CS5010, Fall 2019

Lab1: Setting Up and Getting Started

Therapon Skoteiniotis[1] Tamara Bonaci and Adrienne Slaughter
skotthe@ccs.neu.edu, t.bonaci@northeastern.edu, a.slaughter@northeasteren.edu

1. Summary

In today's lab, we will:

- Configure our machines to use Java and IntelliJ
- Configure our development environment to use a proper code style
- Configure our development environment to use Gradle software management tool
- Walk through a simple example to create a class in Java
- Practice designing simple classes
- Practice writing and running unit tests for classes and methods that we write
- Practice writing documentation, and generating Javadoc
- Practice using Bottlenose to submit our code

Note 1: Labs are intended to help you get started, and give you some practice while the course staff is present, and able to provide assistance. You are not required to finish all the questions during the lab, but you are expected to push your lab work to your designated repo on the Khoury GitHub. Please push package named Lab1 to your individual repo, and once you are done working on the lab, tag it with **Lab1-final**.

The deadline to push your lab work is by **11:59pm Friday, September 13, 2019.**

2. Setup

2.1. Java Tools

Download and install the Java Development Kit (JDK) version 8 (https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html).

1. Go to the section with the title **"Java SE Development Kit 8u221"**

[1] Assignment modified from the original version prepared by Dr. Therapon Skoteiniotis.

2. Click on the radio button with the title **"Accept License Agreement"**
3. Click on the appropriate link based on the operating system that you have on your machine
    - For Windows select Windows x64
    - For Macs select Mac OS X
    - For Linux select the appropriate item, typically Linux x64
4. Allow the download to complete
5. Install the newly downloaded application using the process that you typically use for your operating system to install new software

Note 2: You are welcome to use a different version of the JDK, but if you do, we will not give you any extra time, or amend your grade due to any possible issues that you might experience with that version of the JDK.

2.2. IntelliJ

The instructors will be using IntelliJ in class and labs. You do not have to install IntelliJ. We will try and help you regardless of your IDE choice, but be prepared to do extra work on your own if you select an IDE other than the one used in class.

- IntelliJ (https://www.jetbrains.com/idea/).
    1. As a student you, get the full version of the IDE for free (https://www.jetbrains.com/student/). Apply for the student discount first.
    2. Download IntelliJ Ultimate Edition (https://www.jetbrains.com/idea/download/#section=mac)
    3. Install IntelliJ on your machine using the process that you typically use for your operating system to install new software

Note 3: If you decide to use a different IDE than IntelliJ, we will not give you extra time or amend your grade due to issue that you might face with your IDE's configuration.

2.3. Testing Your Setup

1. Clone your own git repository from the CS5010 GitHub organization to a folder on your computer. CS5010 GitHub organization can be access here: https://github.ccs.neu.edu/cs5010seaF19
2. Open IntelliJ

3. From IntelliJ's main menu select `New → Project`. The New Project Dialog window should appear with two tabs
    - a list of project types on the left
    - a dropdown with the title **"Project SDK"** on the right
    - a list of additional libraries also on the right
4. Select `Java` from the list of project types on the left
5. Using the dropdown on the right select `1.8` as your projects JDK version
6. Click `Next`. The contents of the window will change
7. Select **nothing** in this window and click `Next`. The contents of the window will change
8. You should see two fields
    - **"Project Name"**, provide a name for your project. This name can be anything you like, but today, you probably want it to be Lab1.
    - **"Project Location"**, this is the location on your computer that IntelliJ is going to use to create and store all your code. There should be a button to the far right of this field whose title is three dots, i.e., `...`. Click this button and navigate to the folder created due to the git clone operation in step 1. For the first lab it should be `lab1-<login>`where `<login>` is your Khoury Github login name.
9. Click `Finish`
10. IntelliJ will now open in a new window that will contain two tabs
    - A left tab that is your Project Explorer. This is similar to your file explorer
    - A right tab, the Editor, that will be used to show the contents of files that you select in the Project Explorer.
11. In the Project Explorer you should see a folder with the name you gave to your project. Double click on the folder's name to expand it.
12. You should see a sub-folder named `src`
13. Right click on the folder named `src` to open the context menu. From the context menu select `New → Class` to create a new Java Class.
14. A small window will pop-up asking you to provide a name for your class. Input the name `Author` and click `OK`
15. IntelliJ will detect that you are inside a git repository and will ask you if you would like to add the file to your repo. Please click `No`
16. The right tab of your IntelliJ window should now contain a minimal Java class with
    - a Java comment as the first line
    - an empty Java class definition
17. Notice that in the Project Explorer tab there is now a new file named `Author` under the folder named `src`
18. To test that your setup is working as expected, replace all the contents of the file `Author.java` with the following code

Author.java

```java
/**
 * Represents an Author with their details--name, email and physical address
 *
 * @author therapon
 *
 */

public class Author {

  private String name;
  private String email;
  private String address;

  /**
   * Creates a new author given the author's name, email and address as strings.
   *
   * @param name the author's name
   * @param email the author's email address
   * @param address the authors physical address
   */
  public Author(String name, String email, String address) {
    this.name = name;
    this.email = email;
    this.address = address;
  }

  /**
   * @return the name
   */
  public String getName() {
    return this.name;
  }

  /**
   * @return the email
   */
  public String getEmail() {
    return this.email;
  }

  /**
   * @return the address
   */
  public String getAddress() {
    return this.address;
  }
}
```

19. IntelliJ tries to assist you while you code by popping up an image of a small light bulb inside your editor (the right tab). Move your cursor to be inside the name of class, i.e., the word `Author` in the class header `public class Author`. Wait for a couple of seconds and the yellow light bulb image should appear at the start of that line. You can force the IntelliJ assistant to open using the following keystrokes
    - Windows/Linux : `Alt` + `Enter`
    - Mac : `Option` + `Enter`
20. Click on the yellow light bulb icon and a menu should appear.

2. Code Style

We will be using the Google Codestyle guide (https://google.github.io/styleguide/javaguide.html). From now on, all code that you write for this class **must** follow this Twitter Java Style guide.

2.1. Configuring your IDE

Thankfully, there are saved IDE configurations that enforce **most,** but not **all** coding rules specified in the Twitter Java Style guide.

1. Download the saved configuration file for IntelliJ intellij-java-google-style.xml
2. Open `Preferences`.
    a. On Windows window to `File → Settings` on the main menu.
    b. On Mac go to `IntelliJ → Preferences` on the main menu.
3. Navigate to `Editor → Code Style`. Select `Code Style` but **do not expand the item**. In the right-hand side pane at the top you will see a button with the title `Manage`. Click it to open the `Code Style Schemes` option window.
4. In the `Code Style Schemes` option window click the `Import` button, that will open the `Import Form Selection` window.
5. In the `Import Form Selection` window select `IntelliJIDEA code style XML` and a file explorer window will pop up for you to select the file you wish to import.
6. Select the file you just downloaded named `intellij-java-google-style.xml`. This is the file from step 1.

IntelliJ will **try** and format your code while you type. However, when editing a file at different locations in the file IntelliJ might not indent properly.

To force IntelliJ to re-indent all your code select `Code → Reformat Code` from the main IntelliJ menu. Read the IntelliJ documentation on code formatting for more options (https://www.jetbrains.com/help/idea/2016.3/reformatting-source-code.html).

For Eclipse users these instructions might work.

2.2. IntelliJ Tips - Indentation

You can configure IntelliJ to use 2 spaces instead of 4 for each indentation level.

Go to `Preferences` . Navigate to `Editor → Code Style`. Make sure the following items are set to these numbers

- **Tab size** : 2
- **Indent** : 2
- **Continuation Indent** : 4

3. Unit Testing with Java

In section 2.3, Testing Your Setup, we considered class `Author`.
In this section, we want to test this class by writing unit tests for all of the methods in that class. Here are some steps to help you get started with that.

1. IntelliJ tries to assist you while you code by popping up an image of a small light bulb inside your editor (the right tab). Move your cursor to be inside the name of class, i.e., the word `Author` in the class header `public class Author` . Wait for a couple of seconds and the yellow light bulb image should appear at the start of that line. You can force the IntelliJ assistant to open using the following keystrokes
    a. Windows/Linux : `Alt` + `Enter`
    b. Mac : `Option` + `Enter`
2. Click on the yellow light bulb icon and a menu should appear.
3. From the menu select `Create Test`. This action will cause a new pop-up window to appear.
4. In the new pop-up window, we will configure and create a test for our `Author` class. Starting from the top of the window going down
5. For **"Testing library"** select `JUnit 4`
6. Click the `Fix` button to add JUnit4 to your project. A new pop-up window will appear.
7. Select the first option with the title **"Use JUnit4 from IntelliJ IDEA distribution"**
8. Click `OK`. This will close this pop-up window and takes us back to continue setting up our test for `Author`.
9. Back in the pop-up for setting up our test for `Author` leave **"Superclass"** empty.
10. Leave **"Destination Package"** empty
11. Select the check mark with the title **"setUp/@Before"** only.
12. At the bottom of this pop-up window you should see the list of methods that are available in class `Author` for testing. Select **all** of them.
13. Click `OK`

14. Select JUnit Test Case from the pop-up menu. This will create a new Java class (and a new file) called AuthorTest. If IntelliJ asks you to add this new file to your repo click `No`.
15. The Package Explorer should now have a new file with the name AuthorTest under the folder src
16. The editor should now display

AuthorTest.java

```
public class AuthorTest {
  @org.junit.Before
  public void setUp() throws Exception {

  }

  @org.junit.Test
  public void getName() throws Exception {

  }

  @org.junit.Test
  public void getEmail() throws Exception {

  }

  @org.junit.Test
  public void getAddress() throws Exception {

  }

}
```

3.1. Adding a test for `getName()`

Let's add a test for the method `getName()`.

1. We first create an example of an author. Since we are probably going to use this example in more than one test, we are going to create
    1. a field in the class `AuthorTest` to store the example
    2. create an instance of author and set it to our field inside our `setUp()` method
2. We then use our example inside a test method in order to call methods defined in the `Author` class on our instance of `Author` and verify using JUnit's `Assert.assertEquals` that we get the expected values back. Here is the resulting code

AuthorTest.java

```
import org.junit.Assert;
```

```
public class AuthorTest {

    private Author jane;

    @org.junit.Before
    public void setUp() throws Exception {

        this.jane = new Author("Jane Doe", "j@a.com", "222 Main
St, Seattle, WA, 98980");
    }

    @org.junit.Test
    public void getName() throws Exception {

        Assert.assertEquals("Jane Doe", this.jane.getName());
    }

    @org.junit.Test
    public void getEmail() throws Exception {
        TestCase.fail("Not yet implemented");
    }

    @org.junit.Test
    public void getAddress() throws Exception {
        TestCase.fail("Not yet implemented");
    }

}
```

Practice Exercise

1. Complete the remaining tests in `AuthorTest`
2. Draw the UML class diagram for `Author`

4. Javadoc

For all the code that you have written in the lab up to this point, you should have also written documentation .

If you have not, go back and add documentation now!

Let's generate the HTML version of your code's documentation.

1. Select your project in the Package Explorer tab.

2. On IntelliJ's main menu select `Tools → Generate Javadoc`. A new pop-up window will appear
3. In the new pop-up window
    a. Select **"Whole Project"**
    b. Unselect **"Include test sources"**
    c. Find the line that starts with the text **"Output Directory"**. At the end of that line click the `…` and specify the output folder that you would like IntelliJ to place all the HTML files that will be generated.
    d. Click `OK`. IntelliJ will run Javadoc and show its progress along with any warnings or errors in the bottom tab of the main IntelliJ window. If the Javadoc generation had no errors then IntelliJ will open your browser and point it to the newly generated documentation.

5. Bottlenose

Sign up on https://handins.ccs.neu.edu/ for an account on Bottlenose to submit your assignments.

After you create a profile, ask to enroll in the CS 5010 PPD (Seattle) Course. When your registration is accepted, you'll see the assignment to submit to.

You will need to package all the code up into a .zip file to submit. Please be sure to only package up your code for Lab 1 (or Assignment 1, Assignment 2, etc.), not your entire Github repo.

6. Resources

- IntelliJ User Interface
- IntelliJ Editor
- IntelliJ Testing
- JUnit Assert Class Documentation