CS5001 – Object Oriented Modelling Design and Programming

Practical 1 – Aligning Text

Due Friday 2nd October (week 3) – weighting 15%

Now that you are familiar with your programming environment from having completed the introductory exercise, you are going to write a small program in your first practical that reads in a number of paragraphs of text from a file and aligns the text with a line wrapping at a specified line length.

For this practical, you may develop your code in any IDE or editor of your choice. However, you must ensure that your source code is in a folder named **CS5001-p1-aligntext/src** and your main method is in a file called **AlignText.java** (see Practical 0 for instructions).

Requirements

- Accept two arguments from the command line:
  - First argument is the name (and path) of the file containing the text
  - Second argument is the desired length of the line for wrapping the text
  - If either argument is missing or invalid, you should output the following message:
    `usage: java AlignText file_name line_length`
- Left-align text from the file:
  - Ensure each line contains the maximum number of words possible within the specified limit
  - Ensure no word is split across lines
  - If a single word is longer than `line_length` then it may go over the limit
  - Output the left-aligned text to the standard output
- Use a third command line argument to support one or more of the additional options detailed below:
  - Left-align text from the file
    - Third command line argument = L
    - This should be the default if a third argument is not specified
  - Right-align text from the file
    - Third command line argument = R
  - Centre-align text from the file
    - Third command line argument = C
    - If there are an odd number of spaces, place the extra space at the start of the line
  - Put text in a speech bubble
    - Third command line argument = B
    - Align text to left and surround it with '|', '–' and '_' symbols
    - Add a tail at the bottom made of two '\' characters
    - Every line should be of length `line_length` if possible
    - If a word is too long for this, the whole speech bubble should be widened
    - Should precisely match the examples below, and the automated tests
  - If any argument is missing or invalid, you should output the following message:
    `usage: java AlignText file_name line_length [align_mode]`

A grade of up to 16 can be achieved for an excellent implementation of left alignment. Your grade will be improved with good implementations of any of the R, C and B options. Use of method decomposition and appropriate object-oriented design are desirable.

Implementation

You are going to read from a file in this task. Since we have not covered this yet we have provided you with a *FileUtil.java* class to deal with reading from a file. We have also provided you with two pieces of example text. The files are available at:

https://studres.cs.st-andrews.ac.uk/CS5001/Practicals/p1-aligntext/Resources/

You should save the java file to your **src** directory, i.e. in the same directory that contains your *AlignText.java* file, and you should save the text files to your **CS5001-p1-aligntext** directory.

In order to read in the file specified as the first command line argument into your Java program you should use the following line of code:

```
String[] paragraphs = FileUtil.readFile(args[0]);
```

Each String in the array called 'paragraphs' will then contain a complete paragraph of text from the file.

When running your program on files in different directories remember to use the absolute or relative path to the file. For instance, if you are running your program from `CS5001-p1-aligntext/src` but your test files are in `CS5001-p1-aligntext`, you could use the following command, specifying the relative path to "test_pratchett.txt" (assuming a column length of 80):

```
java AlignText "../test_pratchett.txt" 80
```

Note: you must include the quote marks around the file name if there are spaces in the file name/path.

The result of right-aligning the provided text with a line length of 80 characters is shown in Appendix A – Sample Alignment. You can find further examples of output by examining the tests provided (see Automated Checking below).

As with any programming practical you will probably need to look at the Java API, and in fact I will not mention this again. The API can be found at

https://docs.oracle.com/en/java/javase/11/docs/api/

For this practical you may find the documentation of the *Array* and *String* class particularly useful in showing you how to find out the *length* of an *Array* and of a *String*, how to extract substrings from a string, how to find the occurrence of certain characters in a string, split a string, remove leading and trailing white space, etc.

Automated Checking

You are provided with some basic unit tests to check your code provides the required functionality. The tests can be found at `/cs/studres/CS5001/Practicals/p1-aligntext/Tests`.
In order to run the automated checker on your program before saving it to an archive, ssh into one of the School computers running Linux, then **change directory** to your **CS5001-p1-aligntext** directory and execute the following command:

```
stacscheck /cs/studres/CS5001/Practicals/p1-aligntext/Tests
```

Similarly to the instructions in Practical 0, if the automated checker doesn't run, or the build fails, or all tests fail, you may have mis-typed a command or not have followed the instructions above.

You should open the provided test files and make sure you understand what the tests are doing and try to come up with some interesting tests of your own. You can run your own via the command line, or via the automated checker. You can create new tests in a local sub-directory in your assignment directory and run `stacscheck` with your own directory as the argument e.g.

```
stacscheck ~/CS5001-p1-aligntext/MyTests
```

When you are ready to submit, make sure that you also run the checker on the archive you are preparing to submit, by calling, for example:

```
stacscheck --archive CS5001-p1-aligntext.zip /cs/studres/CS5001/Practicals/p1-aligntext/Tests
```

You should also look at the documentation for the automated checker at:

https://studres.cs.st-andrews.ac.uk/Library/stacscheck/

Deliverables – Software (and ReadMe)

If you have attempted any enhancements beyond those mentioned in the handout, you should include a short ReadMe file in your assignment folder (CS5001-p1-aligntext). You should describe what you have done, including any instructions for compiling, running and using your program. Hand in a .zip archive of your entire assignment folder, which includes your src directory, the ReadMe file (if applicable), and any local test sub-directories, via MMS.

Marking

A very good attempt at satisfying the basic requirement above can achieve a mark of 14–16. This means you should produce very good, re-usable code demonstrating a very good decomposition of the problem into sensible methods with sensible parameters. To achieve a 17 or above, your code must in addition make a very good attempt at one or more enhancements. See the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8-hour period, or part thereof):

http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:

https://www.st-andrews.ac.uk/students/rules/academicpractice/

Appendix A – Sample Alignment

Two sample files are provided on StudRes, containing text from two great works of literature: *Interesting Times* by Terry Pratchett, and *Supercalifragilisticexpialidocious* from the musical *Mary Poppins*. You can use these files to test your work. Some sample desired outputs are shown below, and you can see more by looking at the test files

Pratchett, left-aligned with column width 80

```
Many things went on at the Unseen University and, regrettably teaching had to be
one of them. The faculty had long ago confronted this fact and had perfected
various devices for avoiding it. But this was perfectly all right because to be
fair, so had the students.
The system had worked quite well and, as happens in such cases, had taken on the
status of a tradition. Lectures clearly took place, because they were down there
on the timetable in black and white. The fact that no-one attended was an
irrelevant detail. It was occasionally maintained that this meant that the
lectures did not in fact happen at all, but no-one ever attended them to find
out if this was true. Anyway, it was argued (by the Reader in Woolly Thinking,
which is like Fuzzy Logic, only less so) that lectures had taken place in
essence, so that was all right, too.
And, therefore education at the University mostly worked by the age-old method
of putting a lot of young people in the vicinity of a lot of books and hoping
that something would pass from one to the other, while the actual young people
put themselves in the vicinity of inns and taverns for exactly the same reason.
```

Poppins, left-aligned with column width 20 (note overflow in long words)

```
It's...
supercalifragilisticexpialidocious!
Even though the
sound of it is
something quite
atrocious. If you
say it loud enough
you'll always sound
precocious.
Supercalifragilisticexpialidocious!
```

Poppins, "bubble"-aligned with column width 50 (note the exact number of leading spaces)

```
 __________________________________________________
| It's... supercalifragilisticexpialidocious!    |
| Even though the sound of it is something quite |
| atrocious. If you say it loud enough you'll    |
| always sound precocious.                       |
| Supercalifragilisticexpialidocious!            |
 --------------------------------------------------
        \
         \
```