Getting Started in Java Command Line, Cloud, and IDE

CS 5004 Object Oriented Design

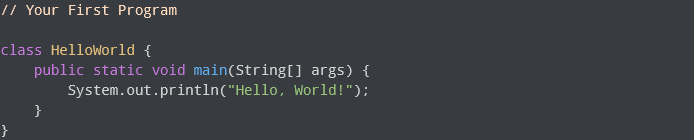
1. **Goals:** 
   * Get familiar with the three ways to compile a program
   * Learn the basics of running a program
   * Learn how to create and run JUnit tests
2. **Instructions:**

There are three ways to compile a program.

1. From the command line
2. From the cloud
3. From an Integrated Development Environment (IDE)

I would like you to be familiar with all three. Your TA will help you get setup and walk you through as much of the lab as time allows, but it will be up to you to take screenshots to show how you have accomplished all of the rubric items. You will also likely have to do some research on your own to get everything to work and in an age where we have online search engines, this is very possible. At the same time, refer back to the “30 Minute Rule” statement in the syllabus.

Here’s a Java Hello World program. You’ll be using this multiple times to experiment with the three different ways to run a Java application.



ICE Part 1: Command Line (5 pts)

Running Java code from the command line allows for remote programming and removes the separation between you and the OS from Java that an IDE does. The instructor will run through some of this in class and your TA will help you with the rest. You will need to attend the recitation section to get the demo from your TA.

For part 1, you will need to accomplish the following:

1. Install the Java JDK appropriate to your machine: <https://www.oracle.com/java/technologies/downloads/>
2. Set the System Environment Variable (Windows only)
3. Install a text editor that can handle .java files
   1. Windows: <https://notepad-plus-plus.org/downloads/>
   2. MAC: <https://www.barebones.com/products/bbedit/>
4. Create a “Hello World” Java program with your text editor
5. Use the *javac* command to translate the text file to a .class file
6. Run the application using the *java* command

Submissible for Part 1 : Take a screenshot showing you running a “Hello World” program from the command line to include in your report.

ICE Part 2: The Cloud (5 pts)

Cloud compiling is a nice way to test small amounts of code. It is quick and convenient, but requires an internet connection and is low on security. You are welcome to explore any cloud compiler out there, but you will need to be able to send links to your code. I will often use a cloud compiler in class to demonstrate code and provide you with in class challenges. The one I will be using this semester is <https://www.onlinegdb.com/>.

For part 2, do the following:

1. Using a search engine locate at least 2 cloud compilers
2. Create a “Hello World” application in each
3. Run the application

Submissible for Part 2 : Take a screenshot showing you running a “Hello World” program in two different cloud compilers to include in your report.

ICE Part 3: Integrated Development Environment (5 pts)

IDE’s provide organization, debugging tools, automatic repository connections, and other bells and whistles. For your more complex lab assignments, I would expect you to use an IDE. You are welcome to use either Eclipse or IntelliJ. In class, I will most likely use Eclipse, mostly because it is the one I’m more familiar with. However, there’s only one section at the end of the semester where you will need Eclipse. The rest of the time feel free to use either. I would encourage you to try both during the semester and discover the one you like the most. I’m here to teach you to code not to use a specific IDE.

1. Install either Eclipse or IntelliJ
   1. <https://www.eclipse.org/downloads/>
   2. <https://www.jetbrains.com/idea/download/>
2. Create a project
3. In this project create a Hello World application
4. Run this application

Submissible for Part 3 : a screenshot showing you running a “Hello World” program in either Eclipse or IntelliJ

ICE Part 4: Identifying Supporting Sources (5 pts)

In this course, there’s no required textbook. I’d like you to identify at least 4 sources you think you might like to use to supplement the course modules and lectures. Those sources can be specific textbooks (some free ones are available to you as a NEU student from O'reilly), YouTube channels, different tutorial websites, or any other sources that you think would work for you. Maybe ask your TA what they recommend.

Submissible for Part 4 : Provide those four sources as citations or links followed by why you think they would be a good resource.

**\*\*\*\* Begin non ICE Points \*\*\*\***

Part 5: Running an Existing Project and Running JUnit Tests

For Part 4, you’ll need to download the following starter files:

<https://www.dropbox.com/s/mwf3xbt9p9074ai/Book.java?dl=0>

<https://www.dropbox.com/s/w3jlufso2st6cyh/Person.java?dl=0>

<https://www.dropbox.com/s/cmetcb74lhp7t4c/PersonTest.java?dl=0>

OK, enough of Hello World. Let’s try a real program. In this course you will be making lots of class objects. It is important that we make sure those objects do what we intend them to do. In fact, you’ll be introduced to the idea of test driven development during this course. That is where we make the tests before we even write the class object. In this course, we’ll be using JUnit tests to do this.

For part 5, I want to make sure you can run JUnit tests. You don’t have to understand everything that’s going on yet. Just make sure you can set up your project to run JUnit tests. We will get into more detail on what’s actually going on later.

1. Create a project in your IDE of choice (For this, you are going to want to use the one your TA is using to demonstrate with.)
2. Add Book.java and Person.java to your project
3. Make sure there are no compilation errors
4. Study the code to make sure understand as much as you can
5. Add the PersonTest.java to your project
6. Study the code to make sure understand as much as you can
7. Verify that all the PersonTests test cases pass. Make any needed corrections.
8. View the level of coverage PersonTests currently offers.
9. Create a JUnit test class for the Book class.
10. Design and write tests that verify that all the public methods of the Book class work as expected.

*Your grader will grade this section by checking for coverage. Make sure you know how to check for coverage yourself. If you missed recitation, check with your cohort TA.*

Part 6: Documenting Your Code

Documentation is very important to your project. A code file is not complete until every file is well documented and contains a header block. For comments you want to include as formal documentation, you will use JavaDoc style comments and for more informal single line comments, you should use regular comments. You need to also include a comment block at the top of every file you submit.

1. Include a comment block at the top of each file in non-JavaDoc style. Example:

/\*\*\*\*\*\*

Name:

Assignment:

Date:

Notes:

\*\*\*\*\*\*/

1. Each class and function should have JavaDoc style comments that can be converted using Java’s built in javadoc command to formal documentation. We’ll be viewing a lot of these online. Person and Book already have these comments, but add a JavaDoc style comment to your BookTest class.
2. Run your IDEs JavaDoc creator to generate JavaDocs then view them in your project folder. (This will likely be the only time I ask you to submit JavaDocs with your assignment.)

Submissible 4 & 5 : Zip up your project folder including the JavaDocs and submit on Canvas

1. **Extensions:**

Outside academics you will not get specific requirements. Each lab assignment is worth 100 points, but the base requirements will only get you to 85% - 90%. If you want an A, you’ll have to find a way to go above and beyond what is asked. I’ll often make some suggestions to you in this section, but it is entirely up to you what you’d like to add to the assignment. Make sure you know who your grader is and discuss extension expectations with them. You won’t have to do all of the extensions to get 100% credit.

Extension suggestions:

* Create your own class object and test class
* Provide screenshots of both Eclipse and IntelliJ
* Have a high quality submission
* Work ahead and create a driver for this application
* Other contribution that’s above and beyond what was asked
* Submit your code as a GitHub link instead

1. **Report:**

Each assignment must include a short report. The generation of this report should take you no more than 15 minutes. This gives you a chance to reflect back on what you learned and it makes grading easier on your grader. For this report, I want the following sections:

1. Reflection (*What did you learn?)*
2. Screenshots (*Requested screenshots labeled and logically organized.)*
3. Extensions (*What extensions are you requesting?)*
4. Grading Statement (*Based on the rubric, what grade do you feel you deserve? Be honest and use the rubric.)*
5. **Submission:**

Please read carefully. Failure to follow submission instructions can result in a reduced score.

Submit all files on Canvas under the appropriate assignment. Make sure to include the following named as follows:

“Your name”\_”Your TA”\_projectFolder.zip - your project folder including all files and generated JavaDocs

“Your name”\_report.pdf - a pdf of your final report

Submission checklist:

* Did you include adequate comments?
* Did you include comment blocks at the top of each file?
* Did you name your files as requested?
* Does your code compile?
* Did you remove any package lines generated by your IDE?
* Did you take care of any warnings presented by your IDE?

**Rubric:**

|  |  |  |
| --- | --- | --- |
|  | **Possible** | **Given** |
| Parts 1 - 4 are ICE Points | | |
| Part 5: | | |
| PersonTests Coverage 100% | 20 | 0 |
| BookTest Coverage 100% | 25 | 0 |
| Part 6: | | |
| Comment blocks correct | 10 | 0 |
| JavaDocs generated | 10 | 0 |
| JavaDoc Line added | 5 | 0 |
| Misc | | |
| Report | 10 | 0 |
| Code Quality (correct indentation, comment blocks, variable naming, etc) | 10 | 0 |
| Not included in total possible: | | |
| Does not compile | -100 | 0 |
| Extensions (Not calculated without report) | 15 | 0 |
| Late penalty | -20 | 0 |
| Creative or went above and beyond | 10 | 0 |
| Code contains warnings | -20 | 0 |
|  | |  |
| TOTAL POINTS POSSIBLE out of 100 | 90 | 0 |