|  |
| --- |
| **Simple Java IDE** |
| By students of Valliammai Engineering College |
|  |
| This project work in done as per the compulsary project in Java lab, by students of CSE 3year Academic year 2011 – 2012, to Successfully complete the course. |
|  |
|  |
| **2011** |
|  |

**Aim**

The aim of the project is to create a IDE which is simple and effective to us. This project will let students to practise their lab programs in a easier way, so that they can concentrate on Learning the language instead on spending time in setting the required environment settings.

The IDE is completely written in java. So it is 100% portable and cross platform. Tested in windows and GNU/Linux.

**Package Details**

+---actionControllers

| +--sidebarControllers

| | +--ListItemSelected.java

| +--Compiler.java

| +--New.java

| +--Open.java

| +--Save.java

| +--SaveAs.java

|

+---base

| +--CompilerOption.java

| +--Pair.java

| +--UndoRedo.java

|

+---main

| +--MainInterface.java

|

+--myEditor

| +--keyEvents

| | +--EditorKeyEventListener.java

| +--syntax

| +--ColorTracer.java

| +--HighLighter.java

| +--Syntax.java

| +--TextColorer.java

|

+--stringParser

| +--TagRemover.java

|

+--uiComponents

| +--dialogBoxes

| | +--AboutDialogBox.java

| | +--JavaPathDialogBox.java

| +--SideBarItem.java

| +--SideBarList.java

**Functionalities of IDE:**

The IDE has all the basic functionality an IDE must have. Namely

* Syntax Highlighting
* Comment Highlighting
* Compiling the Source Code
* Running the compiled program
* Reading and Writing into Inputs of Running programming.
* Searching the code

**Syntax Highlighting**

IDE supports highlighting of all the java keywords and token. The code gets instanstantially highlighted while typing. Keywords , token , comments will be highlighed with different color so that the programmer can have a clear look and view on the code.

**Comment Highlighting**

IDE also supports highlighting of all the Comments used in code. Both comments and documentation comment hightlinght is fully supported. Multiline comments are identified on the code using java inbuilt regular expression engine. So accurate identification of the comments will be done.

**Compiling the source code**

IDE is capable of invoking the java compiler and compile the file being edited. The compiler errors will be displayed at the bottom of the IDE. If the code being edited in IDE is not saved, on clicking the compile button will ask the use to save the code first before compiling. On successfull compilation , Success notification will be displayed.

**Running the compiled program**

IDE is capable of invoking the java Interpreter and run the previously compiled java code. The inputs needed to inserted to running program can be directly typed to the bottom pane.

**Reading and Writing into Running programming standard i/o.**

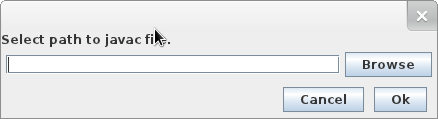
IDE is capable of invoking the java Interpreter and run the previously compiled java code. The running program’s i/o will be processed at the bottom of the IDE.

**Searching the code**

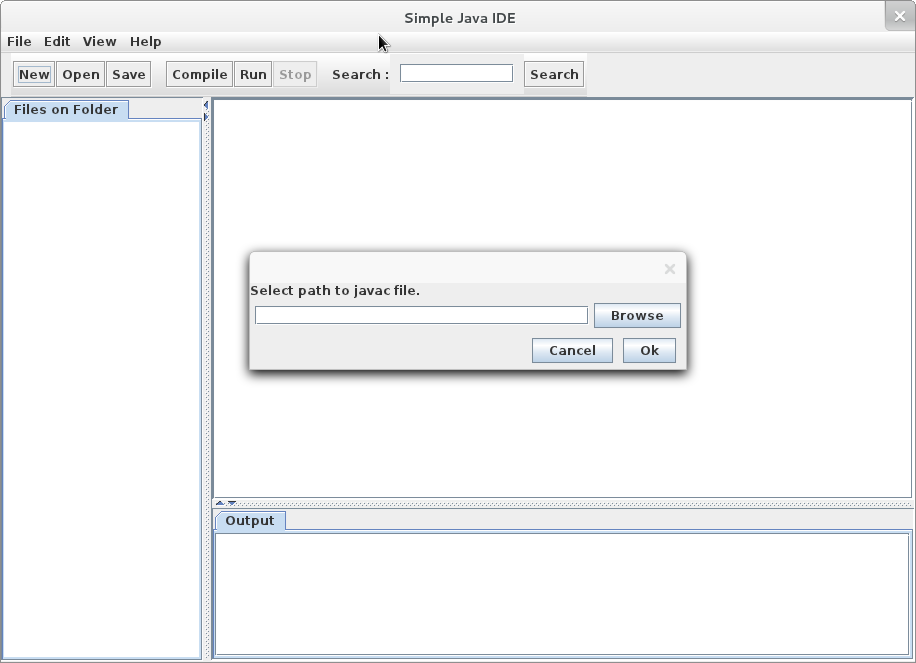
The IDE word be complete without a feature to search the given code for a specific string. Our project have a search option which searches the code and marks the instance of the string with yellow color.

**Screenshots**

**Selecting java Sdk PATH**

****

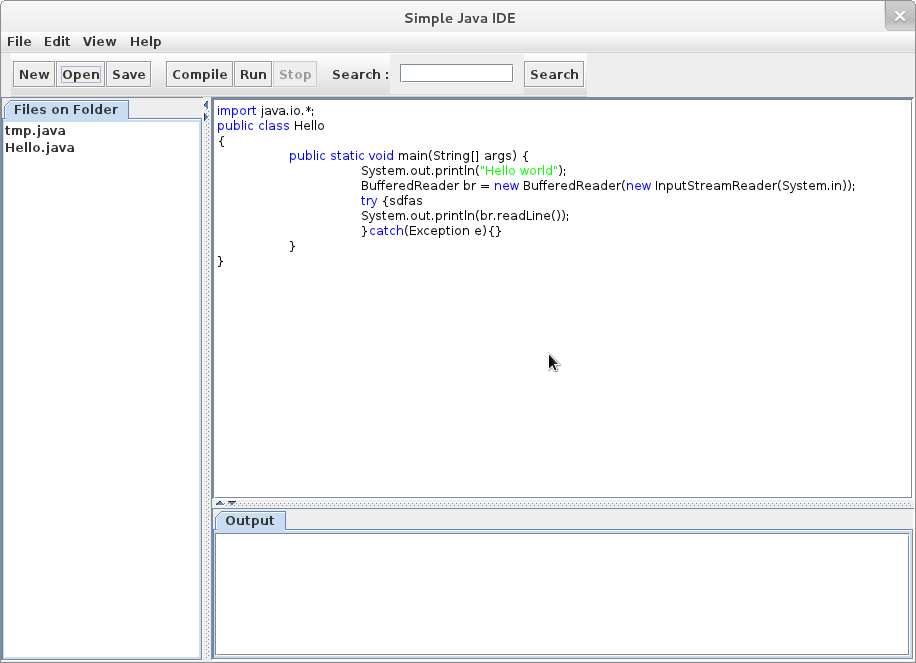
**FULL IDE View**

****

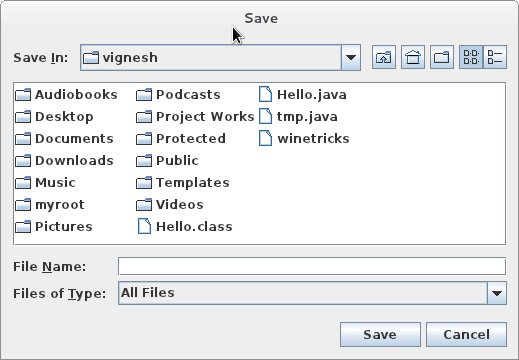
**About Dialog Box**

****

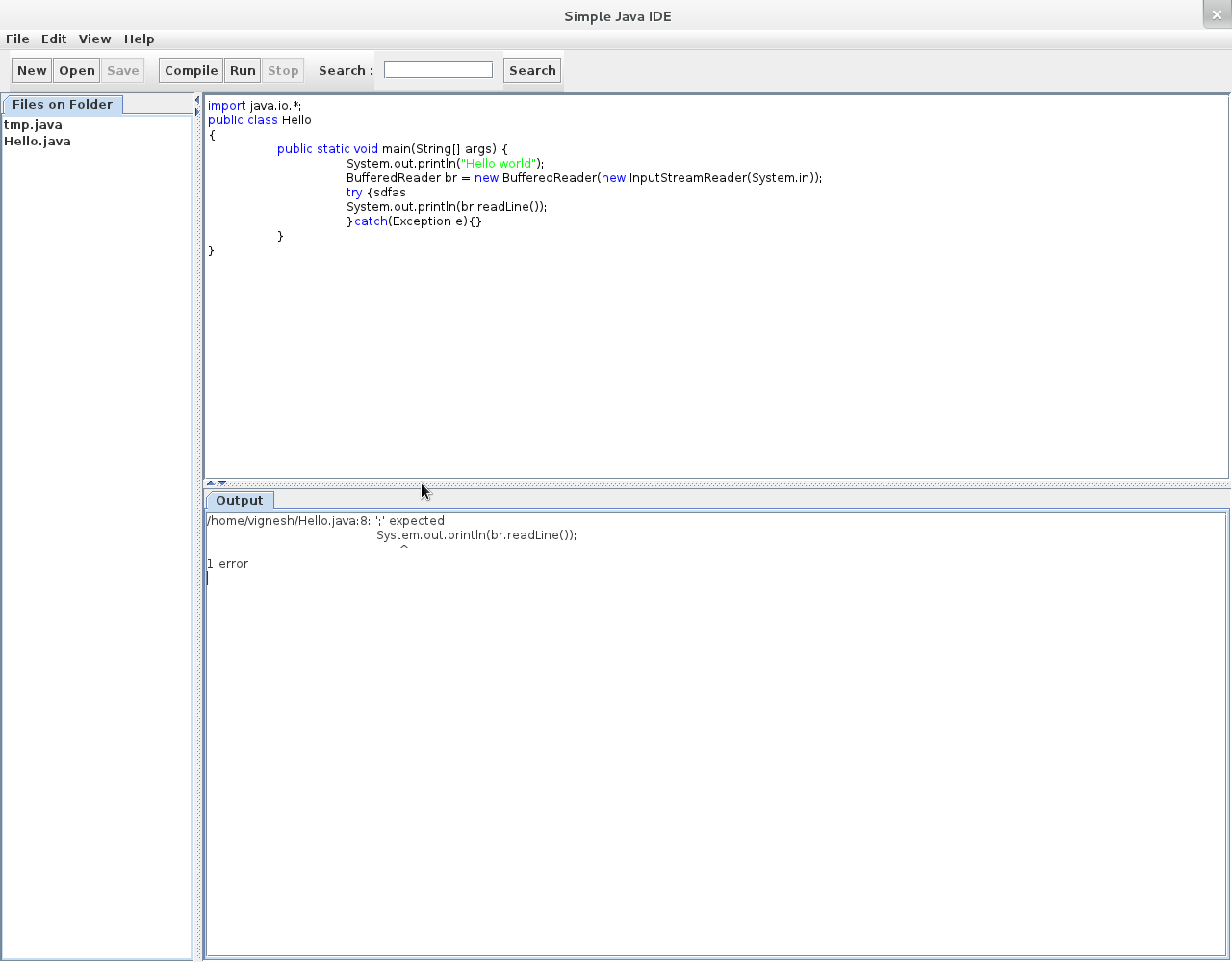
**IDE FuLL View**

****

**Save DialogBox**

****

**IDE Full VIEW**

****

**Source**

**main.MainInterface.java**

package main;

import java.awt.EventQueue;

import javax.swing.JFrame;

import java.awt.BorderLayout;

import javax.swing.AbstractAction;

import javax.swing.JList;

import javax.swing.JMenuBar;

import javax.swing.JMenu;

import javax.swing.JMenuItem;

import javax.swing.JSeparator;

import javax.swing.JSplitPane;

import javax.swing.JTextArea;

import javax.swing.JToolBar;

import javax.swing.JButton;

import javax.swing.KeyStroke;

import javax.swing.SwingConstants;

import javax.swing.JLabel;

import javax.swing.JTextField;

import javax.swing.JTabbedPane;

import javax.swing.JScrollPane;

import javax.swing.JPanel;

import java.awt.Component;

import javax.swing.Box;

import javax.swing.event.CaretEvent;

import javax.swing.event.CaretListener;

import javax.swing.text.Keymap;

import base.CompilerOption;

import actionControllers.sidebarControllers.ListItemSelected;

import uiComponents.SideBarList;

import uiComponents.dialogBoxes.AboutDialogBox;

import uiComponents.dialogBoxes.JavaPathDialogBox;

import java.awt.Color;

import javax.swing.Action;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ComponentAdapter;

import java.awt.event.ComponentEvent;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStreamReader;

public class MainInterface {

private JFrame frmSimpleJavaIde;

private JTextField textField;

private JButton btnSave;

private JButton btnRun;

private JButton btnStop;

public static final int SAVE\_BUTTON = 1;

public static final int RUN\_BUTTON = 2;

public static final int STOP\_BUTTON = 3;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

MainInterface window = new MainInterface();

window.frmSimpleJavaIde.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

public MainInterface() {

initialize();

}

public void disableButton(int buttonType) {

switch(buttonType) {

case MainInterface.SAVE\_BUTTON:

btnSave.setEnabled(false);

break;

case MainInterface.RUN\_BUTTON:

btnRun.setEnabled(false);

btnStop.setEnabled(true);

break;

case MainInterface.STOP\_BUTTON:

btnStop.setEnabled(false);

btnRun.setEnabled(true);

break;

}

}

public void enableButton(int buttonType) {

switch(buttonType) {

case MainInterface.SAVE\_BUTTON:

btnSave.setEnabled(true);

break;

case MainInterface.RUN\_BUTTON:

btnRun.setEnabled(true);

btnStop.setEnabled(false);

break;

case MainInterface.STOP\_BUTTON:

btnStop.setEnabled(true);

btnRun.setEnabled(false);

break;

}

}

/\*\*

\* Initialize the contents of the frame.

\*

\* @wbp.parser.entryPoint

\*/

public void initialize() {

frmSimpleJavaIde = new JFrame();

frmSimpleJavaIde.setTitle("Simple Java IDE");

// frmSimpleJavaIde.setBounds(100, 100, 524, 319);

frmSimpleJavaIde.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

final Toolkit toolkit = Toolkit.getDefaultToolkit();

final Dimension screenSize = toolkit.getScreenSize();

final int x = (screenSize.width - frmSimpleJavaIde.getWidth()) / 2 - 450;

final int y = (screenSize.height - frmSimpleJavaIde.getHeight()) / 2 - 350;

frmSimpleJavaIde.setLocation(x, y);

JMenuBar menuBar = new JMenuBar();

frmSimpleJavaIde.setJMenuBar(menuBar);

frmSimpleJavaIde.getContentPane().setLayout(new BorderLayout(0, 0));

JSplitPane splitPane = new JSplitPane();

splitPane.setContinuousLayout(true);

splitPane.setOneTouchExpandable(true);

frmSimpleJavaIde.getContentPane().add(splitPane, BorderLayout.CENTER);

JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);

SideBarList sbl = new SideBarList();

JList jl = new JList(sbl);

jl.addListSelectionListener(new ListItemSelected());

tabbedPane.addTab("Files on Folder", jl);

tabbedPane.setMinimumSize(new Dimension(200, 0));

tabbedPane.setPreferredSize(new Dimension(200, 0));

splitPane.setLeftComponent(tabbedPane);

JPanel panel = new JPanel();

splitPane.setRightComponent(panel);

panel.setLayout(new BorderLayout(0, 0));

JSplitPane splitPane\_1 = new JSplitPane();

splitPane\_1.setOneTouchExpandable(true);

splitPane\_1.setContinuousLayout(true);

splitPane\_1.setOrientation(JSplitPane.VERTICAL\_SPLIT);

panel.add(splitPane\_1, BorderLayout.CENTER);

JScrollPane scrollPane = new JScrollPane();

splitPane\_1.setLeftComponent(scrollPane);

final myEditor.MyEditor textArea = new myEditor.MyEditor();

scrollPane.setPreferredSize(new Dimension(700, 400));

//scrollPane.setMinimumSize(new Dimension(700, 400));

scrollPane.setViewportView(textArea);

JTabbedPane tabbedPane\_1 = new JTabbedPane(JTabbedPane.TOP);

tabbedPane\_1.setPreferredSize(new Dimension(700, 150));

//tabbedPane\_1.setMinimumSize(new Dimension(700, 150));

final JTextArea compileOutput = new JTextArea();

JScrollPane jsp = new JScrollPane();

jsp.setViewportView(compileOutput);

tabbedPane\_1.addTab("Output",jsp);

compileOutput.addCaretListener(new CaretListener() {

@Override

public void caretUpdate(CaretEvent arg0) {

try {

JTextArea co = (JTextArea)arg0.getSource();

if(co.getText().length() != arg0.getDot())

co.setCaretPosition(co.getText().length());

}

catch(Exception e) {

}

}

});

//final JTextArea runOutput = new JTextArea();

//tabbedPane\_1.addTab("Run Output",runOutput);

// tabbedPane\_1.setMaximumSize(new Dimension(700,150));

splitPane\_1.setRightComponent(tabbedPane\_1);

JPanel panel\_1 = new JPanel();

frmSimpleJavaIde.getContentPane().add(panel\_1, BorderLayout.NORTH);

panel\_1.setLayout(new FlowLayout(FlowLayout.LEFT, 0, 0));

Component rigidArea\_3 = Box.createRigidArea(new Dimension(10, 40));

panel\_1.add(rigidArea\_3);

JToolBar toolBar = new JToolBar();

toolBar.setFloatable(false);

panel\_1.add(toolBar);

JButton btnNew = new JButton("New");

ActionListener nal = new actionControllers.New(textArea);

btnNew.addActionListener(nal);

toolBar.add(btnNew);

JButton btnOpen = new JButton("Open");

ActionListener oal = new actionControllers.Open(frmSimpleJavaIde,sbl,textArea);

btnOpen.addActionListener(oal);

toolBar.add(btnOpen);

btnSave = new JButton("Save");

textArea.initialize(this);

Action sal = new actionControllers.Save(frmSimpleJavaIde,sbl,textArea);

btnSave.addActionListener(sal);

toolBar.add(btnSave);

Component rigidArea\_2 = Box.createRigidArea(new Dimension(10, 40));

toolBar.add(rigidArea\_2);

JToolBar toolBar\_1 = new JToolBar();

toolBar\_1.setFloatable(false);

panel\_1.add(toolBar\_1);

JButton btnCompile = new JButton("Compile");

ActionListener cal = new actionControllers.Compile(frmSimpleJavaIde,textArea,compileOutput);

btnCompile.addActionListener(cal);

toolBar\_1.add(btnCompile);

btnRun = new JButton("Run");

ActionListener execute = new actionControllers.Run(this,textArea,compileOutput);

btnRun.addActionListener(execute);

toolBar\_1.add(btnRun);

btnStop = new JButton("Stop");

btnStop.setEnabled(false);

btnStop.addActionListener(execute);

toolBar\_1.add(btnStop);

Component rigidArea = Box.createRigidArea(new Dimension(10, 40));

toolBar\_1.add(rigidArea);

JToolBar toolBar\_2 = new JToolBar();

toolBar\_2.setFloatable(false);

panel\_1.add(toolBar\_2);

JLabel lblSearch = new JLabel("Search :");

toolBar\_2.add(lblSearch);

JPanel panel\_2 = new JPanel();

FlowLayout flowLayout = (FlowLayout) panel\_2.getLayout();

flowLayout.setVgap(10);

flowLayout.setHgap(10);

toolBar\_2.add(panel\_2);

textField = new JTextField();

textField.setHorizontalAlignment(SwingConstants.LEFT);

panel\_2.add(textField);

textField.setColumns(10);

JButton btnSearch = new JButton("Search");

btnSearch.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

myEditor.syntax.HighLighter.removeAllHighLights(textArea);

myEditor.syntax.HighLighter.highLight(textArea, textField.getText(),Color.YELLOW);

}

});

toolBar\_2.add(btnSearch);

Component rigidArea\_1 = Box.createRigidArea(new Dimension(0, 40));

toolBar\_2.add(rigidArea\_1);

JMenu mnFile = new JMenu("File");

menuBar.add(mnFile);

JMenuItem mntmNew = new JMenuItem("New");

mntmNew.addActionListener(nal);

mnFile.add(mntmNew);

JMenuItem mntmOpen = new JMenuItem("Open");

mntmOpen.addActionListener(oal);

mnFile.add(mntmOpen);

JSeparator separator = new JSeparator();

mnFile.add(separator);

JMenuItem mntmSave = new JMenuItem("Save");

mntmSave.addActionListener(sal);

mnFile.add(mntmSave);

JMenuItem mntmSaveAs = new JMenuItem("Save As");

mntmSaveAs.addActionListener(new actionControllers.SaveAs(frmSimpleJavaIde,sbl,textArea));

mnFile.add(mntmSaveAs);

JSeparator separator\_1 = new JSeparator();

mnFile.add(separator\_1);

JMenuItem mntmQuit = new JMenuItem("Quit");

mntmQuit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

frmSimpleJavaIde.setVisible(false);

System.exit(0);

}

});

mnFile.add(mntmQuit);

JMenu mnEdit = new JMenu("Edit");

menuBar.add(mnEdit);

Action undoAction = new AbstractAction() {

private static final long serialVersionUID = -1676088839051045225L;

public void actionPerformed(ActionEvent arg0) {

EventQueue.invokeLater(new Runnable() {

public void run() {

textArea.undoLastAction();

}

});

}

};

Action redoAction = new AbstractAction() {

private static final long serialVersionUID = 4002647855169900265L;

public void actionPerformed(ActionEvent arg0) {

EventQueue.invokeLater(new Runnable() {

public void run() {

textArea.redoLastAction();

}

});

}

};

JMenuItem mntmUndo = new JMenuItem("Undo");

mntmUndo.addActionListener(undoAction);

mnEdit.add(mntmUndo);

JMenuItem mntmRedo = new JMenuItem("Redo");

mntmRedo.addActionListener(redoAction);

mnEdit.add(mntmRedo);

Keymap km = textArea.getKeymap();

km.addActionForKeyStroke(KeyStroke.getKeyStroke("control Z"), undoAction);

km.addActionForKeyStroke(KeyStroke.getKeyStroke("control Y"), redoAction);

km.addActionForKeyStroke(KeyStroke.getKeyStroke("control S"), sal);

JSeparator separator\_2 = new JSeparator();

mnEdit.add(separator\_2);

JMenuItem mntmNewMenuItem = new JMenuItem("Cut");

mnEdit.add(mntmNewMenuItem);

mntmNewMenuItem.addActionListener(textArea.getActionForKeyStroke(KeyStroke.getKeyStroke("control X")));

JMenuItem mntmCopy = new JMenuItem("Copy");

mnEdit.add(mntmCopy);

mntmCopy.addActionListener(textArea.getActionForKeyStroke(KeyStroke.getKeyStroke("control C")));

JMenuItem mntmPaste = new JMenuItem("Paste");

mnEdit.add(mntmPaste);

mntmPaste.addActionListener(textArea.getActionForKeyStroke(KeyStroke.getKeyStroke("control V")));

JSeparator separator\_3 = new JSeparator();

mnEdit.add(separator\_3);

JMenuItem mntmPreference = new JMenuItem("Set JAVASDK Path");

mntmPreference.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

new JavaPathDialogBox().setVisible(true);

}

});

mnEdit.add(mntmPreference);

JMenu mnView = new JMenu("View");

menuBar.add(mnView);

JMenuItem mntmMedium = new JMenuItem("Medium");

mntmMedium.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

textArea.setFont(new Font(null,Font.PLAIN,14));

}

});

JMenuItem mntmSmall = new JMenuItem("Small");

mntmSmall.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

textArea.setFont(new Font(null,Font.PLAIN,11));

}

});

JMenuItem mntmLarge = new JMenuItem("Large");

mntmLarge.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

textArea.setFont(new Font(null,Font.PLAIN,24));

}

});

mnView.add(mntmMedium);

mnView.add(mntmSmall);

mnView.add(mntmLarge);

JMenu mnHelp = new JMenu("Help");

menuBar.add(mnHelp);

JMenuItem mntmAbout = new JMenuItem("About");

mnHelp.add(mntmAbout);

mntmAbout.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new AboutDialogBox().setVisible(true);

}

});

frmSimpleJavaIde.pack();

File file = new File("pathsetting.plain");

try {

FileInputStream fr = new FileInputStream(file);

BufferedReader br = new BufferedReader(new InputStreamReader(fr));

CompilerOption.setCompilerPath(br.readLine());

} catch (FileNotFoundException e1) {

JavaPathDialogBox jpdb = new JavaPathDialogBox();

jpdb.setVisible(true);

jpdb.addComponentListener(new ComponentAdapter() {

public void componentHidden(ComponentEvent e) {

File file = new File("pathsetting.plain");

try {

FileInputStream fr = new FileInputStream(file);

BufferedReader br = new BufferedReader(new InputStreamReader(fr));

CompilerOption.setCompilerPath(br.readLine());

} catch (FileNotFoundException e2) {

//e2.printStackTrace();

} catch (IOException e1) {

//e1.printStackTrace();

}

}

});

} catch (IOException e) {

e.printStackTrace();

}

}

}

**base.CompilerOptions.java**

package base;

public class CompilerOption {

private static String compiler="javac";

private static String interpretor="java";

public static void setCompilerPath(String path) {

compiler = path;

interpretor =path;

interpretor = interpretor.replace("javac", "java");

}

public static String getCompilerPath() {

return compiler;

}

public static String getInterpretorPath() {

return interpretor;

}

}

**base.Pair.java**

package base;

public class Pair<T> {

private T \_\_first, \_\_second;

public Pair(T first,T second) {

\_\_first = first;

\_\_second = second;

}

public T getFirst() {

return \_\_first;

}

public T getSecond() {

return \_\_second;

}

public void setSecond(T \_\_second) {

this.\_\_second = \_\_second;

}

public void setFirst(T \_\_second) {

this.\_\_first = \_\_second;

}

public String toString() {

return "[" + \_\_first + ", " + \_\_second +"]";

}

}

**base.UndoRedo.java**

package base;

import java.util.ArrayList;

public class UndoRedo {

private ArrayList<String> list = new ArrayList<String>();

private final int max=1000;

private int top=-1;

public UndoRedo() {

this.add("");

}

public synchronized void add(String str) {

if(top+1 == max)

list.remove(0);

if(list.size() > top+1) {

for(int i=list.size()-1;i>top;i--) {

list.remove(i);

}

for(int i=0;i<top;i++) {

list.remove(0);

}

top=0;

}

list.add(str);

top++;

if(top >= max)

top = max-1;

}

public synchronized String previous() {

if(top != 0)

return list.get(--top);

return null;

}

public synchronized String next() {

if(top +1 < list.size())

return list.get(++top);

return null;

}

}

**actionController.Compiler**

package actionControllers;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintStream;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JTextArea;

import base.CompilerOption;

import myEditor.MyEditor;

public final class Compile implements ActionListener {

private JFrame \_\_frame;

private myEditor.MyEditor textArea;

private JTextArea outputArea;

public Compile(JFrame f,MyEditor t, JTextArea o) {

\_\_frame = f;

textArea = t;

outputArea = o;

}

public void actionPerformed(ActionEvent arg0) {

JFileChooser jfc = new JFileChooser();

File file = textArea.getSavedFile();

if (file == null) {

if (jfc.showSaveDialog(\_\_frame) == JFileChooser.APPROVE\_OPTION) {

file = jfc.getSelectedFile();

textArea.contentSavedTo(file);

}

}

if (file != null) {

OutputStream os = null;

try {

os = new FileOutputStream(file);

} catch (FileNotFoundException e1) {

e1.printStackTrace();

}

PrintStream ps = new PrintStream(os);

ps.print(textArea.getText());

try {

os.close();

} catch (IOException e1) {

e1.printStackTrace();

}

textArea.ContentSaved();

outputArea.setText("");

try {

String line;

System.out.println(CompilerOption.getCompilerPath());

Process p = new ProcessBuilder(CompilerOption.getCompilerPath(),"-classpath",file.getParent(),file.getAbsolutePath()).start();

BufferedReader input =

new BufferedReader

(new InputStreamReader(p.getErrorStream()));

boolean flag=false;

while ((line = input.readLine()) != null) {

flag = true;

outputArea.append(line +"\n");

}

if(!flag) {

outputArea.append("Compilation Successful\n");

}

input.close();

} catch (Exception err) {

err.printStackTrace();

}

}

}

}

**actionController.New.java**

package actionControllers;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import myEditor.MyEditor;

public class New implements ActionListener {

private myEditor.MyEditor textArea;

public New(MyEditor t) {

textArea = t;

}

public void actionPerformed(ActionEvent arg0) {

textArea.setText("");

textArea.contentSavedTo(null);

textArea.contentChanged();

}

}

**actionController.Open.java**

package actionControllers;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import uiComponents.SideBarList;

import myEditor.MyEditor;

public class Open implements ActionListener {

private JFrame \_\_frame;

private myEditor.MyEditor textArea;

private SideBarList sbl;

public Open(JFrame f,SideBarList s,MyEditor t) {

\_\_frame = f;

textArea = t;

sbl = s;

}

public void actionPerformed(ActionEvent arg0) {

JFileChooser jfc = new JFileChooser();

if (jfc.showOpenDialog(\_\_frame) == JFileChooser.APPROVE\_OPTION) {

File file = jfc.getSelectedFile();

InputStream is = null;

try {

is = new FileInputStream(file);

} catch (FileNotFoundException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

BufferedReader bf = new BufferedReader(new InputStreamReader(is));

String tmp = "";

try {

String tmp1;

while ((tmp1 = bf.readLine()) != null) {

tmp = tmp + tmp1 + "\n";

}

textArea.setText(tmp);

is.close();

} catch (IOException e1) {

e1.printStackTrace();

}

textArea.contentSavedTo(file);

textArea.colorifyTokens();

sbl.changeListContent(file,textArea);

}

}

}

**actionController.Run,java**

package actionControllers;

import java.awt.event.ActionEvent;

import main.MainInterface;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.OutputStreamWriter;

import javax.swing.JTextArea;

import base.CompilerOption;

import myEditor.MyEditor;

public class Run implements ActionListener,Runnable {

private myEditor.MyEditor textArea;

private JTextArea outputArea;

private MainInterface \_\_mainInterface;

private String \_\_temp="";

private BufferedWriter bw;

private boolean \_\_keyeventRegistered = false;

private Process p=null;

public Run(MainInterface m,MyEditor t, JTextArea o) {

textArea = t;

outputArea = o;

\_\_mainInterface = m;

}

public void actionPerformed(ActionEvent arg0) {

new Thread(this).start();

}

public void run() {

File file = textArea.getSavedFile();

if(p!=null) {

p.destroy();

p=null;

return;

}

if (file != null) {

\_\_mainInterface.disableButton(MainInterface.RUN\_BUTTON);

try {

String line, errorLine = null;

outputArea.setText("");

System.out.println(CompilerOption.getInterpretorPath());

p = new ProcessBuilder(

CompilerOption.getInterpretorPath(), "-classpath",

file.getParent(), file.getName().split("\\.")[0])

.start();

BufferedReader input = new BufferedReader(

new InputStreamReader(p.getInputStream()));

BufferedReader error = new BufferedReader(

new InputStreamReader(p.getErrorStream()));

OutputStream ostream = p.getOutputStream();

bw = new BufferedWriter(new OutputStreamWriter(ostream));

if(!\_\_keyeventRegistered) {

outputArea.addKeyListener(new KeyAdapter() {

public void keyTyped(KeyEvent arg0) {

try {

if(arg0.getKeyChar() == '\b') {

\_\_temp = \_\_temp.substring(0, \_\_temp.length()-2);

} else {

\_\_temp = \_\_temp +new String(new char[] {arg0.getKeyChar()});

}

if(arg0.getKeyChar() == '\n') {

bw.write(\_\_temp);

bw.flush();

\_\_temp ="";

}

} catch (IOException e) {

//e.printStackTrace();

}

}

});

\_\_keyeventRegistered = true;

}

while ((line = input.readLine()) != null

|| (errorLine = error.readLine()) != null) {

if (line != null)

outputArea.append(line + "\n");

if (errorLine != null)

outputArea.append(errorLine + "\n");

}

input.close();

} catch (Exception err) {

//err.printStackTrace();

}

finally {

\_\_mainInterface.enableButton(MainInterface.RUN\_BUTTON);

}

}

}

}

**actionController.Save.java**

package actionControllers;

import java.awt.event.ActionEvent;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.io.PrintStream;

import javax.swing.AbstractAction;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import uiComponents.SideBarList;

import myEditor.MyEditor;

public class Save extends AbstractAction {

/\*\*

\*

\*/

private static final long serialVersionUID = -965327735052681034L;

private JFrame \_\_frame;

private myEditor.MyEditor textArea;

private SideBarList sbl;

public Save(JFrame f,SideBarList s,MyEditor t) {

\_\_frame = f;

textArea = t;

sbl=s;

}

public void actionPerformed(ActionEvent arg0) {

JFileChooser jfc = new JFileChooser();

File file = textArea.getSavedFile();

if (file == null) {

if (jfc.showSaveDialog(\_\_frame) == JFileChooser.APPROVE\_OPTION) {

file = jfc.getSelectedFile();

textArea.contentSavedTo(file);

sbl.changeListContent(file,textArea);

}

}

if (file != null) {

OutputStream os = null;

try {

os = new FileOutputStream(file);

} catch (FileNotFoundException e1) {

e1.printStackTrace();

}

PrintStream ps = new PrintStream(os);

ps.print(textArea.getText());

try {

os.close();

} catch (IOException e1) {

e1.printStackTrace();

}

textArea.ContentSaved();

}

}

}

**actionController.SaveAs.java**

package actionControllers;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.io.PrintStream;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import myEditor.MyEditor;

import uiComponents.SideBarList;

public class SaveAs implements ActionListener{

private JFrame \_\_frame;

private myEditor.MyEditor textArea;

private SideBarList sbl;

public SaveAs(JFrame f,SideBarList s,MyEditor t) {

\_\_frame = f;

textArea = t;

sbl=s;

}

public void actionPerformed(ActionEvent arg0) {

JFileChooser jfc = new JFileChooser();

File file=null;

if (jfc.showSaveDialog(\_\_frame) == JFileChooser.APPROVE\_OPTION) {

file = jfc.getSelectedFile();

textArea.contentSavedTo(file);

}

if (file != null) {

OutputStream os = null;

try {

os = new FileOutputStream(file);

} catch (FileNotFoundException e1) {

e1.printStackTrace();

}

PrintStream ps = new PrintStream(os);

ps.print(textArea.getText());

try {

os.close();

} catch (IOException e1) {

e1.printStackTrace();

}

textArea.ContentSaved();

sbl.changeListContent(file,textArea);

}

}

}

**actionControllers.sidebarControllers. ListItemSelected.java**

package actionControllers.sidebarControllers;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import javax.swing.JList;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import myEditor.MyEditor;

import uiComponents.SideBarItem;

public class ListItemSelected implements ListSelectionListener {

public void valueChanged(ListSelectionEvent arg0) {

JList jl = (JList)(arg0.getSource());

if(jl.getSelectedIndex() <0)

return;

SideBarItem sbi = (SideBarItem)jl.getModel().getElementAt(jl.getSelectedIndex());

MyEditor textArea= sbi.getEditor();

File file = sbi.getFile();

InputStream is = null;

try {

is = new FileInputStream(file);

} catch (FileNotFoundException e1) {

e1.printStackTrace();

}

BufferedReader bf = new BufferedReader(new InputStreamReader(is));

String tmp = "";

try {

String tmp1="";

while ((tmp1 = bf.readLine()) != null) {

tmp = tmp + tmp1 + "\n";

}

textArea.setText(tmp);

is.close();

} catch (IOException e1) {

e1.printStackTrace();

}

textArea.contentSavedTo(file);

textArea.colorifyTokens();

}

}

**myEditor.MyEditor.java**

package myEditor;

import java.awt.\*;

import java.io.File;

import java.util.ArrayList;

import javax.swing.\*;

import javax.swing.text.Style;

import javax.swing.text.StyleConstants;

import base.Pair;

import main.MainInterface;

import myEditor.keyEvents.EditorKeyEventListener;

import myEditor.syntax.HighLighter;

import myEditor.syntax.Syntax;

class ColorifyOnTime implements Runnable {

private MyEditor me;

public ColorifyOnTime(MyEditor m) {

me = m;

}

@Override

public void run() {

while (true) {

try {

Thread.sleep(100);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

// e.printStackTrace();

}

if (!me.istextColored())

me.colorifyTokens();

}

}

}

public class MyEditor extends JTextPane {

private static final long serialVersionUID = -464515284669232442L;

private boolean \_\_contentChanged = false;

private boolean \_\_textColored = true;

private File file = null;

private MainInterface \_\_mainInterface;

private boolean \_\_initialized = false;

private base.UndoRedo \_\_undoRedo;

private myEditor.syntax.TextColorer \_\_textColorer;

public MyEditor() {

}

public MyEditor(MainInterface m) {

initialize(m);

}

public void initialize(MainInterface m) {

\_\_mainInterface = m;

\_\_initialized = true;

\_\_undoRedo = new base.UndoRedo();

\_\_textColorer = new myEditor.syntax.TextColorer(this);

this.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT\_CURSOR));

this.addKeyListener(new EditorKeyEventListener(this));

new Thread(new ColorifyOnTime(this)).start();

Style style = this.addStyle("Keyword:blue", null);

StyleConstants.setForeground(style, Color.BLUE);

Style style1 = this.addStyle("tokens:black", null);

StyleConstants.setForeground(style1, Color.BLACK);

Style style2 = this.addStyle("quotes:green", null);

StyleConstants.setForeground(style2, Color.GREEN);

Style style3 = this.addStyle("quotes:gray", null);

StyleConstants.setForeground(style3, Color.LIGHT\_GRAY);

}

public boolean isContentChanged() {

return \_\_contentChanged;

}

public boolean istextColored() {

return \_\_textColored;

}

public void contentChanged() {

\_\_contentChanged = true;

\_\_textColored = false;

\_\_mainInterface.enableButton(MainInterface.SAVE\_BUTTON);

}

public void backup() {

\_\_undoRedo.add(this.getText());

}

public void ContentSaved() {

\_\_contentChanged = false;

\_\_mainInterface.disableButton(MainInterface.SAVE\_BUTTON);

}

public void contentSavedTo(File f) {

file = f;

}

public File getSavedFile() {

return file;

}

public boolean isInitialised() {

return \_\_initialized;

}

public void undoLastAction() {

String tmp = \_\_undoRedo.previous();

if (tmp != null) {

this.setText(tmp);

}

}

public void redoLastAction() {

String tmp = \_\_undoRedo.next();

if (tmp != null) {

this.setText(tmp);

}

}

public void setColor(ArrayList<Pair<Integer>> pairList) {

\_\_textColorer.setColor(pairList);

}

public void setColor(

ArrayList<ArrayList<Pair<Integer>>> pairListList, String[] StyleKey) {

\_\_textColorer.setColor(pairListList, StyleKey);

}

public void setColor(String[] pattern, String[] StyleKey) {

\_\_textColorer.setColor(pattern, StyleKey);

}

public void removeColor(JTextPane me) {

\_\_textColorer.removeColor(me);

}

public void colorifyTokens() {

String editorContent = this.getText();

this.setColor(new String[] { "\'.\'", "\".\*\"", "//.\*",

"/\\\*(\n|.)\*\\\*/" }, new String[] { "quotes:green",

"quotes:green", "quotes:gray", "quotes:gray" });

this.setColor(Syntax.formatString(editorContent));

/\*

\* ColorTracer.traceOut(editorContent);

\* ArrayList<ArrayList<Pair<Integer>>> al\_al = new

\* ArrayList<ArrayList<Pair<Integer>>>();

\* al\_al.add(Syntax.formatString(editorContent));

\* al\_al.add(ColorTracer.getCommentsLocs());

\* al\_al.add(ColorTracer.getDocuCommentsLocs());

\* al\_al.add(ColorTracer.getDoubleQuotesLocs());

\* al\_al.add(ColorTracer.getSingleQuotesLocs());

\* TextColorer.setColor(jep,al\_al,new

\* String[]{"Keyword:blue","quotes:gray"

\* ,"quotes:gray","quotes:green","quotes:green"});

\*/

\_\_textColored =true;

HighLighter.removeAllHighLights(this);

}

}

**myEditor.keyEvents.EditorKeyEventListener.java**

package myEditor.keyEvents;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

public class EditorKeyEventListener implements KeyListener,Runnable {

private myEditor.MyEditor jep;

private KeyEvent \_\_keyEvent;

public EditorKeyEventListener(myEditor.MyEditor editor) {

jep = editor;

}

public void keyTyped(KeyEvent e) {

\_\_keyEvent = e;

(new Thread(this)).start();

}

public void keyPressed(KeyEvent e) {

}

public void keyReleased(KeyEvent e) {

}

public void run() {

if(\_\_keyEvent.getKeyChar() == ' ' || \_\_keyEvent.getKeyChar() == '\n' || \_\_keyEvent.getKeyChar() == '\b'

|| \_\_keyEvent.getKeyChar() == '.' ||\_\_keyEvent.getKeyChar() == '{' || \_\_keyEvent.getKeyChar() == '}'

|| \_\_keyEvent.getKeyChar() == '(' || \_\_keyEvent.getKeyChar() == ')') {

jep.backup();

}

if(!((int)\_\_keyEvent.getKeyChar() == 19))

jep.contentChanged();

}

}

**myEditor.syntax. ColorTracer.java**

package myEditor.syntax;

import java.util.ArrayList;

import base.Pair;

class InvalidSyntaxException extends Exception {

private static final long serialVersionUID = -468647261150403257L;

}

public class ColorTracer {

private static int hotPointer = -1;

private static int currentState = 0;

private static int makeTransition[][] = {

/\* Transition Table \*/

/\* {other,\n, \, /, \*, ', "} \*/

/\* q0 \*/{ 0, 0, 0, 1, 0, 10, 8 },

/\* q1 \*/{ 0, 0, -1, 2, 3, 0, 8 },

/\* q2 \*/{ 2, 0, 2, 2, 2, 2, 2 },

/\* q3 \*/{ 6, 6, 6, 6, 4, 6, 6 },

/\* q4 \*/{ 5, 5, 5, 0, 7, 5, 5 },

/\* q5 \*/{ 5, 5, 5, 5, 7, 5, 5 },

/\* q6 \*/{ 6, 6, 6, 6, 7, 6, 6 },

/\* q7 \*/{ 7, 6, 6, 0, 7, 6, 6 },

/\* q8 \*/{ 8, -1, 9, 8, 8, 8, 0 },

/\* q9 \*/{ 8, 8, 8, 8, 8, 8, 8 },

/\* q10 \*/{ 10, -1, 11, 10, 10, 0, -1 },

/\* q11 \*/{ 10, -1, 10, -1, -1, 10, 10 } };

private static ArrayList<Pair<Integer>> comments = new ArrayList<Pair<Integer>>();

private static ArrayList<Pair<Integer>> documentation = new ArrayList<Pair<Integer>>();

private static ArrayList<Pair<Integer>> singleQuotes = new ArrayList<Pair<Integer>>();

private static ArrayList<Pair<Integer>> doubleQuotes = new ArrayList<Pair<Integer>>();

public static ArrayList<Pair<Integer>> getCommentsLocs() {

return comments;

}

public static ArrayList<Pair<Integer>> getDocuCommentsLocs() {

return documentation;

}

public static ArrayList<Pair<Integer>> getSingleQuotesLocs() {

return singleQuotes;

}

public static ArrayList<Pair<Integer>> getDoubleQuotesLocs() {

return doubleQuotes;

}

private static void assignPointerValuesInPairs(int index) {

if (hotPointer == -1 || currentState == 0) {

switch (currentState) {

case 2: /\* q2 \*/

hotPointer = 0; // It is a comment

comments.add(new Pair<Integer>(index - 1, 0));

//System.out.printf("\nComment noted at position %d", index - 1);

break;

case 6: /\* q3 \*/

hotPointer = 0; /\* It is a comment \*/// Since Both // and /\* \*/

// will be same color

// both hot pointers are

// zero.

comments.add(new Pair<Integer>(index - 2, 0));

//System.out.printf("\nComment noted at position %d\n", index - 2);

break;

case 5: /\* q5 \*/

hotPointer = 1; // It is a /\*\*\* Documentation \*/

documentation.add(new Pair<Integer>(index - 3, 0));

//System.out.printf("\nDocumentation noted at position %d\n",index - 3);

//System.out.println(documentation.size());

break;

case 8: /\* q8 \*/

hotPointer = 2; // "It is a String"

doubleQuotes.add(new Pair<Integer>(index, 0));

//System.out.printf("\nString noted at position %d\n", index);

break;

case 10: /\* q10 \*/

hotPointer = 3; // 'I't is a 'c'har

singleQuotes.add(new Pair<Integer>(index, 0));

//System.out.printf("\nChar noted at position %d\n", index);

break;

case 0: /\* On initial state q0 \*/

switch (hotPointer) {

case 0: // If I am in end of comment.

(comments.get(comments.size() - 1)).setSecond(index);

break;

case 1: // If I am in end of Documentation.

(documentation.get(documentation.size() - 1))

.setSecond(index);

break;

case 2: // If I am in end of String.

(doubleQuotes.get(doubleQuotes.size() - 1))

.setSecond(index);

break;

case 3: // If I am in end of char.

(singleQuotes.get(singleQuotes.size() - 1))

.setSecond(index);

case -1: /\* Already in q1 only \*/

/\* Do nothing \*/

}

// Resetting the hotPointer.

hotPointer = -1;

}

}

}

private static int symbolToInteger(char c) {

switch (c) {

case '\n':

return 1;

case '\\':

return 2;

case '/':

return 3;

case '\*':

return 4;

case '\'':

return 5;

case '\"':

return 6;

default: /\* on others \*/

return 0;

}

}

public synchronized static void traceOut(String s) {

int index = 0;

comments.clear();

documentation.clear();

singleQuotes.clear();

doubleQuotes.clear();

// Now read the buffered stream.

while (index < s.length()) {

char c = s.charAt(index);

index++;

//if(currentState >= 0)

currentState = makeTransition[currentState][symbolToInteger(c)];

try {

if (currentState == -1)

throw new InvalidSyntaxException();

}

catch (InvalidSyntaxException e) {

System.out.println("InvalidSyntax at index" + index);

return;

}

assignPointerValuesInPairs(index);

}

}

}

**myEditor.syntax. HighLighter.java**

package myEditor.syntax;

import java.awt.Color;

import java.util.ArrayList;

import java.util.regex.\*;

import javax.swing.text.BadLocationException;

import javax.swing.text.DefaultHighlighter;

import javax.swing.text.Highlighter;

import javax.swing.text.JTextComponent;

import base.\*;

public class HighLighter {

private static class PrivatePainter extends DefaultHighlighter.DefaultHighlightPainter {

PrivatePainter(Color c) {

super(c);

}

}

synchronized public static void highLight(JTextComponent jtc,String pattern,Color color) {

//removeAllHighLights(jtc);

try {

Pattern patt = Pattern.compile(pattern);

Matcher m = patt.matcher(jtc.getText());

Highlighter hilite = jtc.getHighlighter();

while(m.find()) {

hilite.addHighlight(m.start(), m.end(), new PrivatePainter(color));

}

} catch (BadLocationException e) {

e.printStackTrace();

}

}

synchronized public static void highLight(JTextComponent jtc,ArrayList<Pair<Integer>> pairList) {

removeAllHighLights(jtc);

try {

Highlighter hilite = jtc.getHighlighter();

for(Pair<Integer> pair : pairList) {

hilite.addHighlight(pair.getFirst(), pair.getFirst()+pair.getSecond(), new PrivatePainter(Color.YELLOW));

}

} catch (BadLocationException e) {

e.printStackTrace();

}

}

public static void removeAllHighLights(JTextComponent jtc) {

Highlighter hilite = jtc.getHighlighter();

Highlighter.Highlight[] hilites = hilite.getHighlights();

for (Highlighter.Highlight hl: hilites) {

if (hl.getPainter() instanceof PrivatePainter) {

hilite.removeHighlight(hl);

}

}

}

}

**myEditor.syntax.Syntax.java**

package myEditor.syntax;

import java.util.\*;

import base.\*;

public class Syntax {

private static ArrayList<String> \_\_JkeywordList = null;

private static boolean mapInitialized = false;

public static void initialized() {

if(mapInitialized)

return;

\_\_JkeywordList = new ArrayList<String>();

String[] key = new String[] {"abstract","assert","boolean","break","byte",

"case","catch","char","class","const","continue","default","do","double",

"else","enum","extends","final","finally","float","for","goto","if","implements",

"import","instanceof","int","interface","long","native","new","package",

"private","protected","public","return","short","static","strictfp","super",

"switch","synchronized","this","throw","throws","transient","try","void","volatile",

"while","false","null","true"};

for(String k: key)

\_\_JkeywordList.add(k);

\_\_JkeywordList.trimToSize();

mapInitialized = true;

}

public static boolean isInitialized() {

return mapInitialized;

}

private static Boolean checkAWordWithoutInitCheck(String str) {

return \_\_JkeywordList.contains(str);

}

public static ArrayList<Pair<Integer>> formatString(String str) {

if(!mapInitialized) {

initialized();

}

ArrayList<Pair<Integer>> result = new ArrayList<Pair<Integer>>();

int count=0;

String tmp[] = str.split("[ ;\\)\\(:\\{\\}\n\t]");

for(String tmp1 : tmp) {

if (checkAWordWithoutInitCheck(tmp1)) {

int temp = count;

count+=tmp1.length();

result.add(new Pair<Integer>(new Integer(temp),new Integer(tmp1.length())));

}else {

count+=tmp1.length();

}

count++;

}

return result;

}

}

**myEditor.syntax.TextColorer.java**

package myEditor.syntax;

import java.util.ArrayList;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.swing.JTextPane;

import javax.swing.text.StyledDocument;

import base.Pair;

public class TextColorer {

private JTextPane me;

public TextColorer(JTextPane m) {

me = m;

}

public synchronized void setColor(ArrayList<Pair<Integer>> pairList) {

StyledDocument doc = me.getStyledDocument();

//doc.setCharacterAttributes(0, doc.getLength(), me.getStyle("tokens:black"), false);

for(Pair<Integer> pair : pairList)

doc.setCharacterAttributes(pair.getFirst(),pair.getSecond(), me.getStyle("Keyword:blue"), false);

}

public synchronized void setColor(ArrayList<ArrayList<Pair<Integer>>> pairListList,String[] StyleKey) {

StyledDocument doc = me.getStyledDocument();

//doc.setCharacterAttributes(0, doc.getLength(), me.getStyle("tokens:black"), false);

int i=0;

removeColor(me);

for(ArrayList<Pair<Integer>> pairList : pairListList) {

for(Pair<Integer> pair : pairList)

doc.setCharacterAttributes(pair.getFirst(),pair.getSecond(), me.getStyle(StyleKey[i]), false);

i++;

}

}

public synchronized void setColor(String[] pattern,String[] StyleKey) {

StyledDocument doc = me.getStyledDocument();

ArrayList<ArrayList<Pair<Integer>>> arraylist = new ArrayList<ArrayList<Pair<Integer>>>();

for(int i=0;i<pattern.length;i++) {

Pattern patt = Pattern.compile(pattern[i]);

Matcher m = null;

try {

m = patt.matcher(me.getText());

}

catch(java.lang.NullPointerException e) {

e.printStackTrace();

return;

}

ArrayList<Pair<Integer>> al = new ArrayList<Pair<Integer>>();

while(m.find()) {

al.add(new Pair<Integer>(m.start(),m.end()-m.start()));

}

arraylist.add(al);

}

removeColor(me);

for(int i=0;i<pattern.length;i++)

for(Pair<Integer> pair:arraylist.get(i))

doc.setCharacterAttributes(pair.getFirst(),pair.getSecond(), me.getStyle(StyleKey[i]), false);

}

public synchronized void removeColor(JTextPane me) {

StyledDocument doc = me.getStyledDocument();

doc.setCharacterAttributes(0, doc.getLength(), me.getStyle("tokens:black"), false);

}

}

**uiComponents. SideBarItem.java**

package uiComponents;

import java.io.File;

import myEditor.MyEditor;

public class SideBarItem {

private static final long serialVersionUID = 2605390333369340753L;

private File file;

private MyEditor textArea;

public SideBarItem(File f,MyEditor ta) {

file=f;

textArea = ta;

}

public File getFile() {

return file;

}

public MyEditor getEditor() {

return textArea;

}

public String toString() {

return file.getName();

}

}

**uiComponents. SideBarList.java**

package uiComponents;

import java.io.File;

import java.io.FileFilter;

import javax.swing.DefaultListModel;

import myEditor.MyEditor;

public class SideBarList extends DefaultListModel {

private static final long serialVersionUID = -1745005884948756451L;

private File[] files;

MyEditor textArea;

public SideBarList() {

}

public SideBarList(File file,MyEditor ta) {

textArea =ta;

initialize(file);

}

public void initialize(File file) {

File dir = file.getParentFile();

FileFilter fileFilter = new FileFilter() {

public boolean accept(File file) {

return file.isFile();

}

};

this.clear();

files = dir.listFiles(fileFilter);

for(File f:files) {

String name = f.getName();

if(name.matches(".\*\\.java"))

this.addElement((new SideBarItem(f,textArea)));

}

}

public void changeListContent(File file,MyEditor ta) {

textArea = ta;

initialize(file);

}

}

**uiComponents.dialogBoxes.AboutDialogBox.java**

package uiComponents.dialogBoxes;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JEditorPane;

import javax.swing.JPanel;

public class AboutDialogBox extends JDialog {

private static final long serialVersionUID = -4979386411951081018L;

public AboutDialogBox() {

initialize();

}

private void initialize() {

this.setPreferredSize(new Dimension(500,200));

this.setResizable(false);

this.setTitle("About");

this.setAlwaysOnTop(true);

final Toolkit toolkit = Toolkit.getDefaultToolkit();

final Dimension screenSize = toolkit.getScreenSize();

final int x = (screenSize.width - this.getWidth()) / 2 - 200;

final int y = (screenSize.height - this.getHeight()) / 2 - 100;

this.setLocation(x, y);

this.setDefaultCloseOperation(HIDE\_ON\_CLOSE);

this.getContentPane().setLayout(new BorderLayout());

JEditorPane jta = new JEditorPane("text/html","<br><h2>&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;A Simple JAVA IDE</h2>" +

"&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;Created by students of CSE Third Year 2011<br>" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +

"&nbsp;&nbsp;&nbsp;&nbsp;" +

"Valliammai Engineering College");

jta.setEditable(false);

this.getContentPane().add(jta,BorderLayout.CENTER);

JButton jbtOK = new JButton("Ok");

JPanel panel = new JPanel();

panel.setLayout(new FlowLayout(FlowLayout.RIGHT,10,5));

this.getContentPane().add(panel,BorderLayout.SOUTH);

panel.add(jbtOK);

jbtOK.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

this.pack();

}

}