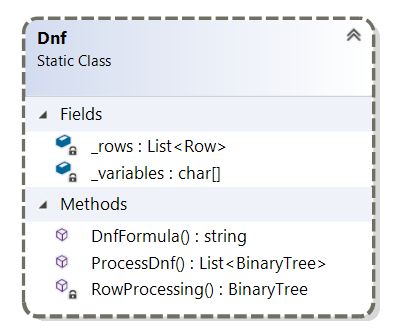


**Simplification**

In the Simplification assignment, the Quine–McCluskey algorithm has been used for minimization of Boolean functions which is being represented as the truth table.

Initially, rows are being divided into two groups whose function value be either 0 or 1 which then the members of each group are sorted based on the number of true values within their Boolean function.

After sorting, each group is then being reclusively simplified where within the simplification process it will try to compare rows to check if rows are being matched pair to each other and then being minimized until the point that it could not be simplified further.

**Normalization**

For Disjunctive normal form, I have used a static ProcessDNF() method which is the entry gate to DNF class that extracts with True value and try to regenerate and equivalent binary tree as the conjunction of variables for each row which later on the elements of this list would be joined by using particular DNFBinaryTree method in BinaryTree class with disjunction operator to form the main DNF binary tree that can deliver the same HashCode.

# **Assignment 5: Nandify**

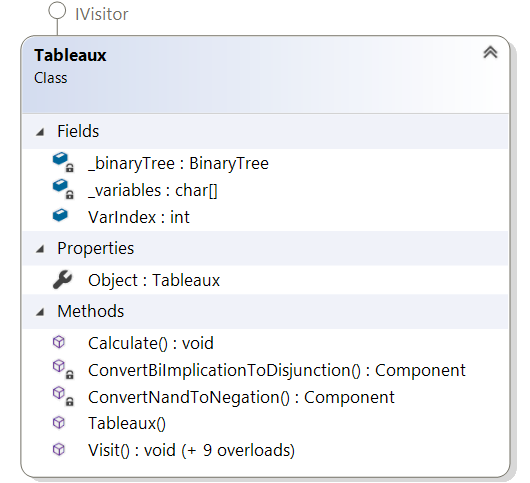
# For Nandification, since a set of recursive operations should be performed on BinaryTree object structure, the logic of transmission has been placed in a concert IVisitor Nandify class in which for each type of connective there would logic for transforming it into a combination of negation and conjunction where final BinaryTree can give same hash code.

# Because of the advantage of using Visitor, the existing object structure would not be modified, and new features are being added to the software that depicts **the maintainability and flexibility of software design**

# **Assignment 6,7,8: Semantic Tableaux**

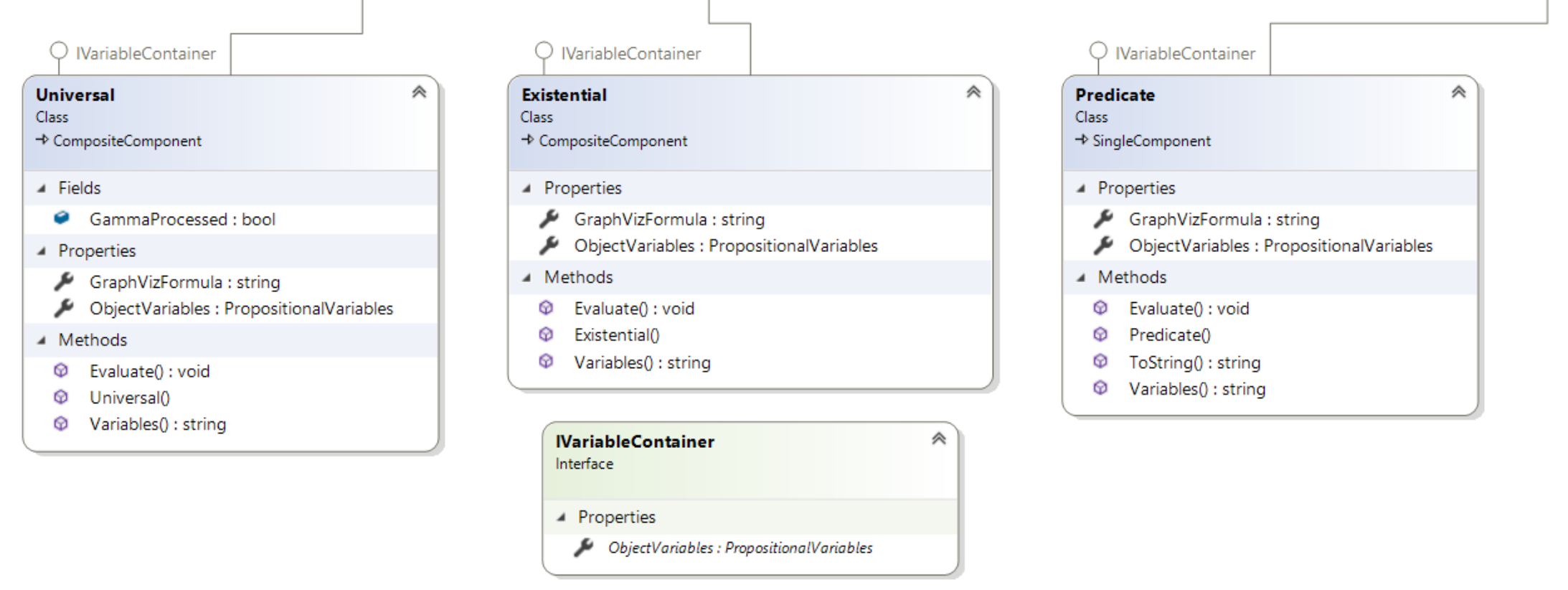
# For Semantic Tableaux I have created a Tableaux Node through binary tree data structure for each node of Tableaux that contains a list of components for each node for both Proposition and Predicates.

It has three different Constructors for creating different kinds of tableaux nodes based on the previous type of Semantic Tableaux rule that is being used which would be called by Tableaux Visitor class in creating further nodes in the case of required simplification.



Tableaux Node has a main method IsClosed() that is responsible for looking in the current node if a contradiction among node components can be found by using FindContradication() method, to close the leaf or it needs to be processed into further smaller formulas using Evaluate() method and when a leaf should be left open.

For performing Semantic Tableaux Rules on each component, the component would be sent to IVisitor cocreate Tableaux class that contains logic and algorithms different types of Tableaux Node which again because of the nature of the Visitor pattern **provide this possibility to extract the algorithm from the object structure**.

****

Those operators that contain bound or non-bound variables also being distinguished from other operators in composite object structure by implementing a specialized IVaraibleContainer interface as a mean of applying [**interface segregation**](https://en.wikipedia.org/wiki/Interface_segregation_principle) **concept of SOLID principle** to avoid forcing other components to override single responsibility principle

Also, for the sake of more **enhanced processing and choosing less expensive algorithms** like alpha or lambda rather than Gamma or Beta, components within each node are being sorted based on the potential type of rule that can be applied to them using ComponentComparer class.

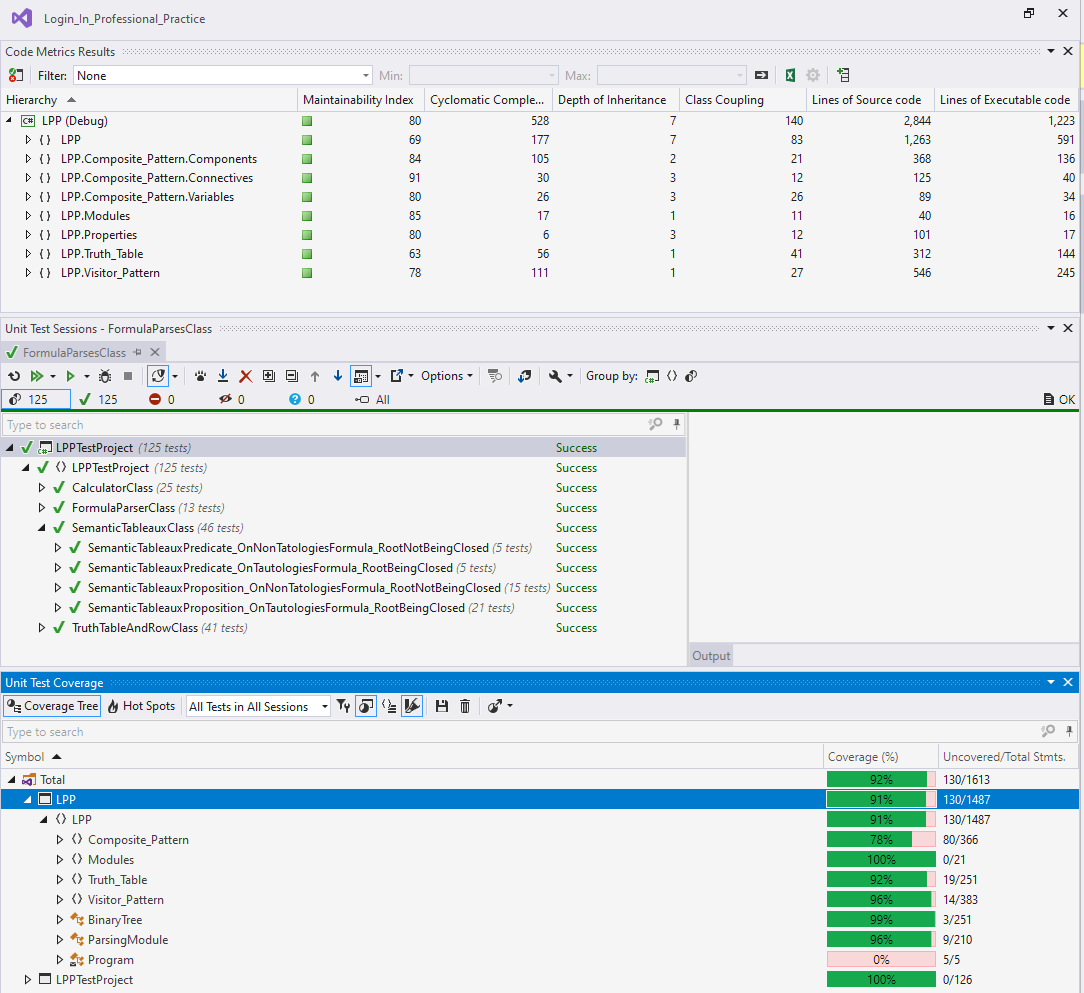
# **GUI**

# 

# Parse Recursively Button become green if the Hash Code of Original truth table equals to its Nandified and DNF HashCode

1. Semantic Tableaux button becomes green if the given formula is tautology otherwise it would turn reddish.
2. On the bottom right side, there is a photo slider that can be used for investigating binary tree of formula, DNF, NAND, and Semantic Tableaux.
3. In Truth Table section the original, simplified representation can be seen next to see the hash code of Original, NAND, and DNF one.

# **Testing**

Software Has been tested intensively and different SUT is being evaluated with bunch different Test cases which in total there would be around 125 unit-tests and 92% percent of Code Coverage as follow next to healthy code metrics:

# **Conclusions and future implementations**

I have learned a lot in this project and enhance both programming and software design skills on how to make a maintainable, extensible, and flexible software solution and take benefit of using Object-Oriented Design Patterns and SOLID principles as ease tools. Next to that my algorithm design capability has been increased considerably and I get interested in the research area of software also.