CMPSC 100-03 Lab Session 3: The Gator Goes to the Ball

- Assigned: 16 September 2019
- Due: 23 September 2019
- Point value: 35 pts

In this laboratory session, we explore `float`, `int`, and `String` (if not more!) data types.

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 (see `gradle build`)
- [ ] Runs successfully (see `gradle run`)
- [ ] Implements `float`, or `int` data types for their correct purpose(s)
- [ ] Performs a correct calculation for the given test case and a hidden instructor test case
- [ ] Contains **no** instances of the `TODO` fragment.
- [ ] Contains **no** instances of the `{YOUR NAME HERE}` fragment.
- [ ] Contains **2** single-line comments which respond to questions asked in them.
- [ ] Uses the following variable names to track the calculations you'll do:
- `volumeGold`,`volumeSilver`,`pctGold`,`pctSilver`

This submission should also include a `reflection.md` file stored in the `writing` directory which:

- [ ] Contains **200** words
- [ ] Uses **1** list
- [ ] Employs **1** code block
- [ ] Has a structure using **3** headings
- [ ] Reponds (directly or indirectly) to these concerns:
- What type of number variable did you eventually have to choose. Why?
- Describe the difficulty of implementing the volume equation you chose. How did you overcome it? How long did it take you? What did you learn? You may add your own areas of investigation here.
- In this exercise we used command line arguments, but had to transform them. Why did we do that?
    - As an optimization, is `int` still the right type for these variables?
    - What happens if we enter decimals in our `run` command: `gradle run --args="1337.1 101.1"`?

- Did Gator Wizard (commonly now know to friends as G. Wiz) get what he wanted?

A story

The Gator Wizard recently got an invitation to a crown ball! As you can probably imagine, he's extremely excited. In fact, he already went out and bought his crown. Of course, as anyone who goes to such an event knows, the real crown jewel of the evening (not a gator pun per se, but I'm working on it) is to show off and brag about how much gold is in your brand new hat.

Now, a brief lesson on economics in the Gator Kingdom. Gold, an extremely precious metal, has been in short supply for some time. While folks can get gold items, often artisans are known for supplementing some reasonable amount of silver in objects they make (as is the case with our fair wizard's crown). So, suffice to say, having a good deal of gold in one's crown is certainly something to brag about.

When buying the crown for the event, the crown seller told the Gator Wizard that there was at least 60% gold in the crown. But, the Gator Wizard's not so convinced that the salesperson gave him the real deal. Here's what he'd like you to do. The gator wizard would like ***you*** to find out how much gold and silver are actually in the crown.

The known knowns

The only crowns allowed at this party are made of gold and silver, perhaps some trace elements, too (no more than 1%!)

Being a diligent magician well-versed in Archimedes' Principle, you've done the following things:

- [x] Weighed the crown: it comes to **1337** grams (the total weight).
- [x] Found how much water is displaced by the crown (the total volume): **100** cm^3
- [x] Looked up the density of gold and silver:
  - 20 grams per cm^3 for **gold**
  - 10 grams per cm^3 for **silver**
- [x] Consulted a few equations in your trusty reference books:
-
  - Total weight as a function of volumes
-
  - Total volume as a sum of volumes

The known unknowns

- [ ] How to implement a function to find either the volume of gold ***or*** silver.
- You don't need both! Just one.
- If you think about it, because there are only two metals involved, the volume of the other is simply the volume of whichever metal you choose, subtracted from the total

volume!

- [ ] Which data type to choose for each number: `float`, `int`, `double`? Some other that we know about?
- [ ] How to calculate the final percentages of gold and silver in the Gator Wizard's crown

Testing

For this lab, to test your program run the following command:

```
gradle run --args="1337 100"
```

You can do this either in the command line terminal (Windows) or in the GatorGrader `container`.

The above is the test case that I'll be looking for in the GatorGrader criteria. There's a hidden case that will only run when you make a `commit` to the GitHub sever. Only I can see that one.

GatorGrader

Docker `container`

If you do not already have the GatorGrader `container`, in a new terminal or Docker Quickstart Terminal, type `docker pull gatoreducator/dockagator` to download the correct `container`.

In the last lab session, we were able to get Docker `container` versions of GatorGrader working! That means that everyone can choose to use the `container` if they'd like. Here are a couple of ways to do it.

Running GatorGrader directly on `container` start

- Be sure that you are in the main directory of your practicals repository when running these commands, or you'll certainly experience issues!
- Remember that if you run `ls -la`, you should see a `.git` folder if you're in the main repository folder.
- To make sure you're in the right repository, run a `pwd` command.
    - If you recieve the expected path, you're in the right place. Else, find your way to the right location.

```
docker run -it --mount type=bind,source="$(pwd)",target="/project" --hostname Gato
rGrader gatoreducator/dockagator
```

Run `gradle` commands in the container`

```
docker run -it --mount type=bind,source="$(pwd)",target="/project" --hostname Gato
rGrader gatoreducator/dockagator /bin/bash
```

- [ ] To `build` your Java work, type `gradle build` at the `command` prompt and press the `Enter` key.
- [ ] To `grade` your Java work, type `gradle grade` at the `command` prompt and press the `Enter` key.

Using `gradle` commands directly

- [ ] In a terminal, ensure you're in the main folder of your practical repository and type `gradle build`
- [ ] After the `command` completes successfully, type `gradle grade`

`commit` your work

When you've finished your work: remember that there are three (3) steps to submitting a `git` repository: to "pack," to "label," and to "ship."

- Note: attempt all of these from the main directory of the repository.
- [ ] To "pack" the submission: `git add .`
- [ ] To "label" the submission: `git commit -m "{ADD MESSAGE DESCRIPTING PURPOSE OF COMMIT HERE}"`
- [ ] To "ship" the submission: `git push`

Attribution

This exercise owes its entire existence to this paper (https://www.math.umd.edu/~jnd/Algebraic_word_problems.pdf) by Jerome Dancis (University of Maryland, College Park).