import java.io.\*;

import java.nio.file.\*;

class InvalidCompressionFormatException extends Exception {

public InvalidCompressionFormatException(String message) {

super(message);

}

}

public class FileCompression {

public static void compressTextFile(String inputFilePath, String outputFilePath) {

try (BufferedReader reader = new BufferedReader(new FileReader(inputFilePath));

BufferedWriter writer = new BufferedWriter(new FileWriter(outputFilePath))) {

int sizeBefore = (int) new File(inputFilePath).length();

StringBuilder compressedData = new StringBuilder();

int count;

int currentChar = reader.read();

while (currentChar != -1) {

count = 1;

int nextChar;

while ((nextChar = reader.read()) == currentChar) {

count++;

}

compressedData.append((char) currentChar).append(count);

currentChar = nextChar;

}

writer.write(compressedData.toString());

int sizeAfter = compressedData.length();

System.out.println("Compression successful. Original size: " + sizeBefore + " bytes, Compressed size: " + sizeAfter + " bytes.");

} catch (IOException e) {

System.err.println("Error during compression: " + e.getMessage());

}

}

public static void decompressTextFile(String inputFilePath, String outputFilePath) {

try (BufferedReader reader = new BufferedReader(new FileReader(inputFilePath));

BufferedWriter writer = new BufferedWriter(new FileWriter(outputFilePath))) {

StringBuilder decompressedData = new StringBuilder();

int character;

while ((character = reader.read()) != -1) {

char currentChar = (char) character;

int count = reader.read() - '0';

if (count < 0 || !Character.isDigit((char) count)) {

throw new InvalidCompressionFormatException("Invalid format in compressed file.");

}

decompressedData.append(String.valueOf(currentChar).repeat(count));

}

writer.write(decompressedData.toString());

System.out.println("Decompression successful. Decompressed data written to " + outputFilePath);

} catch (InvalidCompressionFormatException e) {

System.err.println("Decompression error: " + e.getMessage());

} catch (IOException e) {

System.err.println("Error during decompression: " + e.getMessage());

}

}

public static void compressBinaryFile(String inputFilePath, String outputFilePath) {

}

public static void decompressBinaryFile(String inputFilePath, String outputFilePath) {

}

public static void main(String[] args) {

String inputFilePath = "input.txt";

String compressedFilePath = "compressed.txt";

String decompressedFilePath = "decompressed.txt";

compressTextFile(inputFilePath, compressedFilePath);

decompressTextFile(compressedFilePath, decompressedFilePath);

try {

if (Files.mismatch(Paths.get(inputFilePath), Paths.get(decompressedFilePath)) == -1) {

System.out.println("Verification successful: Original and decompressed files are identical.");

} else {

System.out.println("Verification failed: Files are not identical.");

}

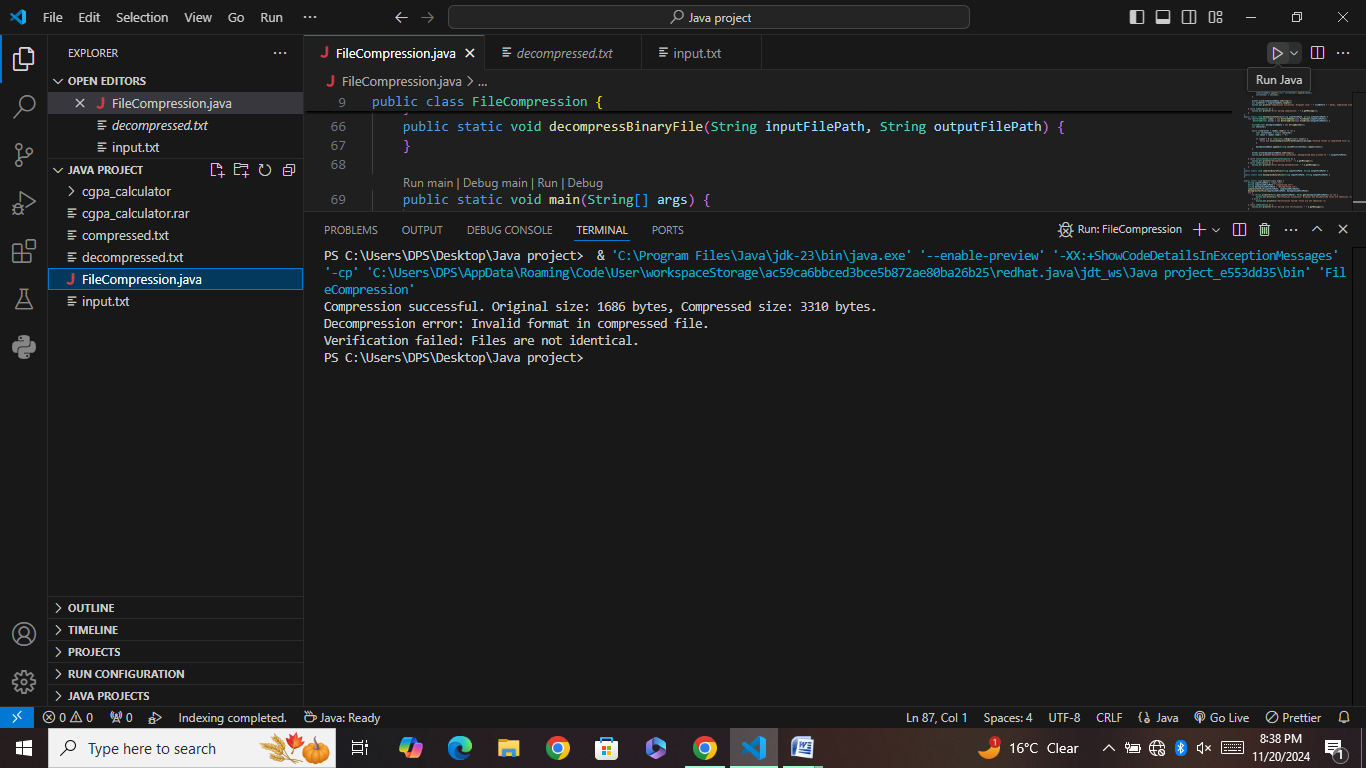
} catch (IOException e) {

System.err.println("Error during file verification: " + e.getMessage());

}

}

}



Explanation ;

This Java program implements text file compression and decompression using a simple run-length encoding (RLE) algorithm. The compressTextFile method compresses the input text file by counting consecutive identical characters and storing them as a character followed by its count. The decompressTextFile method reconstructs the original file by reading the compressed data and repeating characters based on their counts. Additionally, the program handles errors such as invalid compression format and verifies the integrity of the decompressed file by comparing it with the original. The binary compression methods are placeholders and not yet implemented.