Source Code with Comments:

**/\***

**\* Huffman.java**

**\***

**\* Created on May 21, 2007, 1:01 PM**

**\*/**

package makingalicesmall;

import java.util.\*;

import java.lang.\*;

import java.io.\*;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JFrame;

/\*\*

\*

\* @author pbladek

\*/

public class Huffman

{

public static final int CHARMAX = 128;

public static final byte CHARBITS = 7;

public static final short CHARBITMAX = 128;

private GuiJFrame frame;

/\*\*

\* Creates a new instance of Huffman object and also instantiates its GUI

\*/

public Huffman(){

frame = new GuiJFrame(this);

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

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

/\*\*

\* This is the main method of the project which takes in command-line arguement and

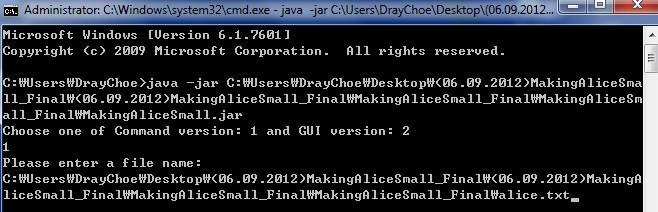
\* encode or decode according to it. This method also instantiate the Huffman object

\* which in turn instantiate its GUI

\*

\* @param args the command line arguments

\*/

public static void main(String[] args){

//----------------------------------------------------

// used for debugging encoding

//----------------------------------------------------

// args = new String[1];

// args[0] = "alice.txt";

//----------------------------------------------------

// used for debugging encoding

//----------------------------------------------------

// args = new String[2];

// args[0] = "-d";

// args[1] = "alice.txt"; +

//----------------------------------------------------

boolean decode = false;

String textFileName = null;

Scanner in = new Scanner(System.in);

int commandNumber;

System.out.println("Choose one of Command version: 1 and GUI version: 2 ");

commandNumber = Integer.parseInt(in.nextLine());

switch(commandNumber)

{

case 1: // Command Version

if(args.length > 0)

{

if(args[0].substring(0,2).toLowerCase().equals("-d"))

{

decode = true;

if(args.length > 1)

textFileName = args[1];

if(textFileName == null)

{

System.out.println("Please enter a file name: ");

textFileName = in.nextLine();

}

}

else

textFileName = args[0];

}

else

{

System.out.println("Please enter a file name: ");

textFileName = in.nextLine();

}

break;

case 2: // GUI VERSION

break;

}

Huffman coder = new Huffman();

if (textFileName != null){

if(!decode)

{

coder.encode(textFileName);

}

else

{

coder.decode(textFileName);

}

}

}

/\*\*

\* This method counts the occurances of each characted in the file passed in.

\*

\* @param fileName The name of the file to be encoded

\*

\* @return An array containing the occurances of each character sorted according

\* to their index

\*

\*/

private int[] countChar(String fileName) {

try{

Scanner in = new Scanner(new File(fileName));

int[] count = new int[CHARMAX];

while(in.hasNextLine()) {

String ln = in.nextLine();

char[] ch = ln.toCharArray();

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

count[ch[i]]++;

count['\n']++;

}

return count;

} catch(IOException ex){

System.out.println(ex.getMessage());

return null;

}

}

/\*\*

\* This method creates an array of HuffmanChar containing information of both the

\* character and its occurances.

\*

\* @param charCount The array containing the occurances of each character sorted according

\* to their index

\* @param ARRAY\_SIZE The size of the array of HuffmanChar

\*

\* @return An array of HuffmanChar

\*/

private HuffmanChar[] createCharCountArray(int[] charCount, int ARRAY\_SIZE){

int sum = 0;

HuffmanChar[] charCountArray = new HuffmanChar[ARRAY\_SIZE];

for(int i = 0, j = 0; i < charCount.length; i++){

sum += charCount[i];

if(charCount[i] != 0)

charCountArray[j++] = new HuffmanChar(

(char)i, charCount[i]);

}

System.out.println("Sum: " + sum);

return charCountArray;

}

/\*\*

\* This method reads in the text file and encode it to binary numbers by using the

\* map provided by the HuffmanTree construted using the array of HuffmanChar.

\*

\* @param fileName The name of the file to be encoded

\* @param theTree HuffmanTree used to encode the file

\*

\* @return Binary numbers which is the encoded text

\*/

private String getTotalBinaryCode(String fileName, HuffmanTree<Character> theTree) {

try{

String tempBinaryCode = "";

String totalBinaryCodes = "";

Scanner in = new Scanner(new File(fileName));

while(in.hasNextLine()){

String line = in.nextLine();

for(int i = 0; i < line.length(); i++) // line by line

tempBinaryCode = tempBinaryCode.concat(

(String)theTree.codeMap.get(line.charAt(i)));

tempBinaryCode = tempBinaryCode.concat((String)theTree.codeMap.get('\n'));

totalBinaryCodes = totalBinaryCodes + tempBinaryCode;

tempBinaryCode = "";

}

return totalBinaryCodes;

} catch (IOException ex){

System.out.println(ex.getMessage());

return null;

}

}

/\*\*

\* This method save the encoded binary numbers into an array of bytes.

\*

\* @param fileName The name of the file to write to

\* @param totalBinaryCodes Encoded binary numbers to be save

\*/

private void createSaveDataArray(String fileName, String totalBinaryCodes) {

byte[] saveDataArray = new byte[(totalBinaryCodes.length()) / 8 + 1 + 1];

loop1:

for(int i = 0, j = 0; j < saveDataArray.length; i += 8, j++){

if (j == saveDataArray.length - 2){

String lastStr = totalBinaryCodes.substring(i);

if (lastStr.length() == 0)

saveDataArray[saveDataArray.length - 2] = (byte) 0;

else

saveDataArray[saveDataArray.length - 2] = (byte) Integer.parseInt(lastStr, 2);

saveDataArray[saveDataArray.length - 1] = (byte) lastStr.length();

break loop1;

}

saveDataArray[j] = (byte) Integer.parseInt(

totalBinaryCodes.substring(i, i + 8), 2); // Converting binarycodes to decimal.

if (i % 2000 == 0)

if (i > 0)

frame.setTextArea("Key file 'alice.cod' successfully created!\nWriting encoded file "

+ Double.toString(i \* 100.0 /saveDataArray.length).substring(0, 5)

+ "%...");

}

frame.setTextArea("Key file 'alice.cod' successfully created!\n"

+ "Writing encoded file completed!\n");

writeEncodedFile(saveDataArray, fileName);// by Dray

frame.appendTextArea("Encoded file 'alice.huf' successfully created!\n");

}

/\*\*

\* This method convert the HuffmanChar passed in and create the three-byte array

\* to be saved on the .cod file.

\*

\* @param fileName The name of the file to write to

\* @param charCountArray The array of HuffmanChar to be converted to three-byte array

\* @param ARRAY\_SIZE The size of the array of HuffmanChar

\*/

private void createKeyByteArray(String fileName, HuffmanChar[] charCountArray, int ARRAY\_SIZE) {

byte[] byteArray = new byte[3 \* (ARRAY\_SIZE)];

int i = 0;

for (HuffmanChar h: charCountArray){

byte[] temp = h.toThreeBytes();

for (int j = 0; j < 3; i++, j++)

byteArray[i] = temp[j];

}

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

System.out.print(byteArray[j] + " | ");

writeKeyFile(fileName, byteArray);

frame.appendTextArea("Key file 'alice.cod' successfully created!\n");

}

/\*\*

\* This method uses the byte array passed in to write the .huf file.

\*

\* @param fileName The name of the file to write to

\* @param bytes The array of byte to be written to the file

\*/

public void writeEncodedFile(byte[] bytes, String fileName){

fileName = fileName.substring(0 , fileName.length() - 3) + "huf";

try // creating .huf file for Object.

{

ObjectOutput out = new ObjectOutputStream(new FileOutputStream(fileName));

out.writeObject(bytes);

out.close();

}

catch(IOException ex)

{

System.err.println(ex);

}

}

/\*\*

\* This method uses the byte array passed in to write the .cod file.

\*

\* @param fileName the name of the file to write to

\* @param byteArray the array of byte to be written to the file

\*/

public void writeKeyFile(String fileName, byte[] byteArray){

fileName = fileName.substring(0 , fileName.length() - 3) + "cod";

try

{

ObjectOutput out = new ObjectOutputStream(

new FileOutputStream(fileName));

out.writeObject(byteArray);

out.close();

System.out.println("\n alice.cod is created");

}

catch (IOException ex)

{

System.err.println(ex);

}

}

/\*\*

\* This is the main encoding method which calls on other methods in order to read in

\* the given file, create its HuffmanTree and then encode the text file according to

\* the HuffmanTree

\*

\* @param fileName The name of the file to be encoded

\*/

public void encode(String fileName){

frame.setTextArea("");

int[] charCount = countChar(fileName);

int nullData = 0;

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

if(charCount[i] == 0)

nullData++;

final int ARRAY\_SIZE = CHARMAX - nullData;

HuffmanChar[] charCountArray = createCharCountArray(charCount, ARRAY\_SIZE);

HuffmanTree<Character> theTree = new HuffmanTree(charCountArray);

createKeyByteArray(fileName, charCountArray, ARRAY\_SIZE);

String totalBinaryCode = getTotalBinaryCode(fileName, theTree);

createSaveDataArray(fileName, totalBinaryCode);

frame.processField.setText("Thank you");

}

/\*\*

\* This method reads in .cod file which contains an array of bytes to construct

\* back the an array of HuffmanChar which contains information on characters and

\* its occurances

\*

\* @param fileName The name of the file to be decoded

\*

\* @return array of HuffmanChar needed to make HuffmanTree

\*/

public HuffmanChar[] readKeyFile(String fileName){

System.out.println("decoding file...");

ObjectInput inCod;

HuffmanChar[] dataArray = null;

try{

inCod = new ObjectInputStream(new FileInputStream(

new File(fileName.substring(0, fileName.length() - 3) + "cod")));

//Replace to above

byte[] tempThreeByte = new byte[3];

byte[] tempThreeByteArray = (byte[]) inCod.readObject();

dataArray = new HuffmanChar[tempThreeByteArray.length / 3];

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

for (int j = 0; j < 3; i++, j++)

tempThreeByte[j] = tempThreeByteArray[i];

dataArray[(i - 2) / 3] = new HuffmanChar(tempThreeByte);

}

System.out.println("\nSorted data");

for (HuffmanChar h: dataArray)

System.out.println(h); // h is sorted data

return dataArray;

} catch (IOException ex){

System.out.println(ex);

return null;

} catch (ClassNotFoundException ex){

System.out.println(ex);

return null;

}

}

/\*\*

\* Reads the encoded file with byte array. Each four bytes are added

\* to integer using left shift and | operations. "0"s are added to the

\* front to recover the key. If the byte less than 0, which means it

\* will have 1s in the front even if converted to integer, we remove the

\* 1 in the front by subtracting 128, and then merging 1 and the

\* subtracted byte individually into the integer value. Very last byte

\* is the length of the last byte. Also array of String is used to

\* store the binary string, so that no string variable will become too

\* big. The method will then return the string that is combination of

\* the string array.

\*

\* @param fileName The name of the file to be decoded

\*

\* @return String of binary numbers read from .huf file

\*/

public String readEncodedFile(String fileName){

ObjectInput inCod;

byte[] tempByteArray;

String[] totalBinaryCodes;

try{

inCod = new ObjectInputStream(new FileInputStream(

new File(fileName.substring(0, fileName.length() - 3) + "huf")));

tempByteArray = (byte[]) inCod.readObject();

} catch (IOException ex){

System.out.println(ex);

return null;

} catch (ClassNotFoundException ex){

System.out.println(ex);

return null;

}

int arraySize = tempByteArray.length / 10000 + 1;

totalBinaryCodes = new String[arraySize];

System.out.println("j size: " + arraySize);

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

totalBinaryCodes[j] = "";

System.out.println("j " + j + ": " + totalBinaryCodes[j]);

}

int fLength = tempByteArray.length / 4;

int leftLength = tempByteArray.length - fLength \* 4;

if(leftLength < 2)

{

leftLength += 4;

fLength -= 1;

}

for(int i = 0, j = 0; i < fLength \* 4; i = i + 4)

{

int fBytes = 0;

if(tempByteArray[i] < 0)

{

tempByteArray[i] -= 128;

fBytes = 128;

}

fBytes = fBytes | (tempByteArray[i] & 0xFF);

if(tempByteArray[i + 1] < 0)

{

tempByteArray[i + 1] -= 128;

fBytes = (fBytes << 8) | (128 & 0xFF);

fBytes = fBytes | (tempByteArray[i + 1] & 0xFF);

}

else

{

fBytes = (fBytes << 8) | (tempByteArray[i+1]& 0xFF);

}

if(tempByteArray[i + 2] < 0)

{

tempByteArray[i + 2] -= 128;

fBytes = (fBytes << 8) | (128 & 0xFF);

fBytes = fBytes | (tempByteArray[i+2]& 0xFF);

}

else

{

fBytes = (fBytes << 8) | (tempByteArray[i+2]& 0xFF);

}

if(tempByteArray[i + 3] < 0)

{

tempByteArray[i + 3] -= 128;

fBytes = (fBytes << 8) | (128 & 0xFF);

fBytes = fBytes | (tempByteArray[i + 3]& 0xFF);

}

else

{

fBytes = (fBytes << 8) | (tempByteArray[i+3]& 0xFF);

}

String temp = Integer.toBinaryString(fBytes);

while(temp.length() < 32)

{

temp = "0" + temp;

}

if (totalBinaryCodes[j].length() > 8 \* 10000){

j++;

System.out.println("new j: " + j);

}

totalBinaryCodes[j] = totalBinaryCodes[j] + temp;

}

System.out.println(tempByteArray.length-fLength \* 4);

loop1:

for (int i = fLength \* 4; i < tempByteArray.length; i++){

byte tempByte = tempByteArray[i];

String temp = Integer.toBinaryString(tempByte);

if (i == tempByteArray.length - 2){

while (temp.length() < tempByteArray[tempByteArray.length - 1])

temp = "0" + temp;

totalBinaryCodes[totalBinaryCodes.length - 1] =

totalBinaryCodes[totalBinaryCodes.length - 1] + temp;

break loop1;

}

if (temp.length() > 8)

{

//temp = temp.substring(temp.length() - 8, temp.length());

tempByte = (byte)(tempByte - 128);

temp = Integer.toBinaryString(tempByte);

while(temp.length() < 7 )

{

temp = "0" + temp;

}

temp = "1" + temp;

}

else if (temp.length() < 8){

while (temp.length() < 8)

temp = "0" + temp;

}

totalBinaryCodes[totalBinaryCodes.length - 1] =

totalBinaryCodes[totalBinaryCodes.length - 1] + temp;

if (i % 250 == 0){

if (i > 0)

frame.setTextArea("Loading encoded file " +

Double.toString(i \* 100.0 /tempByteArray.length).substring(0, 5)

+ "%...");

System.out.println(i \* 100.0 /tempByteArray.length);

}

//System.out.println(i + "/" + tempByteArray.length);

//System.out.println(temp);

//System.out.println(i + "/" + tempByteArray.length);

}

frame.setTextArea("Encoded file loaded 100.00%!");

//System.out.println("totalBinCode: " + totalBinaryCodes);

String finaltotalBinaryCodes = "";

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

System.out.println("j " + j + ": " + totalBinaryCodes[j]);

finaltotalBinaryCodes = finaltotalBinaryCodes + totalBinaryCodes[j];

}

return finaltotalBinaryCodes;

}

/\*\*

\* This method decode the string of binary numbers back into decoded strings by using

\* the map provided by the HuffmanTree and then create a new file to write in the

\* decoded text.

\*

\* @param fileName The name of the file to be decoded

\* @param totalBinaryCodes String of binary numbers read from .huf file

\* @param Htree HuffmanTree constructed from .cod file

\*/

private void writeDecodedFile(String fileName, String totalBinaryCodes, HuffmanTree Htree) {

String tempKey = "";

String originalText = "";

System.out.println("totalBinaryCodes length: " + totalBinaryCodes.length());

final int ORIGINAL\_BIN\_LENGTH = totalBinaryCodes.length();

while(totalBinaryCodes.length() > 0){

tempKey = tempKey + totalBinaryCodes.substring(0, 1);

totalBinaryCodes = totalBinaryCodes.substring(1);

if (Htree.keyMap.containsKey((String) tempKey)){

Character tempChar = (Character) Htree.keyMap.get(tempKey);

originalText = originalText + tempChar.charValue();

tempKey = "";

}

if (totalBinaryCodes.length() % 7500 == 0)

if (totalBinaryCodes.length() - ORIGINAL\_BIN\_LENGTH > 0)

frame.setTextArea("Encoded file loaded 100.00%!\nDecoding encoded file " +

Double.toString( (totalBinaryCodes.length() – ORIGINAL\_BIN\_LENGTH) \* 100.0 / ORIGINAL\_BIN\_LENGTH).substring(0, 5) + "%...");

}

frame.setTextArea("Encoded file loaded 100.00%!\nDecoding encoded file completed!"

+ "\n\nEncoded file:\n\n" + originalText);

System.out.println(originalText);

try{

PrintWriter newWriter = new PrintWriter(new File(

fileName.substring(0, fileName.length() - 4) + "x.txt"));

newWriter.write(originalText);

newWriter.close();

} catch(IOException ex){

System.out.println(ex.getMessage());

}

}

/\*\*

\* This is the main decoding method which calls on other method to read in the key and

\* encoded file, translating encoded file back to strings and write a new file with the

\* decoded string

\*

\* @param inFileName The name of the file to be decoded

\*/

public void decode(String inFileName){

frame.setTextArea("");

HuffmanChar[] dataArray = readKeyFile(inFileName);

HuffmanTree Htree = new HuffmanTree(dataArray); // dataArray is sorted data

String totalBinaryCodes = readEncodedFile(inFileName);

writeDecodedFile(inFileName, totalBinaryCodes, Htree);

frame.processField.setText("Thank you");

}

}

**/\***

**\* HuffmanChar.java**

**\***

**\* Created on May 22, 2007, 5:26 PM**

**\***

**\* To change this template, choose Tools | Template Manager**

**\* and open the template in the editor.**

**\*/**

package makingalicesmall;

import java.io.\*;

/\*\*

\*

\* @author pbladek

\*/

public class HuffmanChar extends HuffmanData<Character>

implements Serializable

{

/\*\*

\* Creates a new instance of HuffmanChar

\*/

public HuffmanChar()

{

super();

}

/\*\*

\* Creates a new instance of HuffmanChar

\* @param c the character

\*/

public HuffmanChar(Character c)

{

super(c);

}

/\*\*

\* Creates a new instance of HuffmanChar

\* @param c the character

\* @param oc the number of occurances

\*/

public HuffmanChar(Character c, int oc)

{

super(c, oc);

}

/\*\*

\* Creates a new instance of HuffmanChar

\* @param hc a HuffmanChar

\*/

public HuffmanChar(HuffmanChar hc)

{

super(hc.getData(), hc.getOccurances());

}

/\*\*

\* Creates a new instance of HuffmanChar

\* @param threeBytes an array of three bytes

\*/

public HuffmanChar(byte[] threeBytes)

{

super(new Character((char)threeBytes[0]),

(((int)threeBytes[1] >= 0 ? (int)threeBytes[1] : (256 + (int)threeBytes[1])) << 8)

| ((int)threeBytes[2] >= 0 ? (int)threeBytes[2] : 256 + (int)threeBytes[2]));

}

/\*\*

\* returns the class converted to a 3-byte array

\* @return the class converted to a 3-byte array

\*/

public byte[] toThreeBytes()

{

byte[] ba = new byte[3];

ba[0] = (byte)(getData().charValue());

short oc = (short)getOccurances();

ba[1] = (byte)(oc >> 8);

ba[2] = (byte)(oc & (byte)(-1));

return ba;

}

}

**/\***

**\* HuffmanData.java**

**\***

**\* Created on May 21, 2007, 2:17 PM**

**\*/**

package makingalicesmall;

import java.lang.\*;

/\*\*

\* @author pbladek

\*/

public class HuffmanData<T extends Comparable<? super T>>

implements Comparable<HuffmanData<T>>

{

private T data;

private int occurances = 0;

/\*\*

\* Creates a new instance of HuffmanData

\*/

public HuffmanData(){}

/\*\* Creates a new instance of HuffmanData

\* @param dataIn the data part

\*/

public HuffmanData(T dataIn)

{

data = dataIn;

}

/\*\* Creates a new instance of HuffmanData

\* @param dataIn the data part

\* @param count the number of occurances

\*/

public HuffmanData(T dataIn, int count)

{

this(dataIn);

occurances = count;

}

/\*

\* accessor

\* @return data

\*/

public T getData()

{

return data;

}

/\*

\* accessor

\* @return occurances

\*/

public int getOccurances()

{

return occurances;

}

/\*\*

\*

\* @param o the other HuffmanData

\* @return -1 if <, 0 if ==, 1 if >

\*/

public int compareTo(HuffmanData<T> o)

{

return (occurances < o.occurances)? -1

: (occurances == o.occurances)? 0 : 1;

}

/\*

\* @return strng version of class

\*/

public String toString()

{

String dataString = "\*";

if(data != null)

dataString = data.toString();

return dataString + ": " + occurances + " ";

}

}

**/\***

**\* HuffmanTree.java**

**\***

**\* Created on May 21, 2007, 2:16 PM**

**\*/**

package makingalicesmall;

import java.util.\*;

/\*\*

\* binary tree for Huffman coding

\* @author pbladek

\*/

public class HuffmanTree<T extends Comparable<? super T>>

extends BinaryTree<HuffmanData<T>>

{

private final T MARKER = null;

SortedMap<T, String> codeMap;

SortedMap<String, T> keyMap;

private int leafCount = 0;

ArrayList<BinaryNode<HuffmanData<T>>> aList =

new ArrayList<BinaryNode<HuffmanData<T>>>();

/\*\*

\* Creates a new instance of HuffmanTree

\*/

public HuffmanTree()

{

super();

}

/\*\*

\* Creates a new instance of HuffmanTree

\* from an array of Huffman Data

\* @param dataArray n array of Huffman Data

\*/

public HuffmanTree(HuffmanData<T>[] dataArray)

{

sortArray(dataArray);

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

{

aList.add(new BinaryNode<HuffmanData<T>>(dataArray[i]));

}

createHuffman();

keyMap = new TreeMap<String, T>();

codeMap = new TreeMap<T, String>();

setMaps(getRootNode(), "");

//to be removed

System.out.println(keyMap.values());

System.out.println(codeMap.values());

}

/\*\*

\* creates two new HuffmanTrees and adds them to the root of this tree

\* @param left

\* @param rightt

\*/

private void add(BinaryNode<HuffmanData<T>> left,

BinaryNode<HuffmanData<T>> right)

{

HuffmanTree<T> leftTree = new HuffmanTree<T>();

leftTree.setRootNode(left);

HuffmanTree<T> rightTree = new HuffmanTree<T>();

rightTree.setRootNode(right);

setTree(new HuffmanData<T>

(MARKER, left.getData().getOccurances()

+ right.getData().getOccurances()), leftTree, rightTree);

}

/\*\*

\* adds 2 new elements to this tree<br>

\* smaller on the left

\* @param element1

\* @param element2

\*/

private void firstAdd(HuffmanData<T> element1, HuffmanData<T> element2)

{

BinaryNode<HuffmanData<T>> left = new

BinaryNode<HuffmanData<T>>(element1);

BinaryNode<HuffmanData<T>> right = new

BinaryNode<HuffmanData<T>>(element2);

add(left,right);

}

/\*\*

\* Creates the huffman tree using the huffman data array.

\* Takes first two elements in the arraylist, and adds them to the root.

\* The latest created root replaces the previous, but the reference of it

\* is saved to the arraylist as a binary node.

\*/

private void createHuffman()

{

while(aList.size() > 1)

{

add(aList.get(0), aList.get(1));

BinaryNode<HuffmanData<T>> temp = (BinaryNode<HuffmanData<T>>) getRootNode();

aList.remove(0);

aList.remove(0);

addAt(temp);

System.out.println("temp: " + temp.getData().getOccurances());

}

//to be removed

System.out.println("Root " + ((BinaryNode<HuffmanData<T>>) getRootNode()).getData().getOccurances());

System.out.println("aList.size(): " + aList.size());

System.out.println(aList.get(0).getData().getData());

System.out.println(aList.get(0).getData().getOccurances());

}

private void addAt(BinaryNode<HuffmanData<T>> temp)

{

for(int i = 0; i < aList.size(); i++)

{

if(temp.getData().getOccurances() <= aList.get(i).getData().getOccurances())

{

aList.add(i,temp);

return;

}

}

aList.add(temp);

}

/\*\*

\* set up the 2 maps

\* @param node

\* @param codeString

\*/

private void setMaps(BinaryNodeInterface<HuffmanData<T>> node,

String codeString)

{

if(node.hasLeftChild())

{

setMaps(node.getLeftChild(),codeString + "0"); // Recursion.

}

if(node.hasRightChild())

{

setMaps(node.getRightChild(),codeString + "1");

}

if(node.getData().getData() != null) // getData twice?

{

keyMap.put(codeString, node.getData().getData()); // a character representation of the chars

codeMap.put(node.getData().getData(), codeString); // a string representation of the binary digits

}

}

private void sortArray(HuffmanData<T>[] data)

{

HuffmanData smallest;

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

int smallestIndex = i;

smallest = data[i];

for (int j = i + 1; j < data.length; j++){

if (data[j].compareTo(smallest) < 0){

smallestIndex = j;

smallest = data[j];

}

}

if (i != smallestIndex){

HuffmanData temp = data[i];

data[i] = smallest;

data[smallestIndex] = temp;

}

}

}

/\*

\* accessor for codeMap

\* @ return codeMap

\*/

public SortedMap<T, String> getCodeMap()

{

return codeMap;

}

/\*

\* accessor for keyMap

\* @ return keyMap

\*/

public SortedMap<String, T> getKeyMap()

{

return keyMap;

}

}

**/\***

**\* TreeInterface.java**

**\***

**\* Created on May 21, 2007, 2:17 PM**

**\*/**

package makingalicesmall;

/\*\*

\* @author Carrano

\*/

public interface TreeInterface<T>

{

public T getRootData();

public int getHeight();

public int getNumberOfNodes();

public boolean isEmpty();

public void clear();

}

**/\***

**\* TreeIteratorInterface.java**

**\***

**\* Created on May 21, 2007, 1:31 PM**

**\*/**

package makingalicesmall;

import java.util.\*;

/\*\*

\* @author Carranno

\*/

public interface TreeIteratorInterface<T>

{

public Iterator<T> getInOrderIterator();

}

**/\***

**\* BinaryNode.java**

**\***

**\* Created on May 21, 2007, 1:08 PM**

**\*/**

package makingalicesmall;

import java.io.Serializable;

/\*\*

\*

\* @author Carrano

\*/

public class BinaryNode < T > implements BinaryNodeInterface < T >,

Serializable

{

private T data;

private BinaryNode < T > left;

private BinaryNode < T > right;

/\*\*

\* default constructor

\*/

public BinaryNode ()

{

this (null); // call next constructor

} // end default constructor

/\*\*

\* constructor

\* @param dataPortion the data portion

\*/

public BinaryNode (T dataPortion)

{

this (dataPortion, null, null); // call next constructor

} // end constructor

/\*\*

\* constructor

\* @param leftChild The left child

\* @param rightChild The Right child

\*/

public BinaryNode (T dataPortion, BinaryNode < T > leftChild,

BinaryNode <T> rightChild)

{

data = dataPortion;

left = leftChild;

right = rightChild;

} // end constructor

/\*\*

\* Accessors for the data

\* @return data

\*/

@Override

public T getData ()

{

return data;

} // end getData

/\*\*

\* Set a new data

\* @param newData new element or data

\*/

@Override

public void setData (T newData)

{

data = newData;

} // end setData

/\*\*

\* Accessors for a left child

\* @return A left child

\*/

@Override

public BinaryNodeInterface < T > getLeftChild ()

{

return left;

} // end getLeftChild

/\*\*

\* Set a left child

\* @param leftChild Element to be the left child

\*/

@Override

public void setLeftChild (BinaryNodeInterface < T > leftChild)

{

left = (BinaryNode < T > ) leftChild;

} // end setLeftChild

/\*\*

\* Checking if a left child exist

\* @return True if a left child existed; otherwise, return False

\*/

@Override

public boolean hasLeftChild ()

{

return left != null;

} // end hasLeftChild

/\*\*

\* Checking a node if it is leaf

\* @return True if the node is a leaf; otherwise, return false

\*/

@Override

public boolean isLeaf ()

{

return (left == null) && (right == null);

} // end isLeaf

/\*\*

\* Accessors for right child

\* @return The right child

\*/

@Override

public BinaryNodeInterface < T > getRightChild ()

{

return right;

} // end getRightChild

/\*\*

\* Set a right child

\* @param rightChild Element to be the right child

\*/

@Override

public void setRightChild (BinaryNodeInterface < T > rightChild)

{

right = (BinaryNode < T > ) rightChild;

} // end setRightChild

/\*\*

\* Checking if a right child exist

\* @return True if a right child existed; otherwise, return False

\*/

@Override

public boolean hasRightChild ()

{

return right != null;

} // end hasRightChild

/\*\*

\* Implementations of getRightChild, setRightChild, and hasRightChild are

\* analogous to their left-child counterparts.

\* @return

\*/

@Override

public BinaryNodeInterface < T > copy ()

{

BinaryNode < T > newRoot = new BinaryNode <> (data);

if (left != null)

newRoot.left = (BinaryNode < T > ) left.copy ();

if (right != null)

newRoot.right = (BinaryNode < T > ) right.copy ();

return newRoot;

} // end copy

/\*\*

\* Get the height of the tree

\* @return the height of the tree

\*/

@Override

public int getHeight ()

{

return getHeight (this); // call private getHeight

} // end getHeight

/\*\*

\* Get the height of the tree beginning from a specific node

\* @param node

\* @return

\*/

private int getHeight (BinaryNode < T > node)

{

int height = 0;

if (node != null)

height = 1 + Math.max (getHeight (node.left),

getHeight (node.right));

return height;

} // end getHeight

/\*\*

\* Get the number of nodes in the tree

\* @return the number of the nodes

\*/

@Override

public int getNumberOfNodes ()

{

int leftNumber = 0;

int rightNumber = 0;

if (left != null)

leftNumber = left.getNumberOfNodes ();

if (right != null)

rightNumber = right.getNumberOfNodes ();

return 1 + leftNumber + rightNumber;

} // end getNumberOfNodes

} // end BinaryNode

**/\***

**\* BinaryNodeInterface.java**

**\***

**\* Created on May 21, 2007, 1:07 PM**

**\*/**

package makingalicesmall;

/\*\*

\*

\* @author Carrano

\*/

interface BinaryNodeInterface< T >

{

/\*\* Task: Retrieves the data portion of the node.

\* @return the object in the data portion of the node \*/

public T getData ();

/\*\* Task: Sets the data portion of the node.

\* @param newData the data object \*/

public void setData (T newData);

/\*\* Task: Retrieves the left child of the node.

\* @return the node that is this nodes left child \*/

public BinaryNodeInterface < T > getLeftChild ();

/\*\* Task: Retrieves the right child of the node.

\* @return the node that is this nodes right child \*/

public BinaryNodeInterface < T > getRightChild ();

/\*\* Task: Sets the nodes left child to a given node.

\* @param leftChild a node that will be the left child \*/

public void setLeftChild (BinaryNodeInterface < T > leftChild);

/\*\* Task: Sets the nodes right child to a given node.

\* @param rightChild a node that will be the right child \*/

public void setRightChild (BinaryNodeInterface < T > rightChild);

/\*\* Task: Detects whether the node has a left child.

\* @return true if the node has a left child \*/

public boolean hasLeftChild ();

/\*\* Task: Detects whether the node has a right child.

\* @return true if the node has a right child \*/

public boolean hasRightChild ();

/\*\* Task: Detects whether the node is a leaf.

\* @return true if the node is a leaf \*/

public boolean isLeaf ();

/\*\* Task: Counts the nodes in the subtree rooted at this node.

\*@returnthenumberof nodes in the subtree rooted at this node \*/

public int getNumberOfNodes ();

/\*\* Task: Computes the height of the subtree rooted at this node.

\* @return the height of the subtree rooted at this node \*/

public int getHeight ();

/\*\* Task: Copies the subtree rooted at this node.

\* @return the root of a copy of the subtree rooted at this node \*/

public BinaryNodeInterface < T > copy ();

} // end BinaryNodeInterface

**/\***

**\* BinaryTree.java**

**\***

**\* Created on May 21, 2007, 1:12 PM**

**\*/**

import java.util.Iterator;

import java.util.NoSuchElementException;

import java.util.Stack;

/\*\*

\* @author Carrano

\*/

public class BinaryTree<T> implements BinaryTreeInterface<T> ,

java.io.Serializable

{

private BinaryNodeInterface<T> root;

/\*\*

\* default constructor

\*/

public BinaryTree ()

{

root = null;

} // end default constructor

/\*\*

\* constructor

\* @param rootData data for root node

\*/

public BinaryTree (T rootData)

{

root = new BinaryNode <> (rootData);

} // end constructor

/\*\*

\* constructor

\* @param rootData data for root node

\* @param leftTree left subtree to attach

\* @param leftTree right subtree to attach

\*/

public BinaryTree (T rootData, BinaryTree<T> leftTree,

BinaryTree<T> rightTree)

{

privateSetTree (rootData, leftTree, rightTree);

} // end constructor

/\*\*

\* sets the root node

\* @param rootData data for root node

\*/

@Override

public void setTree (T rootData)

{

root = new BinaryNode <> (rootData);

} // end setTree

/\*\*

\* sets the root node

\* @param rootData data for root node

\* @param leftTree left subtree to attach

\* @param leftTree right subtree to attach

\*/

@Override

public void setTree (T rootData, BinaryTreeInterface < T > leftTree,

BinaryTreeInterface < T > rightTree)

{

privateSetTree (rootData, (BinaryTree < T > ) leftTree,

(BinaryTree < T > ) rightTree);

} // end setTree

/\*\*

\* @return copy of the tree

\*/

public BinaryNodeInterface<T> copy ()

{

BinaryNode < T > newRoot = new BinaryNode <> (root.getData());

if (root.getLeftChild() != null)

newRoot.setLeftChild((BinaryNode <T>)root.getLeftChild().copy());

if (root.getRightChild() != null)

newRoot.setRightChild((BinaryNode <T>)root.getRightChild().copy());

return newRoot;

} // end copy

/\*

\* Gets the tree height

\* @return height of tree

\*/

@Override

public int getHeight()

{

return root.getHeight();

} // end getHeight

/\*

\* gets the number of nodesw

\* @return numberOfNodes

\*/

@Override

public int getNumberOfNodes()

{

return root.getNumberOfNodes();

} // end getNumberOfNodes

private void privateSetTree(T rootData, BinaryTree < T > leftTree,

BinaryTree < T > rightTree)

{

root = new BinaryNode <> (rootData);

if ((leftTree != null) && !leftTree.isEmpty ())

root.setLeftChild (leftTree.root.copy ());

if ((rightTree != null) && !rightTree.isEmpty ())

root.setRightChild (rightTree.root.copy ());

} // end privateSetTree

/\*\*

\* Accessors for the root data

\* @return A root data

\*/

@Override

public T getRootData ()

{

T rootData = null;

if (root != null)

rootData = root.getData ();

return rootData;

} // end getRootData

/\*\*

\* Checking if the tree is empty

\* @return True if the tree is empty; otherwise, return false

\*/

@Override

public boolean isEmpty ()

{

return root == null;

} // end isEmpty

/\*\*

\* Clear the binary tree

\*/

@Override

public void clear ()

{

root = null;

} // end clear

/\*\*

\* Setup a root data for the binary tree

\* @param rootData Supposely to be the root

\*/

protected void setRootData (T rootData)

{

root.setData (rootData);

} // end setRootData

/\*\*

\* Setup a root data for the binary tree

\* @param rootNode Supposely to be the root

\*/

protected void setRootNode (BinaryNodeInterface < T > rootNode)

{

root = rootNode;

} // end setRootNode

/\*\*

\* Accessors for the root

\* @return

\*/

protected BinaryNodeInterface<T> getRootNode ()

{

return root;

} // end getRootNode

/\*\*

\* In Order Traversal

\*/

public void inorderTraverse ()

{

Stack< BinaryNodeInterface<T>> nodeStack =

new Stack<>();

BinaryNodeInterface < T > currentNode = root;

while (!nodeStack.isEmpty () || (currentNode != null))

{

// find leftmost node with no left child

while (currentNode != null)

{

nodeStack.push (currentNode);

currentNode = currentNode.getLeftChild ();

} // end while

// visit leftmost node, then traverse its right subtree

if (!nodeStack.isEmpty ())

{

BinaryNodeInterface < T > nextNode = nodeStack.pop ();

assert nextNode != null; // since nodeStack was not empty

// before the pop

System.out.println (nextNode.getData ());

currentNode = nextNode.getRightChild ();

} // end if

} // end while

}

/\*\*

\* Internal iterator

\*/

private class InorderIterator implements Iterator<T>

{

private Stack<BinaryNodeInterface<T>> nodeStack;

private BinaryNodeInterface < T > currentNode;

/\*\*

\* Default constructor

\*/

public InorderIterator ()

{

nodeStack = new Stack<>();

currentNode = root;

}

/\*\*

\* Returns true if there is a next element; false otherwise

\* @return true if there is a next element; false otherwise

\*/

@Override

public boolean hasNext ()

{

return !nodeStack.isEmpty () || (currentNode != null);

}

/\*\*

\* Returns the next element and moves forward

\* @return the next element

\*/

@Override

public T next ()

{

BinaryNodeInterface < T > nextNode = null;

// find leftmost node with no left child

while (currentNode != null)

{

nodeStack.push (currentNode);

currentNode = currentNode.getLeftChild ();

} // end while

// get leftmost node, then move to its right subtree

if (!nodeStack.isEmpty ())

{

nextNode = nodeStack.pop ();

assert nextNode != null; // since nodeStack was not empty

// before the pop

currentNode = nextNode.getRightChild ();

}

else

throw new NoSuchElementException ();

return nextNode.getData ();

} // end next

/\*\*

\* unsupported

\*/

@Override

public void remove ()

{

throw new UnsupportedOperationException ();

}

}

/\*\*

\* returns a new iterator over the class

\* @return a new iterator

\*/

@Override

public InorderIterator getInOrderIterator()

{

return new InorderIterator();

}

}

**/\***

**\* BinaryTreeInterface.java**

**\***

**\* Created on May 21, 2007, 1:17 PM**

**\*/**

/\*\*

\* @author Carranno

\*/

public interface BinaryTreeInterface < T > extends TreeInterface<T>,

TreeIteratorInterface<T>

{

/\*\* Task: Sets an existing binary tree to a new one-node binary tree.

\* @param rootData an object that is the data in the new trees root

\*/

public void setTree (T rootData);

/\*\* Task: Sets an existing binary tree to a new binary tree.

\* @param rootData an object that is the data in the new trees root

\* @param leftTree the left subtree of the new tree

\* @param rightTree the right subtree of the new tree \*/

public void setTree (T rootData, BinaryTreeInterface < T > leftTree,

BinaryTreeInterface < T > rightTree);

}

**/\***

**\* GuiJFrame.java**

**\*/**

package makingalicesmall;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

import javax.swing.\*;

/\*\*

\*

\* @author Sereyvathanak Khorn

\*/

public class GuiJFrame extends JFrame

{

private static final int FRAME\_WIDTH = 800;

private static final int FRAME\_HEIGHT = 600;

private static final int TEXT\_WIDTH = 50;

private static final int TEXT\_HEIGHT = 25;

private static final int STRUT\_SIZE = 32;

private static final int TEXTPANEL\_WIDTH = 584;

private static final int TEXTPANEL\_HEIGHT = 600;

private JFileChooser openChooser;

private File file;

private String textTile = "";

private String textTitleforDisplay = "";

private Huffman coder;

private JLabel titleLabel;

private JLabel bookTitle;

private JTextField bookTitle1;

private JPanel textPanel;

private JPanel buttonPanel;

private JButton encodeButton;

private JButton decodeButton;

private JTextArea information;

private JLabel processLabel;

public JTextField processField;

private int lightBlue = 0X63B1FF;

private StringBuilder display = new StringBuilder();

JScrollPane scrollPane;

/\*\*

\* Setting up the GUI

\* @param coder Huffman

\*/

public GuiJFrame(Huffman coder)

{

this.coder = coder;

setTitle("Making Alice Smaller");

setSize(FRAME\_WIDTH, FRAME\_HEIGHT);

setLayout(new FlowLayout());

//Menu

JMenuBar menuBar = new JMenuBar();

setJMenuBar(menuBar);

menuBar.add(fileMenu());

menuBar.add(info());

//Button

textBox();

add(textPanel);

button();

add(buttonPanel);

encodingButton();

decodingButton();

}

/\*\*

\* Create panel for buttons, (encoding button) and (decoding button)

\*/

private void button()

{

buttonPanel = new JPanel();

Font buttonFont = new Font("Serif", Font.PLAIN, 18);

buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y\_AXIS));

buttonPanel.add(Box.createHorizontalStrut(STRUT\_SIZE));

buttonPanel.setPreferredSize(new Dimension(200, 600));

buttonPanel.setBackground(new Color(lightBlue));

encodeButton = new JButton(" Encode ");

encodeButton.setFont(buttonFont);

encodeButton.setSelected(true);

encodeButton.setMnemonic('E');

encodeButton.setToolTipText("Encoding the file!");

encodeButton.setPreferredSize(new Dimension(0, 100));

decodeButton = new JButton(" Decode ");

decodeButton.setFont(buttonFont);

decodeButton.setMnemonic('d');

decodeButton.setToolTipText("Decoding the file!");

decodeButton.setPreferredSize(new Dimension(0, 100));

buttonPanel.add(encodeButton);

buttonPanel.add(Box.createHorizontalStrut(STRUT\_SIZE));

buttonPanel.add(decodeButton);

buttonPanel.add(Box.createHorizontalStrut(STRUT\_SIZE));

}

/\*\*

\* The method create a the panel for the text area, and the title of the GUI

\*/

private void textBox()

{

textPanel = new JPanel();

textPanel.setPreferredSize(new Dimension(TEXTPANEL\_WIDTH,

TEXTPANEL\_HEIGHT));

textPanel.setLayout(new FlowLayout());

titleLabel = new JLabel("Making Alice Smaller");

titleLabel.setForeground(new Color(lightBlue));

Font titlefont = new Font("Serif", Font.BOLD, 50);

titleLabel.setFont(titlefont);

bookTitle = new JLabel(" Book Title: ");

JPanel blank = new JPanel();

blank.setPreferredSize(new Dimension(600, 600));

blank.setBackground(new Color(lightBlue));

information = new JTextArea(TEXT\_HEIGHT, TEXT\_WIDTH);

information.setEditable(false);

bookTitle1 = new JTextField(30);

bookTitle1.setHorizontalAlignment(JTextField.CENTER);

bookTitle1.setEditable(false);

processLabel = new JLabel("Processing: "

+ " ");

processField = new JTextField(20);

processField.setEditable(false);

processField.setHorizontalAlignment(processField.CENTER);

blank.add(processLabel);

blank.add(processField);

textPanel.add(titleLabel);

textPanel.add(bookTitle);

textPanel.add(bookTitle1);

scrollPane = new JScrollPane(information,

JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,

JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

textPanel.add(scrollPane, BorderLayout.CENTER);

textPanel.add(blank);

}

/\*\*

\* Creating a listener for the encoding button

\*/

private void encodingButton()

{

class changeStatusListener implements ActionListener

{

/\*

\* Creating the listener for the encoding button

\*/

@Override

public void actionPerformed(ActionEvent event)

{

setTextArea("");

if (!textTile.equals(""))

coder.encode(textTile);

}

}

ActionListener listener = new changeStatusListener();

encodeButton.addActionListener(listener);

}

/\*\*

\* Creating a listener for the decoding button

\*/

private void decodingButton()

{

class changeStatusListener implements ActionListener

{

/\*

\* Creating a listener for the decoding button

\*/

@Override

public void actionPerformed(ActionEvent event)

{

if (!textTile.equals(""))

coder.decode(textTile);

}

}

ActionListener listener = new changeStatusListener();

decodeButton.addActionListener(listener);

}

/\*\*

\* Creating JMenu for the menu bar "File"

\* File menu contains "Open" and "Exit"

\* @return File menu

\*/

private JMenu fileMenu()

{

JMenu menu = new JMenu(" File ");

menu.add(openMenu());

menu.add(exitMenu());

return menu;

}

/\*\*

\* Creating a menu "Info"

\* @return the info menu

\*/

private JMenu info()

{

JMenu info = new JMenu(" Info ");

info.add(infoMenu());

return info;

}

/\*\*

\* Creating an Information menu that hold all the information about Group A

\*/

public JMenuItem infoMenu()

{

JMenuItem item = new JMenuItem(" Group Information ");

class MenuItemListener implements ActionListener

{

/\*\*

\* Showing information dialogue box

\*/

@Override

public void actionPerformed(ActionEvent event)

{

JOptionPane.showMessageDialog(null,

"The program is written by \"Group A\"\n\n"

+ "CHOE, DEUK\nENKBET, TAMIR\nJANLIE, RYOKS\n"

+ "KHORN, SEREYVATHANAK\n",

"GROUP INFORMATION", 1);

}

}

ActionListener listener = new MenuItemListener();

item.addActionListener(listener);

return item;

}

/\*\*

\* Creating a sub-menu for an open chooser that will look for alice.txt

\* @return the sub-menu "Open"

\*/

public JMenuItem openMenu()

{

JMenuItem item = new JMenuItem(" Open ");

class MenuItemListener implements ActionListener

{

/\*\*

\* Choosing a new file

\*/

@Override

public void actionPerformed(ActionEvent event)

{

openChooser = new JFileChooser("Books/");

Scanner in = null;

if (openChooser.showOpenDialog(null) ==

JFileChooser.APPROVE\_OPTION)

{

file = openChooser.getSelectedFile();

textTile = file.getAbsolutePath();

textTitleforDisplay = file.getName();

String title = textTitleforDisplay.substring(0, textTitleforDisplay.length() - 4);

title = title.toUpperCase();

bookTitle1.setText(title);

try

{

in = new Scanner(file);

while(in.hasNextLine())

{

String line = in.nextLine();

display.append(line);

display.append("\n");

}

information.setText(display.toString());

display.delete(0, display.length());

}

catch (FileNotFoundException ex)

{

JOptionPane.showMessageDialog(null,

"Either the file does not exists or readable!",

"Error", JOptionPane.INFORMATION\_MESSAGE);

}

}

}

}

ActionListener listener = new MenuItemListener();

item.addActionListener(listener);

return item;

}

/\*\*

\* Creating sub-menu for exit sub-menu

\* @return the "Exit" menu item

\*/

public JMenuItem exitMenu()

{

JMenuItem item2 = new JMenuItem(" Exit ");

class MenuItemListener implements ActionListener

{

@Override

public void actionPerformed(ActionEvent event)

{

System.exit(0);

}

}

ActionListener listener = new MenuItemListener();

item2.addActionListener(listener);

return item2;

}

/\*\*

\* Set the GUI to be unable to resize

\* @return false for no resizing

\*/

@Override

public boolean isResizable()

{

return false;

}

/\*\*

\* Giving TextArea to set String

\* @param str String that will appear in the information textArea

\*/

public void setTextArea(String str)

{

information.setText(str);

}

/\*\*

\* Append more String onto the old String in the information textArea

\* @param str String that will appear in the information textArea

\*/

public void appendTextArea(String str)

{

String oldStr = information.getText();

information.setText(oldStr + str);

}

/\*\*

\* Accessors for the information textArea

\* @return Text in information

\*/

public String getTextAreaString()

{

return information.getText();

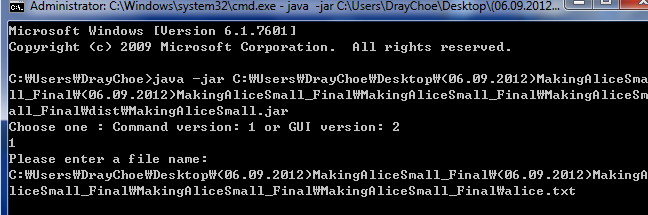
}

}

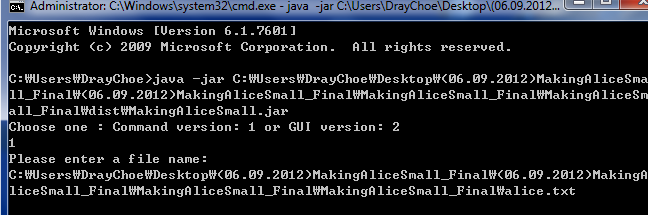
Sample Output:

1. Command line

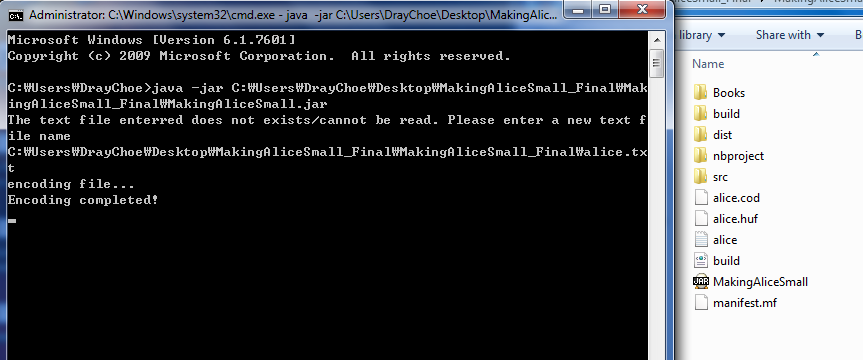
1-1. Coding mode



- Our program has two things with Command line version and GUI version, so you should choose one of them.

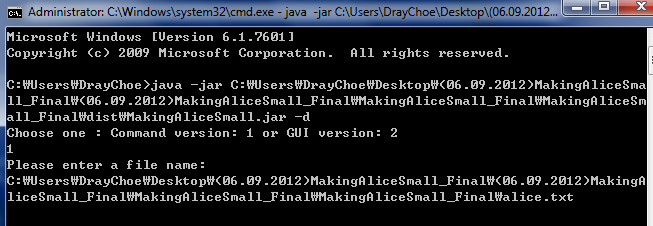


- Run our program without a path of text file. Then, GUI appears to choose a text file.



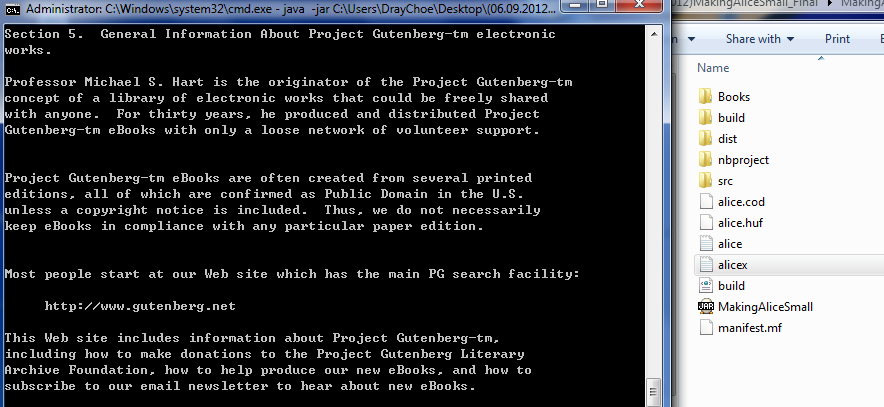
- You can check encoding completed.

1-2. Decoding mode



- Decoding mode is required with “-d”.

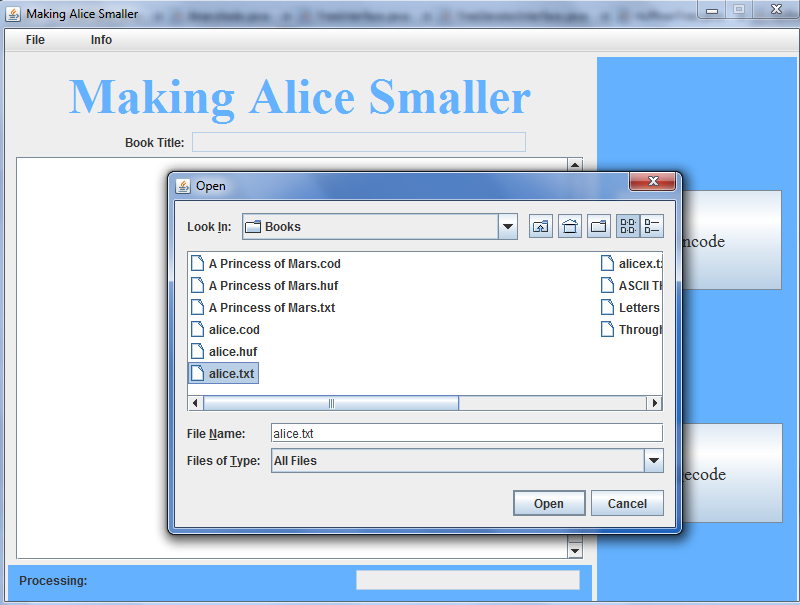
- Also, if I do not enter a file name, the program asks me to enter a file name.



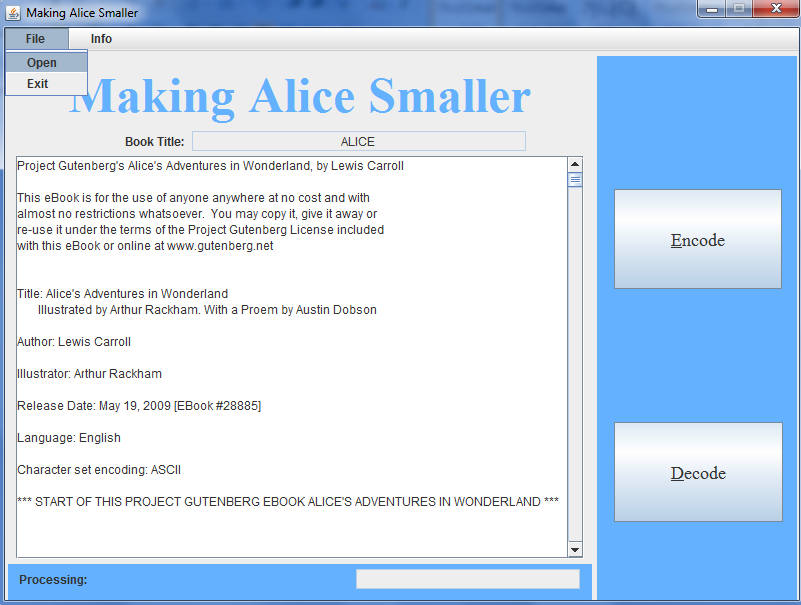
- Decoding has completed successfully!! You can see a new file, “alicex.txt”.

2. Graphic User Interface (GUI)

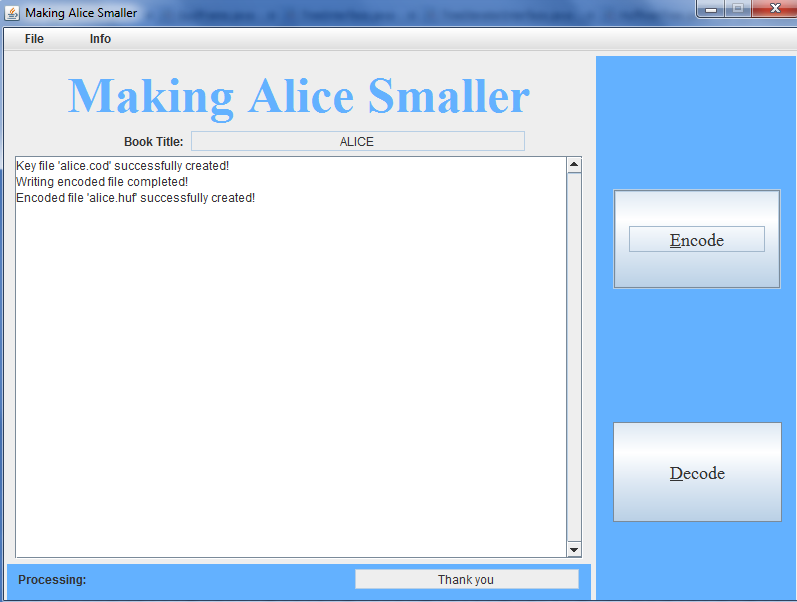
2-1. Coding mode



- Choose any file with ASCII code.

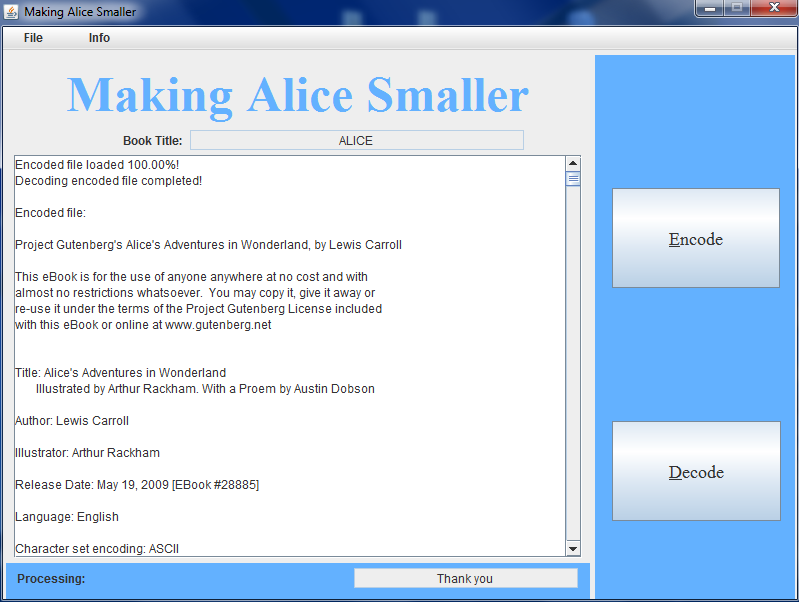


- The file you choose displays its contents.



- In GUI version, it encodes well. You can check alice.cod and alice.huf in the folder.

2-2. Decoding mode



- As you can see, decoding works successfully. And you can see the strings decoded in the display.

Test Plan:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Expected Output** | **Actual Output** | **Reason for Test** | **Reason for Difference** |
| java –jar a path of jar file | “Please enter a file name” | “Please enter a file name” | If the file name is not passed in as an argument, then the program should ask for the file name in coding mode | n/a |
| java –jar a path of jar file alice.txt | alice.cod and alice.huf | alice.cod and alice.huf | Testing for encoding | n/a |
| java –jar a path of jar file –d | “Please enter a file name” | “Please enter a file name” | If the file name is not passed in as an argument, then the program should ask for the file name in decoding mode | n/a |
| java –jar a path of jar file –d alice.txt | Alicex.txt | Alicex.txt | Testing for decoding | n/a |
| Choose A Princess of Mars file for encoding | A Princess of Mars. cod and A Princess of Mars.huf | A Princess of Mars. cod and A Princess of Mars.huf | Testing whether bigger file works or not | n/a |
| Choose A Princess of Mars file for decoding | A Princess of Marsx.txt | A Princess of Marsx.txt | Testing whether bigger file works or not | n/a |