

Department Of Information Technology

           National Institute Of Technology

       Karnataka, Surathkal

18th November 2016

**REPORT**

ON

**TEXT EDITOR (Notepad) USING JAVA**

**SUBMITTED TO:**

Mr.Pradeep Jagannath

**SUBMITTED BY:**

1. Harichandana B. S. S. (15IT118)

        2. Raut Kajal Dattatray (15IT153)

      3. Padala Preethi (15IT225)

       4. Navya W Prakash (15IT222)

**CONTENTS:**

1. **INTRODUCTION ……………………3**
2. **CODE ……………………3**
3. **EXPLANATION ……………………15**
4. **METHODS USED ……………………17**
5. **UML DIAGRAMS ……………………19**

* **CLASS DIAGRAM**
* **SEQUENCE DIAGRAM**
* **ACTIVITY DIAGRAM**
* **ARCHITECTURAL DIAGRAM**

1. **OUTPUT ……………………21**

**INTRODUCTION:**

A useful text editor is an essential component of any personal computer. Every day we need to make notes, compose documents, and record vital pieces of information. We depend on our text editor. Microsoft provided *Notepad,*a superbly useful software tool, for precisely these purposes.

This project aims to create a simple model of a notepad . In this project a text editor which has two menus namely file and edit has been created using the java swing classes and awt classes.

The project has also implemented File handling techniques inorder to save and open the required file as per the user’s choice. This project has a wide range of applications which are :

\*Creating new files .

\* Viewing the contents of a particular file.

\* Editing a existing file and saving the modified content.

These are only some of many applications of the text editor.

**CODE:**

package notepad;

import java.awt.Frame;

import java.awt.event.ActionEvent;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

public class Notepad extends Frame {

private java.awt.Menu mEdit;

private java.awt.MenuItem mExit;

private java.awt.Menu mFile;

private java.awt.MenuItem mNew;

private java.awt.MenuItem mOPen;

private java.awt.MenuItem mSave;

private java.awt.MenuBar mbar;

private java.awt.MenuItem mCut;

private java.awt.MenuItem mCopy;

private java.awt.MenuItem mPaste;

private javax.swing.JTextArea txtContent;

private String filename;

private File file;

private boolean filenameProvided,modified;

public Notepad() {

initComponents();

filenameProvided=modified=false;

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

txtContent = new javax.swing.JTextArea();

mbar = new java.awt.MenuBar();

mFile = new java.awt.Menu();

mNew = new java.awt.MenuItem();

mOPen = new java.awt.MenuItem();

mSave = new java.awt.MenuItem();

mExit = new java.awt.MenuItem();

mEdit = new java.awt.Menu();

mCut=new java.awt.MenuItem();

mCopy=new java.awt.MenuItem();

mPaste=new java.awt.MenuItem();

addWindowListener(new java.awt.event.WindowAdapter() {

public void windowClosing(java.awt.event.WindowEvent evt) {

exitForm(evt);

}

});

txtContent.setName("txtcontent"); // NOI18N

add(txtContent, java.awt.BorderLayout.CENTER);

mbar.setName("mbar");

mFile.setLabel("File");

mFile.setName("mFile");

mFile.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

action1Performed(evt);

}

});

mNew.setLabel("New");

mNew.setName("New");

mFile.add(mNew);

mOPen.setLabel("Open");

mOPen.setName("mOpen");

mFile.add(mOPen);

mSave.setLabel("Save");

mSave.setName("Save");

mSave.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

action1Performed(evt); }

});

mFile.add(mSave);

mExit.setLabel("Exit");

mExit.setName("Exit");

mExit.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

action1Performed(evt);

}

});

mFile.add(mExit);

mbar.add(mFile);

mEdit.setLabel("Edit");

mEdit.setName("Edit");

mCut.setLabel("Cut");

mEdit.add(mCut);

mCopy.setLabel("Copy");

mEdit.add(mCopy);

mPaste.setLabel("Paste");

mEdit.add(mPaste);

mCut.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

action1Performed(evt);

}

});

mCopy.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

action1Performed(evt);

}

});

mPaste.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

action1Performed(evt);

}

});

mbar.add(mEdit);

setMenuBar(mbar);

mbar.getAccessibleContext().setAccessibleName("mbar");

pack();

}// </editor-fold>

/\*\*

\* Exit the Application

\*/

private void exitForm(java.awt.event.WindowEvent evt) {

System.exit(0);

}

private void mFileActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

}

private void mSaveActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

save();

}

private void mExitActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

if(modified)

{

if(saveConfirm())

{

save();

}

}

System.exit(0);

}

public boolean saveConfirm(){

int option;

option=JOptionPane.showConfirmDialog(this,"Contents not saved,save?","save",JOptionPane.YES\_NO\_OPTION);

if(option==JOptionPane.YES\_OPTION)

return true;

else

return false;

}

public boolean getFileName(int so){

JFileChooser jfc=new JFileChooser();

int option;

if(so==1)

option=jfc.showSaveDialog(jfc);

else

option=jfc.showOpenDialog(jfc);

if(option==JFileChooser.APPROVE\_OPTION)

{

file=jfc.getSelectedFile();

filename=file.getPath();

filenameProvided=true;

return true;

}

else

return false;

}

public void saveContents(){

String str=txtContent.getText();

try{

FileOutputStream fout=new FileOutputStream(file);

fout.write(str.getBytes());

fout.close();

}catch(Exception e)

{

System.out.println("Error"+e);

}

}

public void save(){

if(!filenameProvided)

{

if(!getFileName(1))

return ;

}

saveContents();

}

public void action1Performed(ActionEvent ae){

String cmd=ae.getActionCommand();

if(cmd.equals("New"))

{

if(modified)

{

if(saveConfirm())

{

save();

}

}

txtContent.setText(" ");

filenameProvided=false;

modified=true;

}

else if(cmd.equals("Open"))

{

if(modified)

{

if(saveConfirm())

{

save();

}

}

if(getFileName(2))

{

try

{

FileInputStream fin;

fin=new FileInputStream(file);

byte[] b=new byte[(int)file.length()];

fin.read(b);

String str=new String(b);

txtContent.setText(str);

fin.close();

}catch(Exception e)

{

System.out.println("Error"+e);

}

modified=false;

filenameProvided=true;

}

}

else if(cmd.equals("Save"))

{

save();

}

else if(cmd.equals("Cut"))

{

txtContent.cut();

}

else if(cmd.equals("Copy"))

{

txtContent.copy();

}

else if(cmd.equals("Paste"))

{

txtContent.paste();

}

else if(cmd.equals("Exit"))

{

if(modified)

{

if(saveConfirm())

{

save();

}

}

System.exit(0);

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new Notepad().setVisible(true);

}

});

}

}//end of code

**EXPLANATION:**

This programme helps us create our own notepad by making use of java swing package.

The frame for the notepad window is created using the Frame class in java.awt package .

To create this, a Notepad class is created which inherits the Frame class. the internal compnents of the frame includes a menubar & a text area both of which are compnents of the javax.swing package. The menu bar contains two menus file and edit. Both the menus show multiple options when the curser is placed on them.

The multiple options shown in the form of a dropdown list contains menuitems . These menuitems are a component of the menu. The file menu contains four menuitems :

1)New

2)Open

3)Save

4)Exit

Each of these MenuItems are registered to their own actionListeners which direct the programme toward the action1Performed method which inturn performs the required task as per the command.

The edit menu contains three MenuItems:

1)Cut

2)Copy

3)Paste

Like the other menuitems these too are registered to their own actionListener which again inturn directs the programme to perform the required task.

The initComponents method contains all the initialization and declaration of all the components in the Frame of the notepad. This includes the design of the frame .

The saveConfirm method opens a dialogue box which is a JOptionPane also part of swings , this asks the user to confirm if he wants to save the file or not.

The getFileName opens a JFileChooser dialogue box which takes the filename as input . this allows the file to be saved by that file name.

The saveContents method takes the text written in the textarea portion and writes it to a file using the file output stream.

The save method saves the file if the filename is provided and the contents have been saved.

The action1Performed method is the most significant method among all the other. This method allows the programme to perform a particular task when a specific command is choosen.

The New button upon being selected checks if the file has been modified and asks the user to save the modified contents if not saved. Then , this clears the text area.

The open button opens the JFileChooser to allow the user to select the desired file .this then reads the contents of the selected file and writes it to the text area.

The Save button saves the file.

The Cut, Copy and Paste commands cut ,copy or Paste the selected text to the file.

The Exit button terminates the programme if the file has been saved , otherwise asks the user to save the modified file.

**METHODS USED:**

**1]javax.swing.JFileChooser**

public class JFileChooser

extends JComponent

implements Accessible

JFileChooser provides a simple mechanism for the user to choose a file.

File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list, or entering the name of a file or directory. To display a file chooser, you usually use the JFileChooser API to show a modal dialog containing the file chooser. Another way to present a file chooser is to add an instance of JFileChooser to a container.

1.**static int** APPROVE\_OPTION

Return value if approve (yes, ok) is chosen.

2.**int showOpenDialog(Component parent)**

Pops up an "Open File" file chooser dialog.

3.**int showSaveDialog(Component parent)**

Pops up a "Save File" file chooser dialog.

4.**File getSelectedFile()**

Returns the selected file.

**2]javax.swing.JOptionPane**

public class JOptionPane

extends JComponent

implements Accessible

JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.

1.showConfirmDialog :Asks a confirming question, like yes/no/cancel.

2.YES\_NO\_OPTION:optionType that defines the set of option buttons that appear at the bottom of the dialog box.

**3]java.awt.Menu and java.awt.MenuItem**

public class Menu

extends MenuItem

implements MenuContainer, Accessible

A Menu object is a pull-down menu component that is deployed from a menu bar.

Each item in a menu must belong to the MenuItem class. It can be an instance of MenuItem, a submenu (an instance of Menu), or a check box (an instance of CheckboxMenuItem).

1.**public void setLabel(String label)**

Sets the label for this menu item to the specified label.

2.**public void setName(String name)**

Sets the name of the component to the specified string.

3.**MenuItem add(MenuItem mi)**

Adds the specified menu item to this menu.

4.**public void addActionListener(ActionListener l)**

Adds the specified action listener to receive action events from this menu item. If l is null, no exception is thrown and no action is performed.

**4]. Class javax.swing.JTextArea**

**1.getText()**

Returns the text contained in this TextComponent

**2.setText()**

Sets the text of this TextComponent to the specified text.

**3.cut()**

Transfers the currently selected range in the associated text model to the system clipboard, removing the contents from the model.

**4.copy()**

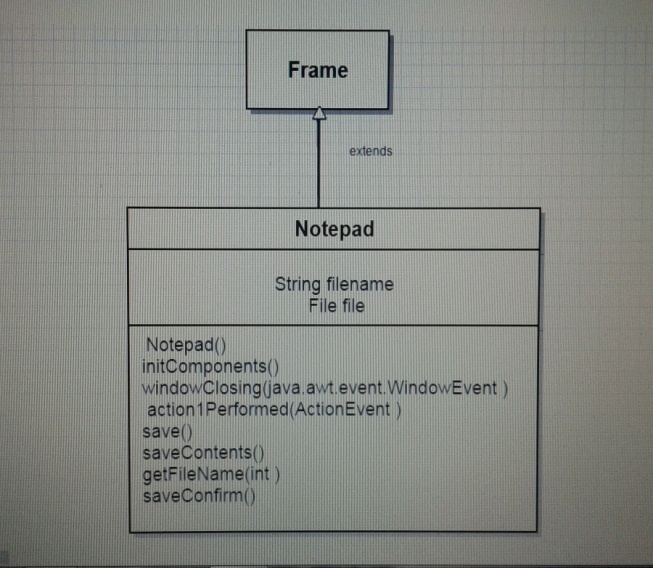
Transfers the currently selected range in the associated text model to the system clipboard, leaving the contents in the text model.

**5.paste()**

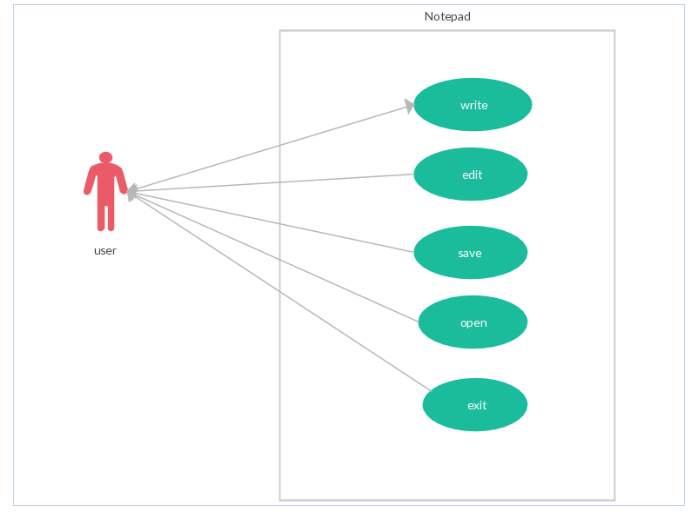
Transfers the contents of the system clipboard into the associated text model

**UML DIAGRAMS:**

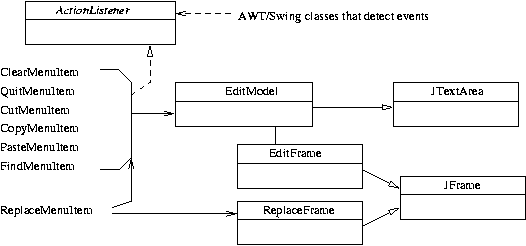
**1.CLASS DIAGRAM:**



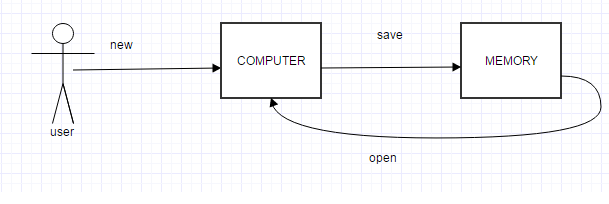
**2.SEQUENCE DIAGRAM**

****

**3.ACTIVITY DIAGRAM**

****

**4.ARCHITECTURAL DIAGRAM:**



**OUTPUTS:**

