A1: Command-Line Interaction

Rename Utility

Build a command-line application that can rename files.

```
Syntax:
java rename [–option argument1 argument2 ...]
```

With this syntax, `option' refers to one or more options that determine how renaming is handled. `arguments' are parameters that a particular option requires. Details are below.

Options

Options are specific arguments provided to the program on the that determine how the rename executes. Options always start with a dash ("-"), and typically require one or more additional arguments to function. The program itself requires at least two options to function: '-file' to designate the file to process, and one or more additional options to specify the rename rules.

The following options are supported:

```
  –help                      :: print out a help page and exit the program.
  –prefix [string]           :: rename [filename] so that it starts with [string].
  –suffix [string]           :: rename [filename] so that it ends with [string].
  –replace [str1] [str2]     :: rename [filename] by replacing all instances of [str1] with [str2].
  –file [filename]           :: indicate the [filename] to be modified.
```

Parameter

Most options require one or more additional parameters to operate, which immediately follow the option on the command-line. Parameters cannot start with a dash ("-"), since that symbol is used to designate options. Parameters consist of one or more of the following:

- a string, or
- @date which should be replaced by your program with the current date as a string (MM-DD-YYYY format), or
- @time which should be replaced by your program with the current time as a string (HH-MM-SS format).

In the case of options that support multiple parameters (described below), they should be separated by a space. e.g. `java rename -replace a b -file file1.txt'

The `-help' option does not require additional parameters.

The `-prefix' and `-suffix' options both support one or more parameters. If more than one parameter is provided, they should be concatenated together and treated as a single input string (e.g. `-prefix a b c' is the same as '-prefix abc').

The `-replace` option is unique, in that it must be passed exactly two arguments, representing the source and destination for the rename operation.

The `-file' option supports one or more arguments, representing multiple filenames. If multiple filenames are passed, it should be assumed that the operations are performed on all files, in order.

[Ed: if you are using a Unix shell (e.g. bash, zsh), any wildcards (* or ?) will be expanded by the shell into the full filenames and passed to your program. This means that if you handle multiple files as described above, you get wildcard support "for free"! We will not be testing wildcards, and there are no marks allocated... but it's cool].

Processing Options & Parameters

Specifying the `-help' option will cause the program to ignore all other options, display the help text, and exit.

Options are always processed in the order listed, from left to right. e.g. 'java rename -prefix a -suffix b' would process the prefix first, and the suffix second. The results of one option should be passed as input to the next option. e.g. `java rename -prefix a -replace a b -file file.txt' would rename the file to `bfile.txt', since the prefix adds `a' to the filename before the rename operation is executed.

Options can be specified in any order. i.e. `java rename -prefix a -file f1' and `java rename -file f1 -prefix a' are equally valid and represent the same operation.

The user is expected to provide a minimum number of options required to execute (-file to denote a file and at least one other option describing how the rename should be performed). If the user doesn't provide the minimum number of options, then you should display the help text and exit.

If the user supplies invalid arguments or options, you should report an appropriate error and exit e.g. "'-flyingcows' is an invalid option. Please supply valid options from the following list..."

If the options are valid but the there are missing or invalid parameters, you should provide a detailed error on how to address it and exit. e.g. `java rename -replace a -file file1.txt' might report "Invalid option: replace requires exactly 2 arguments to be specified."'

When performing a file rename, you should echo the operations as you perform them, so that the user has immediate feedback (e.g. "renaming `f1' to `af1'). Ideally, this would be a progress bar or something similar as the operation is performed; for this assignment, echoing the operation that was performed is sufficient.

Valid examples (given the existence of files named `f1' and `f2' before the program is executed):

- java rename -prefix a -file f1 —> af1
- java rename -prefix a -suffix b -file f1 —> af1b
- java rename -prefix x y _ -file f1 —> xy_f1
- java rename -prefix @date _ -file f1 —> 08-26-2019_f1
- java rename -replace f abcd -file f1 —> abcd1
- java rename -prefix a -prefix b -file f1 —> baf1
- java rename -suffix a b -file f1 —> f1ab

Invalid examples:

- java rename f1 // no options provided
- java rename -prefix a // no filename provided
- java rename -prefix a -suffix b f1 // no filename provided (`f1' interpreted as argument to suffix instead of a filename)
- java rename -prefix a -suffix b -file f3 // error if file f3 does not exist

The `-help' option should display all of the options and their required arguments. The format should resemble this:

```
(c) 2020 Jeff Avery. Last revised: Jan 2, 2020.
Usage: java rename [-option argument1 argument2 ...]

Options:
-help                   :: display this help and exit.
-prefix [string]        :: rename the file so that it starts with [string].
-suffix [string]        :: rename the file so that it ends with [string].
-replace [str1] [str2]  :: rename [filename] by replacing all instances of [str1] with [str2].
-file [filename]        :: indicate the [filename] to be modified.
```

Renaming Files

You can assume that files are always in the current directory (i.e. you do not need to support relative paths when renaming files).

If you encounter a file system error when renaming the file, you should display the appropriate error (e.g. `File not found') and exit without making any changes.

If you are renaming more than one file, and one of the file operations fail, it is acceptable to process some of the files, and report the files that didn't process [note: this is an exception to general behaviour, simply because it's difficult to predict if a rename will fail ahead of time, and I don't want you to have to worry about "rolling back" a file rename if a later rename fails].

Technical Requirements

- You must use Java 11 and IntelliJ 2019.3 or later. You are required to submit the source code and project files.
- You may use classes from the JDK, and samples provided in the public repo. You cannot use any other source code without permission from the instructor.
- Your main class should be named `rename' and reside, with a Main() method, in a file named `rename.java' [Ed: yes I'm aware that this breaks naming conventions, but we want a lower-case name on the command-line].

Submission

Your directory structure for your assignment submission should look something like this.

```
a1/
├── a1.iml
├── out
│   └── src
│       ├── rename.class
│       └── production
├── readme.md
└── src
    └── rename.java
```

Your submission needs to include:

- All source code, resources required to build your project.
- An IntelliJ project compiles everything using the specified JDK.
- A `readme.md' file with any details required to help the TA grade your assignment.

Your `readme.md' file needs to include, at a minimum, your name, student number, the version of Java that you used (from `java -version'), and your operating system. If the TA needs to know anything else to run your assignment (e.g. undocumented hotkeys), include them in this file. For example:

```
Jeff Avery
12345678 j2avery
openjdk version "11.0.2" 2019-04-16
macOS 10.14.6 (MacBook Pro 2017)
```

Assessment

Your submission will be assessed roughly as follows:

5%
Complete submission including project file and `readme'. Code compiles and runs.
20%
Correctly parse and identify options and arguments.
25%
Provide feeback on incorrect options and arguments.
20%
Chain multiple operations in the correct order.
30%
Rename files correctly using the operations above.

Versions

1.2 Jan 23, 2020. Removed example that used wildcards, since they aren't required in this assignment.

1.1 Jan 8, 2020. Fixed typo in example within section Processing Options & Arguments.

1.0 Dec 6, 2019. Initial draft.