**/\* CS 6413 : Compiler Design & Construction Spring 2015 Instructed By : Dr. R. M. Siegfried**

**Code developed by :**

**Sumitra Kannan N12952143**

**TERM PROJECT : PART 2 :**

**AIM OF THE PROJECT :** Implement a symbol table for the Pascal subset specified in the Appendix of the “Dragon book” (downloadable from **My Poly**) You may use the code from the JASON compiler as a basis for your compiler, however, keep in mind that you need to add an auxiliary table to keep track of upper and lower bounds for array.

The attribute table now needs a pointer to the auxiliary table as well as an indication of how many auxiliary table entries it is pointing to (2 - one pointing to lower bound and one pointing to upper bound). The auxiliary table simple contains pointers to the appropriate attribute table entries.

**GRAMMAR SPECIFICATIONS FOR THE PROJECT :**

Program ::= program id ( IdList ) ; Declarations

SubprogramDeclarations CompoundStatement .

IdList ::= id | Idlist , id

Declarations ::= Declarations var IdList : Type ; | <nil>

Type ::= StandardType | array [ num .. num ] of StandardType

StandardType ::= integer | real

SubprogramDeclarations ::= SubprogramDeclarations SubprogramDeclaration ;

| <nil>

SubprogramDeclaration ::= SubprogramHead Declarations CompoundStatement

SubprogramHead ::= function id Arguments : StandardType ;

| procedure id Arguments ;

Arguments ::= ( ParameterList ) | <nil>

ParameterList ::= IdList : Type | ParameterList ; IdList : Type

CompoundStatement ::= begin OptionalStatements end

OptionalStatements ::= StatementList | <nil>

StatementList ::= Statement | StatementList ; Statement

Statement ::= Variable assignop Expression

| ProcedureStatement

| CompoundStatement

| if Expressions then Statement else Statement

| while Expressions do Statement

Variable ::= id | id [ Expression ]

ProcedureStatement ::= id | id ( ExpressionList )

ExpressionList ::= Expression | ExpressionList , Expression

Expression ::= SimpleExpression | SimpleExpression relop SimpleExpression

SimpleExpression ::= Term | Sign Term | SimpleExpression addop Term

Term ::= Factor | Term mulop Factor

Factor ::= id | id ( ExpressionList ) | num | ( Expression ) | not factor

Sign ::= + | -

**CODE :** /\* The aim of the code is to Implement a symbol table for the given Pascal subset.

\* Using the code from the JASON compiler as a basis for our Java compiler, an auxiliary table

\* has been added to keep track of upper and lower bounds for array.

The attribute table now needs a pointer to the auxiliary table as well as an indication

of how many auxiliary table entries it is pointing to (2 - one pointing to lower bound and

one pointing to upper bound). The auxiliary table simple contains pointers to the appropriate attribute table entries.

\*/

**import** java.\*;

**import** java.io.\*;

**import** java.util.\*;

**import** java.util.regex.\*;

**import** java.lang.\*;

**class** Symbol\_Table {

**public** **static** String *Number\_Token* = "tokNumber";

**public** **static** String *Word\_Token* = "tokWord";

**public** **static** String *Operator\_Token* = "tokOp";

**public** **static** Vector *Number\_TokVector*= **new** Vector(3,2);

**public** **static** Enumeration *Number\_Enum* = *Number\_TokVector*.elements();

**public** **static** Vector *Word\_TokVector*= **new** Vector(3,2);

**public** **static** Enumeration *Word\_Enum* = *Word\_TokVector*.elements();

**public** **static** Vector *Op\_TokVector*= **new** Vector(3,2);

**public** **static** Enumeration *Op\_Enum* = *Op\_TokVector*.elements();

String tok1;

**public** **void** getTokens(String Tokens) //getTokens() function breaks the input string into Tokens

{

String relops[]={"=","<>","<","<=",">=",">"};

String addops[]={"+","-"};

String mulops[]={"\*","/"};

**int** i;

StringTokenizer newtokens = **new** StringTokenizer(Tokens);

**while** (newtokens.hasMoreTokens()) { // Checks whether there is another coming up token in the code

String token = newtokens.nextToken() ;

String tokenType = getTokenType(token) ;

//System.out.println(token + "\t" + tokenType); // Returns the type of token entered, to the user

**if**(tokenType.equals("tokNumber")){

*Number\_TokVector*.addElement(**new** String(token)); // Stores the token of Type- Number for future use

System.***out***.println(token + "\t" + tokenType);

tok1=token; }

**if**(tokenType.equals("tokWord")) {

*Word\_TokVector*.addElement(**new** String(token)); // Stores the token of Type- Word for future use

**if**(token.equalsIgnoreCase("program"))

System.***out***.println(token + "\t" + "tokProgram");

**else**

**if**(tok1.equalsIgnoreCase("program"))

System.***out***.println(token + "\t" + "tokIdentifier");

**else**

**if**(tok1.equalsIgnoreCase("var"))

System.***out***.println(token + "\t" +"tokIdentifier");

**else**

**if**(tok1.equalsIgnoreCase(":"))

System.***out***.println(token + "\t" + "toktype");

**else**

**if**(token.equalsIgnoreCase("or"))

System.***out***.println(token + "\t" + "tokaddop");

**else**

**if**(token.equalsIgnoreCase("div"))

System.***out***.println(token + "\t" + "tokmulop");

**else**

**if**(token.equalsIgnoreCase("mod"))

System.***out***.println(token + "\t" + "tokmulop");

**else**

**if**(token.equalsIgnoreCase("and"))

System.***out***.println(token + "\t" + "tokmulop");

**else**

**if**(token.equalsIgnoreCase("begin"))

System.***out***.println(token + "\t" + "tokBegin");

**else**

**if**(token.equalsIgnoreCase("End"))

System.***out***.println(token + "\t" + "tokEnd");

**else**

System.***out***.println(token + "\t" + "tokWord");

tok1=token; }

**if**(tokenType.equals("tokOp")){

*Op\_TokVector*.addElement(**new** String(token)); // Stores the token of Type -Operator for future use

**if**(token.equalsIgnoreCase(";"))

System.***out***.println(token + "\t" + "toksemicolon");

**if**(token.equalsIgnoreCase("."))

System.***out***.println(token + "\t" + "tokperiod");

**if**(token.equalsIgnoreCase(":="))

System.***out***.println(token + "\t" + "tokassignop");

**for**(i=0;i<6;i++){

**if**(token.equalsIgnoreCase(relops[i]))

System.***out***.println(token + "\t" + "tokrelop"); }

**for**(i=0;i<2;i++){

**if**(token.equalsIgnoreCase(addops[i]))

System.***out***.println(token + "\t" + "tokaddop"); }

**for**(i=0;i<2;i++){

**if**(token.equalsIgnoreCase(mulops[i]))

System.***out***.println(token + "\t" + "tokmulop"); }

tok1=token; }

}

}

/\*

if(token.equalsIgnoreCase("begin"))

System.out.println(token + "\t" + "tokBegin");

f(token.equalsIgnoreCase("End"))

System.out.println(token + "\t" + "tokEnd");\*/

/\* In the below code, after identifying the token stored in a Vector ,

\* while creating symbol table it can for future use.\*/

**private** String getTokenType(String token) {

**if**(token != **null**) {

**if** (Pattern.*matches*("[\\d]+", token)) { // Pattern.matches() Checks whether the input token is a Number Token

**return** *Number\_Token*;

} **else** **if** (Pattern.*matches*("[\\w]+", token)) { // Pattern.matches() Checks whether the input token is a Word Token

**return** *Word\_Token*;

} **else** {

**return** *Operator\_Token*; // Returns the value of Operator Token

}

}

**return** **null**;

}

**public** **static** **void** main(String s[]) **throws** IOException {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***)); //Creates BufferedReader using System.in to read statement

**int** i;

Vector vtok = **new** Vector(3, 2);

Enumeration vEnum = vtok.elements();

String getstmt; //getstmt To get input string from user and store the same

Symbol\_Table st = **new** Symbol\_Table();

System.***out***.println("Input your Program Code: ");

System.***out***.println("When you have finished typing your code, enter Stop: ");

**do** {

getstmt = br.readLine();

vtok.addElement(**new** String(getstmt));

} **while** (!getstmt.equalsIgnoreCase("Stop"));

getstmt = "";

**while** (!getstmt.equalsIgnoreCase("Stop")) {

**while** (vEnum.hasMoreElements()) {

getstmt = (String) vEnum.nextElement();

**if** (getstmt.equalsIgnoreCase("Stop"))

**break**;

**else**

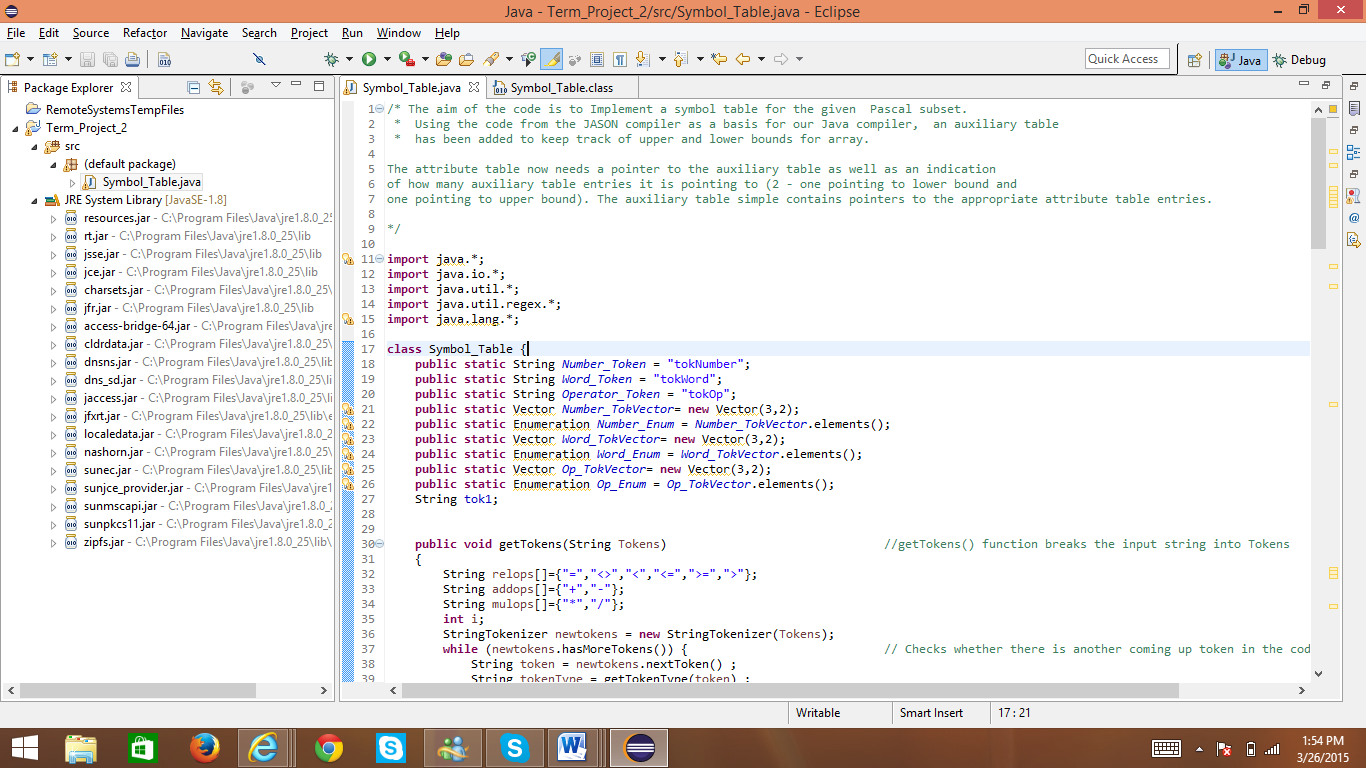
st.getTokens(getstmt); //getTokens() is called to identify and parse the tokens

}

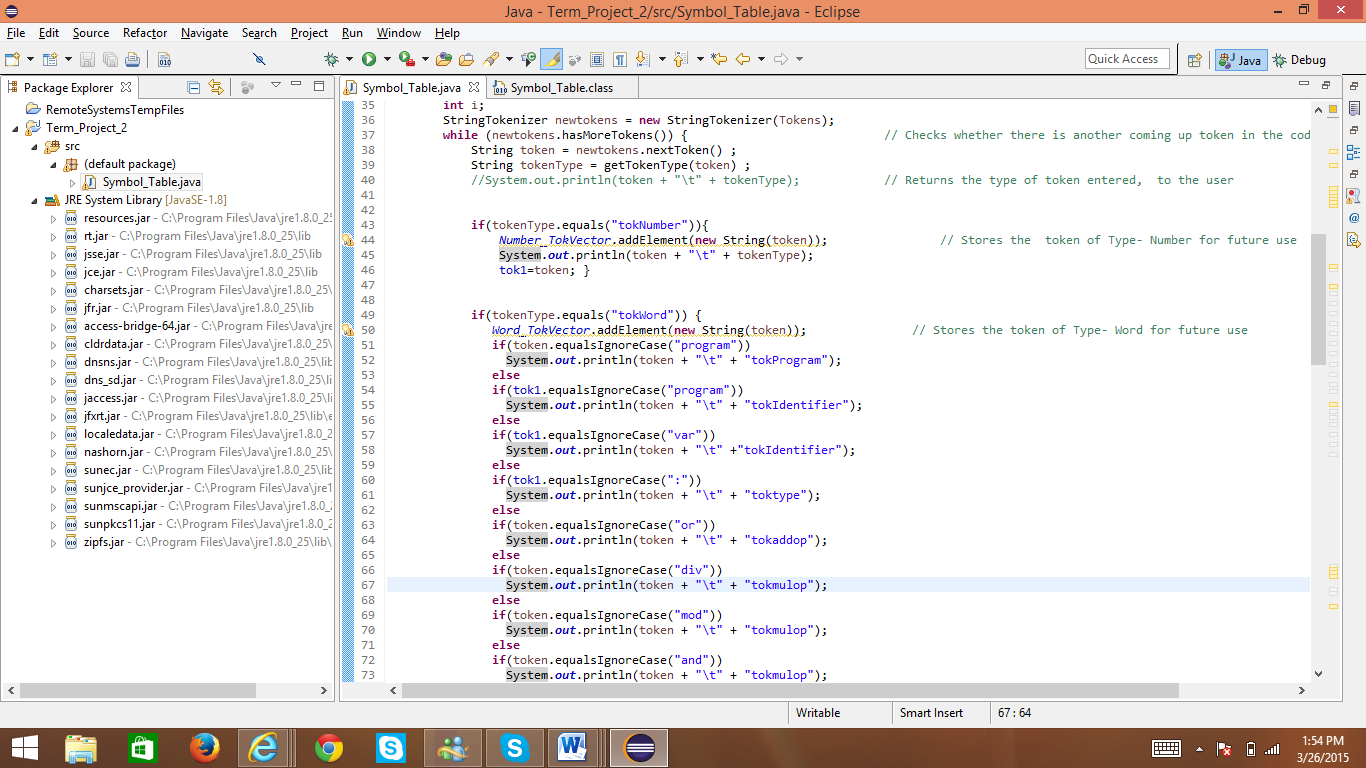
}

}

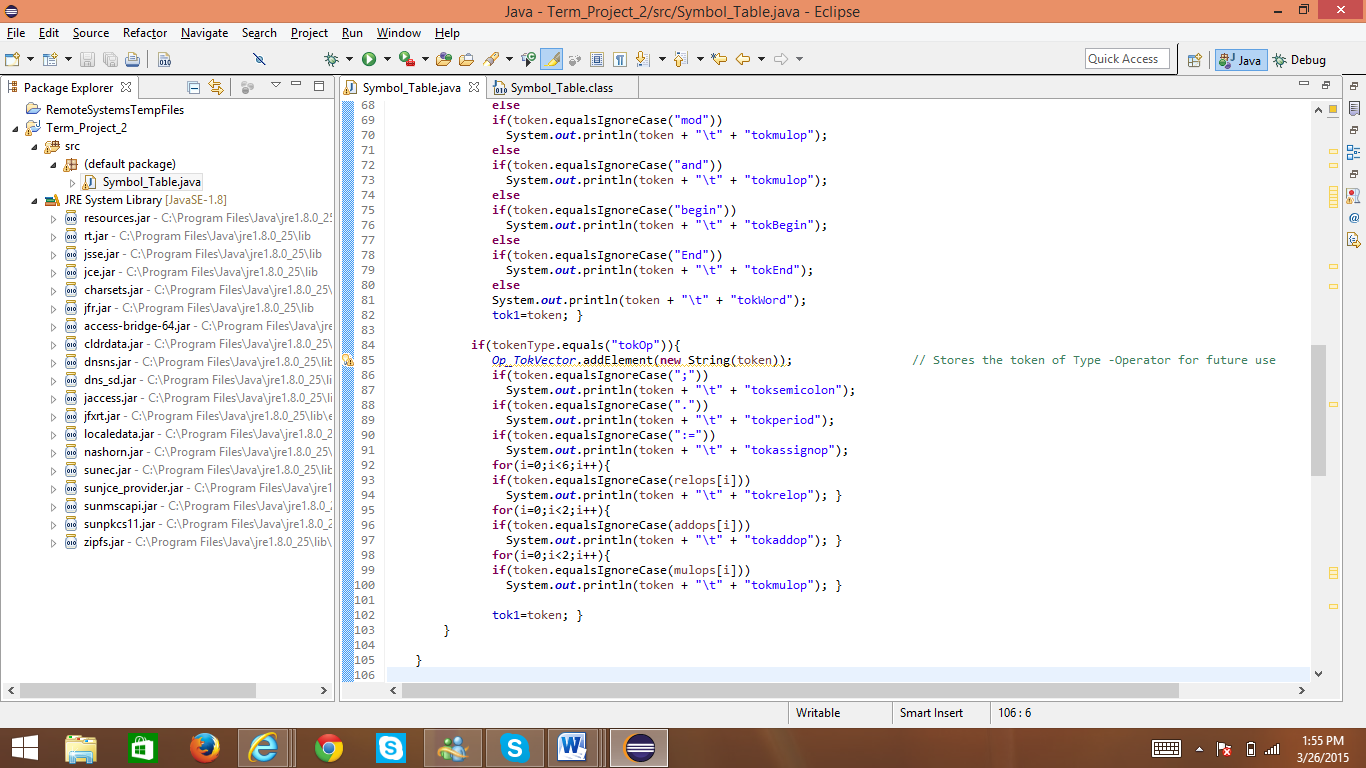
}



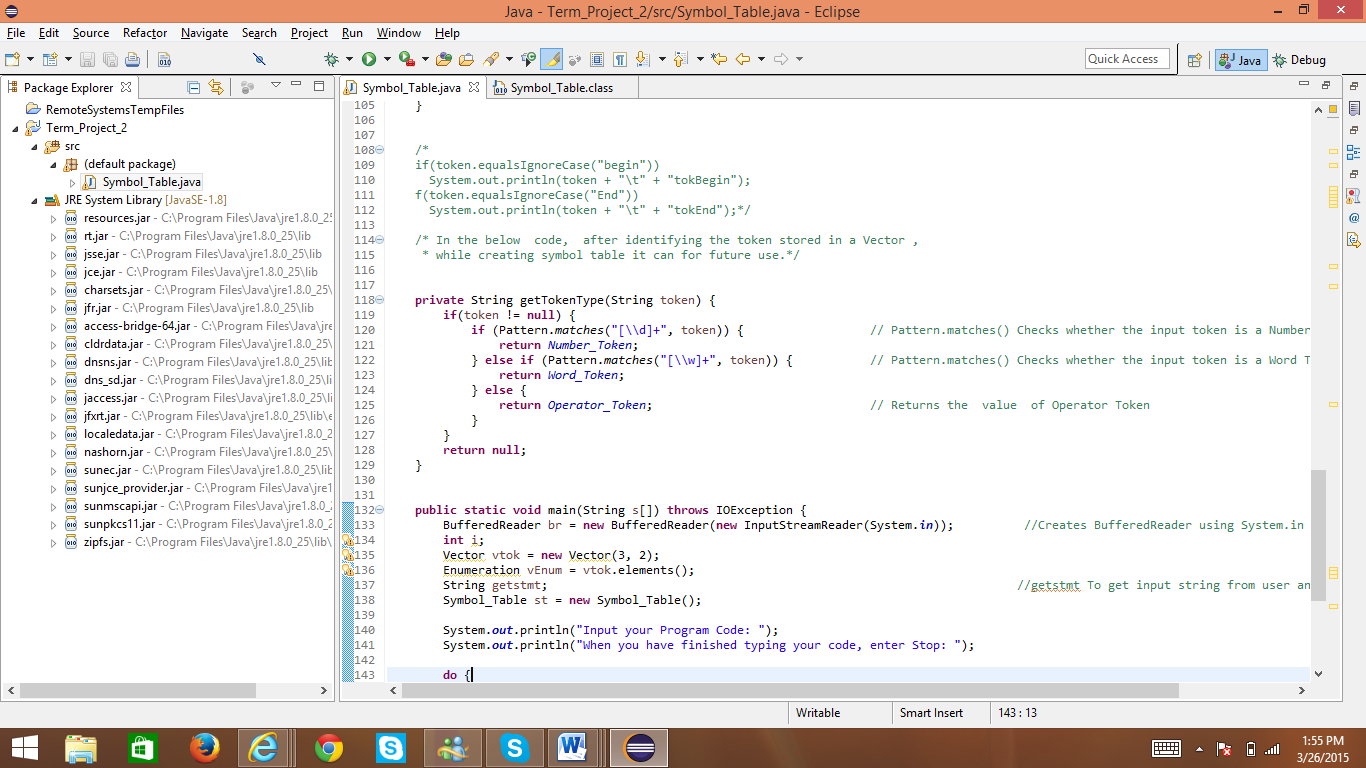
**Fig 1 : Screenshot of Symbol\_Table.java class in Eclipse Compiler**



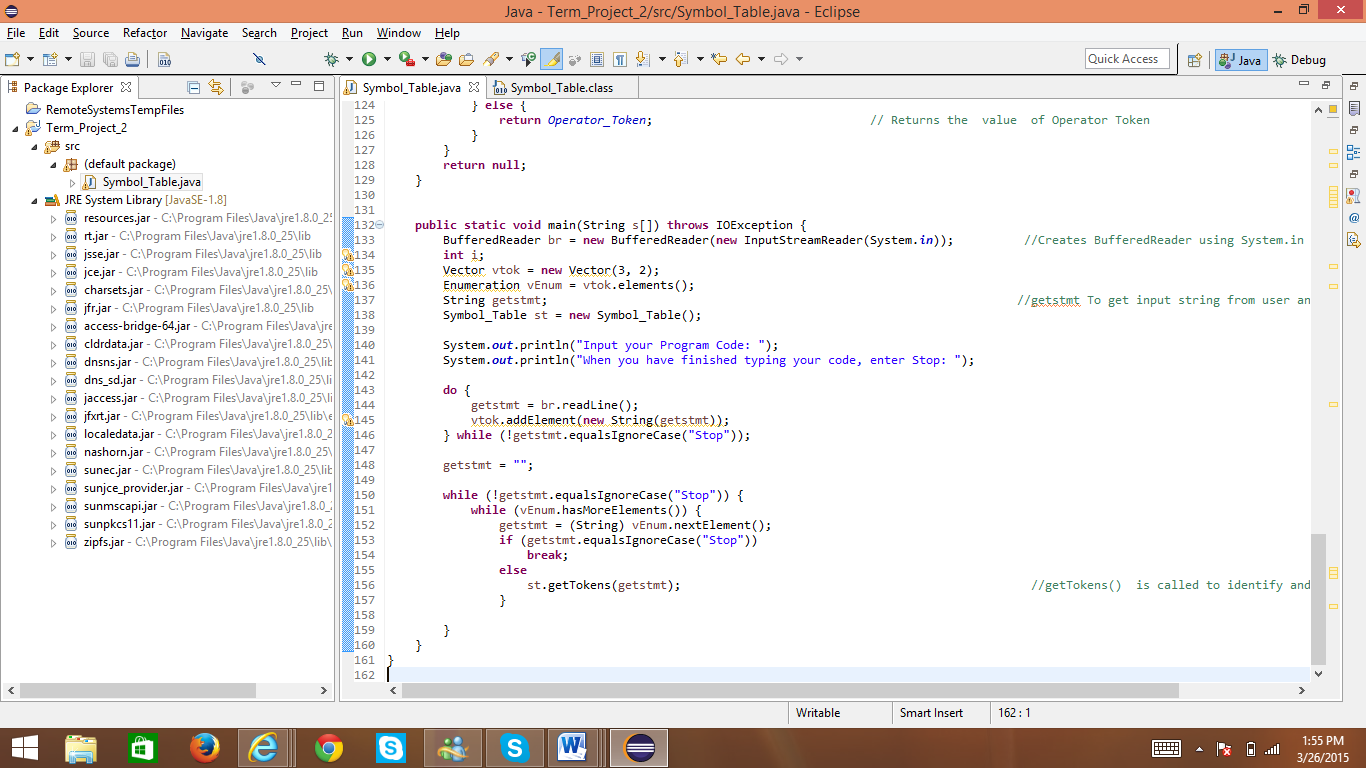
**Fig 2 : Screenshot of Symbol\_Table.java class in Eclipse Compiler**



**Fig 3 : Screenshot of Symbol\_Table.java class in Eclipse Compiler**



**Fig 4 : Screenshot of Symbol\_Table.java class in Eclipse Compiler**



**Fig 5 : Screenshot of Symbol\_Table.java class in Eclipse Compiler**

**Input for the Symbol\_Table.java Class :**

PROGRAM Sample ;

CONST

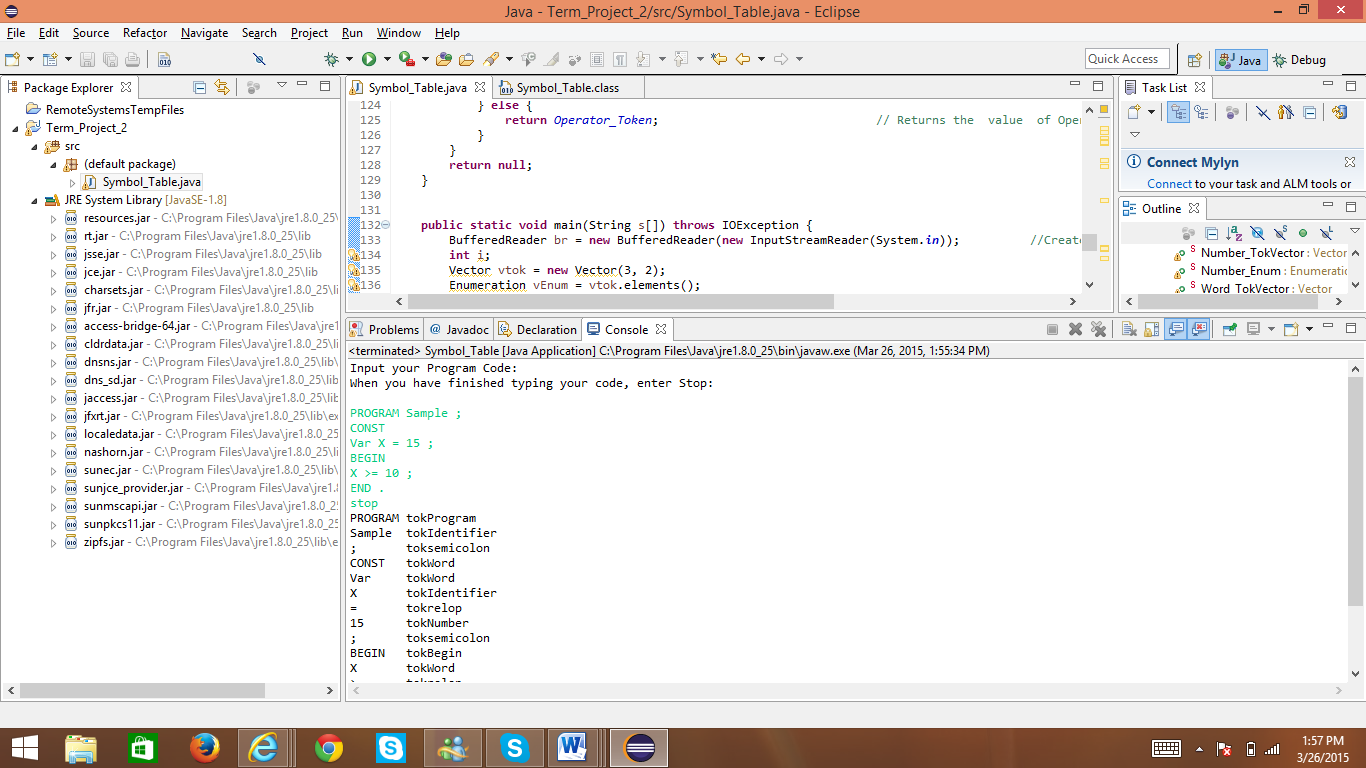
Var X = 15 ;

BEGIN

X >= 10 ;

END .

STOP



**Fig 6 : Screenshot of Execution of Symbol\_Table.java class in Eclipse Compiler ; The Screenshot Provides the input for the Class**

**Output for the Symbol\_Table.java Class :**

PROGRAM tokProgram

Sample tokIdentifier

; toksemicolon

CONST tokWord

var tokWord

X tokIdentifier

= tokrelop

15 tokNumber

; toksemicolon

BEGIN toksemicolon

X tokWord

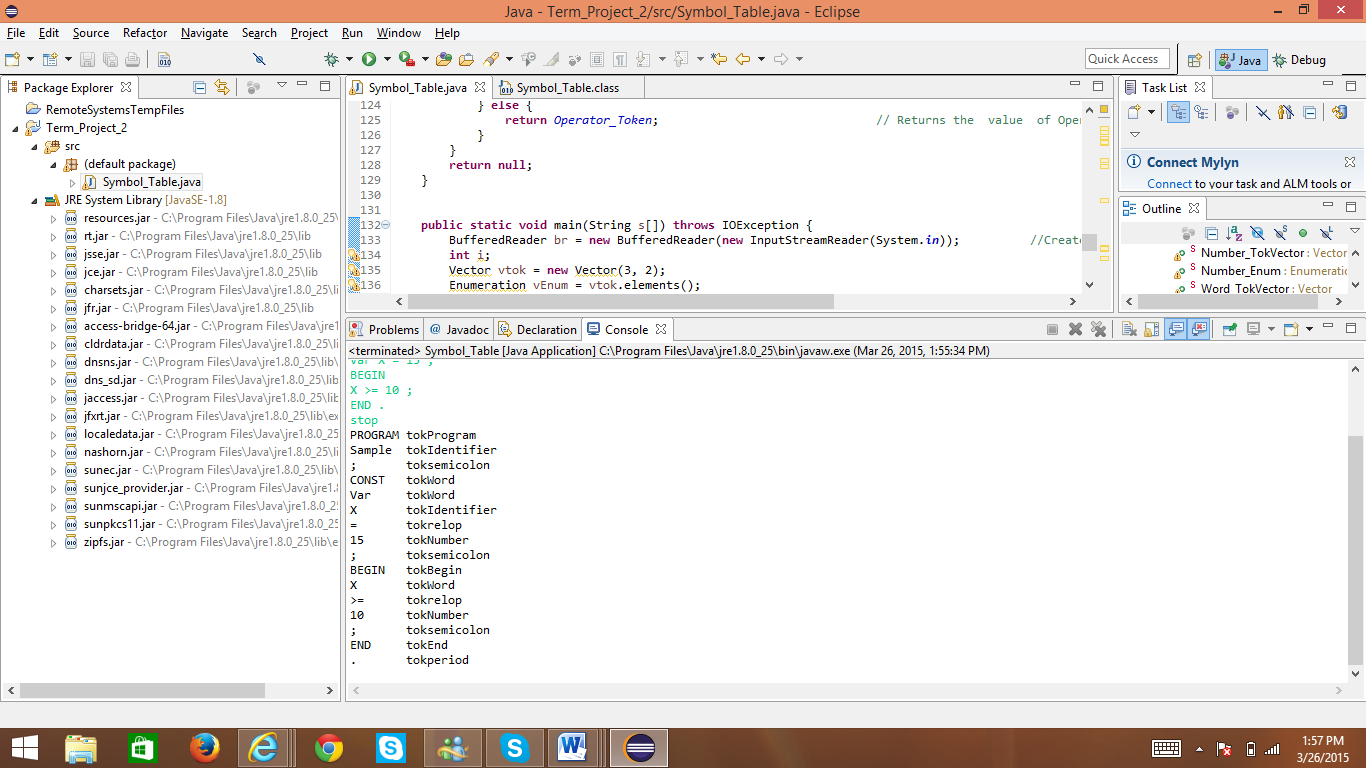
>= tokrelop

10 tokNumber

; toksemicolon

END tokEnd

. tokperiod



**Fig 7 : Screenshot of Execution of Symbol\_Table.java class in Eclipse Compiler ; The Screenshot Provides the output for the Class**