Seneca College

Applied Arts & Technology
SCHOOL OF COMPUTER STUDIES

JAC444 **Submission date:** **March 08, 2023**

Workshop 5

Description:
The following workshop lets you practice basic java coding techniques, creating classes, methods, using arrays, inheritance, polymorphism, Exceptional Handling and Java I/O.

This assignment focuses on file input/output and strings. Turn in a file named `MadLibs.java` and `mymadlib.txt`.

"Mad Libs" are short stories that have blanks called *placeholders* to be filled in. In the non-computerized version of this game, one person asks a second person to fill in each of the placeholders without the second person knowing the overall story. Once all placeholders are filled in, the second person is shown the resulting story, often with humorous results.

In this assignment you present a menu to the user with three options: create a new mad lib, view a previously created mad lib, or quit. These are shown as C, V, and Q, case-insensitively. If any other command is typed, the user is re-prompted.

When creating a new mad lib, the program prompts the user for input and output file names. Then the program reads the input file, prompting the user to fill in any placeholders that are found without showing the user the rest of the story. As the user fills in each placeholder, the program is writing the resulting text to the output file. The user can later view the mad lib that was created or quit the program. The log below shows one sample execution of the program:

Console Interaction

```
Welcome to the game of Mad Libs.
I will ask you to provide various words
and phrases to fill in a story.
The result will be written to an output file.

(C)reate mad-lib, (V)iew mad-lib, (Q)uit? c
Input file name: oops.txt
File not found. Try again: mabldib1.txt
File not found. Try again: madlib1.txt
Output file name: out1.txt

Please type an adjective: silly
Please type a plural noun: apples
Please type a noun: frisbee
Please type an adjective: hungry
Please type a place: Tacoma, WA
Please type a plural noun: bees
Please type a noun: umbrella
Please type a funny noise: burp
Please type an adjective: shiny
Please type a noun: jelly donut
Please type an adjective: beautiful
Please type a plural noun: spoons
Please type a person's name: Keanu Reeves
Your mad-lib has been created!

(C)reate mad-lib, (V)iew mad-lib, (Q)uit? x
(C)reate mad-lib, (V)iew mad-lib, (Q)uit? I don't understand.
(C)reate mad-lib, (V)iew mad-lib, (Q)uit? V
Input file name: OUT001.txt
File not found. Try again: i forget the file name
File not found. Try again: something.DOC
File not found. Try again: out1.txt

One of the most silly characters in fiction is named
"Tarzan of the apples ." Tarzan was raised by a/an
frisbee and lives in the hungry jungle in the
heart of darkest Tacoma, WA . He spends most of his time
eating bees and swinging from tree to umbrella .
Whenever he gets angry, he beats on his chest and says,
" burp !" This is his war cry. Tarzan always dresses in
shiny shorts made from the skin of a/an jelly donut
and his best friend is a/an beautiful chimpanzee named
Cheetah. He is supposed to be able to speak to elephants and
spoons . In the movies, Tarzan is played by Keanu Reeves .

(C)reate mad-lib, (V)iew mad-lib, (Q)uit? Q
```

Notice that if an input file is not found, either for creating a mad-lib or viewing an existing one, the user is re-prompted. No re-prompting occurs for the output file. If it does not already exist, it is created. If it already exists, overwrite its contents. (These are the default behaviors in Java.) You may assume that the output file is not the same file as the input file.

Input Data Files:

You will need to download the example input files from BlackBoard and save them to the same folder as your program. Mad lib input files are mostly just plain text, but they may also contain placeholders. Placeholders are represented as input tokens that begin and end with < and > . For example, the file `madlib1.txt` used in the previous log contains:

Input File madlib1.txt

```
One of the most <adjective> characters in fiction is named
"Tarzan of the <plural-noun> ." Tarzan was raised by a/an
<noun> and lives in the <adjective> jungle in the
heart of darkest <place> . He spends most of his time
eating <plural-noun> and swinging from tree to <noun> .
Whenever he gets angry, he beats on his chest and says,
" <funny-noise> !" This is his war cry. Tarzan always dresses in
<adjective> shorts made from the skin of a/an <noun>
and his best friend is a/an <adjective> chimpanzee named
Cheetah. He is supposed to be able to speak to elephants and
<plural-noun> . In the movies, Tarzan is played by <person's-name> .
```

Your program should break the input into lines and into tokens using a `Scanner` so that you can look for all its placeholders. Normal non-placeholder word tokens can be written directly to the output file as-is, but placeholder tokens cause the user to be prompted. The user's response to the prompt is written to the output file, rather than the placeholder itself. You should accept whatever response the user gives, even a multi-word answer or a blank answer.

You may assume that each word/token from the input file is separated by neighboring words by a single space. In other words, when you are writing the output, you may place a single space after each token to separate them. You do not need to worry about blank spaces at the end of lines of the output file. It's okay to place a space after each line's last token.

Sometimes a placeholder has multiple words in it, separated by a hyphen (-), such as `<proper-noun>`. As your program discovers a placeholder, it should convert any such hyphens into spaces. Any hyphens that appear outside of a placeholder, such as in the other text of the story, should be retained and not converted into spaces.

When prompting the user to fill in a placeholder, give a different prompt depending on whether the placeholder begins with a vowel (a, e, i, o, or u, case-insensitively). If so, prompt for a response using "an". If not, use "a". For example:

Placeholder	Resulting Prompt
`<noun>`	`Please type a noun:`
`<adjective>`	`Please type an adjective:`
`<plural-noun>`	`Please type a plural noun:`
`<Emotional-Actor's-NAME>`	`Please type an Emotional Actor's NAME:`

Do not make unnecessary assumptions about the input. For example, an input file could have < and > characters in it that are not part of placeholders, and these should be retained and included in the output. You may assume that a placeholder token will contain at least one character between its < and > (in other words, no file will contain the token <>). You may assume that a placeholder will appear entirely on a single line; no placeholder will ever span across multiple lines.

Your output mad lib story must retain the original placement of the line breaks from the input story.

When you are viewing an existing mad lib story, you are simply reading and echoing its contents to the console. You do not need to do any kind of testing to make sure that the story came from a mad lib input file; just output the file's contents.

Your program's menu should work properly regardless of the order or number of times its commands are chosen. For example, the user should be able to run each command (such as C or V) many times if so desired. The user should also be able to run the program again later and choose the V option without first choosing the C option on that run. The user should be able to run the program and immediately quit with the Q option if so desired. And so on.

Workshop Header

/**
Workshop #
Course:*<subject type> - Semester*
Last Name:*<student last name>*
First Name:*<student first name>*
ID:*<student ID>*
Section:*<section name>*
This assignment represents my own work in accordance with Seneca Academic Policy.
Signature
Date:*<submission date>*
**/

Code Submission Criteria:

Please note that you should have:

- Appropriate indentation.
- Proper file structure
- Follow java naming convention
- Document all the classes properly using JavaDoc
- JavaDoc should be generated properly in the project
- Do Not have any debug/ useless code and/ or files in the assignment

Deliverables and Important Notes:

All these deliverables are supposed to be uploaded on the blackboard once done.

- You are supposed to create **video with voice** of your running solution. **(50%)**
  - Screen Video captured file should state your last name and id, like Ali_123456.mp4 (or whatever the extension of the file is)

    OR
  - Show your work during the lab
- A text file which will reflect on learning of your concepts in this workshop. **(20%)**
  - Should state your Full name and Id on the top of the file and save the file with your last name and id, like Ali_123456.txt
- JavaDocs must be used for proper documentation of each task. **(15%)**

- Submission of working code. **(15%)**
    - Make sure your follow the "**Code Submission Criteria"** mentioned above.
    - You should zip your whole working project to a file named after your Last Name followed by the first 3 digits of your student ID. For example, **Ali123.**zip.
- Your marks will be deducted according to what is missing from the above-mentioned submission details.
- Late submissions would result in additional 10% penalties for each day or part of it.
- Remember that you are encouraged to talk to each other, to the instructor, or to anyone else about any of the assignments, but the final solution may not be copied from any source.