CS112 - Spring 2024

Lab05

Instructor: Paul Haskell

INTRODUCTION

This week you will develop programs in which you create your own classes and in which you explore the capabilities of String class. The programs you write will be a bit longer and trickier than those in earlier labs. A few of the programs will be adapted for use in the big projects we do later in the semester.

Reading command line arguments

First, something simple. As we discussed in lecture, the `args` variable in

```
        static public void main(String[] args)
```

contains the "arguments" or parameters that are part of the command line when a program is called. For example,

```
        %java ReadArgs Argument1 Arg2 3 FourthArgument
```

`args[]` is a list, with entries `args[0], args[1],` … up to `args.length-1`. Your first program for this lab shall be called "**ReadArgs.java**" and it should output:

Program called with <<number of arguments>>[1] arguments:

Then, your program should print each argument, with one argument per output line.

Be sure to test this program yourself, with different numbers of arguments. The TA's and I grade your programs with test software that does exactly that: runs your program with different numbers of arguments, collects the output, and checks if the output is correct.

Be sure to test with zero, one, and more than one command-line argument!

How do we set command-line arguments in Eclipse?

- go to the Run menu and select "Run Configurations…"
- Click the "Arguments" tab
- Enter what you want into the "Program Arguments" subwindow and click "Apply". You can click 'Run' right here if you want to

[1] Recall what the angle brackets mean? Replace the angle brackets and everything inside with the correct answer

String class capabilities

The String class offers many useful functions for modifying Strings, querying the contents of Strings, comparing Strings, etc. The online documentation for String (as well as other Java classes) can be found online simply by searching for "java String" in your favorite search engine.

- The docs.oracle.com site has official documentation
- Other high-ranked search results will give useful examples and tutorials

You will need to review this material for your next program, "**FixCapitalization.java**".

Book Editing

Some very famous authors have rejected standard rules of punctuation and capitalization. For example, an excerpt from William Faulkner's *The Sound and the Fury* reads:

> told me the bone would have to be broken again and inside me it began to say Ah Ah Ah and I began to sweat what do I care I know what a broken leg is all it is it wont be anything I'll just have to stay in the house a little longer that's all and my jaw-muscles getting numb and my mouth saying Wait Wait just a minute through the sweat ah ah ah behind my teeth and Father damn that horse damn that horse…

As assistant editor for a book publisher, you have been given a manuscript in which the author followed standard rules of punctuation but capitalized aT RAnDoM. It is your job to clean up the capitalization, and you will write a program called "**FixCapitalization.java**" to do it. The input to FixCapitalization.java is a single command-line argument. That argument is a String--a long string--with multiple words and with punctuation. The output will be the same text, but with capitalization corrected. You will use simple rules of capitalization, only requiring at the output that:

the first word of every sentence be capitalized,

no other capital letters exist in the output

Here are some hints for your program:

- Words are separated by spaces and possibly punctuation
- Sentences end with '.' or '!' or '?'
- *There will be one or more spaces between sentences*

Your first step is to think and sketch out in words how the program should work. When do you capitalize? When do you get rid of capitalization?

Be sure to test your program well! In the terminal window or in Eclipse, you can send multiple words to your java program as a single command-line argument if you put all the words in double-quotes, e.g.

```
% java FixCapitalization "thIS iS A single cOMMand LINE arguMent. Yes, it is LoNG."
```

Your Own Class

You did so well using Java in your first editing assignment that your boss at the book publisher gave you another job. This time, you are editing a British author and are asked to modify a text for an American audience--what you must do is to replace all uses of the word "tea" with the word "coffee".

- Only replace whole words. For example, the word "tease" should not become "coffeese"
- You should preserve uppercase/lowercase from the original input

- You should handle punctuation properly

As before, the main `class Americanize` reads in a single command line argument with a long String. But inside the **Americanize.java** file, you will add your own class, called `WordConvert`. `class Americanize` feeds words from the input one-at-a-time to an object of type `class WordConvert`. The `WordConvert` does the actual conversion. And the `class Americanize` prints out the converted text.

Please test your program well!

ParseWords

This program might be helpful for our first big Project.

You will write a program called **ParseWords.java**. The program takes its input from the first command-line argument, i.e. `args[0]`. The input consists of words separated by colons (':'). There may be zero or one word between colons. The input may start with a word or with a colon. The input may end with a word or with a colon. There may be zero or more colons in the input. Please start by making a good list of test inputs!

Ok, what should the program do?

- it finds the words and prints them out, each on its own line
- If there are back-to-back colons in the input, the program prints the word "BLANK" on its own output line
- If the input starts with a colon or ends with a colon, the program also prints "BLANK" (for each)

Your second step should be to figure out with a paper and pencil what the output should be for each of your test cases.

Ok, now you can finally plan your Java code to solve this. Don't start programming yet! Instead, think about and write down your steps to solve the problem. Try them out in your head, or with a paper and pencil. Do they work?

Now, go ahead and code your Java program. And test it, of course!

SphereInfo

In your program **SphereInfo.java**, you will write your own class, called `class Sphere`. (There should also be a `class SphereInfo` that has your `main()` function.) The `Sphere` class should have a member variable called `diameter`. It shall have several methods:

- `setDiameter(double) // sets the diameter, of course!`
- `radius() // returns the radius`
- `diameter() // returns the diameter`
- `surfaceArea() // returns the surface area of the sphere`
- `volume() // returns the volume of the sphere`

Your `main()` method should read the diameter, a `double` value, from the first command-line argument. (You can use the `Double.parseDouble()` method to convert a `String` to a `double`.)

The `main()` method then should construct an object of type `Sphere`, and then use the object to print the following output, using `System.out.println()` :

> **A sphere of radius <<print the sphere's radius>> has surface area <<fill in proper answer>> and volume <<fill in proper answer>>**

Reminder

Put useful comments into your code. Design and organize your code so that it is easy for others to understand.

Put your programs into a subdirectory called **Lab05** inside your **MyWork** directory, and remember to push your **Lab05** to GitHub before the deadline. This assignment must be turned in before Sunday Feb 11th at 11:59pm.

Even with Eclipse, <u>you must push your code to GitHub</u> – Eclipse does not do that for you automatically.

Conclusion

In this lab you developed Java programs that are a bit more sophisticated than the earlier ones, relying on some of the design and coding experience you developed in earlier classes. These programs are a bit tricky, so feel free to contact the instructor and/or TA's with any questions.

Rubric

The first program is worth 5 points:

- One point if it has the correct name, is located in the correct directory, and compiles
- One point each if the output is correct for each of four test cases we will run

The other four programs are worth 20 points each:

- One point if it has the correct name, is located in the correct directory, and compiles.
- Three point each if the output is correct for each of the five test cases we will run.
- Zero to four points, based on the graders' judgment of the software design quality, software readability, comments, etc.