**Faculty of Engineering, Alexandria University.**

**Computer & Systems Engineering Dpt.**

**Discrete Mathematics**

**Truth Table Constructor**

**Final Project Report**

***By***

**Marina Zakaria**

**Index**

1. Introduction 3
2. Overview 5
3. Features 5
4. Data Structures 6
5. Functions 9
6. User Manual
7. Sample runs
8. References

**Introduction**

This project was assigned in the Discrete Mathematics course

on : Tuesday 14th Nov, 2017.

And due : Tuesday 28thNov, 2017.

This delivery is by :

*Marina Zakaria Boules Youssef Zakaria*

The source of this project is available on github on this link:

**Overview**

This program intends to take a logic expression and draw the corresponding truth table. Then user can find the truth value of the expression given the values of its propositions, determine if the expression is a tautology or a contradiction, and test if two expressions are equivalent by comparing the truth table for both of them.

**Features**

*In our implementation,*

The program can perform:

**Input validation**: The program makes sure that the expression syntax is valid and if the syntax was invalid the program shows an error message specifying the exact index where the error occurs and how to fix it.

**Output Format**: the program shows a visual view of truth table .

**Save the truth table :** the user has the ability to save the truth table in a text file to get back to it any time.

**Operation on expressions:**

The program support the following operations on the expression:

* expression truth value
* test for tautology or contradiction
* test for equivalence.

**Algorithms**

**1)validation Algorithm**

Get rid of all spaces and turn all characters to lower case

Check each character in the expression in order

If(the character is a letter from a to z )then

Check the following character

If(another letter Or opened bracket)then

Invalid expression , must be operator

Else if (the character is an operator )then

Check the following character

If(another operator other then (not)or closing bracket)then

Invalid expression , must be operand

Else if(the character is the operator (not))then

Check the following character

If(another operator Or closing bracket)then

Invalid expression , must be operand

Else if(the character is an opening bracket)

Check the following character

If(operator other then not)then

Invalid expression , must be operand

Else push to a Stack

Else if(the character is a closing bracket)

Check the following character

If(a letter or (not)operator)then

Invalid expression , must be an operator other then (not)

Else pop from the Stack

Else

Invalid expression , you entered an unsupported character

Check the Stack if not empty

Invalid expression , unclosed bracket

**1)Evaluation Algorithm**

1)get number of propositions by looping through the expression and count number of letters

2)fill the truth table with all possible combinations of the propositions and save the values of each proposition in a separate array

3)turn the infix expression to postfix expression to support the order of operations (brackets then not then and then or then implies then bi-conditional)

4)evaluate the post fix expression by converting it to stages each contains simpler expression of 2 operands and store the result in an intermediate array to use it in the next stage

**1)Check tautology Algorithm**

Loop through the result truth values if all is true then the expression is a tautology

**1)Check Contradiction Algorithm**

Loop through the result truth values if all is false then the expression is a contradiction

**1)Check Equivalence Algorithm**

Loop through the result truth values of the 2 expressions if all the corresponding values are identical then the expressions are equivalent

**Data Structure Used**

**Array list** for the proposition names : because its number can’t be predicted

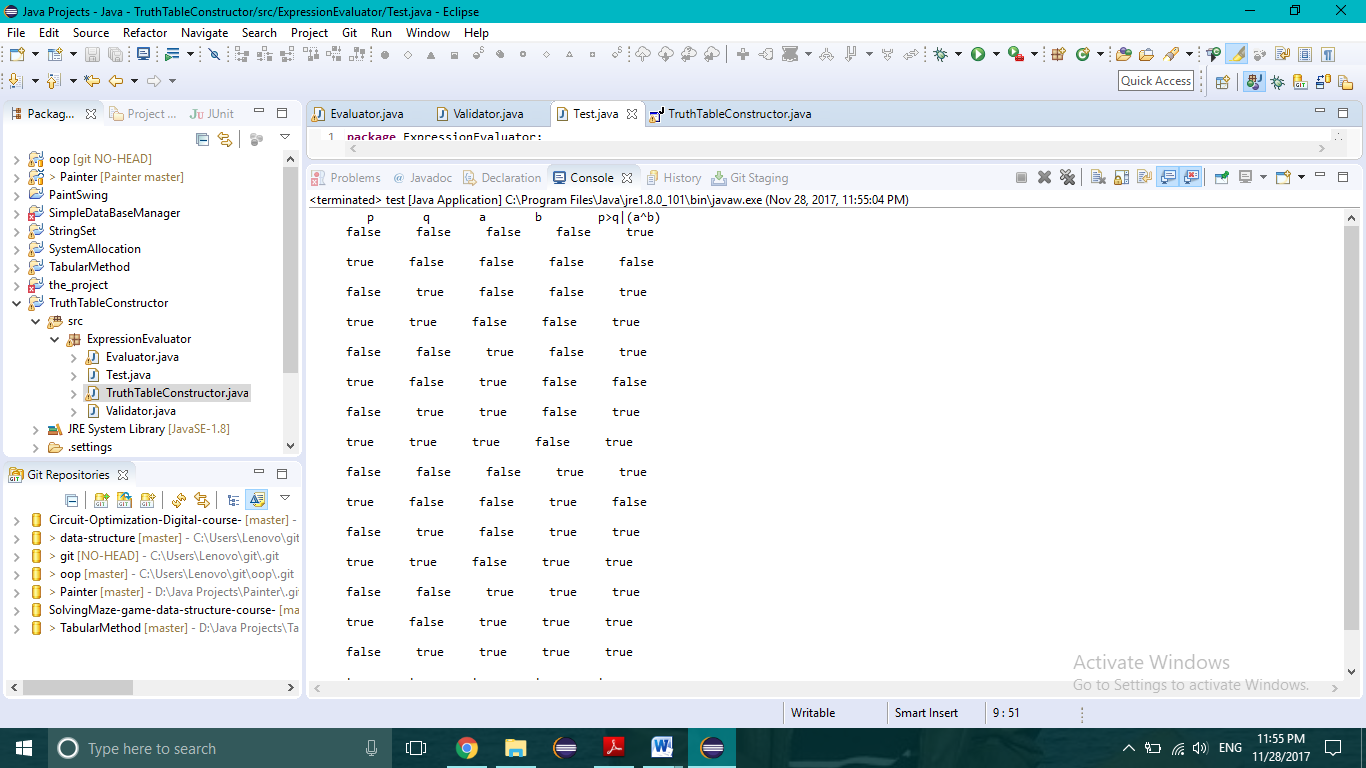
**Boolean arrays** :for truth values of the propositions and the result

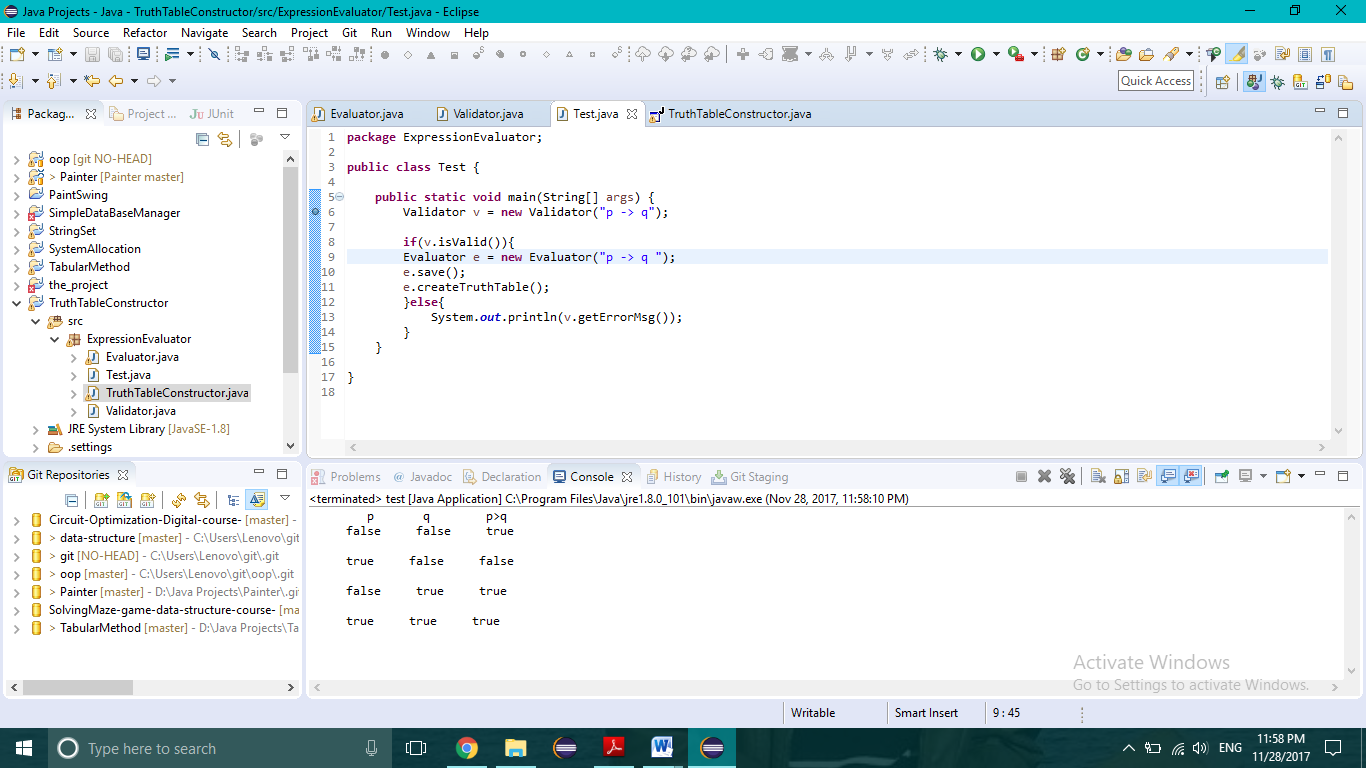
**Stack** : to convert from infix to postfix and to evaluate postfix expressions and check validation of brackets

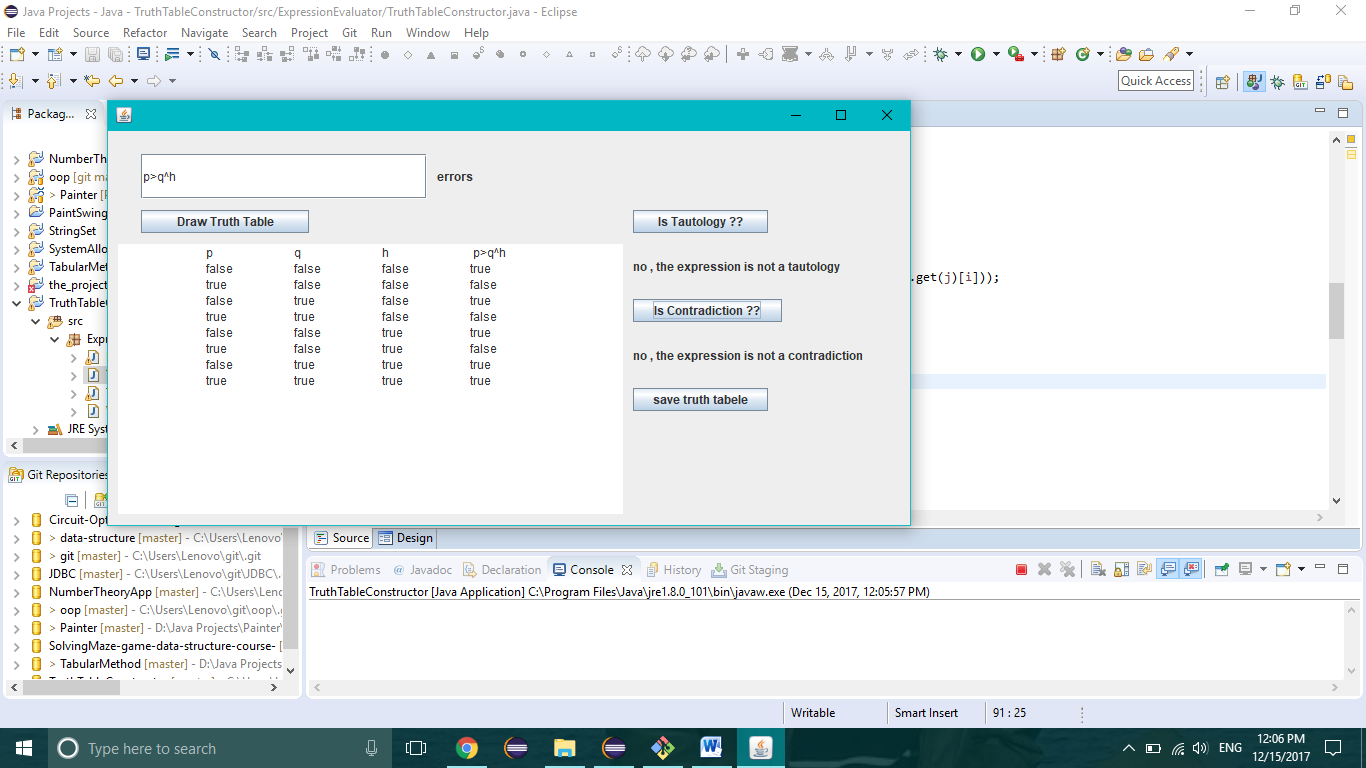
**Assumptions**

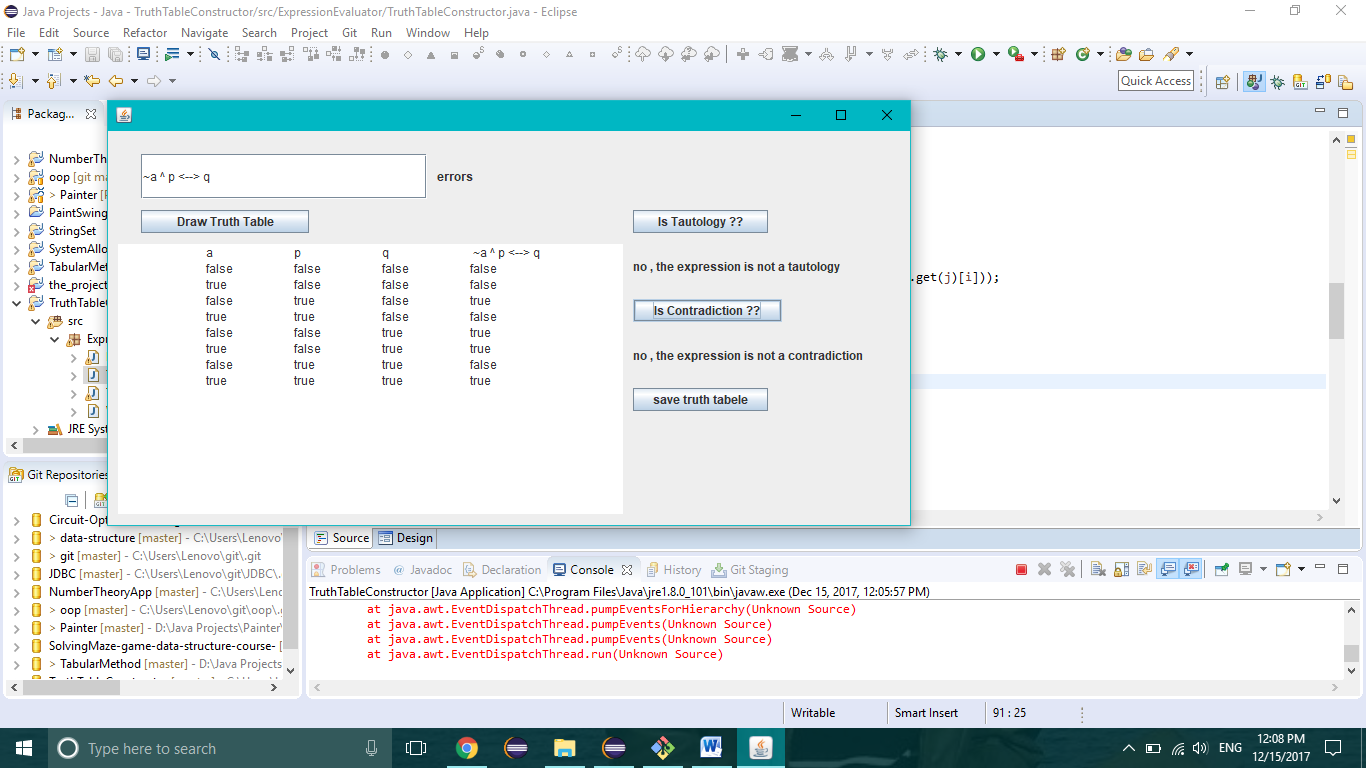
* The brackets allowed are ( and ) only
* Not is expressed as ~
* and is expressed as ^
* or is expressed as |
* implies is expressed as ->
* bi-condition is expressed as <->
* the program is case insensitive and spaces insensitive

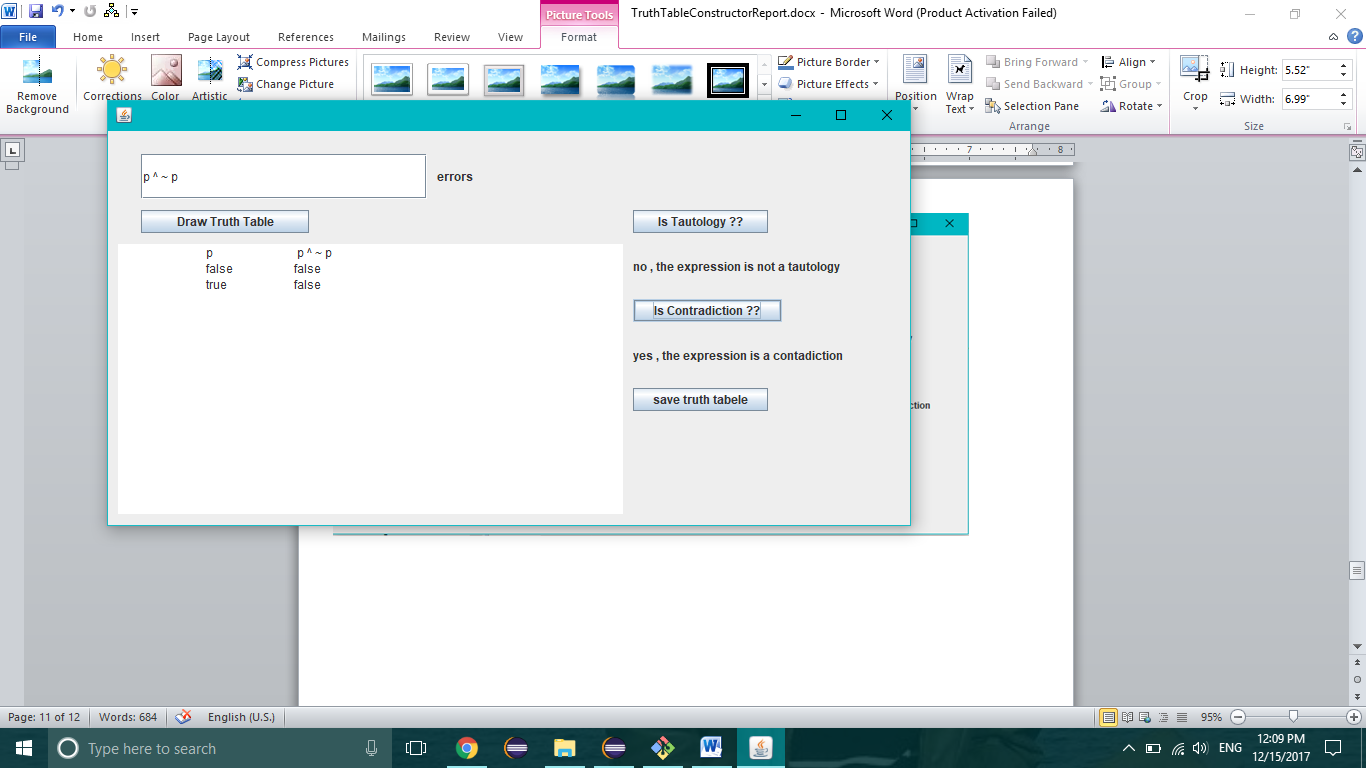
Sample Runs :











Analytical :

The program breaks down after 20 variables or becomes very slow