完成

截止 5月13日 由 23:00 编辑 **得分** 40

Lab 0 - Getting Started with an IDE

Goals: The goals of this lab are to familiarize yourself with Github Classroom, to start working in an IDE, to successfully submit files to the submission server, to review the basics of running a program in Java, and to write basic tests using the JUnit testing framework.

Accepting Repo for Lab Assignments

In this class, we require that all of your assignments be maintained in a Github repo. Since we have two types of assignments, we will be providing you with two separate repos for this class: one for your lab assignments and one for your semester-long project.

We will be using a tool called **Github Classroom** (https://classroom.github.com) to manage all of the student repos. This is beneficial to both you and us because it allows us to glance at your entire source for an assignment during office hours. To get started:

1. To create your lab assignment repo, you need to *accept the assignment* by using **this assignment link** (https://classroom.github.com/a/1Kxm2RM3)
2. Log into **Github.com** (https://github.com/) . If you do not already have an account, create a new one. I recommend that you associate your northeastern.edu email account with this account so that you can access your repos using *single-sign on*.
3. Associate your Github account with your student email so that we can identify you by selecting your email from the list of students. If your email is not there, just send us a

private message on Piazza!

4. Use the link to the your newly created repository for maintaining your lab assignments.
5. The first thing you should do is update the provided README.md file with your personal information including your name, your northeastern email address, and your preferred name. You should also update this file with information about each of the lab assignments as the course progresses.
6. The other file that is in the initial repo is a *.gitignore* file. This file specifies **files that should be typically not be included in your repo (https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files)**. We have populated it with specifics for Java, Eclipse, IntelliJ, MacOSx, and Windows. If you are using a different IDE, you should add the files specific to your IDE using the **official list of recommended *.gitignore* files (https://github.com/github/gitignore)**.
7. Before you can start using your repo, you will need to *clone it* into a directory on your computer. Be sure that you *commit* and *push* files frequently into your repo as you work on your assignments so that you never lose anything! Remember that we will be able to see your repos when you need help so be sure that you have pushed the latest version of your code before coming to office hours.

To use this repo, you will need to get familiar with git (if you aren't already). Here are some resources for you:

- **https://guides.github.com/introduction/git-handbook/ (https://guides.github.com/introduction/git-handbook/)**
- Here's a great tutorial video on **Git Command Line Basics (https://www.youtube.com/watch?v=HVsySz-h9r4&list=PL-osiE80TeTuRUfjRe54Eea17-YfnOOAx)**.
- Github also has these Github Training & Guides: **https://skills.github.com/ (https://skills.github.com/)** and **https://github.com/git-guides (https://github.com/git-guides)**

Getting Started with your IDE

Each of your assignments should be in a separate directory in your Github repo. To make it easier for us to know where to look to help you, each assignment will provide you with a recommended directory name:

Assignment Directory: lab00-getting-started

Some assignments will provide *starter files*. This is a starting place for your assignment and you will need to download them and add them to the assignment:

Starter Files: Book.java (https://northeastern.instructure.com/courses/146237/files/21110121?wrap=1) (https://northeastern.instructure.com/courses/146237/files/21110121/download?download_frd=1) , Person.java

(https://northeastern.instructure.com/courses/146237/files/21110136?wrap=1) (https://northeastern.instructure.com/courses/146237/files/21110136/download?download_frd=1) , PersonTest.java (https://northeastern.instructure.com/courses/146237/files/21110079?wrap=1) (https://northeastern.instructure.com/courses/146237/files/21110079/download?download_frd=1)

Finally, in order for the automatic tests to run successfully, you will need to make sure that your files are in the correct package. Recall that a package is a way of organizing files within specific directory structure in your project. If your files are not in the correct package, then the automatic tests will not run successfully:

Package: person

Please note the videos in this section were made for an earlier instance of this course and there have been some updates since they were made. Pay particular attention to these yellow boxes for what change has been made.

Choosing an IDE

This course requires that you download, install, and configure a professional *Integrated Development Environment* for use in this course. Since this class is being taught in Java, there are two that most popularly used in industry: IntelliJ and Eclipse. While you are free to use either of these IDEs or another IDE of your choice.

Eclipse IntelliJ

Eclipse is the most frequently used IDE used in industry for Java development. It is a highly configurable, open-source IDE that has lots of plugins that can be added that make it a great choice for developers who want control over their development environment. One of it's main advantages is that it provides a single workspace in which multiple projects can be managed. Follow these steps for setting up Eclipse for this course:

Step 1: Getting and installing Eclipse

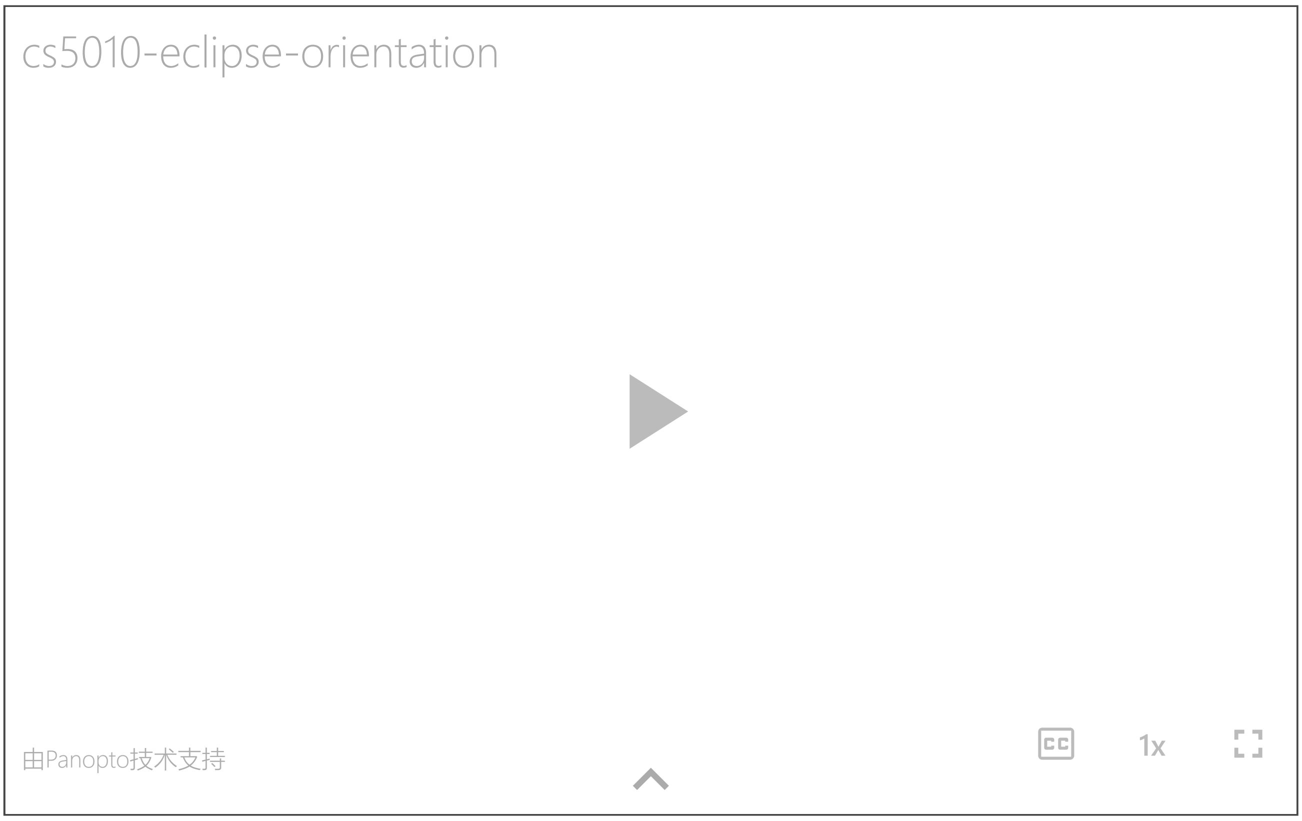
Ensure that you have installed the current version of the Eclipse by going to the **Eclipse website** (https://www.eclipse.org/) and clicking the "Download" button. This should give you an appropriate installer depending upon your system. Once it has downloaded, execute and and go through the steps. You will want to install the "Eclipse IDE for Java Developers". Other options can be added at a later date through the Marketplace inside of Eclipse.

Like most professional IDEs, Eclipse includes a source code editor, build automation tools, and a debugger. It allows you to organize your work into several files that together comprise a *project*, and organizes several projects into a *workspace*.

Step 2: Creating your first project

The following video will walk you through how to create a new workspace, how to organize your projects, and how to write new classes in a project.

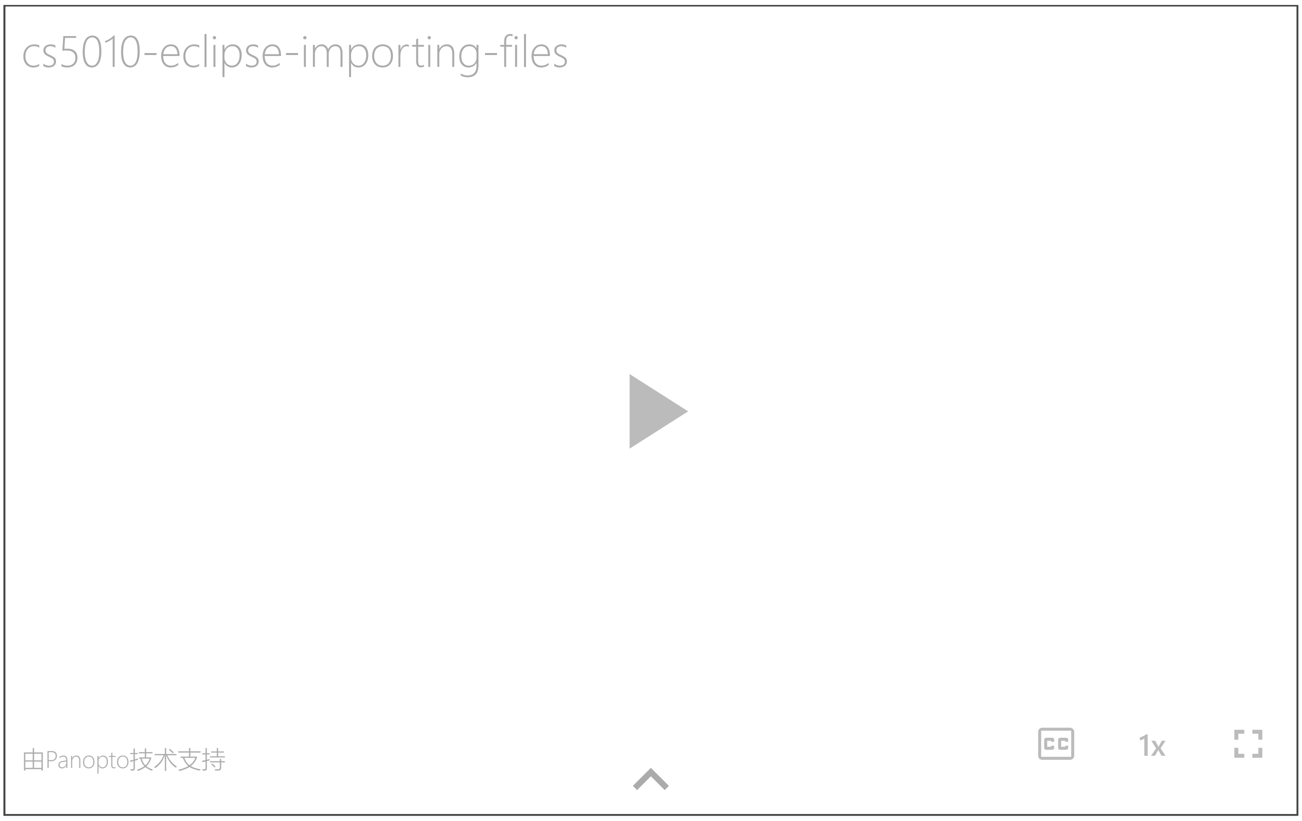
Please note that this course requires Java 11 and JUnit 4, NOT Java 8 as indicated in the following video.



Step 3: Adding files to a project

The next step we want to take is learn how to add files into an existing project. To do this, download the provided files and then follow the steps in the following video.

Please note that you should use the provided **Assignment Directory** for your project name and you should use the **Package** indicated instead of those used in the following video.

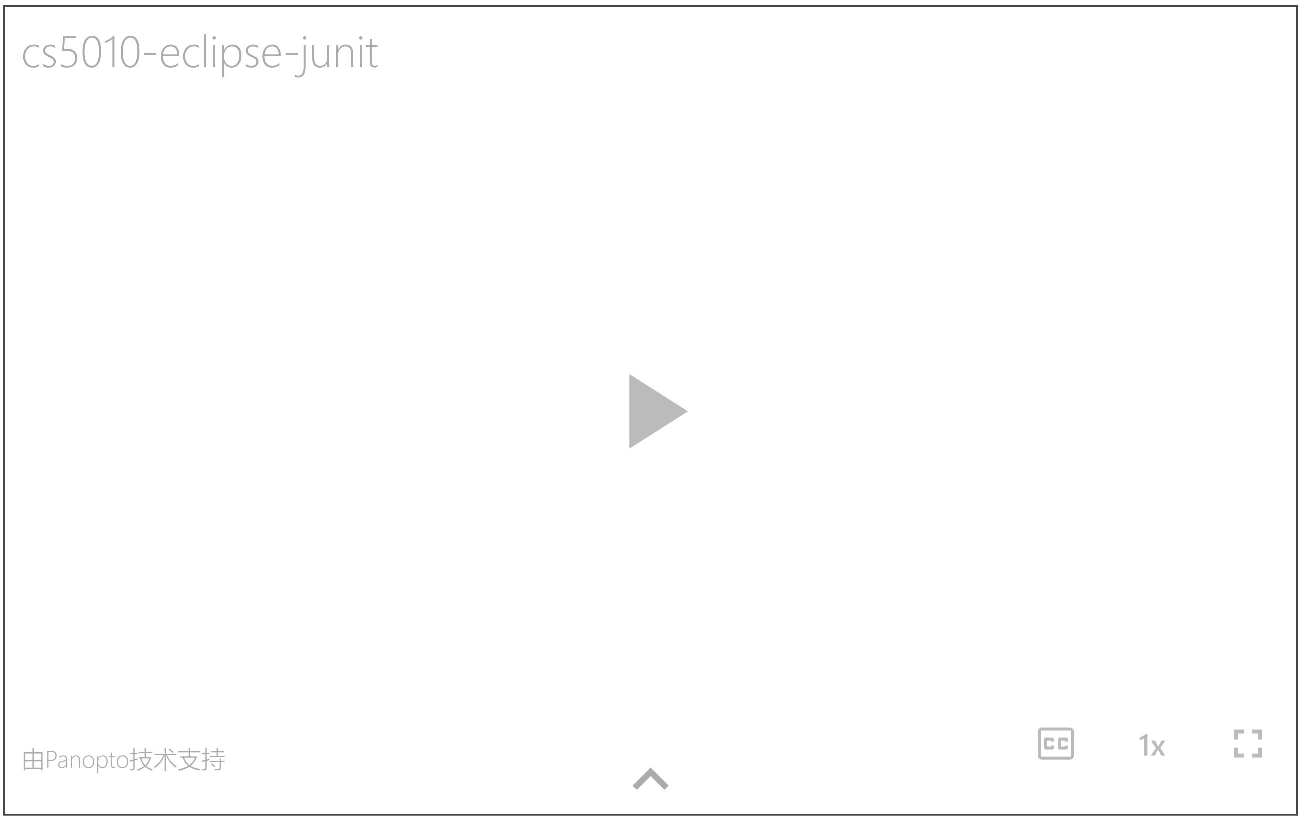


Verify that none of the files show any errors by double-clicking on them correcting any errors you might find before moving on.

Step 4: Writing test classes

Whenever we write a class, we also want to ensure that our class works correctly. We do this using the JUnit testing framework. Watch the following video to learn how to create and add JUnit test classes to your project.

Please not that the following video is using Java 8 with JUnit 4. This course requires Java 11 with JUnit 4.



Verify that all of the test cases and make corrections to be sure that all tests pass.

Step 5: JUnit for Book class

Create a JUnit test class for the Book class, using the same process as you did in the previous section to create tests for the Person class. Design and write tests that verify that all the public methods of the Book class work as expected. For each test case:

- Decide the objective of each test: *What exactly are you testing?*
- Decide *how* you will fulfill the above objective: *What must you do to ensure that your test indeed tests what it is supposed to?*
- Write the test.

Be sure that you execute the tests to ensure they is working correctly. Recall that the primary objective of any test class is ***convince someone else that your code works correctly.***

Step 6: Generate Javadocs

The provided files already have Javadoc-style comments. You will now generate html documentation from these files from within Eclipse.

1. Choose **Project** --> **Generate Javadoc...** from the menu.
2. If there is nothing in the **Javadoc command:** box, click on **Configure** and navigate the the directory where Java is installed on your computer. On my computer, it is

installed in `C:\Program Files\Java\jdk1.8.0_231\bin`. In this directory, you will find a `javadoc` executable. Select it.

3. Press **Finish** and confirm to update if prompted. While Eclipse generates the `html` files, it will report any errors in the Console. After verifying that there are no errors, you can use Eclipse to open the `index.html` file so you can browse them. You can also open them directly in your browser.

4. Correlate the Javadoc comments in the `Person.java` file with the output in `Person.html` file to see how Javadoc comments affect the documentation.

This works because the Java SDK comes with a tool called *javadoc*. Eclipse uses this tool and passes it all your *.java files. You will find this tool in the `bin` folder inside your Java SDK directory (wherever you installed it on your machine).

Step 7: Configuring for Style

Style of code is important for its readability and uniformity. Most companies impose a code style that makes all the code created by numerous developers have the same formatting. To this end, we will be enforcing a coding style that will be automatically assessed by the submission server when you submit your assignment (see directions later in this assignment).

The tool that the submission server uses is called **Checkstyle (https://checkstyle.sourceforge.io/)**, a tool that you can install from the Eclipse Marketplace (found on the Help menu in Eclipse) and configure to run checking on your local setup. **Checkstyle** checks that your code matches against a configurable set of rules. We use a styling convention that is derived from **the one used at Google. (https://google.github.io/styleguide/javaguide.html)**

- **How to install and configure Checkstyle (https://www.baeldung.com/checkstyle-java#eclipse-plugin)** (see Section 3)
- Configuration file: `northeastern_checks.xml` **(https://northeastern.instructure.com/courses/146237/files/21110168?wrap=1) (https://northeastern.instructure.com/courses/146237/files/21110168/download?download_frd=1)**

Finally, there are a few things that need to be adjusted in Eclipse itself. These include things like adjust Eclipse to default to 2 spaces and 100 character line lengths. You can change this by selecting **Windows ... Preferences** from the menu bar. Then select **Java ... Code Style ... Formatter**. Add a new profile by clicking the Import button and navigate to this **eclipse-formatter.xml (https://northeastern.instructure.com/courses/146237/files/21110147?wrap=1) (https://northeastern.instructure.com/courses/146237/files/21110147/download?**

download_frd=1) file to configure your Eclipse as close to the expectation of the submission server as we can get it

IntelliJ is another popular choice of many professional Java programmers. The environment provides an editor, allows you to organize your work into several files that together comprise a project. Follow these steps for setting up IntelliJ for this course:

Throughout part of the this lab, we will refer to the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)**. This video was originally made but the professor that teaching CS 3500 for an earlier instance of his course; replace those references with CS 5010.

Step 1: The First Project

You will want to go through this in detail so that you can gain familiarity with the entire process.

1. Follow the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)** **up to about the 6:00 mark**. It will walk you through how to create a new project, how to organize your projects, and how to setup your Java SDK.
2. The video creates a new project called "HelloWorld". Use the **Assignment Directory** provided above as your project name
3. Add Person.java and Book.java from the provided **Starter Files** to your project placing them in the provided **Package** rather than following the package structure in the video. Follow the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)** from **9:30--10:25**. Verify that none of the files show any errors by double-clicking on them.
4. Understand the folder structure imposed by IntelliJ by listening to the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)** from **11:35--12:30**.
5. Add JUnit tests for the Person class. Follow the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)** from **12:30--17:10**, but create a test class for the Person class instead of what the tutorial is showing you.
6. Replace the contents of the newly created PersonTest.java with what is in the file with the same name provided to you in the related files.
7. Run this test case, by following the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)** from **18:40--19:22**. Verify that all tests pass.

Step 2: Tests for Book

1. Create a JUnit test class for the Book class, using the same process as you did in the previous section to create tests for the Person class.
2. Design and write tests that verify that all the public methods of the Book class work as expected.
    - Decide the objective of each test: *What exactly are you testing?*
    - Decide *how* you will fulfill the above objective: *What must you do to ensure that your test indeed tests what it is supposed to?*
    - Write the test.
3. Execute the test to ensure it is working correctly.

Step 3: Generate Javadocs

The provided files already have Javadoc-style comments. You will now generate html documentation from these files from within IntelliJ.

1. Within your IntelliJ project folder, create a new folder called "docs". IntelliJ will pull all the generated documentation within this folder.
2. From the IntelliJ menu, choose **Tools** --> **Generate Javadoc...**.
3. In the **Output Directory** field, point it to the "docs" folder you created in step 1.
4. Press **OK**. IntelliJ will now generate the html files and open them in your browser. If the files do not open automatically, go to the "docs" folder and open "index.html" in your browser.
5. Correlate the Javadoc comments in the "Person.java" file with the output in "Person.html" file to see how Javadoc comments affect the documentation.

How does this work? The Java SDK comes with a tool called *javadoc*. IntelliJ uses this tool and passes it all your *.java files. You will find this tool in the "bin" folder inside your Java SDK directory (wherever you installed it on your machine).

Step 4: Configuring for Handins Style

Style of code is important for its readability and uniformity. Most companies impose a code style that makes all the code created by numerous developers have the same formatting. In this course we will use a styling convention from Google and employ IntelliJ to enforce this style on our code. All the assignments in this course will have a part of their grade determined by style, so it is important that you enforce this style for all source code!

1. Download the style file: **intellij-java-google-style.xml (http://khoury.northeastern.edu/~mjump/java/intellij-java-google-style.xml)**
2. Follow the **tutorial video for IntelliJ (http://khoury.northeastern.edu/~mjump/java/Intellij-setup.mp4)** from **20:43--28:12**

It is important to note that while the provided style file for IntelliJ does quite a bit of the styling for you in your file, ***it does not do everything that is expected of you by the submission server.***

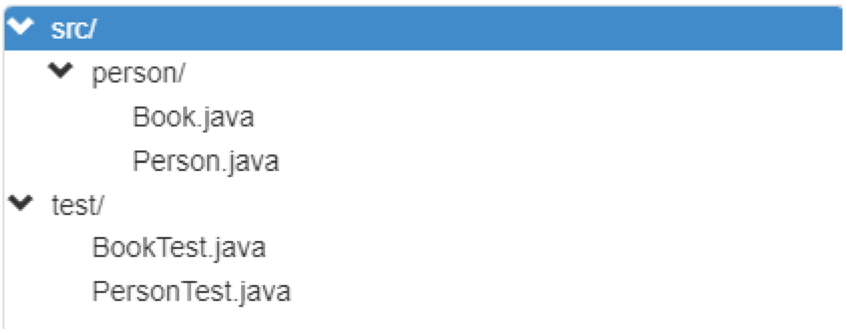
However, you can get most of the server style feedback by installing the **Checkstyle** plugins in IntelliJ. First, download the configuration files for these: **Checkstyle (https://northeastern.instructure.com/courses/146237/files/21110168?wrap=1) (https://northeastern.instructure.com/courses/146237/files/21110168/download?download_frd=1)** .

Next, install the plugins and configure them using those configuration files: **Video on how to install & configure Checkstyle (https://northeastern.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0beb1c42-bb6c-4e03-9ded-ac3100f751ab)** (you can ignore the PMD part)

Submitting to Gradescope

Once you have completed all steps for setting up your IDE, you should:

1. Create a zip file that contains the ***src*** and ***test*** folders of your IDE as the **top-level folders** of the zip. For example, your submission for this assignment should have the following file structure:



```
src/
  person/
    Book.java
    Person.java
test/
  BookTest.java
  PersonTest.java
```

2. Submit it to the assignment called *Lab 0 - Getting Started* on the submission server.
3. Inspect your submission and look at the style grader feedback. Correct your code accordingly and resubmit, until the style grader no longer complains.
4. One you are completely happy with your submission, complete the corresponding *Lab 0 - Report* assignment on the submission server.