Assignment 6: YazInterpreter (20 points) *due August 4, 2020, 11:59pm*

This assignment focuses on file input and output and string processing, as well as reinforcing previous concepts such as loops, conditionals, and methods. Turn in the following TWO files using the link on the course website:

- `YazInterpreter.java` – A program that interprets commands from the programming language YazLang.
- `my-command.txt` – A proposal for a new YazLang command.

Background

Note: You do not need to read this section to complete the assignment, but it provides some helpful context that may make the assignment easier to understand.

Throughout the quarter, we have been working with the programming language Java. Java is an example of a **compiled language**, meaning that before we can run our code, we need to run it through a tool called a *compiler* to translate it into a language that the computer itself can understand and execute. But not all languages work this way. Some languages are what are called **interpreted languages**, meaning that the source code in the language can be read and executed directly using a tool called an *interpreter*. The language you will work with on this assignment, YazLang, is an example of an interpreted language.

Program Behavior

In this assignment, you will create an interpreter for the programming language YazLang. (This language was named the current CSE 142 Summer Instructor, Ayaz, who led development of the language and this assignment.) When interpretting a YazLang file, the program prompts the user for input and output file names. Then the program reads and executes the YazLang commmands in the input file and outputs the results to a different file. The user can later view the output file that was created or quit the program.

Sample Execution

```
Welcome to YazInterpreter!
You may interpret a YazLang program and output
the results to a file or view a previously
interpreted YazLang program.

(I)nterpret .yzy program, (V)iew .yzy output, (Q)uit? I
Input file name: input.yzy
File not found. Try again: interpret.txt
File not found. Try again: interpret.yzy
Output file name: interpret-out.txt
YazLang interpreted and output to a file!

(I)nterpret .yzy program, (V)iew .yzy output, (Q)uit? View
(I)nterpret .yzy program, (V)iew .yzy output, (Q)uit? vi
(I)nterpret .yzy program, (V)iew .yzy output, (Q)uit? v
Input file name: interpret-out.txt

-9 -6 -3 0 3 6
39F
gucci ganggucci ganggucci ganggucci ganggucci ganggucci gang
11C
humuhumunukunukuapua'a
5 12 19 26 33
24F

(I)nterpret .yzy program, (V)iew .yzy output, (Q)uit? q
```

Menu

The 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 I 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 I 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.

Interpreting YazLang Files

Sample Input File (interpret.yzy)

```
RANGE -9 9 3
CONVERT 4 C
REPEAT "gucci_gang" 7
CONVERT 53 F
REPEAT "humu" 2 "nuku" 2 "apua'a" 1
RANGE 5 35 7
CONVERT -4 C
```

When the user enters `I` from the menu, they should then be prompted to enter an input file and an output file. The input file should contain YazLang commands. Your program should then read the input file, execute each command, and print the output to the output file. If the input file does not exist, the user should be reprompted until they enter a file that does exist. See the Sample Execution above for an example of this reprompting behavior.

No reprompting is necessary for the output file. If the output file does not exist, it should be created. If it does already exist, its contents should be overridden. (These are the default behaviors for the file input/output approaches we use.) You should assume that the input and output files are not the same.

Sample Output File (interpret-out.txt)

```
-9 -6 -3 0 3 6
39F
gucci ganggucci ganggucci ganggucci ganggucci ganggucci ganggucci gang
11C
humuhumunukunukuapua'a
5 12 19 26 33
24F
```

Viewing YazLang Files

When the user enters `V` from the menu, they should then be prompted to enter an input file to view an interpretted YazLang program. If the input file does not exist, the user should be reprompted until they enter a file that does exist. See Output Comparison Tool for examples of this reprompting behavior.

When you are viewing an interpretted YazLang program, 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 it is a YazLang interpretted file; just output the file's contents.

YazLang Commands

The syntax for YazLang is much simpler (and more limited) than Java's. Every YazLang command follows this pattern:

$$\texttt{COMMAND}\ arg_1\ arg_2\ \ldots\ arg_n$$

That is, every command consists of a single token indicating the command be executed, followed by some number of arguments. Some commands take a specific number of arguments, while others may take any number of arguments. Some commands may also take no arguments, in which case the command token itself is considered a complete command. There will be one or more spaces or tabs between the command and the arguments, and between each argument. In a YazLang program file, each command is on its own line.

The three commands will always appear exactly as CONVERT, RANGE and REPEAT (case sensitive) with the appropriate arguments

YazLang includes three commands: `CONVERT`, `RANGE`, and `REPEAT`. These commands are described in the table on the next page.

Command	Arguments	Description	Examples	Example Output

`CONVERT`	Always takes exactly two arguments:
	• arg_1: the temperature to convert, as an integer.
	• arg_2: either `C` or `F`, indicating the *current* units of the temperature.
	arg_2 will always be either `C` or `F` (case-insensitive).
		Converts a temperature from Celsius to Fahrenheit or vice versa using the following formulas:
		$F = 1.8 * C + 32$
		$C = (F - 32)/1.8$
		If the temperature is currently in Celsius (that is, arg_2 is `C`), it should be converted to Fahrenheit. If the temperature is currently in Fahrenheit, it should be converted to Celsius. The output should be given as an integer, with any decimal places truncated, and should indicate the new units.
			`CONVERT 0 C`
			`CONVERT 32 F`
			`CONVERT 9 C`
			`CONVERT 9 F`
				`32F`
				`0C`
				`48F`
				`-12C`

`RANGE`	Always takes exactly three arguments:
	• arg_1: the first number to be printed.
	• arg_2: the first number to not be printed.
	• arg_3: the amount to increment by.
	arg_3 will always be greater than zero.
		Prints a sequence of numbers starting from arg_1 and incrementing by arg_3 until a number greater than or equal to arg_2 is reached. Does not print arg_2 or any number greater than it.	`RANGE 0 5 1`
			`RANGE 1 10 9`
			`RANGE 2 1 1`
				`0 1 2 3 4`
				`1`

`REPEAT`	Takes an arbitrary number of arguments, alternating between strings and integers. The number of arguments will always be even (but might be zero). String arguments will be enclosed in quotation marks, and may contain underscores. Integer arguments will be greater than or equal to zero.	Prints out each string argument repeated the number of times indicated by the following integer argument. The string arguments should have the outer quotation marks removed and underscores replaced with spaces before printing.	`REPEAT "a" 5 "B" 2`
			`REPEAT "yo_yo" 1 "_a" 1`
			`REPEAT "a" 1 "b" 0 "c" 2`
			`REPEAT`
				`aaaaaBB`
				`yo yo a`
				`acc`

Creative Aspect (my-command.txt)

There are only three commands in YazLang at the moment, and to come up with more we have decided to crowdsource! Along with your program, submit a file called `my-command.txt` with a proposal for a new command to add to YazLang. Your proposal must include the following elements:

- The name of the command
- The arguments the command will take
- A description of what the command does
- At least one sample input and sample output

You should format your proposal like the example below. We have also posted examples on the course website. You do not need to provide an implementation for your custom command.

`repeat-proposal.txt`

```
REPEAT

REPEAT takes an arbitrary number of pairs of Strings and integers and
creates one large string with each string repeated the number of times
indicated by the following integer.

Input: REPEAT "ha" 3 "_" 1 "lol" 2
Output: "hahaha lollol"
```

Development Strategy

To be able to use the `Scanner` and `File` classes in your code, you will need to include the following lines of code at the beginning of your program (before your `public class` declaration):

```
import java.util.*;
import java.io.*;
```

Approach

Once again, this assignment will be best approached in smaller chunks. We recommend the following strategy:

(1) **Read and print the contents of a file:** This is for viewing an interpreted YazLang program (even though you haven't interpreted one yet.)

(2) **Execute commands to the console:** When working on interpreting a YazLang program, work on one command at a time, and test each one before moving on to the next. We suggest working through the commands in the order they appear in the table above. We also suggest outputting the commands to the console first to test your implementaiton of each command.

(3) **Execute commands to a file:** Modify the code to execute the YazLang commands from a file to output to another output file.

(4) **Menu/Reprompting:** Add code to allow the user to select their mode (interpret, view, quit) and to handle reprompting for missing input files.

In the final version of the program, you should *not* print out the interpretted commands to the console when the user types I in the menu.

Hints

The following suggestions and hints may help you be more successful on this assignment:

- When reading input from a file, you may need to use a mixture of line-based and token-based processing as shown in class and described in chapter 6 of the textbook.
- To check if a file exists, you should use methods from the `File` class. The textbook describes an alternate technique for dealing with missing files using `try/catch` statements, but you should **NOT** use this approach on this assignment.
- You may find the `startsWith` method of the `String` class useful for determining which type of command you are processing.
- You may also find the `replace` method of the `String` class useful for replacing occurrences of one character with another. For example, the code:

  ```
  String str = "mississippi";
  str = str.replace("s", "*");
  ```

  will result in the string `str` containing the value `"mi**i**ippi`.
- If your program is generating `InputMismatchException` errors, you are likely reading the wrong type of values from your `Scanner` (for example, using `nextInt` to read a string).
- If your program is generating `NoSuchElementException` errors, you are likely attempting to read past the end of a file or line.

Debugging Tips

You may want to initially "hard-code" the input and output filenames; in other words, you may want to just use fixed file names in your code rather than prompting the user to enter the file names. You may also want to temporarily print extra "debug" text to the console while developing your program, such as printing each command or argument as you read it. Be sure to remove this extra output before submitting your program.

It is easier to debug this problem using a smaller input file with fewer commands and arguments. The file `simple.yzy` on the course website has a short YazLang program that will be useful for testing your program at first.

Implementation Guidelines

User Input/File Input

All console input should be processed using a `Scanner` and should be read using the `nextLine` method. All file input should be processed using a `File` object and a `Scanner` as shown in class. File output should be performed using a `File` and a `PrintStream` as shown in class.

When interpretting a YazLang program, your program should break the input into lines and then into tokens using Scanner objects so that you can identify the command and look for all its arguments.

You may assume anytime a YazLang command is expected, it will be valid. Specifically, you may assume that:

- each command will be on its own line
- the first word on each line will be a valid YazLang command (`CONVERT`, `RANGE`, or `REPEAT`)
- each command will have an appropriate number of arguments
- all arguments will be of the correct type and will meet the requirements outlined above

Permitted Java Features

For this assignment, you are restricted to Java concepts covered in chapters 1 through 6 of the textbook. In particular, you **ARE NOT** allowed to use arrays on this assignment.

Style Guidelines

You should follow all guidelines in the Style Guide and on the General Style Deductions page of the course website. Pay particular attention to the following elements:

Capturing Structure

Your `main` method in this program may have more code than it has in previous assignments. In particular, you may include a limited amount of output and some control flow constructs (e.g. a loop to drive the menu) in `main`. However, your `main` method must remain a concise summary of your program's structure, and you must still utilize methods to both capture structure and eliminate redundancy.

Each method should perform a single, coherent task, and no method should do too much work. To receive full credit, your program must include a separate method to execute each type of command, plus four (4) other non-trivial method besides `main`. (Therefore, your program should have a total of at least seven (7) non-trivial methods.)

Your program *must* include a single method to process each type of YazLang command.

Using Parameters and Returns

Your program should utilize parameters and return values effectively to produce a well-structured program as described above. Your methods should not accept unnecessary or redundant parameters. In particular, your program should include only a single `Scanner` connected to `System.in`, though you may have additional `Scanner`s as well. You can (and probably should) use objects (such as `Scanner`, `File`, or `PrintStream`) as parameters and/or return values.

Code Aesthetics

Your code should be properly indented, make good use of blank lines and other whitespace, and include no lines longer than 100 characters. Your class, methods, and variables should all have meaningful and descriptive names and follow the standard Java naming conventions. (e.g. `ClassName`, `methodOrVariableName`). See the Style Guide for more information.

Commenting

Your code should include a header comment at the start of your program, following the same format described in previous assignments. Your code should also include a comment at the beginning of each method that describes that method's behavior and any parameters or return value. You should also include inline comments for any complex or confusing code to further explain what that code is doing. Comments should be written in your own words (i.e. not copied and pasted from this spec) and header comments should not include implementation details. See the Style Guide for examples.

Getting Help

If you find you are struggling with this assignment, make use of all the course resources that are available to you, such as:

- Reviewing relevant lecture examples
- Reviewing this week's section handouts
- Reading the textbook
- Visiting the IPL
- Posting a question on the message board

Academic Integrity

Remember that, while you are encouraged to use all resources at your disposal, including your classmates, **all work you submit must be entirely your own**. In particular, you should **NEVER** look at a solution to this assignment from another source (a classmate, a former student, an online repository, etc.). Please review the full policy in the syllabus for more details, and ask the course staff if you are unclear on whether or not a resource is OK to use.