Name: Salik Ashraf

ID: 11349

Course: Software construction

Department: BS-CS

Project: Compiler

**Table of content**

Contents

[Choice of programming language 3](#_Toc61572933)

[Language of compiler 3](#_Toc61572934)

[Phases of compiler 6](#_Toc61572935)

[Lexical Errors 8](#_Toc61572936)

[Types of Parser 9](#_Toc61572937)

[Role of Parser 9](#_Toc61572938)

[Properties of intermediate code 10](#_Toc61572939)

[Level of designs 11](#_Toc61572940)

[Code Sample 11](#_Toc61572941)

[Screenshots 11](#_Toc61572942)

[GitHub Link 11](#_Toc61572943)

## Choice of programming language

The language on which this compiler is design is Java and I used that language as it is an object oriented language so it will be more beneficial to design the compiler in Java because of its object oriented capability as well as it is the requirement of this project.

## Language of compiler

Our language will have the following specifications.

**Identifier Rules**

1. Identifier can be of maximum length 6.
2. Identifiers are not case sensitive.
3. An Identifier can only have alphanumeric characters( a-z , A-Z , 0-9 ) and underscore(\_).
4. The first character of an identifier can only contain alphabet( a-z , A-Z ).
5. Keywords are not allowed to be used as Identifiers.
6. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

**Data Types:**

Our language supports only 3 datatypes

1. Integer
2. String
3. Character

**Expressions**

1. Arithmetic operators (+, -, \*, /, %)
2. Uniray operator
3. Paranthesis
4. Only Integer supported
5. Relational expression to be supported (>, <, >=, <=, ==, !=)
6. Character string and integer constants

e.g.    int const 4

char const ‘4’

string const “4”

**Statements**

1. Declaration statement : int a;
2. Declaration and Initialisation : int a=5;
3. Assingment Statement : a=6;
4. Conditional statement

**Simple if (nesting not allowed)**

if then

Endif

**Switch Statement (nesting not allowed)**

Switch()

Cases

Value 1:

Break;

 Value n:

break;

Endcase

**Repetition Statement (nesting not allowed)**

* 1. Repeat

Until ()

* 1. While (relational expression)

Endwhile

* 1. For = start value, end value, inc/dec

………

Endfor

**I/O Statement**

1. Input ;
2. Output ;

**Program Structure**

Decleration:

Start

End

# Phases of compiler

**Overview**

Analysis part of compiler breaks the source program into constituent pieces and imposes a grammatical structure on them which further uses this structure to create an intermediate representation of the source program. It is also termed as front end of compiler.

**Lexical analyser**

Lexical analysis is the process of converting a sequence of characters from source program into a sequence of tokens. A program which performs lexical analysis is termed as a lexical analyzer (lexer), tokenizer or scanner.

Lexical analysis consists of two stages of processing which are as follows:

• Scanning

• Tokenization

Token is a valid sequence of characters which are given by lexeme. In a programming language,

• keywords,

• constant,

• identifiers,

• numbers,

• operators and

• punctuations symbols

are possible tokens to be identified.

For example : c=a+b;

In this c,a and b are identifiers and ‘=’ and ‘\*’ are mathematical operators.

## Lexical Errors

• A character sequence that cannot be scanned into any valid token is a lexical error.

• Lexical errors are uncommon, but they still must be handled by a scanner.

10

• Misspelling of identifiers, keyword, or operators are considered as lexical errors.

Usually, a lexical error is caused by the appearance of some illegal character, mostly at the beginning of a token.

**Syntax analyser**

Syntax analysis is the second phase of compiler. Syntax analysis is also known as parsing.

Parsing is the process of determining whether a string of tokens can be generated by a grammar.

It is performed by syntax analyser which can also be termed as parser.

In addition to construction of the parse tree, syntax analysis also checks and reports syntax errors accurately. Parser is a program that obtains tokens from lexical analyser and constructs the parse tree which is passed to the next phase of compiler for further processing.

Parser implements context free grammar for performing error checks.

## Types of Parser

* Top down parsers Top down parsers construct parse tree from root to leaves.
* Bottom up parsers Bottom up parsers construct parse tree from leaves to root.

## Role of Parser

• Once a token is generated by the lexical analyser, it is passed to the parser.

• On receiving a token, the parser verifies the string of token names that can be generated by the grammar of source language.

• It calls the function getNextToken(), to notify the lexical analyser to yield another token.

• It scans the token one at a time from left to right to construct the parse tree.

• It also checks the syntactic constructs of the grammar.

**Semantic analyser**

• Semantic analysis is the third phase of compiler.

• It checks for the semantic consistency.

• Type [information](https://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) is gathered and stored in symbol table or in syntax tree.

• Performs type checking.

**Intermediate code generator**

Intermediate code generation is the process by which a compiler's code generator converts some intermediate representation of source code into a form (e.g., machine code) that can be readily executed by a machine.

Intermediate code generation produces intermediate representations for the source program which are of the following forms:

     o Postfix notation

     o Three address code

     o Syntax tree

Most commonly used form is the three address code.

**t1 = inttofloat (5)**

**t2 = id3\* tl**

**t3 = id2 + t2**

**id1 = t3**

### **Properties of intermediate code**

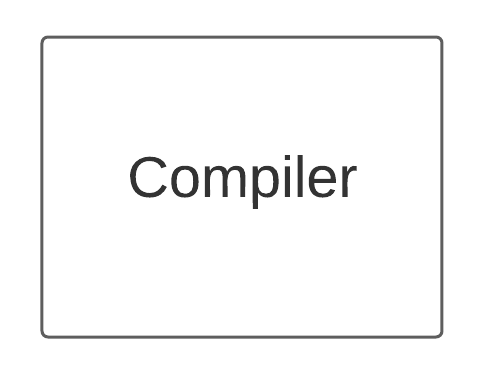
• It should be easy to produce.

• It should be easy to translate into target program.

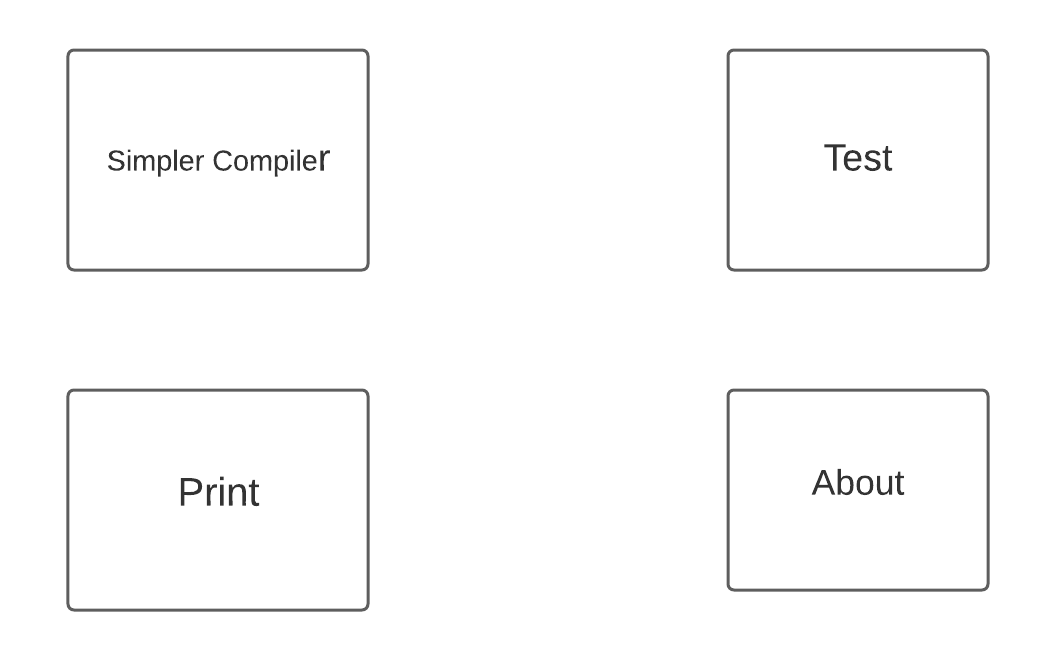
After intermediate code generation the front end part of compiler finishes.The output to intermediate code generated is fed as input to back end of compiler , which converts this Intermediate code to machine code.

# Level of designs

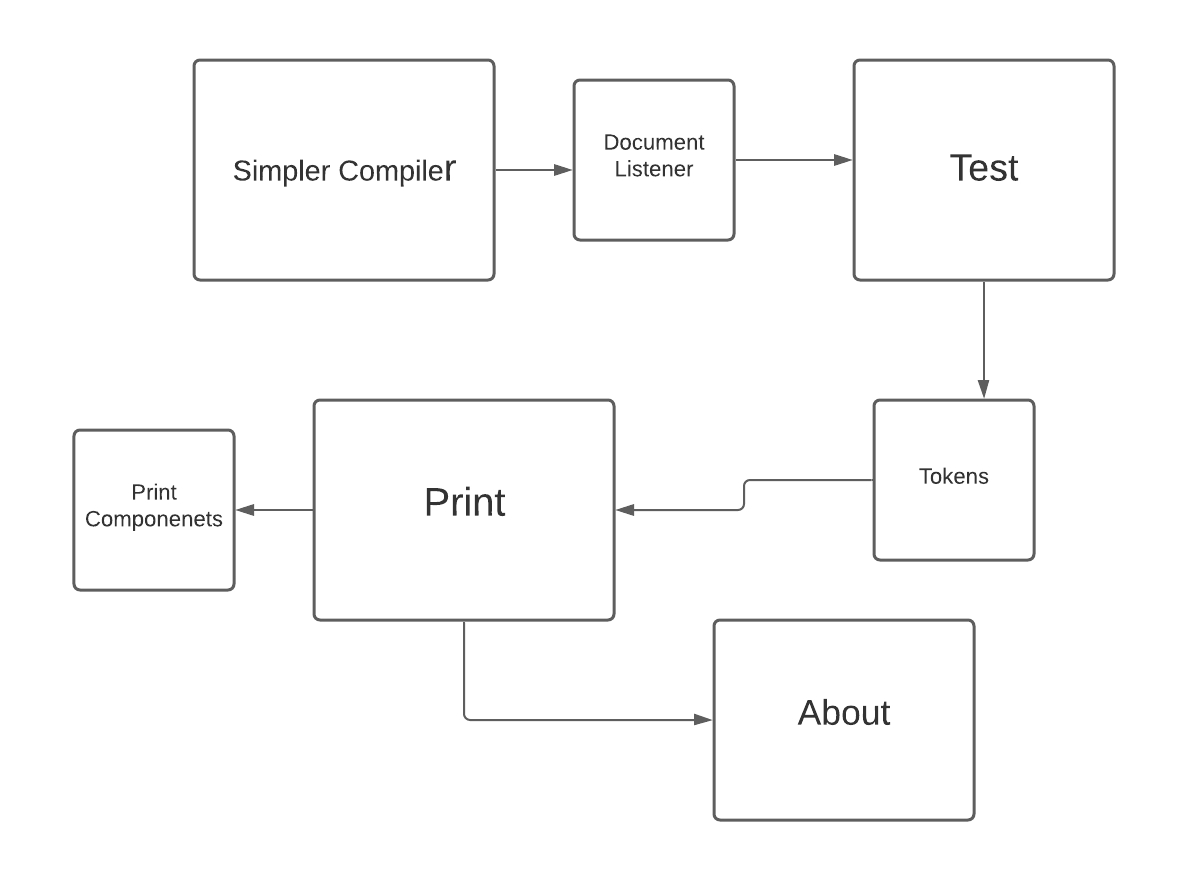
Software system



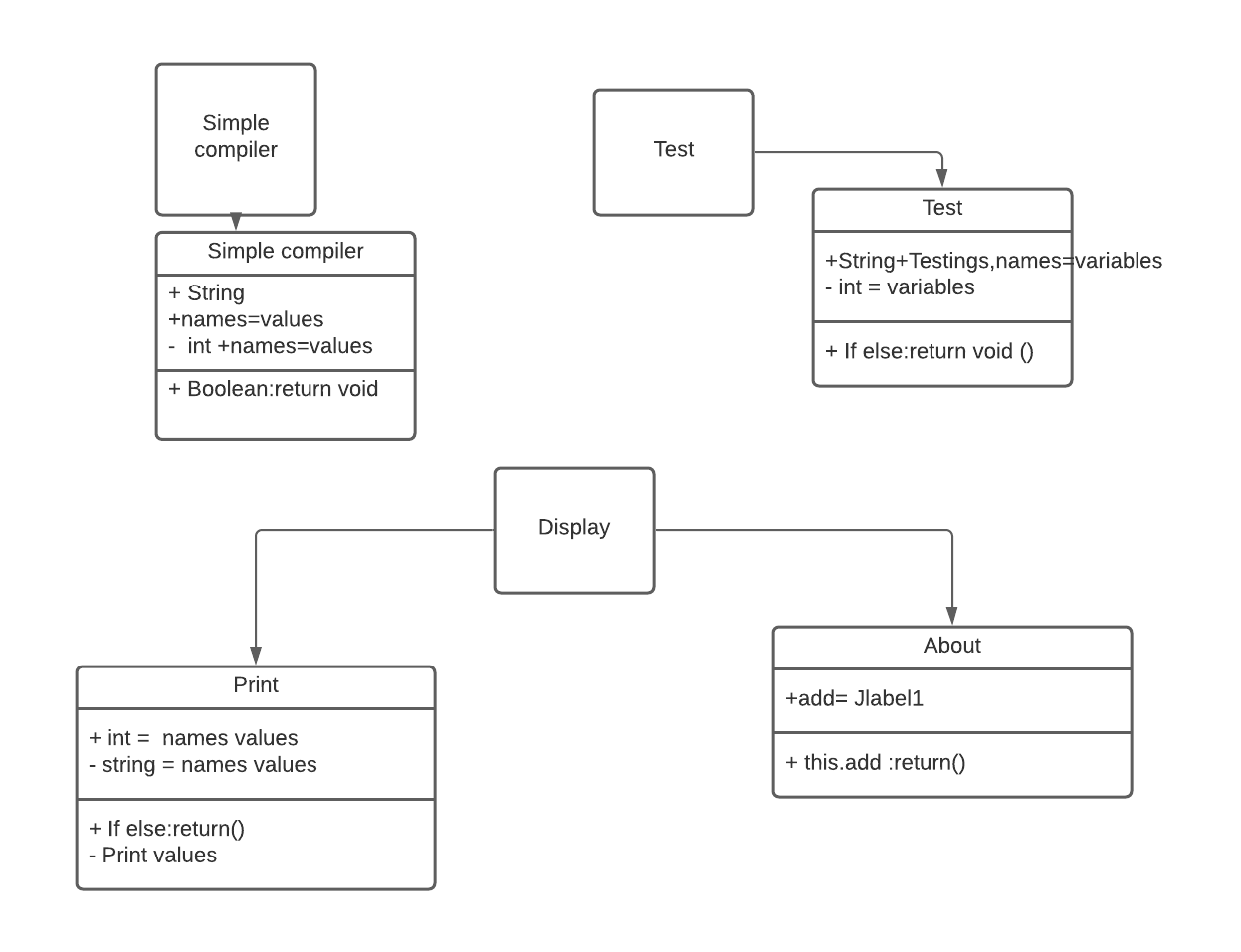
Division into sub systems



Division into Classes within subsystems



Routine Diagram



Internal Routine Design:

Classes:

simple compiler

In this class first of all I have to import java.awt for Jframe then use JText Area , JFile chooser , JFrame Bar and other components as well after that I have to add content pane and use some “try catch “ blocks for the functionality purpose , windows listener and windows adapter as well after that I have to input events , keys and handle Jmenu Items and have to if else and while loops as well where they are necessary.

Test

In Public class test I have to use JOption Pane , Tokens using if else , while statements and use print statements like syste.out.println or something instead of that which is used in Java to display anything on the screen.

Print

In public class Print I have to handle specially the printing functionality and Printing Exceptions like when error appears on the screen due to some reason and other printing functionalities aswell.

About

This class will tell the information about the project.

### Source Code:

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import java.util.\*;

import java.util.StringTokenizer;

import javax.swing.\*;

import javax.swing.border.\*;

import javax.swing.event.\*;

public class SimpleCompiler

extends JFrame {

public static final String APP\_NAME = "Simple Compiler Version 1.0";

protected JTextArea m\_editor;

protected JTextArea errorMsg;

protected JFileChooser m\_chooser;

protected File m\_currentFile;

protected boolean m\_textChanged = false;

protected JToolBar m\_toolBar;

protected JComboBox m\_cbFonts;

String status;

public SimpleCompiler() {

super(APP\_NAME+": Part IV - Custom Menu Components");

setSize(500, 350);

m\_editor = new JTextArea(12,10);

errorMsg =new JTextArea(4,10);

JScrollPane pl = new JScrollPane(errorMsg);

JScrollPane ps = new JScrollPane(m\_editor);

getContentPane().add(ps, BorderLayout.CENTER);

getContentPane().add(pl, BorderLayout.SOUTH);

JMenuBar menuBar = createMenuBar();

setJMenuBar(menuBar);

m\_chooser = new JFileChooser();

try {

File dir = (new File(".")).getCanonicalFile();

m\_chooser.setCurrentDirectory(dir);

} catch (IOException ex) {}

newDocument();

WindowListener wndCloser = new WindowAdapter() {

public void windowClosing(WindowEvent e) {

if (!promptToSave())

return;

System.exit(0);

}

};

addWindowListener(wndCloser);

}

protected JMenuBar createMenuBar() {

final JMenuBar menuBar = new JMenuBar();

JMenu mFile = new JMenu("File");

mFile.setMnemonic('f');

ImageIcon iconNew = new ImageIcon("icons/New.gif");

Action actionNew = new AbstractAction("New", iconNew) {

public void actionPerformed(ActionEvent e) {

if (!promptToSave())

return;

newDocument();

}

};

JMenuItem item = new JMenuItem(actionNew);

item.setMnemonic('n');

item.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_N, InputEvent.CTRL\_MASK));

mFile.add(item);

ImageIcon iconOpen = new ImageIcon("icons/Open.gif");

Action actionOpen = new AbstractAction("Open...", iconOpen) {

public void actionPerformed(ActionEvent e) {

if (!promptToSave())

return;

openDocument();

}

};

item = new JMenuItem(actionOpen);

item.setMnemonic('o');

item.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_O, InputEvent.CTRL\_MASK));

mFile.add(item);

ImageIcon iconSave = new ImageIcon("icons/Save.gif");

Action actionSave = new AbstractAction("Save", iconSave) {

public void actionPerformed(ActionEvent e) {

if (!m\_textChanged)

return;

saveFile(false);

}

};

item = new JMenuItem(actionSave);

item.setMnemonic('s');

item.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_S, InputEvent.CTRL\_MASK));

mFile.add(item);

ImageIcon iconSaveAs = new ImageIcon("icons/saveall.gif");

Action actionSaveAs = new AbstractAction("Save As...", iconSaveAs) {

public void actionPerformed(ActionEvent e) {

saveFile(true);

}

};

item = new JMenuItem(actionSaveAs);

item.setMnemonic('a');

mFile.add(item);

mFile.addSeparator();

Action actionExit = new AbstractAction("Exit") {

public void actionPerformed(ActionEvent e) {

System.exit(0);

}

};

item = new JMenuItem(actionExit);

item.setMnemonic('x');

mFile.add(item);

menuBar.add(mFile);

// Add Project Menu

JMenu mOpt = new JMenu("Project");

mOpt.setMnemonic('p');

ImageIcon iconCompile = new ImageIcon("icons/compile.gif");

Action actionCompile = new AbstractAction("Compile File...", iconCompile) {

public void actionPerformed(ActionEvent e) {

status="ok";

pile();

}

};

item = new JMenuItem(actionCompile);

item.setMnemonic('C');

mOpt.add(item);

menuBar.add(mOpt);

ImageIcon iconPrint = new ImageIcon("icons/print.gif");

Action actionPrint = new AbstractAction("Print...", iconPrint) {

public void actionPerformed(ActionEvent e) {

Print.printComponent((m\_editor));

}

};

item = new JMenuItem(actionPrint);

item.setMnemonic('P');

mOpt.add(item);

// Help Project Menu

JMenu mHelp = new JMenu("Help");

mFile.setMnemonic('H');

ImageIcon iconAbout = new ImageIcon("icons/help.gif");

Action actionAbout = new AbstractAction("About ?", iconAbout) {

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(null, new About(),"About Simple Compiler",JOptionPane.PLAIN\_MESSAGE);

}

};

item = new JMenuItem(actionAbout);

item.setMnemonic('A');

mHelp.add(item);

menuBar.add(mHelp);

// Create toolbar

m\_toolBar = new JToolBar("Commands");

JButton bNew = new SmallButton(actionNew,

"New File");

m\_toolBar.add(bNew);

JButton bOpen = new SmallButton(actionOpen,

"Open File");

m\_toolBar.add(bOpen);

JButton bSave = new SmallButton(actionSave,

"Save file");

m\_toolBar.add(bSave);

m\_toolBar.addSeparator();

JButton bPrint=new SmallButton(actionPrint,"Print File");

m\_toolBar.add(bPrint);

JButton bCompile = new SmallButton(actionCompile,"Compile program");

m\_toolBar.add(bCompile);

m\_toolBar.addSeparator();

JButton bAbout = new SmallButton(actionAbout,"About Compiler");

m\_toolBar.add(bAbout);

getContentPane().add(m\_toolBar, BorderLayout.NORTH);

return menuBar;

}

protected String getDocumentName() {

return m\_currentFile==null ? "Untitled" :

m\_currentFile.getName();

}

protected void newDocument() {

m\_editor.setText("");

m\_currentFile = null;

setTitle(APP\_NAME+" ["+getDocumentName()+"]");

m\_textChanged = false;

m\_editor.getDocument().addDocumentListener(new UpdateListener());

}

protected void openDocument() {

if (m\_chooser.showOpenDialog(SimpleCompiler.this) !=

JFileChooser.APPROVE\_OPTION)

return;

File f = m\_chooser.getSelectedFile();

if (f == null || !f.isFile())

return;

m\_currentFile = f;

try {

FileReader in = new FileReader(m\_currentFile);

m\_editor.read(in, null);

in.close();

setTitle(APP\_NAME+" ["+getDocumentName()+"]");

}

catch (IOException ex) {

showError(ex, "Error reading file "+m\_currentFile);

}

m\_textChanged = false;

m\_editor.getDocument().addDocumentListener(new UpdateListener());

}

protected boolean saveFile(boolean saveAs) {

if (saveAs || m\_currentFile == null) {

if (m\_chooser.showSaveDialog(SimpleCompiler.this) !=

JFileChooser.APPROVE\_OPTION)

return false;

File f = m\_chooser.getSelectedFile();

if (f == null)

return false;

m\_currentFile = f;

setTitle(APP\_NAME+" ["+getDocumentName()+"]");

}

try {

FileWriter out = new

FileWriter(m\_currentFile);

m\_editor.write(out);

out.close();

}

catch (IOException ex) {

showError(ex, "Error saving file "+m\_currentFile);

return false;

}

m\_textChanged = false;

return true;

}

protected boolean promptToSave() {

if (!m\_textChanged)

return true;

int result = JOptionPane.showConfirmDialog(this,

"Save changes to "+getDocumentName()+"?",

APP\_NAME, JOptionPane.YES\_NO\_CANCEL\_OPTION,

JOptionPane.INFORMATION\_MESSAGE);

switch (result) {

case JOptionPane.YES\_OPTION:

if (!saveFile(false))

return false;

return true;

case JOptionPane.NO\_OPTION:

return true;

case JOptionPane.CANCEL\_OPTION:

return false;

}

return true;

}

public void showError(Exception ex, String message) {

ex.printStackTrace();

JOptionPane.showMessageDialog(this,

message, APP\_NAME,

JOptionPane.WARNING\_MESSAGE);

}

public static void main(String argv[]) {

SimpleCompiler frame = new SimpleCompiler();

frame.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

frame.setVisible(true);

}

class SmallButton extends JButton implements MouseListener {

protected Border m\_raised = new SoftBevelBorder(BevelBorder.RAISED);

protected Border m\_lowered = new SoftBevelBorder(BevelBorder.LOWERED);

protected Border m\_inactive = new EmptyBorder(3, 3, 3, 3);

protected Border m\_border = m\_inactive;

protected Insets m\_ins = new Insets(4,4,4,4);

public SmallButton(Action act, String tip) {

super((Icon)act.getValue(Action.SMALL\_ICON));

setBorder(m\_inactive);

setMargin(m\_ins);

setToolTipText(tip);

setRequestFocusEnabled(false);

addActionListener(act);

addMouseListener(this);

}

public float getAlignmentY() {

return 0.5f;

}

public Border getBorder() {

return m\_border;

}

public Insets getInsets() {

return m\_ins;

}

public void mousePressed(MouseEvent e) {

m\_border = m\_lowered;

setBorder(m\_lowered);

}

public void mouseReleased(MouseEvent e) {

m\_border = m\_inactive;

setBorder(m\_inactive);

}

public void mouseClicked(MouseEvent e) {}

public void mouseEntered(MouseEvent e) {

m\_border = m\_raised;

setBorder(m\_raised);

}

public void mouseExited(MouseEvent e) {

m\_border = m\_inactive;

setBorder(m\_inactive);

}

}

class UpdateListener implements DocumentListener {

public void insertUpdate(DocumentEvent e) {

m\_textChanged = true;

}

public void removeUpdate(DocumentEvent e) {

m\_textChanged = true;

}

public void changedUpdate(DocumentEvent e) {

m\_textChanged = true;

}

}

public void pile( )

{

errorMsg.setText("");

StringTokenizer st=new StringTokenizer(m\_editor.getText());

String akhir=new String();

errorMsg.append("Compiling 1 file.....\n");

String mula=new String(st.nextToken());

if(mula.equals("Start"))

{

while (st.hasMoreTokens())

{

akhir=st.nextToken();

}

if(akhir.equals("End"))

DecState(m\_editor.getText());

else

errorMsg.append("ERROR Missing End statement\n");

}

else

errorMsg.append("ERROR Missing Start statement\n");

}

public void DecState(String stat)

{

String pencam;

StringTokenizer st=new StringTokenizer(stat);

st.nextToken();

pencam=st.nextToken();

if (("Set").equals(pencam))

{

pencam=st.nextToken();

if (Character.isLetter(pencam.charAt(0)))

{

checkPencam(pencam);

if(status.equals("ok"))

{

pencam=st.nextToken();

if (("As").equals(pencam))

{

pencam=st.nextToken();

if( (("int").equals(pencam))|| (("char").equals(pencam)) ||(("double").equals(pencam)) || (("String").equals(pencam)) )

{

st.nextToken();

if(st.hasMoreTokens())

errorMsg.append("\nERROR : Syntax Error --> Set VariableName As DataType");

else

errorMsg.append("\nCompiler Successful");

}

else

errorMsg.append("ERROR "+pencam+" Not a valid datatype");

}

else

errorMsg.append("ERROR "+pencam + " Not reconized");

}

else;

}

else

errorMsg.append("ERROR "+pencam + " Illegal start of variable name!!");

}

else

errorMsg.append("ERROR "+pencam + " Not reconized");

}

public String checkPencam(String s)

{

int i;

for(i=1;i<s.length();i++)

{

if((Character.isLetterOrDigit(s.charAt(i))) || (s.charAt(i)=='\_') );

else

{

errorMsg.append("\nERROR "+ s +" - Illegal character ("+ s.charAt(i)+") in variable name");

status="error";

}

}

return status;}

}

# 

import javax.swing.JOptionPane;

import java.util.StringTokenizer;

public class test

{

public static void main (String[] args)

{

String stat = JOptionPane.showInputDialog(null,

"Enter a statement","Height",JOptionPane.QUESTION\_MESSAGE);

System.out.println("Testing Compiler for statement " + stat+"\n");

StringTokenizer st=new StringTokenizer(stat);

String akhir=new String();

System.out.println("The total number or words is " + st.countTokens());

String mula=new String(st.nextToken());

if(mula.equals("Start")){

while (st.hasMoreTokens())

{

akhir=st.nextToken();

}

if(akhir.equals("End"))

DecState(stat);

else

System.out.println("Missing End statement");

}

else

System.out.println("Missing Start statement");

}

public static void DecState(String stat)

{

String pencam;

StringTokenizer st=new StringTokenizer(stat);

st.nextToken();

pencam=st.nextToken();

if (("Set").equals(pencam))

{

pencam=st.nextToken();

if (Character.isLetter(pencam.charAt(0)))

{

pencam=st.nextToken();

if (("As").equals(pencam))

{

pencam=st.nextToken();

if( (("int").equals(pencam))|| (("char").equals(pencam)) ||(("double").equals(pencam)) )

System.out.println("Compiler Successful");

}

else

System.out.println(pencam + " Not reconized");

}

else

System.out.println(pencam + " Not a valid variable name");

}

else

System.out.println(pencam + " Not reconized");

}

}

import java.awt.\*;

import java.awt.print.\*;

import javax.swing.\*;

/\*\*

\*A class for creating a printer option.

\*/

public class Print implements Printable{

private Component componentToBePrinted;

public static void printComponent(Component c){

new Print(c).print();

}

public Print(Component componentToBePrinted){

this.componentToBePrinted = componentToBePrinted;

}

public void print(){

PrinterJob printJob = PrinterJob.getPrinterJob();

printJob.setPrintable(this);

if(printJob.printDialog())

try{

printJob.print();

}

catch(PrinterException pe){

System.out.println("Error printing: " + pe);

}

}

public int print(Graphics g, PageFormat pageFormat, int pageIndex){

if(pageIndex > 0){

return(NO\_SUCH\_PAGE);

}

else{

Graphics2D g2d = (Graphics2D)g;

g2d.translate(pageFormat.getImageableX(), pageFormat.getImageableY());

disableDoubleBuffering(componentToBePrinted);

componentToBePrinted.paint(g2d);

enableDoubleBuffering(componentToBePrinted);

return(PAGE\_EXISTS);

}

}

public static void disableDoubleBuffering(Component c){

RepaintManager currentManager = RepaintManager.currentManager(c);

currentManager.setDoubleBufferingEnabled(false);

}

public static void enableDoubleBuffering(Component c){

RepaintManager currentManager = RepaintManager.currentManager(c);

currentManager.setDoubleBufferingEnabled(true);

}

}

import java.awt.\*;

import javax.swing.\*;

/\*\*

\*A CLASS FOR CREATING ABOUT PANEL

\*/

public class About extends JPanel{

public About(){

//Create a Label & an image icon in it

JLabel label1 = new JLabel(new ImageIcon(""));

//adding label1 to the JPanel

this.add(label1);

//Create a Label & put a HTML script

JLabel label2 = new JLabel("<html><li>Simple Compiler™</li><li><p>Ver 1.0</li>"

+"<li><p>Developed by: Salik Ashraf </li><li><p>ID:11349</li><li>"

+"<p>Course - Software Construction</li>"

+"<li><p>Faculty : Dr Farooq Iqbal & Dr Nadeem </li>"

+"<li><p>By Salik Ashraf(11349)</li></html>");

//adding label2 to the JPanel

this.add(label2);

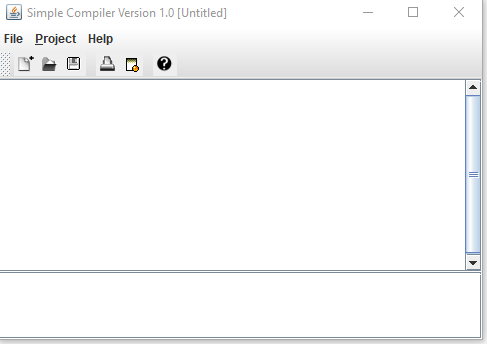
}

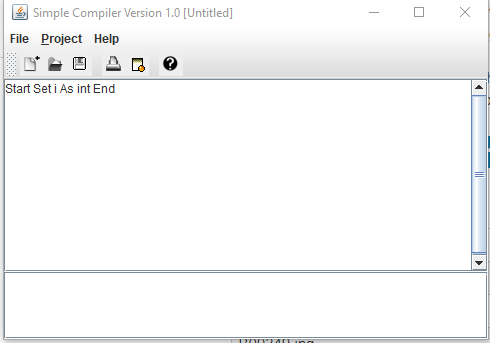
}

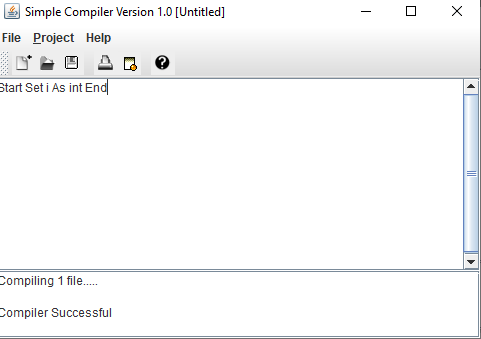
# Code Sample

Start Set i As int End

# Screenshots







# GitHub Link

[**https://github.com/Sonic146/SimpleCompiler.git**](https://github.com/Sonic146/SimpleCompiler.git)