CMPSC 100-03 Lab Session 2: Code Poems

- Assigned: 9 September 2019
- Due: 16 September 2019
- Point value: 30 pts

In this laboratory session, we explore the Java Standard Library and begin to learn about creating more complex objects. In particular, this lab assesses:

- Creating an object with multiple `System.out.println` statements
- Understanding of the role and function of the `main` method in a Java `class`.
- Developing a working solution to an abstract request.

General guidelines for laboratory sessions

- **Follow steps carefully.** Laboratory sessions often get a bit more complicated than their preceeding Practical sessions. Especially for early sessions which expose you to platforms with which you may not be familiar, take notes on `commands` you run and their corresponding effects/outputs. If you find yourself stuck on a step, let a TL or the professor know! Laboratory sessions do not mean that we won't help you in the same way we do during Practicals.
- **Regularly ask and answer questions.** Some of you may have more experience with the topics we're discussing than others. We can use this knowledge to our advantage. But, like in Practicals, let students try things for a while before offering help (**always *offer* first**). To ask questions, use our Slack (https://cmpsc100fall2019.slack.com) 's `#labs` channel.
- **Store and transfer files using GitHub.** Various forms of file storage are more or less volatile. *You* are responsible for backing up and storing files. If you're unsure of files which have been changed, you can always type `git status` in the terminal for your working folder to determine what you need to back up.
- **Keep all of your files.** See above, but also remember that you're responsible for the files you create.
- **Back up your files regularly.** See above (& above-above).
- **Review the Honor Code (https://sites.allegheny.edu/about/honor-code/) reguarly when working.** If you're taking a solution from another student or the Internet at-large (*especially* Stack Overflow (https://stackoverflow.com)), bear in mind that using these solutions *can* constitute a form of plagiarism that violates the Allegheny Honor Code. While it may seem easy and convenient to use these sources, it is equally easy and convenient to rely on them and create bad habits which include not attributing credit or relying exclusively on others to solve issues. Neither are productive uses of your intellect. Really.

Further helpful reading for this assignment

If you have not already done so, I recommend reading the GitHub Guides (https://guides.github.com) which GitHub makes available. In particular, the guides:

- Mastering markdown (https://guides.github.com/features/mastering-markdown/)
- Documenting your projects on GitHub (https://guides.github.com/features/wikis/)
- GitHub Handbook (https://guides.github.com/introduction/git-handbook/)
- GitHub handouts distributed at the beginning of the lab session

As for a markdown cheatsheet, this GitHub repository (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) serves as a useful online, single-page guide.

Required deliverables

A successful submission for this lab will include a Java program which:

- Compiles
- Runs successfully
- Contains a minimum of:
    - 1 multi-line comment containing each item below on a separate line, clearly labeled:
        - The title of the poem selected
        - The author of the poem selected
        - The source from which the poem was taken
    - **At Least** 10 `System.out.println` statements which contain individual lines of the work chosen that:
        - Display the title, author, and source of work chosen, separated from the body of the poem by a blank line
        - Reproduce each line of the poem paying attention to spacing, syntax, and correct grammar using individual `System.out.println` commands
            - Keep in mind that "spacing" here means both horizontal and vertical spacing
        - Function as syntactically-correct Java statements
- A 250-word reflection included in a file named `reflection.md`, located in a directory called `writing` located in the main directory of your `committed` repository.
    - This file must contain:
        - One (1) level one header which should read "CMPSC 100-03: Lab 2"
        - Two (2) level two headers which describe the processes of finding the poem you used and your experience programming it
            - What interested in you in that particular work?
            - What challenges or difficulties did you face in reproducing it?

        - If we consider Java an "object-oriented programing" (OOP) language, how does your program model an actual poem as an object?
            - It may be productive to think about this very basic, uncomprehensive definition (https://www.poetryarchive.org/glossary/line) of the poetic "line."

Note: Some poems contain italicized or otherwise-styled language. Though most formatting is possible in text-only Java applications, it is not a required part of the submission. However, students interested in achieving this formatting can read more about formatting and non-printing characters in Java in this Stack Overflow discussion (https://stackoverflow.com/questions/30310147/how-to-print-an-string-variable-as-italicized-text) .

If you do not have a poem in mind, you can always use a poem I've selected for this assignment. If the below does not suit you, sites like Poetry Daily (https://www.poems.com) or the Poetry Foundation (https://www.poetryfoundation.org) have some great examples. Perhaps you'll find something unexpected that you like and can share with everyone you know, including your professor.

Write your code

- [ ] If you have not already cloned the repository, do so now.
- Don't forget to add your SSH key first:
    - Check if you have a key already loaded: `ssh-add -l`
    - `eval "$(ssh-agent -s)"`
    - `ssh-add ~/.ssh/id_rsa`
- Find an easily-accessible and easy to remember location to do this. It may require using commands like `cd`, `pwd`, and the `~` to find the best place for the repository.
- [ ] When you've selected the poem you're going to replicate, locate the DisplayPoem.java file.

Using GatorGrader

GatorGrader is a software utility actively developed and maintained by students in consultation with Allegheny faculty, Prof. Kapfhammer. This software allows you to check your work ***before*** turning it in, so that you know--at any point--what your grade would be on an assignment if turned in at the moment you run GatorGrader.

When you turn assignments in for this class, the action triggers a "build" in a utility called Travis (https://travis-ci.com) . One of the steps it includes is running GatorGrader against the submitted code. There aren't any hidden criteria; the grader will grade the assignment the same way, regardless of who run it because it is based on files which are contained ***in the very repository you commit***. Once a repository is cloned, grading critieria will never change.

Installation

Mac

- [ ] In a terminal window, type `docker pull gatoreducator/dockagator`.
- This will contact Docker Hub to download the GatorGrade docker `image`.
- [ ] Type `docker images`
- The newly-acquired `image` should appear in the list as `gatoreducator/dockagator`
- [ ] To run the GatorGrader, `cd` to the main directory of your repository and type:

```
docker -it run --mount type=bind,source="$(pwd)",target="/project" gatoredua
cator/dockagator /bin/bash
```

- You should now be in the GatorGrader `container`.
- [ ] Type `gradle build` to begin setting up your assignment for grading.
- [ ] When the above command completes, type `gradle grade`
- This will begin the grading process.
- [ ] When the grading process completes, call the professor over to discuss.

Windows

For those Windows users interested in using a Docker image to implement GatorGrader, please book student hours (https://cs.allegheny.edu/sites/dluman) to discuss the process with me.

Note: For all of the following steps, you will need administrator privileges to install these programs.

Install `gradle`

Using the simplest definition, Gradle is a build automation tool. In plain terms, it's a program that orchestrates complex processes so that you, the end user, don't have to do them.

The faculty have written scripts to automate the grading process--a process which can take many, many individual commands to complete. As you'll see, there are really only three things you need to do when running a `gradle` grading process instead of the many that the process may actually require. The tool makes things easy for you.

- [ ] In a `cmd` window, use `choco` to install Gradle by typing `choco install gradle -y`
- [ ] Once complete, test your installation by typing `gradle --version`
- A typical response would be similar to: `Gradle` followed by a version number

Install `python` and `ruby`

If you're not familiar with these two languages, Python and Ruby are 3rd generation languages--similar to Java. Some of you may have experience with them while others of you may never have heard of them. You do not need to know anything about either in this course except that they need to exist on your computer to run GatorGrader.

If you're so inclined, you might begin to tinker in these languages to learn a bit more about how different languages operate. Python and Ruby, while similar to each other, are different from Java.

- [ ] Use `choco` to install Python by typing `choco install python -y`
- [ ] Use `choco` to install Ruby by typing `choco install ruby -y`

To facilitate GatorGrader finding the right `python` command, we need to use `mklink` to create a "symbolic link" (a.k.a. a fake file).

- [ ] In the same `cmd` window, type:

```
mklink "c:\Python37\python3.exe" "c:\Python37\python.exe"
```

We're in the home stretch.

- [ ] Close the current `cmd` window and open another with administrative privileges
- [ ] Use `pip`, a "package manager" like `choco` to install a couple of tools:

```
pip install proselint
```

```
pip install pipenv
```

- `pip` is exclusive to Python, and installs extensions and applications for the Python language.
- [ ] Use `gem`, another "package manager" to install one tool:

```
gem install mdl
```

Grading

- [ ] `cd` to the main directory of the repository where you've been saving your code
- [ ] Type `gradle build` and press `Enter`.
- [ ] Once the build operating ends, type `gradle grade`

The GatorGrader will use pre-defined criteria included in the repository you cloned to assess the assignment. Output for a successfuly lab will look something like this:

Sample poem

```
"The Naming of Cats"
T.S. Eliot
Old Possum's Book of Practical Cats

The Naming of Cats is a difficult matter,
  It isn't just one of your holiday games;
You may think at first I'm as mad as a hatter
When I tell you, a cat must have THREE DIFFERENT NAMES.
First of all, there's the name that the family use daily,
  Such as Peter, Augustus, Alonzo, or James,
```

Such as Victor or Jonathan, George or Bill Bailey--
  All of them sensible everyday names.
There are fancier names **if** you think they sound sweeter,
  Some **for** the gentlement, some **for** the dames:
Such as Plato, Admetus, Electra, Demeter--
  But all of them sensible everyday names.
But, I tell you, a cat needs a name that's particular,
  A name that's peculiar, and more dignified,
Else how can he keep up his tail perpendicular,
  Or spread out his whiskers, or cherish his pride**?**
Of name of **this** kind, I can give you a quorum,
  Such as Munkustrap, Quaxo, or Coricopat,
Such as Bombalurina, or **else** Jellyorum--
  Names that never belong to more than one cat.
But above and beyond there's still one name left over,
  And that is the name that you will never guess;
The name that no human research can discover--
  But THE CAT HIMSELF KNOWS, and will never confess.
When you notice a cat **in** profound meditation,
  The reason, I tell you, is always the same:
His mind is engaged **in** a rapt contemplation
  Of the thought, of the thought, of the thought of his name:
    His ineffable effable
    Effanineaffable
Deep and inscrutable singlar Name.