package searchapp;

import java.awt.\*;

import java.awt.event.\*;

import java.io.BufferedInputStream;

import java.io.DataInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.FileNotFoundException;

import java.util.Arrays;

import java.util.Collection;

import java.util.EventObject;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Scanner;

import java.util.Set;

import java.util.List;

import java.util.ArrayList;

import javax.swing.\*;

import javax.swing.event.TableModelEvent;

import javax.swing.event.TableModelListener;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableColumnModel;

import javax.swing.table.TableModel;

public class IndexSetupWindow extends BaseWindow implements TableModelListener{

private static final ActionListener ActionListener = null;

JFrame frame = new JFrame("Maintain index");

public IndexSetupWindow(){

initWindow();

JFrame frame;

JButton add, remove;

JPanel dynamicButtonPane, addRemovePane;

JFrame.setDefaultLookAndFeelDecorated(true);

frame = new JFrame ("maintenance index");

}

///TODO

// Get the indexes visible in the JTable

// save the last modification date to detect if outdated

// get it so it detects deleted files

// save the index to a file so it can be accessed anytime

// utilize an update method of some kind to check if the indexFileMap is valid

///

//this is what will be affected when inserting and removing data from the table

//it will be affected everytime there is a change to the indexFileStatus

//this will change to be expandable

Object [][] fileData = new Object[10][2];

protected boolean deleteNow;

//this will keep in store all the data per text file

public static Map<File , Set<String> > indexFileMap = new HashMap<File, Set<String>>();

public static Map<File, FileIndexStatus> indexFileStatus = new HashMap<File,FileIndexStatus>();

public static Map<File, Long> indexFileModified = new HashMap<File,Long>();

public void initWindow() {

// TODO Auto-generated method stub

//initialize frame size

// make frame..

frame.setSize(700,700);

frame.setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE);

frame.setBackground(Color.RED);

frame.setBounds(700,560,500,700);

frame.setLayout(new BorderLayout());

//frame.setSize(200, 200);

frame.setVisible(false);

//create the panels

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new FlowLayout());

JPanel topLayout = new JPanel();

topLayout.setLayout(new BorderLayout());

//create the buttons to work with the maintenance window

//add button

JButton frm = new JButton("Add your File");

// frm.setBounds(334, 126, 90, 25);

buttonPanel.add(frm);

//Create a table to view files available in index

//next up is to create a file index table model

//note, make sure that it loads in data from an index file for the app

final String [] columnNames = {"File","Index Status"};

final DefaultTableModel fileModel = new DefaultTableModel(fileData,columnNames);

// final JTable fileIndex = new JTable(fileData, columnNames);

final JTable fileIndex = new JTable(fileModel);

fileIndex.getModel().addTableModelListener(this);

frame.add(fileIndex,BorderLayout.CENTER);

frm.addActionListener(new ActionListener () {

@Override

public void actionPerformed ( ActionEvent ae){

SwingUtilities.invokeLater (new Runnable (){

public void run (){

// TODO: Replace with JFile Chooser;

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter filter = new FileNameExtensionFilter(

"Text Files", "txt");

chooser.setFileFilter(filter);

int returnVal = chooser.showOpenDialog(frame);

if(returnVal == JFileChooser.APPROVE\_OPTION) {

//this is where the file operation occurs to

//a) add the file to the list of indexes, set to indexed

//b)it is rendered on the table as such

System.out.println("You chose to open this file: " +

chooser.getSelectedFile().getName());

System.out.println("File to save: " + chooser.getSelectedFile());

try {

//read the contents of the file

FileInputStream fis = new FileInputStream(chooser.getSelectedFile());

// Here BufferedInputStream is added for fast reading.

BufferedInputStream bis = new BufferedInputStream(fis);

DataInputStream dis = new DataInputStream(bis);

//now create a temporary set to put all the extracted data into

Set<String> tempSet = new HashSet<String>();

String tempString = "";

// dis.available() returns 0 if the file does not have more lines.

while (dis.available() != 0) {

// this statement reads the line from the file and print it to

// the console. we know it is presently deprecated and will be actively looking

//for a more efficient solution, in the meantime this works for plain text files

tempString += dis.readLine();

}

// dispose all the resources after using them.

fis.close();

bis.close();

dis.close();

//use this to separate all the words, this will separate them by phrase, and from there the searching will be done word by word

String \_delimiters = "[.?!:\\-=+,&()<>@#$%^\\\*~`]+";

String[] tokens = tempString.split(\_delimiters);

//now each word goes into the set

for(String str : tokens){

tempSet.add(str);

}

// When the temporary set is made, add the new map entry to the map index

indexFileMap.put(chooser.getSelectedFile(), tempSet);

indexFileStatus.put(chooser.getSelectedFile(), FileIndexStatus.INDEXED);

indexFileModified.put(chooser.getSelectedFile(), chooser.getSelectedFile().lastModified());

//add to filedata to be used in jtable

//this will add the filename as a string and the file index status

//we now fill the fileData with the current info to update the JTable

for(File f : indexFileStatus.keySet()){

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

fileData[i][0] = f.getName();

fileData[i][1] = indexFileStatus.get(f);

}

//update the JTable, first remove the rows

for(int i = 0;i<=fileModel.getRowCount();i++){

fileModel.removeRow(i);

}

//now add them in for the updated fileData

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

for(int j=0; j< fileData.length;j++)

fileModel.insertRow(i, fileData[j]);

}

}

//chooser.getSelectedFile().lastModified(); --we need a way to store this so that it can be utilized in checking for outdated files

//chooser.getSelectedFile().exists(); --use this when checking if the file exists

//reset the vars for future use

} catch (FileNotFoundException e) {

System.out.println("no file selected");

}catch(IOException e){

System.out.println("no file selected");

}

}

}

});

}

});

//rebuild button

Button from = new Button("Rebuild of-out Date");

// frm.setBounds(334, 126, 90, 25);

buttonPanel.add(from);

class RebuildButtonHandler implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

//add rebuild button code

}

}

//remove button

//TableModel data = null;

//TableColumnModel column = null;

JTable table=new JTable(fileData,columnNames);

JButton button1=new JButton("Remove Selected Files");

buttonPanel.add(button1);

JFileChooser jf = new JFileChooser();

button1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

DefaultTableModel model=new DefaultTableModel(fileData,columnNames);

JTable table=new JTable(model);

JOptionPane.showMessageDialog(null, "Are you sure? you want to delete this file"

, "Remove Option : ", JOptionPane.INFORMATION\_MESSAGE);

deleteNow = true;

}

});

JButton resetB = new JButton("Reset");

//ResetButtonHandler rbHandler = new ResetButtonHandler();

//resetB.addActionListener((ActionListener) );

buttonPanel.add(resetB);

class ResetButtonHandler implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

//add reset button code

}

}

JLabel labelOptions = new JLabel("SEARCH ENGINE-INDEX MAINTENANCE");

labelOptions.setFont(new Font("Arial",2 , 24));

labelOptions.setForeground(Color.red);

labelOptions.setBounds(220, 60, 250, 70);

labelOptions.setText("<html>SEARCH <font color='blue'><html>ENGINE<font color='green'><html>-INDEX MAINTENANCE</font></html>");

labelOptions.setLocation(50, 30);

labelOptions.setSize(86, 14);

topLayout.add(labelOptions, BorderLayout.CENTER);

topLayout.add(buttonPanel, BorderLayout.PAGE\_END);

frame.add(topLayout,BorderLayout.PAGE\_START);

//we will use this as the primary means to update the window

frame.addFocusListener(new FocusListener(){

@Override

public void focusGained(FocusEvent arg0) {

// TODO Auto-generated method stub

//use this to check if files are outdated

if(indexFileStatus.keySet() != null)

for(File f : indexFileMap.keySet()){

//check if the file is deleted

if(!f.exists()){

indexFileMap.remove(f);

indexFileStatus.remove(f);

indexFileModified.remove(f);

}

//check if last modified of file is not equal to current index, if so set it to outdated

else if(f.lastModified() != indexFileModified.get(f)){

indexFileStatus.put(f, FileIndexStatus.OUTDATED);

}

}

//we now fill the fileData with the current info to update the JTable

if(indexFileStatus.keySet() != null)

for(File f : indexFileStatus.keySet()){

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

fileData[i][0] = f.getName();

fileData[i][1] = indexFileStatus.get(f);

}

//update the JTable, first remove the rows

for(int i = 0;i<=fileModel.getRowCount();i++){

fileModel.removeRow(i);

}

//now add them in for the updated fileData

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

for(int j=0; j< fileData.length;j++)

fileModel.insertRow(i, fileData[j]);

}

}

}

@Override

public void focusLost(FocusEvent arg0) {

// TODO Auto-generated method stub

}

});

}

private void add(Button fro) {

// TODO Auto-generated method stub

}

private void registerDelAction() {

// TODO Auto-generated method stub

}

public void setVisibility(boolean visibility){

frame.setVisible(visibility);

}

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

}

@Override

public void tableChanged(TableModelEvent e) {

// TODO Auto-generated method stub

}

}