**A Text Editor**

**CODING :01**

import java.util.LinkedList;

import java.util.Queue;

public class Rope {

protected static class Node {

Node left;

Node right;

int lengthWeight;

String data;

public Node() {}

public Node(String str) {

data = str;

lengthWeight = str.length();

}

private Node(Node left, Node right) {

if (left.lengthWeight == 0) {

left = right;

right = null;

}

this.left = left;

this.right = right;

lengthWeight = getWeight(left);

}

protected static int getWeight(Node leftChild) {

int weight = leftChild.lengthWeight;

if (!leftChild.isLeaf() && leftChild.right != null) {

do {

weight += leftChild.right.lengthWeight;

leftChild = leftChild.right;

} while (leftChild.right != null);

}

return weight;

}

public boolean isLeaf() {

return data != null;

}

public boolean isEmpty() {

return lengthWeight == 0;

}

}

private static class Pair {

Node left;

Node right;

public Pair(Node left, Node right) {

this.left = left;

this.right = right;

}

}

private Node root;

public void insert(String str, int index) {

if (str == null || str.length() == 0) {

return;

}

Node newNode = new Node(str);

if (root == null) {

root = newNode;

} else {

Pair splitNodes = split(index, root);

root = concatenate(splitNodes.left, concatenate(newNode, splitNodes.r

}

}

public void delete(int from, int to) {

if (root == null) {

return;

}

if (from == 0) {

Pair splitNodes = split(to, root);

root = splitNodes.right;

if (root.isEmpty()) {

root = null;

}

} else {

Pair splitNodesLeft = split(from, root);

Pair splitNodesRight = split(to - from, splitNodesLeft.right);

root = concatenate(splitNodesLeft.left, splitNodesRight.right);

}

}

private Node concatenate(Node left, Node right) {

return new Node(left, right);

}

private Pair split(int index, Node startingRoot) {

Queue<Node> prunedNodes = new LinkedList<>();

goDownPathAndUpdate(index, startingRoot, prunedNodes);

return new Pair(startingRoot, rightSideSplit(prunedNodes));

}

private void goDownPathAndUpdate(int index, Node current, Queue<Node> pruneNodes) {

if (current.isLeaf()) {

pruneNodes.add(new Node(current.data.substring(index)));

current.data = current.data.substring(0, index);

current.lengthWeight = current.data.length();

} else {

Node selectedChild = null; //Remember which child we went down

if (current.lengthWeight <= index && current.right != null) {

selectedChild = current.right;

goDownPathAndUpdate(index - current.lengthWeight, selectedChi

} else {

selectedChild = current.left;

goDownPathAndUpdate(index, selectedChild, pruneNodes);

}

//Update the parent node based on child node

if (current.right != selectedChild) { //only prune right side if chil

pruneNodes.add(current.right);

current.right = null;

} else if (current.left.isEmpty()) { //fixing the child links

current.left = current.right;

current.right = null;

} else if (current.right.isEmpty()) {

current.right = null;

}

current.lengthWeight = Node.getWeight(current.left);

}

}

private Node rightSideSplit(Queue<Node> prunedNodes) {

Node rightSide = prunedNodes.remove();

while (!prunedNodes.isEmpty()) {

rightSide = concatenate(rightSide, prunedNodes.remove());

}

return rightSide;

}

public char charAt(int index) {

return charAt(index, root);

}

private char charAt(int index, Node current) {

if (current == null) {

return '\0';

}

if (!current.isLeaf()) {

if (current.lengthWeight <= index && current.right != null) {

return charAt(index - current.lengthWeight, current.right);

} else {

return charAt(index, current.left);

}

} else {

return current.data.charAt(index);

}

}

public String report(int start, int end) {

return report(start, end, root);

}

private String report(int start, int end, Node current) {

if (current == null || end < 1) {

return "";

}

if (current.isLeaf()) {

if (start > current.lengthWeight) {

return "";

}

if (end > current.lengthWeight) {

end = current.lengthWeight;

}

if (start < 0) {

start = 0;

}

return current.data.substring(start, end);

}

return report(start, end, current.left)

+ report(start - current.lengthWeight, end - current.lengthWe

}

public static void main(String[] args) {

Rope rope = new Rope();

rope.insert("llo\_", 0);

rope.insert("na", 4);

rope.insert("my\_", 4);

rope.insert("me\_i", 9);

rope.insert("s", 13);

rope.insert("\_Simon", 14);

rope.insert("He", 0);

String finalInput = "Hello\_my\_name\_is\_Simon";

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

System.out.print(rope.charAt(i));

} //Hello\_my\_name\_is\_Simon

rope.delete(0, finalInput.length()/2); //delete "Hello\_my\_na",

System.out.println("\n" + rope.report(0, finalInput.length())); //me\_is\_Simon

rope.delete(3, 5); //delete "is", result = me\_\_Simon

System.out.println(rope.report(0, finalInput.length())); //me\_\_Simon

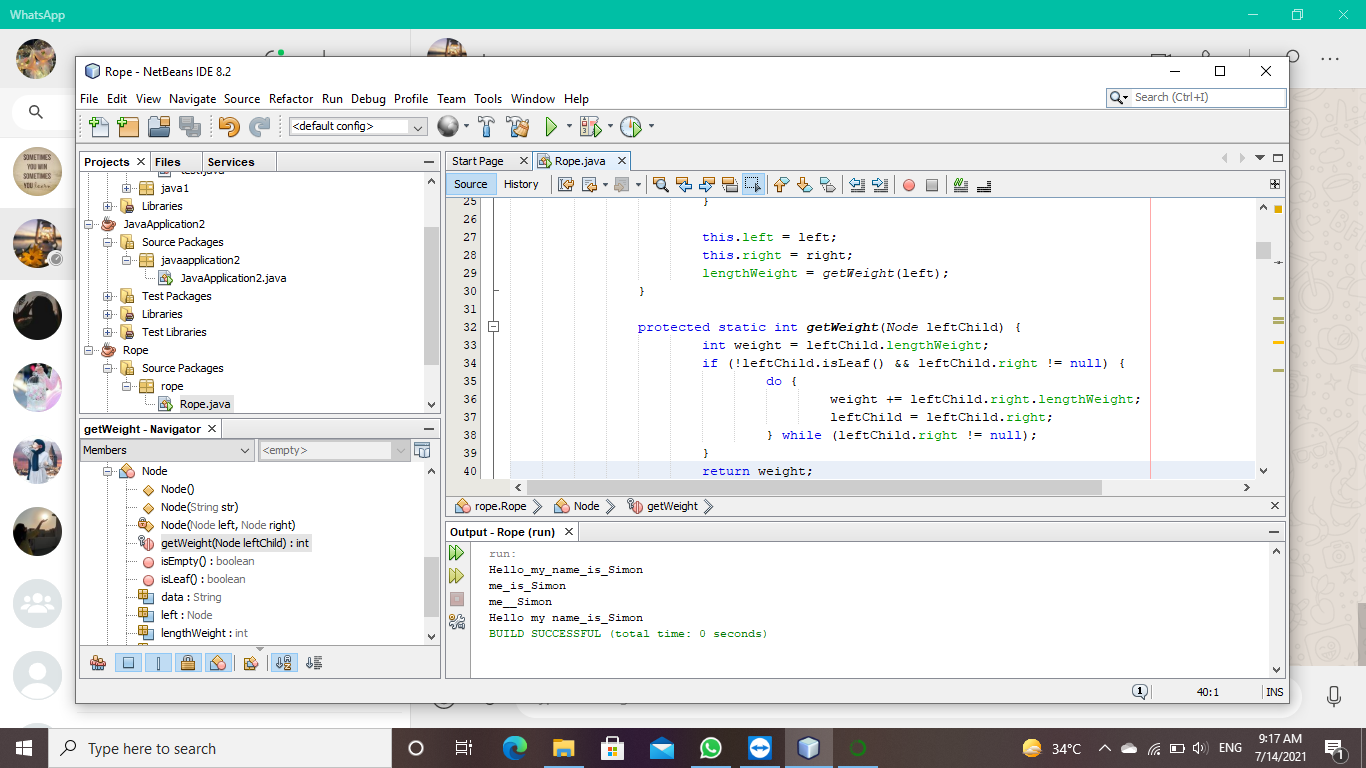
rope.insert("Hello my na", 0);

rope.insert("is", 14);

System.out.println(rope.report(0, finalInput.length())); //Hello\_my\_name\_is\_S

}

}



**CODING:02**

// Java Program to create a text editor using java

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JTextArea;

import javax.swing.UIManager;

import javax.swing.plaf.metal.MetalLookAndFeel;

import javax.swing.plaf.metal.OceanTheme;

public class Editor extends JFrame implements ActionListener{

// Text component

JTextArea t;

// Frame

JFrame f;

// Constructor

Editor()

{

// Create a frame

f = new JFrame("editor");

try {

// Set metal look and feel

UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

// Set theme to ocean

MetalLookAndFeel.setCurrentTheme(new OceanTheme());

}

catch (Exception e) {

}

// Text component

t = new JTextArea();

// Create a menubar

JMenuBar mb = new JMenuBar();

// Create amenu for menu

JMenu m1 = new JMenu("File");

// Create menu items

JMenuItem mi1 = new JMenuItem("New");

JMenuItem mi2 = new JMenuItem("Open");

JMenuItem mi3 = new JMenuItem("Save");

JMenuItem mi9 = new JMenuItem("Print");

// Add action listener

mi1.addActionListener(this);

mi2.addActionListener(this);

mi3.addActionListener(this);

mi9.addActionListener(this);

m1.add(mi1);

m1.add(mi2);

m1.add(mi3);

m1.add(mi9);

// Create amenu for menu

JMenu m2 = new JMenu("Edit");

// Create menu items

JMenuItem mi4 = new JMenuItem("cut");

JMenuItem mi5 = new JMenuItem("copy");

JMenuItem mi6 = new JMenuItem("paste");

// Add action listener

mi4.addActionListener(this);

mi5.addActionListener((ActionListener) this);

mi6.addActionListener(this);

m2.add(mi4);

m2.add(mi5);

m2.add(mi6);

JMenuItem mc = new JMenuItem("close");

mc.addActionListener(this);

mb.add(m1);

mb.add(m2);

mb.add(mc);

f.setJMenuBar(mb);

f.add(t);

f.setSize(500, 500);

f.show();

}

// If a button is pressed

public void actionPerformed(ActionEvent e)

{

String s = e.getActionCommand();

if (s.equals("cut")) {

t.cut();

}

else if (s.equals("copy")) {

t.copy();

}

else if (s.equals("paste")) {

t.paste();

}

else if (s.equals("Save")) {

// Create an object of JFileChooser class

JFileChooser j = new JFileChooser("f:");

// Invoke the showsSaveDialog function to show the save dialog

int r = j.showSaveDialog(null);

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

// Set the label to the path of the selected directory

File fi = new File(j.getSelectedFile().getAbsolutePath());

try {

// Create a file writer

FileWriter wr = new FileWriter(fi, false);

// Create buffered writer to write

BufferedWriter w = new BufferedWriter(wr);

// Write

w.write(t.getText());

w.flush();

w.close();

}

catch (Exception evt) {

JOptionPane.showMessageDialog(f, evt.getMessage());

}

}

// If the user cancelled the operation

else

JOptionPane.showMessageDialog(f, "the user cancelled the operation");

}

else if (s.equals("Print")) {

try {

// print the file

t.print();

}

catch (Exception evt) {

JOptionPane.showMessageDialog(f, evt.getMessage());

}

}

else if (s.equals("Open")) {

// Create an object of JFileChooser class

JFileChooser j = new JFileChooser("f:");

// Invoke the showsOpenDialog function to show the save dialog

int r = j.showOpenDialog(null);

// If the user selects a file

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

// Set the label to the path of the selected directory

File fi = new File(j.getSelectedFile().getAbsolutePath());

try {

// String

String s1 = "", sl = "";

// File reader

FileReader fr = new FileReader(fi);

// Buffered reader

BufferedReader br = new BufferedReader(fr);

// Initialize sl

sl = br.readLine();

// Take the input from the file

while ((s1 = br.readLine()) != null) {

sl = sl + "\n" + s1;

}

// Set the text

t.setText(sl);

}

catch (Exception evt) {

JOptionPane.showMessageDialog(f, evt.getMessage());

}

}

// If the user cancelled the operation

else

JOptionPane.showMessageDialog(f, "the user cancelled the operation");

}

else if (s.equals("New")) {

t.setText("");

}

else if (s.equals("close")) {

f.setVisible(false);

}

}

// Main class

public static void main(String[] args) {

Editor e = new Editor();

}

}

