CMPSC 100-03 Practical Session 2

In this practical session, we focus on writing our initial Java program, the "Hello, World!"

General guidelines for practical sessions

- **Experiment!** We design practical sessions to create a space for you to ***try things***. Given the expertise of our classroom TLs and my interest in fixing stuff, I am sure that even if something breaks, we can fix it.
- **Complete *something*.** Grading for practical assignments hinges on ***completion***. As long as you provide a good faith effort to finish a task, your grade should reflect your effort.
- **Practice skills.** If you work in the discipline of computer science, many of the skills you revisit or establish here are industry standard practice. Learning and practicing them often helps prepare you for either other classes or professional work.
- **Try to finish during the class session** While I provide extra time to complete the work, these assignments can be completed in 50 minutes. This will help you develop your awareness and management of time.
- **Help one another!** We're a community of users here, not competitors. If you grasp something quickly, but a neighbor does not, offer to help them after they've tried for a bit. Conversely, ***ask for help*** from either me, our lab TLs, or your neighbor.

Table of Contents

Slack

Before beginning the practical session, log into our shared Slack (https://cmpsc100Fall2019.slack.com) workspace and navigate to the `#practicals` channel. Remain in this channel for the duration of the session to accept the assignment and ask and answer questions.

GitHub

An individualized version of this assignment is available on our GitHub Classroom platform. The link for accepting your individual repository is in the `#practicals` channel of our Slack.

Installing Java

Windows

- [ ] Windows users who have installed `chocolatey` can install Java using the `choco` package manager by opening a `cmd` (`Command Prompt`) window and typing:

```
choco install jdk8
```

- Note that the install may take some time, during which you may continue with writing your Java code using the instructions below.
- [ ] Once the install finishes, type `javac -version`
- If the system responds with a line resembling `javac` follwed by a version number, the install completed successfully.

Mac

- [ ] Check to see if you already have Java installed. In a terminal window, type `javac -version`.
- If the system responds with a version number appearing similar to `java version` followed by a number, you already have Java and do not need to complete these steps. Continue with the programming instructions below.

homebrew

- [ ] In a terminal window, install `homebrew`, a package manager which facilitates downloading applications for Mac. Type:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

- [ ] Next, update the package manager to the latest version/configuration settings by typing `brew update`
- [ ] Install the `cask` extension for `homebrew` to make downloading packages just a bit easier. Type:

```
brew tap caskroom/cask
```

Java

- [ ] To install Java, type `brew cask install java` and wait for the process to complete.

General note on style

This course follows the guidance of the Java Google Style Guide (https://google.github.io/styleguide/javaguide.html) . Bookmark this page, as it will be helpful to you as you begin to write code on your own. Though many of the agreements here may seem arbitrary, the style guidelines provided will create code which satisfies two goals of writing code, that it be:

- Legible
- Understandable

While you may not yet fully grasp all of the sections and what they mean, as we learn more you will recognize and be able to interpret how to best implement new syntax.

The "Hello,World!" Java Program

Creating a program

- [ ] In a text editor (many of you downloaded Atom (https://atom.io)), create a new file and save it under the name `HelloWorld.java`
- `*.java` files represent raw Java source code that is not compiled. When writing new Java programs, create your files as `*.java` files.
- [ ] Open the file with the following code: ```java public class HelloWorld {

}

~~~
* What does this code represent? What exactly are we declaring here?
- [ ] In the space between the opening `{` and closing `}`, continue with the fol
lowing code, indented by two spaces:
```java
  public static void main(String[] args) {

  }
~~~

- This code represents the next building block of a Java program. What is it called?
- [ ] In between the opening `{` and closing `}` brackets of the `main` declaration, type the following statement, indented two additional spaces:

```
    // This is a comment. Does it display?
    System.out.println("Hello, World!");
```

Your final program should look something like this:

```
public class HelloWorld {
  public static void main(String[] args) {
    // This is a comment. Does it display?
    System.out.println("Hello, World!");
  }
```

```
}
```

Running your program

- [ ] In a terminal window, `cd` or navigate to the folder where you saved your work.
- [ ] Type `ls` to list the contents of the directory.
- Make a note of the files in the directory.
- [ ] Type `java HelloWorld.java`
- What happens? If you encounter an error, call a TL or the professor over to help figure out what is happening.
- [ ] Did you expect anything else to display? What happened to the line:

```
// This is a comment. Does it display?
```

- [ ] Type `ls` to list the contents of the directory.
- Did anything change?

This is because we used one side of the Java ***compiler*** to run our program; it merely ***interpreted*** our instructions and demonstrated that the program works (or productively didn't). In either of these cases, we have actionable information.

If your build worked (without errors)

- [ ] In your open terminal window, type `javac HelloWorld.java`.
- What happens? If you encounter an error, call a TL or the professor over to help figure out what is happening.
- [ ] Type `ls` to list the contents of the directory.
- Did anything change?

There should be a new file called `HelloWorld.class`. This is Java ***bytecode***, a "compiled" version of your program ready to run in what developers refer to as a "staging" environment.

- [ ] Turn to page 39 in your textbook and call the professor over to have a brief discussion about what we see in the diagram at the top of the page.
- [ ] You can test your binary by typing `java HelloWorld` in the terminal window.
- If you receive the expected output, congratulations! You're done developing your code for this exercise. Proceed on.
- If you encountered errors, call a TL or the professor over to discuss and figure out what is happening.

If the build "failed"

- [ ] Review the steps for creating the "Hello, World" program above and call a TL or the professor over.

`Commmit`ing your code

- [ ] Be sure you're in the repository folder and type `git add .` to stage all of the files in your repository.
- After this resolves, all new and changes files have been ***staged*** for transmission to GitHub.
- [ ] Type `git commit -m "Submitting code."` to `commit` your code with the helful message that you're submitting the code.
- [ ] Get Prof. Luman's attention before your do this step. We'll have a look at some more magic.
- [ ] To transmit, type `git push origin master`.
- [ ] When you've completed this step, message Professor Luman on our class Slack (http://cmpsc100fall2019.slack.com) .
- There's a reward in it for you: the coveted `build passing` badge!

You're done! Feel free to continue to work on your lab assignment if you haven't finished it, or get ahead with reading for Monday's class.