Lab 01 Specification – Lab Tools Review; Compilers
Due (via your git repo) no later than 8 a.m., Tuesday, 10th September 2019.
50 points

Lab Goals

- Quick review of Docker, GitHub, and git commands.
- Take a closer look at the output of the Java compiler.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at:
  `https:/www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

Learning Assignment

If you have not done so already, please read all of the relevant ”GitHub Guides”, available at https:/guides.github.com/, that explain how to use many of the features that GitHub provides. In particular, please make sure that you have read guides such as ”Mastering Markdown” and ”Documenting Your Projects on GitHub”; each of them will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- PLP chapter 01, section (1.1 - 1.3)

1. **[Docker Setup.]** At this point, I expect that you had already completed this step based on our previous class discussions. Those who had not completed this step, the documentation below should provide more details.
   - Get Docker setup completed on your laptops:
   - Docker Mac Setup:
     `https:/docs.docker.com/docker-for-mac/install/`
   - Docker Ubuntu Setup
     `https:/www.digitalocean.com/community/`
     `tutorialshow-to-install-and-use-docker-on-ubuntu-18-04`
   - Docker Windows Setup:
     `https:/docs.docker.com/docker-for-windows/install/`
   - If the setup goes correctly as desired, you should be able to get started and validate the Docker version and run the hello world docker container using the following commands:
     docker –version
     docker run hello-world
   - There are some more documentation for Docker get started to test your installation in the link provided below:

`https:/docs.docker.com/docker-for-mac/`

`https:/docs.docker.com/docker-for-windows/`

Take a look at the detailed documentation for getting started with GitHub, which is available at: `https:/www.cs.allegheny.edu/sites/amohan/resources/github.pdf`

2. **[VNC Viewer Setup.]** Take a look at the detailed documentation for downloading and installing VNC Viewer at:

   Mac:`https:/www.realvnc.com/en/connect/download/viewer/macos//`
   Windows:`https:/www.realvnc.com/en/connect/download/viewer/windows//`
   Ubuntu:`https:/www.realvnc.com/en/connect/download/viewer/linux//`

3. **[Loading Docker Container.]** There are three steps in loading the container, namely:

   - Build the container
   - Run the container
   - Connect to the container

   Build the container: So in order to build the container. the following steps should be performed.

   (a) First accept the lab url provided in the Slack. After downloading the lab folder from the GitHub classroom, navigate to the cmpsc201-fall-19-lab01 directory using termimal or Docker quick start terminal (windows users).

   (b) At this point, you should find a directory called myimages inside the parent directory. This can be checked by typing in *ls* command in terminal window. Moving forward in this document, Terminal refers to Docker Quickstart Terminal for windows users.

   (c) Navigate to the myimages directory using the **cd** command.

   (d) Provide the executable privelege to the entrypoint.sh file by running the following command:

   chmod +x entrypoint.sh

   (e) Build the docker image using the following command:

   docker build -t jbe .

   (f) Note: In the command above, jbe is the user provided container name. This could be random. But it is recommended to use the same name so as to easily follow the rest of this document. Additionally, it is required to be inside the myimages directory in order to run the build command. If you are not inside the myimages directory, you may receive an error message.

   (g) Upon successful build, it is recommended to verify the correctness of image creation by using the following command:

   docker image ls

   (h) The image named ”jbe” should be listed as one of the output from the command above.

   Run the container: So in order to create and run the container. the following steps should be performed.

   (a) Run docker container based on the image created in the previous steps using the following command:

   ```
   docker run -t -d -P -v /Users/amohan/myimages/_data:/usr/share/_data
   --rm -ti -p 5900:5900 --name jbe01 jbe
   ```

   (b) Note: In the command above, replace the directory path "`/Users/amohan/myimages/_data`" with the path of the directory in YOUR machine. It is required to physically create the _data folder inside the directory structure provided. Let us suppose you are using a windows laptop, then you may use the following command.

   ```
   docker run -t -d -P -v /c/_data:/usr/share/_data --rm -ti -p 5900:5900
   --name jbe01 jbe
   ```

In the above command, it is required to create the _data folder inside the **c** directory.

(c) In addition to creating the container, the run command above creates a mount between the host machine and the contianer for a shared folder space. So, any files placed inside the host mount directory can be easily accessible inside the container mount directory and vice versa.

(d) If you are using the lab workstations, then use the /tmp directory on the host machine side as the mount directory. Note: Other directories in the lab workstation cannot be used for mount purpose. In my most recent testing, I had received Permission Denied error message while using directory other than /tmp. So the command for creating and running the container in lab workstations is as follows:

```
docker run -t -d -P -v /tmp/_data:/usr/share/_data --rm -ti -p
5900:5900 --name jbe01 jbe
```

(e) Upon successful run, it is recommended to verify the correctness of container creation by using the following command:

docker container ls

(f) A container with its ID number should be listed as one of the output from the command above.

Connect to the container: So in order to connect to a running container, we will use the VNC viewer. The details are outlined through the following steps:

(a) Download and install VNC viewer based on the instructions provided in step 2.

(b) If you are using the lab workstation, then a tool called remmmina can be used as an alternative to VNC viewer. T

(c) After successful installation, open the VNC viewer and create a new connection.

(d) In the Server section, type in the following if you are a MAC or Linux user:

localhost:5900

If you are a windows user, type the following command in the Docker quick terminal:

docker-machine ip

(e) Copy the IP address displayed as part of the output from the above command.

(f) Next go to the vnc viewer and type down in the URL path:

IPADDRESS:5900

(g) Note: Here you need to replace the IPADDRESS with your own ip address from the docker command.

(h) At this point this should take you to a prompt for providing password. The password is **1234**.

(i) After providing the password, this should take you to the container console.

4. **[GitHub Setup.]** Take a look at the detailed documentation for getting started with GitHub, which is available at:
`https:/www.cs.allegheny.edu/sites/amohan/resources/github.pdf`

5. **[Java Experiment 1.]**

(a) Create the following Java program in a fresh directory in your repository. I will also place a copy on the lab repository.

```
public class Lab1 {
  public static void main(String[] args) {
    int i = 5, j = 6, k = 127, l = 128, m = 255,
    n = 32767, o = 32768;
    System.out.println("i="+i);
    System.out.println("j="+j);
    System.out.println("k="+k);
    System.out.println("l="+l);
    System.out.println("m="+m);
```

```
        System.out.println("n="+n);
        System.out.println("o="+o);
      }
    }
```

Add header comments to include your name and the Honor Code pledge, but do not change anything else in the file!

(b) Open the VNC Viewer and login into the container using the instructions provided above.

(c) Place a copy of the Lab1.java file inside your host machine mount directory. So that the file is accessible inside the container.

(d) **Note:** JBE requires Java version 8. Any version above Java 8 is not fully supported with the use of JBE editor. This is the reason why the container has Java 8 installation!

(e) In the white terminal screen, type the following commands:

cd /usr/share/_data

Compile and execute it as you normally do ("javac Lab1.java", "java Lab1") just to make sure it's working.

(f) Next, let us open the Java Byte Editor using the following commands:

cd /tools/jbe/bin

java ee.ioc.cs.jbe.browser.BrowserApplication

(g) You should see a window something like Figure 1. Use the "File/Open" command to open "Lab1.class" (NOT Lab1.java!), then from the menu on the left expand "Methods/main/Code" (see Figure 2).

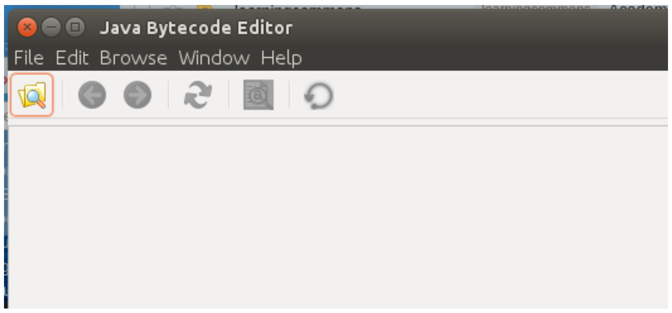


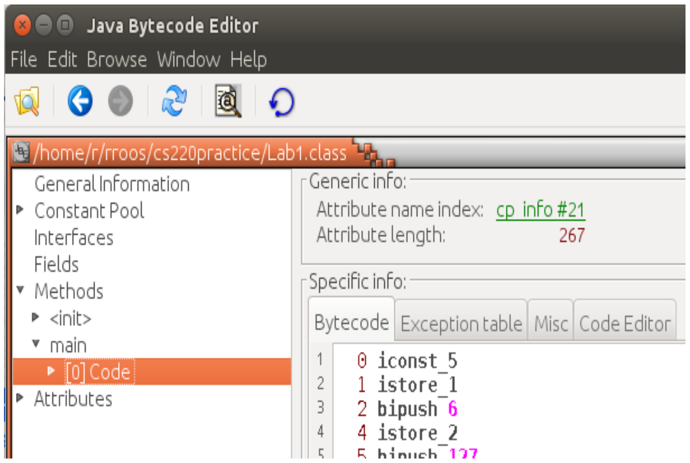
Figure 1: Result of typing jbe command

(h) Click on the "Code Editor" button and locate the line that says "iconst_5". Change it to "iconst_m1" and then click the "Save Method" button. Dont worry if you get a warning in the terminal window about JDK 1.5just ignore it.

(i) In your terminal window, run the program again. DON'T recompile it! You should see a different result for i. Congratulations! You have just edited some Java bytecode!

(j) You will notice that different constant initial values compile into different JVM commands. For instance, all values from -1 through 5 have one-word bytecode commands ("iconst_m1" through "iconst_5"), while values between 6 and 127 translate into "bipush" commands. Another change occurs between 32767 and 32768. Use the jbe code editor so that the bytecode will set variable i to -2, variable j to 5000, variable k to 65000, variable l to 3. Don't change the Java program, only change the bytecode! Run your program to see if it works.

(k) **[To Hand In.]** Commit the Lab1.java (with modified header comments) and the modified Lab1.class files to your repo. Be sure that the .class file reflects the changes you made to the Java bytecode in the previous step. Be sure you have not changed the code in the .java file (apart from the header comments).



Specific info:

Attribute name index: cp_info #21
Attribute length: 267

```
0 iconst_5
1 istore_1
2 bipush 6
4 istore_2
5 bipush 127
```

Figure 2: Getting to the bytecode

6. **[Java Experiment 2.]**
Now imitate what you just did in a brand new file, e.g,. ”Lab1Part4.java” (do not modify the file from the previous part), but with double rather than int variables and constants. Through experimentation, try to determine the different ways that double constants are assigned to double variables. You needn't try to be exhaustive, but find at least two distinct JVM commands that assign double values to double variables. What happens if you assign an int constant to a double?

   Edit the bytecode so that your double variables are assigned different values from what is in the source code (as in the previous exercise) and verify that running the program without recompiling produces the new results.

   Write up your findings in a plain text document. Be sure to put your name in the document and to write in the Honor Code pledge. Commit your Java program and your modified .class file in your repository for submitting work. Make sure you do not recompile your Java program after you have edited the bytecode.

7. **[Speculate.]**
At the bottom of your text document from the previous step, speculate as to the reasons why the Java compiler uses several different types of instructions for the assignments in the previous two questions.

8. **[Java Experiment 3.]**
Experiment with simple integer arithmetic operators (addition, multiplication, etc.). What is the Java bytecode for adding two int variables? What is the Java bytecode for multiplying two int variables? What about subtraction and division? Place your answers at the end of your text file. In your text file, paste in the (short) portions of the source code and byte code that show the operators. You do not have to submit the Java program or the class file.

9. **[Java Experiment 4.]**
Do some research on ”constant folding”. Does the Java compiler do any ”constant folding”. Write a program where constant folding could be performed, then generate the bytecode and see if the optimization occurred. Add the relevant pieces of the Java source code and the corresponding bytecode to the end of your text file, along with your answer.

10. **[Reset Docker.]**
It is important to realize that Docker takes up a lot of space on your machines. So, it is highly recommended to remove all the docker images, containers, and reset the Docker environment after finishing the lab completely.
Note: Once the Docker environment is reset then the container will be removed from your machine. So make sure to finish your lab before reseting the Docker environment.
The following commands are used to reset the Docker environment:

- Get the container id by running the following command:

  docker container ls
- Stop and remove the container by using the following commands:

  docker container stop YOURCONTAINERID

  docker container rm YOURCONTAINERID
- In the command above, replace the container id with your container id from running the first command in this section. Follow this steps to remove all the containers. If you get any error stating container does not exist, then ignore it.
- Run the following command to clear up all the stopped and dangling containers.

  docker system prune
- This may prompt you to type in ”y” in the console.
- Next step is to remove all the images. Get the image id by running the following command:

  docker image ls
- Copy the image ID and remove all the images one by one, so as to free up space on your machine.
- The following command is used to remove the image from your machine:

  docker rmi -f YOURIMAGEID
- In the command above, replace the image id with your image id. Follow this steps to remove all the images. If you get any error stating images does not exist, then ignore it.

Submission Details

1. Push your programs into the repository you shared with me.
2. Add a Reflection document (in PDF format) to the repository. List out the biggest learning points and any challenges that you have encountered during this lab.
3. **Note:** Build and run the container, should install Git automatically in the container. So it is allowed to do the git push either by installing git on your host machine or by doing the same using the container. If you are using the container, please copy the lab repo to the shared mount directory. In this way, you are able to access the folder inside the container and execute git commands.
4. Place all your comleted files inside the submission-files directory within the repo.
5. Your two programs must be uploaded to your repository no later than 8 a.m., Tuesday, Sep. 10.
6. Questions about the lab? Bring them to class on Wednesday morning!

Grading Rubric

1. Task [1 - 4] in this assignment is a preliminary requirement to complete this lab assignment. There is no explicit points awarded for completion of these tasks. It is highly recommended to take this seriously and complete it as per the instructions, so that it can set you up nicely for the entire semester.
2. If you complete Task 5 completely as per the requirement outlined above, you will receive 20 points.
3. If you complete Task 6 completely as per the requirement outlined above, you will receive 10 points.
4. If you complete Task 7 completely as per the requirement outlined above, you will receive 5 points.
5. If you complete Task 8 completely as per the requirement outlined above, you will receive 10 points.
6. If you complete Task 9 completely as per the requirement outlined above, you will receive 5 points.
7. Failure to upload the lab assignment code and text file to your git repo, will lead to no points awarded for the lab.