Lab 02 Specification – Explore Compilers
50 points

Due by: 09/13/2021 2:00 PM

Lab Goals

- Examine the output of the Java compiler and understand the Just in Time compiling process.
- Practice implementing new features in a Programming Language using SLY library.

Learning Assignment

If not done previously, it is strongly recommended to read all of the relevant "GitHub Guides", available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading assignment is useful to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, it is also recommended to do the reading assignment from the section of the course textbook outlined below:

- **PLP chapter 01, section 1.2-1.4**
- **SLY documentation: `https://sly.readthedocs.io/en/latest/`**

Assignment Details

Now that we have discussed some basics of Programming Languages and Compilers, we are ready to explore Just in Time compiling and developing new features in PL's. In this lab, we are going to program using two new tools namely, **JBE, and SLY**.

In this lab, we will practice developing a technical solution to implement Java Byte code and learning the intricacies of producing intermediate code. This will help retain the knowledge from the class discussions so far. This also includes learning how to develop a new programming language and use the SLY online library.

Students are not expected to achieve mastery in any single language in this course. Instead, we are trying to learn different languages in the context of studying the principles of programming languages. In other words, our goal is to put the principles of programming languages at the forefront and not achieving language mastery. We may use new tools and techniques which is an extension of the theoretical discussion in class. We should understand that the class time is dedicated to theoretical discussions and we may experience and explore new tools and techniques that were not directly discussed in class during the lab timings. To achieve these goals, this lab expects you to be open minded and accept new challenges and be ready to have fun and learn new techniques that are connected to the principles of programming languages.

At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL(s) to clarify if there is any confusion related to the lab and/or class materials. The Professor proofread the document more than once, if there is an error in the document, it will be much appreciated if you can communicate that to the Professor. The class will be then informed as soon as possible regarding the error in the document. Additionally, it is highly recommended that students will reach out to the Professor in advance of the lab submission with any questions. Waiting till the last minute will minimize the student's chances to get proper assistance from the Professor and the Technical Leader(s).

Students are recommended to get started with this part in the laboratory session, by discussing ideas and clarifying with the Professor and the Technical Leader(s). It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other's program(s) and technical reports is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill tests, exams, etc · · ·

Preliminary Steps

It is important that you can set up Docker and GitHub to complete the rest of the lab. Please follow the guidelines below to complete the preliminary steps.

1. **[Docker Setup.]** At this point, I expect the MAC, Linux, and Windows Pro users, to have this step completed based on our previous class discussions. For those who had not completed this step, the documentation below should provide more details regarding the download and installation setup.
   - 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/tutorials/how-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/`

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

   - Build the container
   - Connect and Run the container

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

   (a) First, accept the lab URL provided in Slack. After downloading the lab folder from the GitHub classroom, navigate to the cmpsc201-fall-21-lab02 directory using terminal (Mac/Ubuntu) or Command Prompt/Docker quick start terminal (windows).

   (b) If you have a **Windows** laptop, you are required to run this command from Powershell as an administrator. If you run the command from a regular command prompt, the entrypoints will not be created correctly and then you can't connect to the container. Build the docker image using the following command in terminal (Ubuntu/Mac) and PowerShell (Windows):

   docker build -t cs201lab02 .

   Please note, you are required to have the period in the command above.

   (c) Note: In the command above, cs201lab02 is the user-provided image name. This could be random. But it is recommended to use the same name to easily follow the rest of this document. Additionally, it is required to be inside the `cmpsc201-fall-21-lab02` directory to run the build command. If you are not inside the `cmpsc201-fall-21-lab02` directory, you may receive an error message.

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

   docker image ls

   (e) The image named "cs201lab02" should be listed as one of the outputs from the command above.

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

   (a) If you have a windows laptop, you can run the following command from the command prompt. Please note only the build command need to be run using the Powershell and the run container command need to be run using the command prompt. If you have a Mac or Ubuntu you can both build and run the container using your terminal window.

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

   Mac/Ubuntu:

   ```
   docker run -t -d -P -v $(pwd)/src:/root/src --rm -ti -p 5900:5900
   --name lab02 cs201lab02
   ```

   Windows:

   ```
   docker run -t -d -P -v "%cd%/src":/root/src --rm -ti -p 5900:5900
   --name lab02 cs201lab02
   ```

   If you get an error while running these commands, then the port may be in use. You may want to try to change the port from 5900 to 5910. [Only on the left side]. For example:

   Mac/Ubuntu:

   ```
   docker run -t -d -P -v $(pwd)/src:/root/src --rm -ti -p 5910:5900
   --name lab02 cs201lab02
   ```

   Windows:

   ```
   docker run -t -d -P -v "%cd%/src":/root/src --rm -ti -p 5910:5900
   --name lab02 cs201lab02
   ```

   Keep in mind that if you change it to 5910 then you need to use that port when connecting using VNC viewer.

   (c) To run the above command, it is required to be inside the `cmpsc201-fall-21-lab02` directory. And, please note, you will log in to the container after entering the above command.

(d) After creating the container, the run command above creates a mount between the host machine and the container with 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.

(e) **[VNC Viewer Setup.]** We will use VNC Viewer to access the container. This lab requires the use of Graphical User Interface (GUI). Unlike the previous labs, we need to connect differently to access the GUI applications inside the docker container. You are required to download and install VNC Viewer. 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//`

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

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

localhost:5900

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

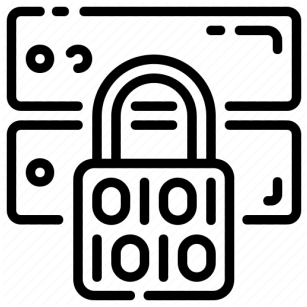
(i) After providing the password, this should take you to the container console. Congratulations, you are physically connected to the Docker container and we should be able to see the GUI at this point.

3. **[GitHub Setup.]** Take a look at the detailed documentation for getting started with GitHub, which is available at: `https://docs.github.com/en/get-started`

   You are required to know the procedure to git clone, git pull, git add, git commit, and git push to access the lab specification folder and to submit your lab for grading purposes. If there is an issue with your GitHub setup please discuss it with your Technical Leader(s) and/or the Professor.

PS next page ...

Part 01 - Java Byte Code Editor (35 points)



We will do a series of tasks in this part to implement Java Byte code and explore the JBE editor to realize the concept of Just in Time compiling. In this part, we will have five tasks to be completed and this part is worth 35 points.

- **Task 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 JBE1.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 src

  Compile and execute it as you normally do (”javac JBE1.java”, ”java JBE1”) just to make sure it’s working.

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

  chmod +x jbe.sh

  ./jbe.sh

  (g) You should see a window something like Figure 1. Use the ”File/Open” command to open ”JBE1.class” (NOT JBE1.java!), then from the menu on the left expand ”Methods/main/Code” (see Figure 2).

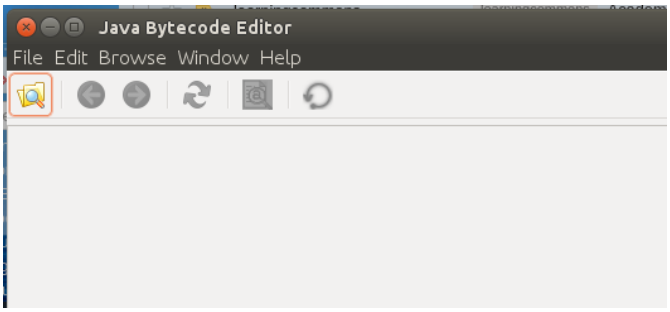


Figure 1: Result of typing jbe command

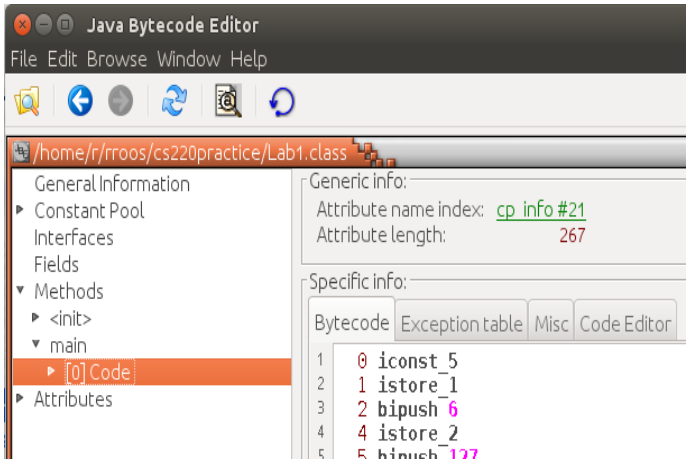


Figure 2: Getting to the bytecode

(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. Don’t worry if you get a warning in the terminal window about JDK 1.5—just 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) **Submit:** Commit the JBE1.java (with modified header comments) and the modified JBE1.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).

PS next page ...

- **Task 2:**
  Now imitate what you just did in a brand new file, named ”JBE2.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.

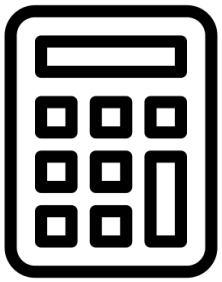
  Write up your findings in the `Technical-report` file provided in the lab repository.

- **Task 3:**
  Speculate as to the reasons why the Java compiler uses several different types of instructions for the assignments in the previous two questions. Add your speculations in the `Technical-report` file provided in the lab repository.

- **Task 4:**
  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 `Technical-report` file, include 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.

- **Task 5:**
  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 in the `Technical-report` file provided in the lab repository.

Part 02 - Calculator (15 points)



First make sure that you stop the container from the previous part and run a new container using the commands provided in Lab01. That is you are not connecting to the container using GUI and instead you are connecting using the traditional approach as listed in the lab-1 sheet. The commands are provided here for your reference:

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

Mac/Ubuntu:

```
docker run --rm -v $(pwd)/src:/root -it cs201lab01
```

Windows:

```
docker run --rm -v "%cd%/src":/root -it cs201lab01
```

Modify the code files in the `CMPSC201-calculator.py` program to implement additional operators as new features of the PL. The Calculator PL currently features 4 operators, namely, addition, subtraction, multiplication, and division. The PL provides a custom lexer and parser to define the operators and the respective mathematical precedence in the context of expressions.

You are required to add 7 new operators namely, Exponent (^), Greater than (>), Lesser than (<), Negation (!), AND (&), OR (|), and Equal (=). To complete this part, you are required to modify the CMPSC201-calculator.py file and test it with the expressions in the next page. Make sure that you also test the new operators with the existing operators such as add, subtract, multiply and divide by creating expressions on your own that combine the new and existing operators in the PL.

1. $10 > 5 \Rightarrow 1$ // 1 implies True
2. $5 > 10 \Rightarrow 0$ // 0 implies False
3. $5 < 2 \Rightarrow 0$ // 0 implies False
4. $2 < 5 \Rightarrow 1$ // 1 implies True
5. $2 = 2 \Rightarrow 1$ // 1 implies True
6. $2 = 3 \Rightarrow 0$
7. $!1 \Rightarrow 0$
8. $!0 \Rightarrow 1$
9. $1 \,\&\, 0 \Rightarrow 0$
10. $0 \,\&\, 1 \Rightarrow 0$
11. $0 \,\&\, 0 \Rightarrow 0$
12. $1 \,\&\, 1 \Rightarrow 1$
13. $1 \mid 0 \Rightarrow 1$
14. $0 \mid 1 \Rightarrow 1$
15. $0 \mid 0 \Rightarrow 0$
16. $1 \mid 1 \Rightarrow 1$

    1 implies True and 0 implies False in all the above logic expressions.

Part 03 - Honor Code

Make sure to **Sign** the following statement in the `honor-code.txt` file in your repository. To sign your name, simply replace Student Name with your name. The lab work will not be graded unless the honor code file is signed by you.

This work is mine unless otherwise cited - Student Name

Part 04 - Reflection

Add a Reflection to the repository by modifying the `reflection` file in the lab repository. List out the biggest learning points and any challenges that you have encountered during this lab.

PS next page ...

Submission Details

For this assignment, please submit the following to your GitHub repository by using the link shared to you by the Professor:

1. Modified files in `JBE1.java` and `JBE1.class`.
2. Modified files in `JBE2.java` and `JBE2.class`.
3. Modified files in `CMPSC201-calculator.py`.
4. The technical report in `Technical-report` file.
5. A document containing the reflection of the lab in the file named `Reflection`.
6. A signed honor code file, named `Honorcode`.
7. To reiterate, it is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits may be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your GitHub repository will lead to receiving no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn't compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student's responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student's submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If a student needs any clarification on their lab grade, it is strongly recommended to talk to the Professor. The lab grade may be changed if deemed appropriate.