KINGA ZMUDA - Keepit Technical Assignment Solution - Running Instructions & Details

HOW TO RUN THE PROGRAM

(Those are for Windows operating system, for UNIX-based systems the mechanism is quite the same but the exact syntax of the commands used might be a little different).

0. For the program to run, the computer has to have Java installed, as I decided to code in Java. To install Java, go to https://www.java.com/en/download/help/download_options.html

1. To successfully run the program, the Main.java file should be in the same directory as the message.txt and magazine.txt files.

2. In this directory, open the Windows Command Line and type:
javac Main.java

3. Then to to run the program on any message.txt and magazine.txt test files type:
java Main message.txt magazine.txt

The program utilizes main function arguments, so it is crucial to run it this way.

*For multiple magazines.txt as a source of letters the third step would look accordingly:
java Main message.txt magazineOne.txt magazineTwo.txt magazineThree.txt
And so on, depending on how many magazines can be a source for letters for my message. This is explained below in the **** INPUT ASSUMPTIONS **** and in the comments in my code.

INPUT ASSUMPTIONS

Inspired by the example inputs provided in the task description, I decided to assume that when putting my ransom message I do not use punctuation marks or special characters. Another assumption I made is that I do not cut out the empty spaces between letters from my magazines to put as dividers on my message, as that seems tedious and redundant to the task of glueing each letter individually on the paper. The only characters that matter for the ability to build the given message are digits and letters (I count both lower and upper case letters as the same, so i.e. "w" and "W" are treated like the same character).

I decided for my solution to accept the varying number of arguments.
This considers both the situation in which I have only one magazine to work with, as well as the situation in which I want to consider more than one magazine as a source for letters for my message, as in real life that might be the case. This comes in handy especially if I use my magazine for multiple messages and I have some letters left - I can use them in my future messages. However, it would be necessary to scan the used magazine once more to

get the updated number of letters, as my program does not make any changes to the contents of the given files.

When running, there must be provided exactly one message.txt file name and at least one magazine.txt file name;

DIFFERENT OUTPUTS - MEANING

Important note: message.txt and magazine.txt are here as the arguments of the main function (args[]), so when less than two are given to the program, it will only send a message "Please provide two filenames as command-line arguments." for the output, and the program will end.

Otherwise, when the number of arguments provided is correct, they are correct file names that can be accessed, and the program returns one of two answers:
"true" if the message can be created from the letters from the magazine.
"false" if the message cannot be created from letters from the magazine.

The program output will be sent directly to the console output.

MY TESTS AND EDGE/PATHOLOGICAL CASES

I tested the code against my prepared test cases, including some more tricky ones.
Those tests can be found in the "tests" folder. Each test consists of 3 files:
message<x>.txt
magazine<x>.txt
output<x>.txt

Where message<x>.txt and magazine<x>.txt are the input files and output<x>.txt contains the predicted program output (true/false). x is the number of the test.

Regarding edge cases, I included three tests that contain empty files:
message_empty<x>.txt
magazine_empty<x>.txt
output_empty<x>.txt

Tests for more than one magazine file are in their own folders in the tests folder.

Those test cases where one or both of the input files are empty. An empty message string can be created from anything, so I assumed that for cases where message_empty<x>.txt is empty, the answer is true. In a case where message_empty<x>.txt is not empty and the magazine_empty<x>.txt is empty, the answer is false, as there is no way to create a non-empty message when there are no letters to work with.

The way I tested my program was after compiling it as above, I sent the output of my program to the myoutput<x>.txt or myoutput_empty<x>.txt, depending on the tested case, and, using another command, compared with fc command (file compare) this output file to output<x>.txt or output_empty<x>.txt respectively.

So to run in the directory of the program on message1.txt and magazine1.txt:
javac Main.java
java Main tests/message1.txt tests/magazine1.txt

For the example test with 3 files:
javac Main.java
java Main tests/test-for-3-files-no1/message.txt tests/test-for-3-files-no1/magazine-no1.txt tests/test-for-3-files-no1/magazine-no2.txt

For the "many files" test:
javac Main.java
java Main tests/test-for-many-files/message.txt tests/test-for-many-files/magazine1.txt tests/test-for-many-files/magazine2.txt tests/test-for-many-files/magazine3.txt tests/test-for-many-files/magazine4.txt tests/test-for-many-files/magazine5.txt tests/test-for-many-files/magazine6.txt tests/test-for-many-files/magazine7.txt tests/test-for-many-files/magazine8.txt tests/test-for-many-files/magazine9.txt tests/test-for-many-files/magazine10.txt tests/test-for-many-files/magazine11.txt tests/test-for-many-files/magazine12.txt

The program handles well files with many spaces or new lines too.

If the files by given names are not found program throws a FileNotFoundException.

Pathological case handling (where input is incorrect) where explained earlier.

COMPUTATIONAL COMPLEXITY

In order to determine the complexity of my solution, I analysed both the canCreateMessage function and a helper function used inside it - countTheCharacters.
countTheCharacters returns the number of each character in the argument text in the form of a HashMap. Its time complexity for a single run depends on the size of the text. If the character is already in the HashMap, the update of its count is performed on average in constant time - O(1). The function takes every character one by one in a for loop (so the loop runs n times). That means the time complexity is O(n) where n is the number of unique characters in the given text.

For a valid input (explained above) this function will be executed at least twice.
If the number of magazine files is big enough (like really big, significantly bigger than the number of characters), it might be the dominant factor for determining the time complexity too, but it is safe to assume, that the combined time complexity for counting all the characters will be O(m) where m is number of characters in both message and (all) magazine(s), as every character needs to be looked at and counted. Space complexity for a single HashMap is O(n) where n is the number of entries. There will be at least 2 HashMaps for a valid input.

The canCreateMessage function reads the contents of a file line by line. The time complexity for reading the entire file depends on the number of lines and the length of each line, so it will be O(m) where m is the total size of a file. Additionally, the function constructs a string

from the file contents, which also takes O(m). Looking up the character count from a HashMap later is in O(1) time.

For multiple magazine files the time complexity goes like O(m + m1 + m2 + …), where each m1, m2, ... is the size of a magazine file (depending on the number of the files), and m is the size of the message file. Summing up, O(m + total size of magazine files combined). This function's space complexity is primarily defined by StringBuilder used to store the message, requiring O(m) space. Similarly for the magazine file(s) - making it O(m + total size of magazine files) combined.