Comp215:
Lab 1: "Hello World"

Dan S. Wallach and Zoran Budimlić
dwallach@rice.edu and zoran@rice.edu

Fall 2018

Contents

1 Deadline **2**

2 Preliminaries **2**

3 Introduction **2**

4 Install Git **3**
4.1 Mac OS X . . . 3
4.2 Linux . . . 3
4.3 Windows . . . 3

5 Install Java8 **3**

6 Install IntelliJ **4**

7 Getting Started with IntelliJ **5**
7.1 IntelliJ + JDK8 . . . 5
7.2 Hello, world . . . 8

8 IntelliJ, Git, and GitHub **9**
8.1 IntelliJ + GitHub . . . 9
8.2 "Cloning" your repository . . . 12
8.3 Make sure it's working . . . 14

9 Do the "lab assignment" **15**

10 Install a few plugins **15**

11 Commit and push your changes to GitHub **19**

12 On your own time **21**
12.1 Fonts 22
12.2 Advanced: Using the command-line 22
12.3 Advanced: Digging into Git 22

1 Deadline

Lab check-off no later than Friday, August 24. You must have been present in one of the lab sessions.

2 Preliminaries

GitHub. We're using GitHub this year to manage how all the code goes from us to you and back to us again. That means you'll need to create yourself an account at https://github.com. If you haven't already done so, you need to it now.

Web form. We have a web form to track all of the students in the class. Please fill it out ASAP if you haven't already done so. If you don't fill out the form, we won't be able to give you a grade in the class. https://www.surveymonkey.com/XXXXXXXXX

(The initial page of questions is mandatory. The rest are optional. See the course syllabus for details on how you can opt-out of the survey.)

Piazza. We'll populate the course Piazza based on the email addresses that we get from Esther. There may be some bumps along the way. If you mysteriously disappear, don't panic. We'll get it fixed. We post weekly assignments and slides on Piazza. Some of you have a short NetID (your initials and a number) as well as a longer "vanity" email address. We typically use the longer email for your Piazza account. Piazza allows you to "merge" these together[1], so you'll be easily able to move back and forth across the Piazza forums for each of your classes.

3 Introduction

Your lab this week has almost nothing to do with Java and everything to do with making sure that you've installed IntelliJ properly, including the necessary libraries and plugins. Our goal is to make sure that we've sorted out all the first-week jitters, both for you and for the graders.

For starters, *don't panic*. This is a seemingly long assignment, but it's really just walking you through getting everything installed properly on your computer so, for subsequent assignments, things will go much smoother.

[1] http://support.piazza.com/customer/portal/articles/1672367-add-an-email-address-or-merge-two-accounts

4 Install Git

First things first. "Git" is a *distributed version control system*, used to manage code repositories. "GitHub" is a commercial service that hosts Git repositories in the same way that "Gmail" is a commercial service that hosts email. We're not going to teach you all the gory details of how Git works, since there's a lot of complexity that only really happens when you work in teams. Nonetheless, Git is widely used in industrial and open-source software development. You might as well get started.

To get Git installed on your computer, each operating system you might be using has its own quirks for this. We've got labbies and TAs who use each of these who can help you if you get stuck.

4.1 Mac OS X

Run the **Terminal** application and type this:

```
xcode-select --install
```

This will install a variety of "command-line developer tools" that will be used under the hood by IntelliJ, including Git. More details in this Apple tech note[2]. Alternately, you can install the full-blown XCode application from Apple's App Store. We won't be using this in Comp215.

4.2 Linux

Most Linux distributions come with pretty much everything you're going to need. You should check that you have Git installed by bringing up a shell and simply typing `git --version` and hitting return. If it says version 2 or later, you're good to go. If not, you'll need to look up how to install packages with your system. For Ubuntu, you'd type something like:

```
sudo apt-get install git
```

4.3 Windows

There are many different ways to install Git. Probably the easiest is to download and install this: https://git-scm.com/download/win. Select the default choices for the installation.

5 Install Java8

You need to install the latest version of the Java Development Kit (JDK). You can get it here: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. You want the "Java SE Development Kit 8u181" or newer for whatever your computer happens to be. Oracle supports you for Windows, Mac OS X, and Linux. Don't install JDK9 or anything higher than that.

[2] https://developer.apple.com/library/ios/technotes/tn2339/_index.html

We're using JDK8. Also, for Windows, make sure you get the "x64" (64-bit) version, unless you've got a *very* old computer, which we wouldn't recommend for Comp215.

If you're using an "old" version of Java8 that you may have installed in prior months or years, please update to the latest and greatest. Prof. Wallach started out writing code for this class with the "8u25" version and managed to get the Java compiler to blow up on his perfectly reasonable and beautiful code. It's better now, but please do make sure you have the latest and greatest.

If you've previously installed a newer version of Java (Java9, Java10, or even Java11), you'll have to do some additional work to make sure that IntelliJ is indeed using Java8 (see Section 7.1 below). Unfortunately, some of the tools we use in Comp215 don't yet support these newer Java versions, and you'll get weird errors if you're using the wrong version. Find a labbie or TA if you have problems here and we'll help you with your computer's configuration.

There are more detailed platform-specific instructions available for you[3]. Make sure you follow the JDK ("Java development kit") instructions, not the JRE ("Java runtime environment") instructions. You need the JDK.

Windows: You may get some scary warnings from Oracle about how they're changing their support model for Java. Please ignore these warnings.[4]

6 Install IntelliJ

Visit the IntellJ IDEA download site: https://www.jetbrains.com/idea/download/ and download the free "community edition" (CE). (There's nothing in the "ultimate" edition that will make any difference to your work in Comp215.) Versions are available for Windows, Mac OS X, and Linux. IntelliJ is an example of an integrated development environment (IDE). The other popular Java IDE you may have heard of or used in the past is Eclipse. *For Comp215, you will use IntelliJ. If you run anything else, we won't help you when it breaks, and if you submit something that "works for you" but doesn't work for us, your grade will suffer.*

If this is the first time you have installed IntelliJ IDEA on your computer, the software will ask you to customize it a bit before starting. You can pick a color scheme that you like, then it will ask you to customize different plugins. Leave that alone for now, and start using the IDE.

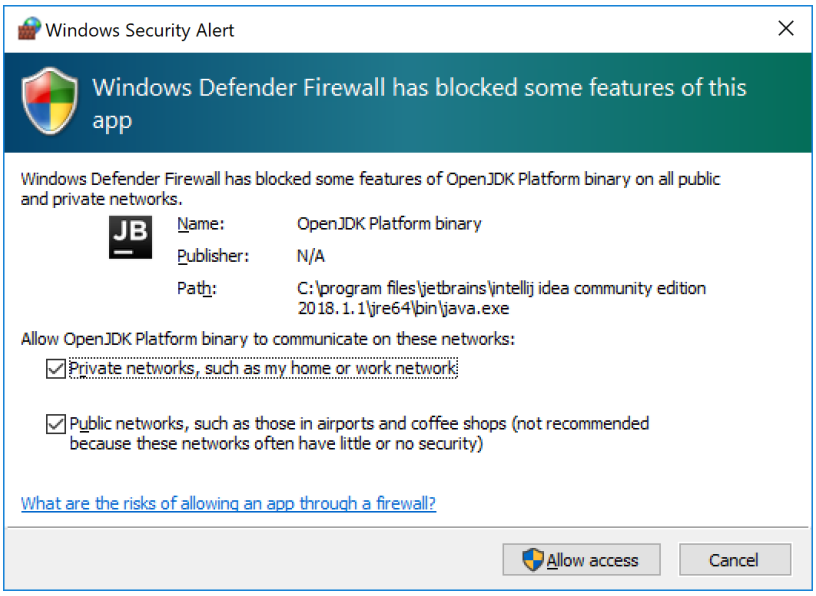
If you've installed IntelliJ in previous semesters, it's time to upgrade. You want to be running "IntelliJ IDEA CE 2018.2" or newer, which only just came out. Why the absolute latest and greatest? IntelliJ keeps getting better every year, automatically finding and sometimes even offering to fix your bugs for you. Also, newer versions of IntelliJ simplify some of the steps that you're about to go through, relative to earlier years. (Crazy, but true.) Please make sure you're running the newest version.

Windows: IntelliJ will ask you if you want to uninstall a previous version. This is fine, and it's helpful to check the "do it quietly" box. You'll also get a checkbox asking if you want to "Download and install JRE x86 by JetBrains". Please select this. It's also fine to check the boxes to associate

[3] https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html

[4] In future years, we may need to switch from the "Oracle JDK" to the "OpenJDK", but otherwise there are no meaningful impacts here on Comp215, and probably no meaningful impact on the use of Java elsewhere.

IntelliJ with Java and other files it understands. Also, as we go on with these instructions, the Windows Defender Firewall may occasionally have some questions for you:



When you see questions like this while following our instructions, please just check all the boxes and Allow access when prompted.

7 Getting Started with IntelliJ

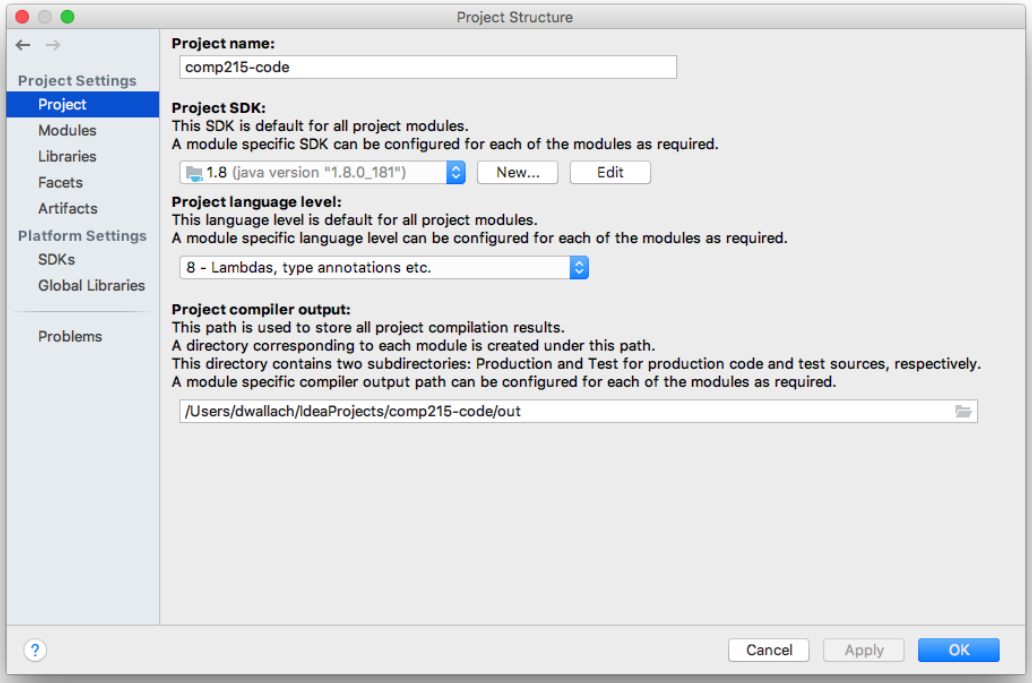
If you've previously used IntelliJ and have a project open, you'll see a menu entry, File → New → Project... Otherwise, if you're staring at the "Welcome to IntelliJ IDEA" launch screen, there's a Create New Project button. Click it. Either way, you'll see many different kinds of projects to choose from. In the left column, it should offer you a "Java project" as the default option, and there's a drop-down from the top that should let you select "1.8" as the Project SDK. If it's only offering you an older version, don't worry, we'll fix that below. Otherwise, you just give it a name ("Hello") and tell IntelliJ where to put the files (typically inside the "IdeaProjects" directory).

If IntelliJ is telling you it has Project SDK: <No SDK> at the top of the window and gives you no other choices, you should hit the New... box. On my Mac, it automatically found /Library/Java/-JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home but on a Windows machine, I had to drive it to C:\Program Files\Java\jdk1.8.0_181 instead.

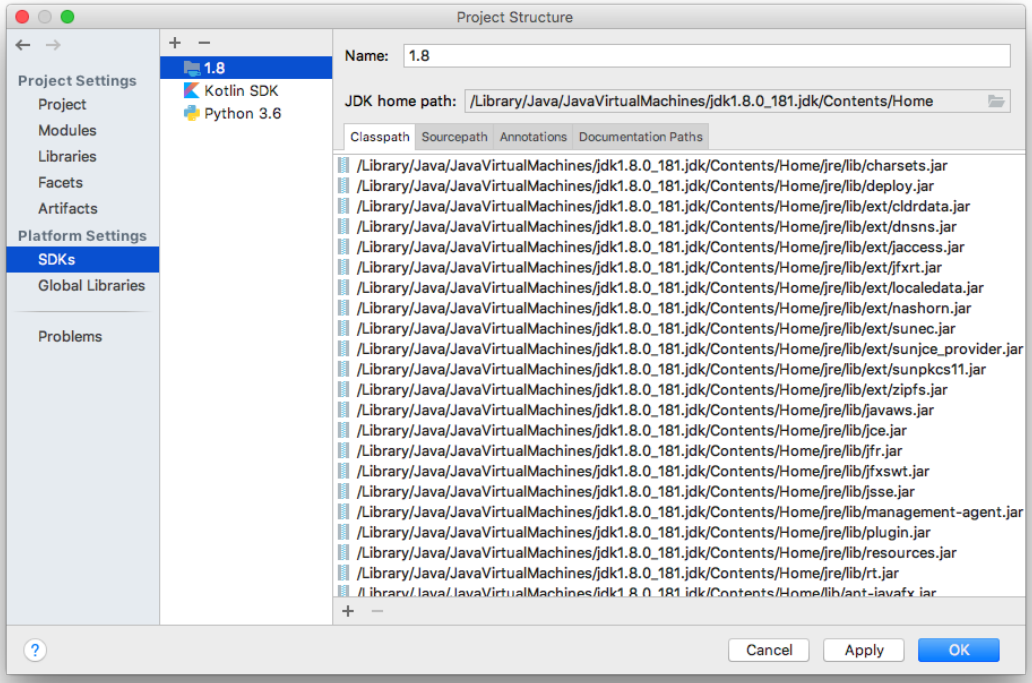
7.1 IntelliJ + JDK8

Hopefully the short instructions above worked. But if not, here are lots more details. Do this to make sure everything is working.

You'll need to configure IntelliJ to use the Java8 JDK that you just installed. IntelliJ has two different kinds of preferences, for the "current project" and for the IDE as a whole. Let's start with the whole IDE. You should see something like this:



If you see an older JDK8 than the one you just installed, then hit the “SDKs” tab below “Platform Settings” and you should see this:

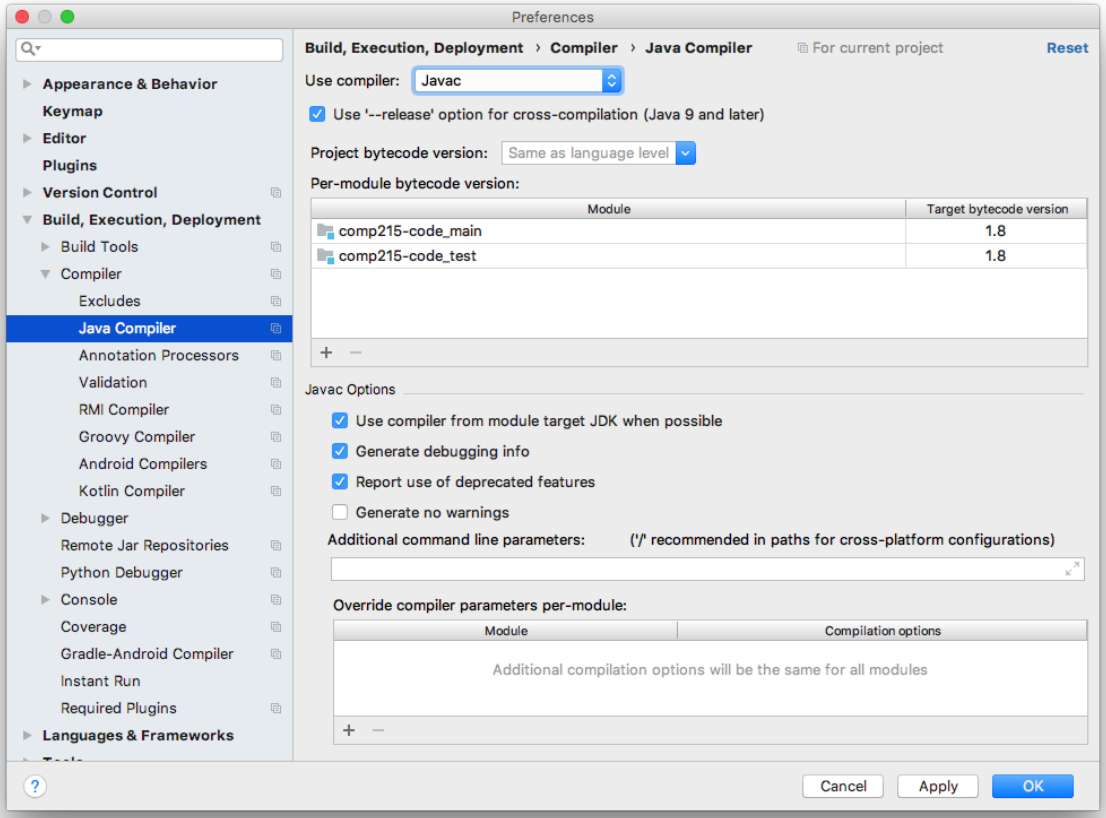


Before you install a "new" SDK, it's useful to avoid confusion by removing the old one first. Select the "1.8" and hit the [−] button[5]. At that point, you can hit the [+] button. As described above, you'll want to drive it to the newest JDK8 on your computer, typically /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home) on a Mac or C:\Program Files\Java\jdk1.8.0_181 on Windows.

Next, it's time to make sure that your current project is using the JDK that you just installed. Flip back to the initial "Project settings" and make sure that it's set to use "1.8" as the Project SDK and that it includes the proper release number ("181"). Don't forget that the "language level" needs to be "8", allowing you to use all the new Java8 features.

Now that IntelliJ knows where to find Java8, you need to make sure your current project is properly set to use it. You'll see something like this:

[5]If you don't remove the old JDK8 before installing the new JDK8, you'll end up in a goofy situation where you have to select between "1.8" and "1.8 (1)" You only need the latest and greatest linked into IntelliJ. Nothing here removes the JDK from your computer, just from IntelliJ's settings



It should indeed say that "1.8" is the target bytecode version and you should be good to go.

If you experience problems here, don't panic. These instructions can't possible imagine everything that might go wrong. Any labbie or TA can help you get your machine set up properly.

7.2 Hello, world

At this point, you're ready to write a program that actually does something. We're going to keep this super simple and verify that you can create a Java8 "Hello, world" program. You should be able to right-click on the left column where it has "src" and create a new Java class. Go ahead and do that with a simple name like "Hello". It should go ahead and make that for you and will fill it out with something hopefully like this:

```
public class Hello {
}
```

Rewrite that to look like this:

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world");
    }
}
```

Right-click on "src/Hello" in the left column, and select "Run Hello.main()". It should go split-screen on you and you'll see "Hello, world" printed at the bottom. Now for something silly, but it's enough to verify that you've got Java8 working, rather than something older that might be leftover on your computer:

```
import java.util.function.Supplier;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world");

        Supplier<String> x = () -> "Hello, lambda!";
        System.out.println(x.get());
    }
}
```

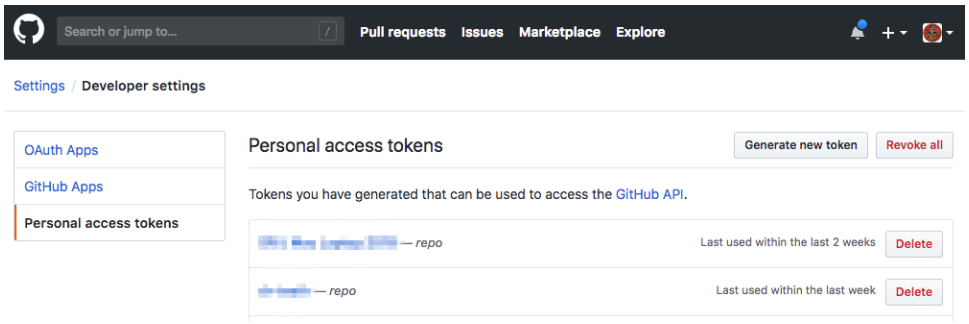
If that worked, then you're running Java8. Congratulations. (Fear not. We'll teach what all those weird symbols mean in the first few weeks of class.)

8 IntelliJ, Git, and GitHub

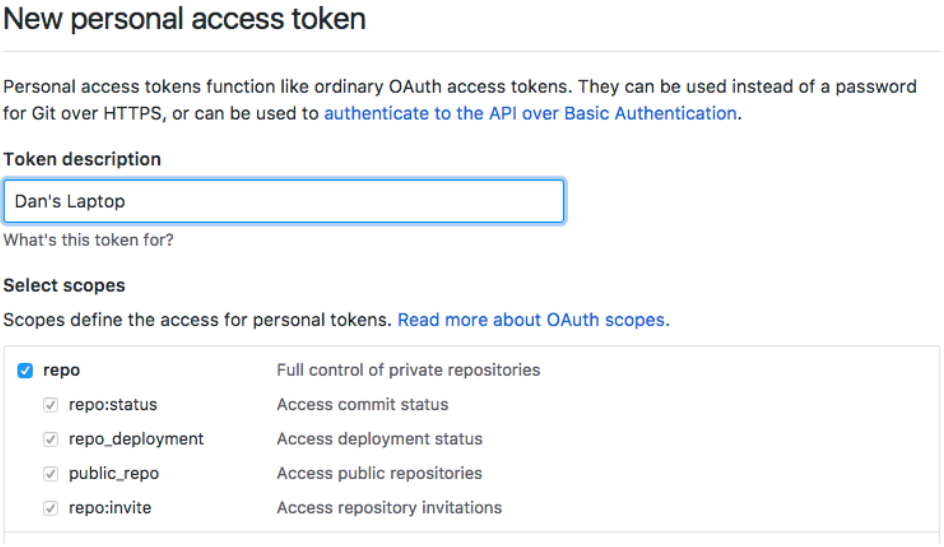
We already had you install Git in Section 4. Now you're going to connect it to IntelliJ and use it to copy our reference code from GitHub to your own computer.

8.1 IntelliJ + GitHub

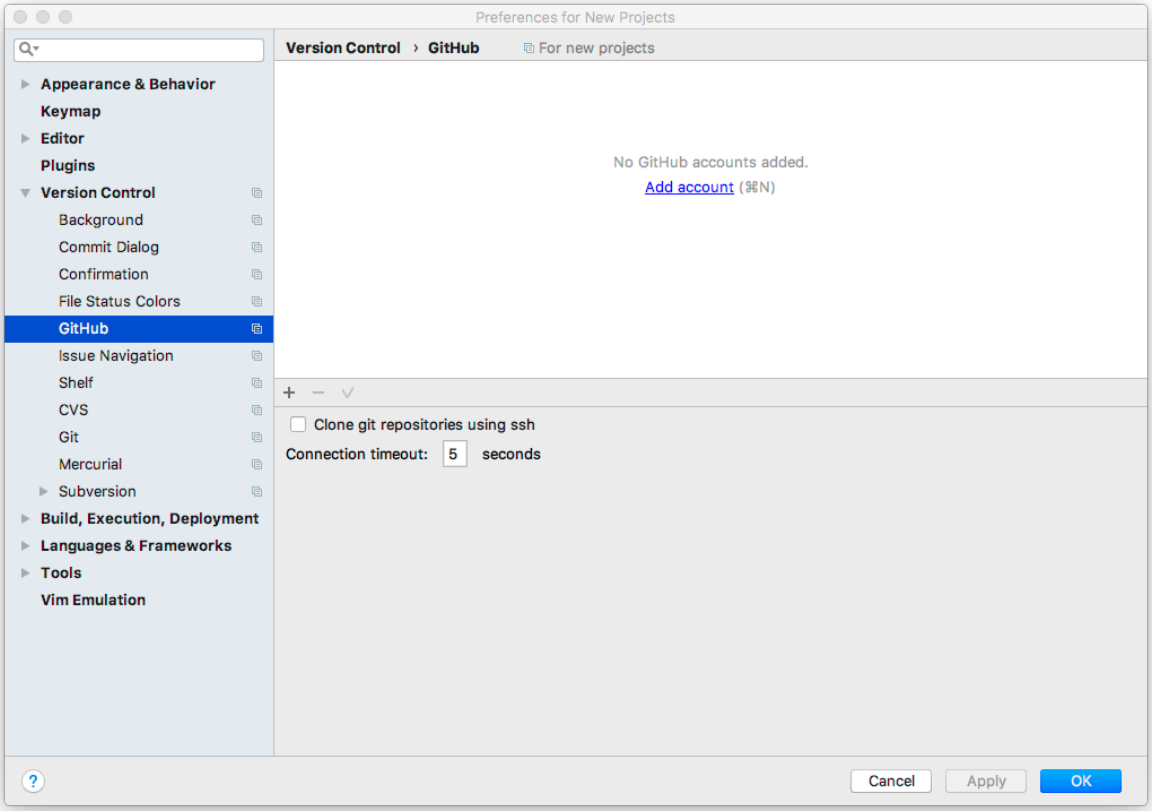
IntelliJ has special support to talk to GitHub, avoiding all the mess of usernames and passwords you'd use for other Git servers. After you log into GitHub's web page, the icon in the upper right has a Settings selection, and from there you can select the Personal access tokens tab on the left.



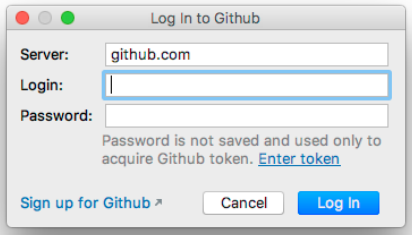
At that point, you select the Generate new token button in the upper right, and give the token a name (e.g., "Dan's Laptop"). It's important, when GitHub asks you, that you give this token all the "repo" permissions.

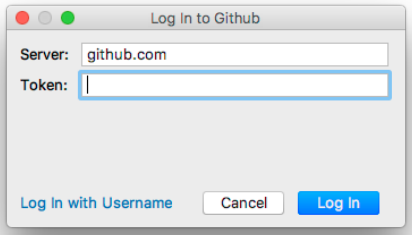


After you generate it, your token is displayed as a long string of random digits. Leave this on-screen, then flip over to IntelliJ. You're going to need to tell IntelliJ how to use this token. You do this, one time, and IntelliJ will remember it. Go to your preferences (IntelliJ IDEA → Preferences... on a Mac, File → Preferences... on Windows) and select Version Control → GitHub. You should see this:

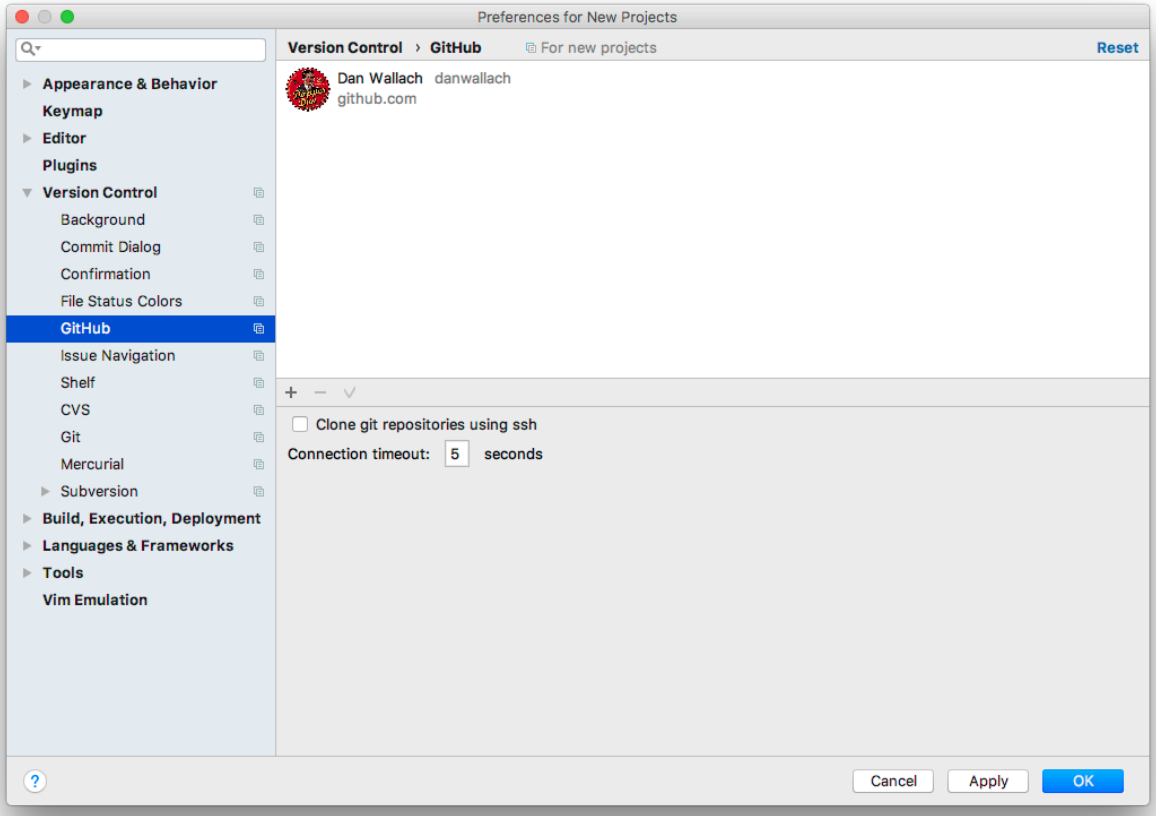


Click the "Add account" link. You'll see the first dialog box, below. Click the "Enter token" link and you'll see the second dialog box, also below. That's what you want. Paste the "personal access token" into the "token" box and hit "Log in."





If you're successful, you'll end up looking at something like this:



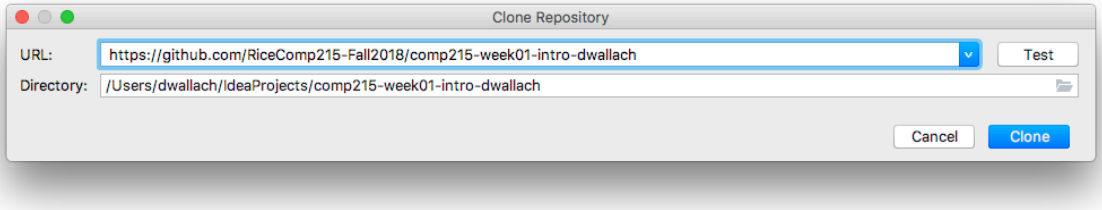
IntelliJ might also ask you for a "master password" that it uses to encrypt a database of all the different usernames and passwords that you might use with it. You can just use your Rice NetID password if it asks for this, or you can use something else. That's up to you. This password will never leave your laptop.

8.2 "Cloning" your repository

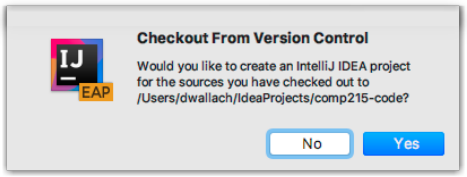
Every week, we'll give you a new link. You'll click on it and GitHub will create a copy of our reference code for you. For this week, go ahead and click this:

https://classroom.github.com/a/XXXXXXXXXX

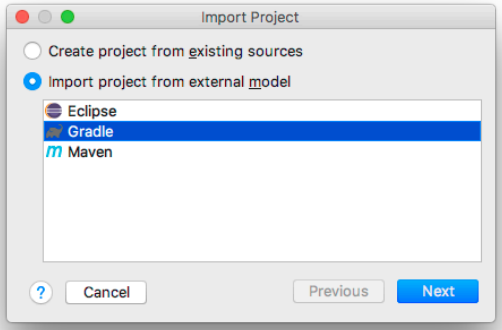
You may see a permission dialog from GitHub, which you should approve. Next, click the button to Accept this assignment and you're ready to make a local copy by "checking out" this code from GitHub. You'll want to select File → New → Project from Version Control → Git. You'll then get a dialog like this:



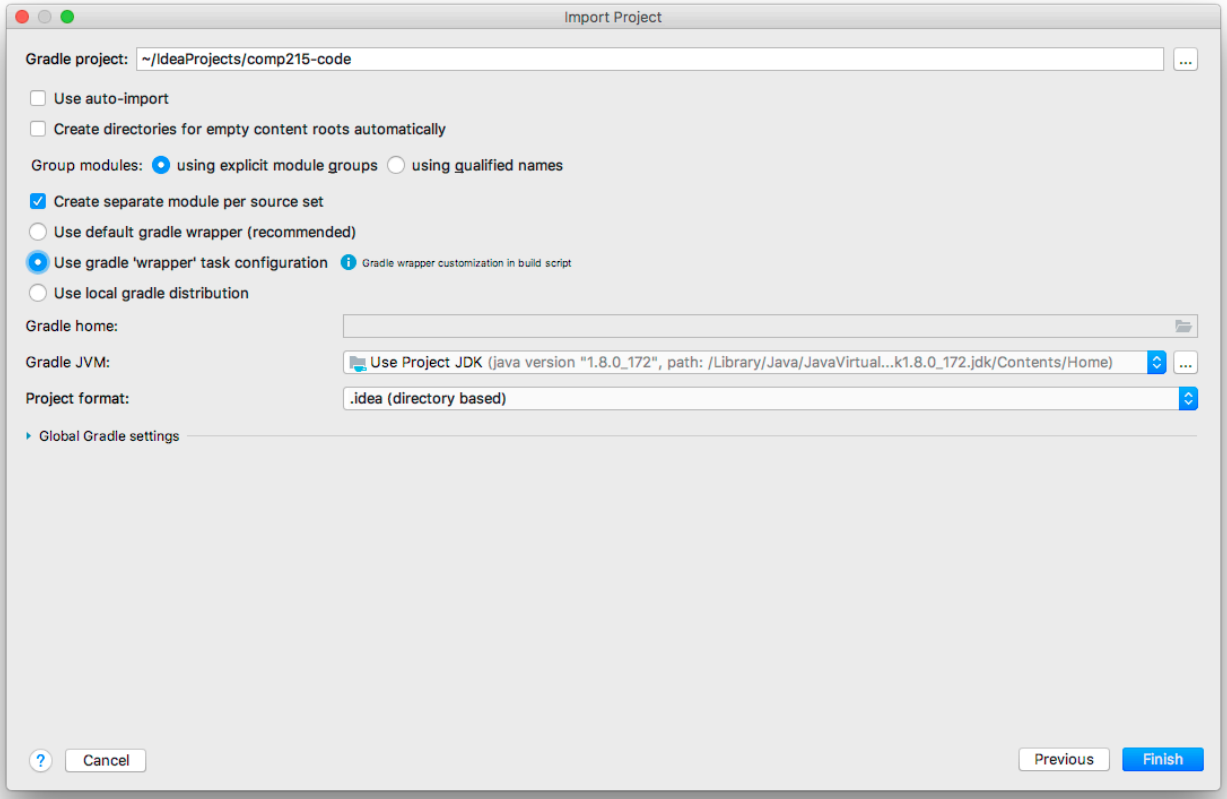
You'll use the URL that GitHub created for you. It probably looks something like https://github.com/RiceComp215-Fall2018/comp215-week01-dwallach/. Copy-and-paste it into the dialog box, then hit Clone. You'll get some pretty obvious questions like this:



Select Yes. You'll also get a somewhat more complex dialog like this:



We use a tool called "Gradle" to help configure IntelliJ and many other things. Select "Gradle", hit Next, and you'll see this:



This is the confusing part. Make sure you select *Use gradle 'wrapper' task configuration* and make sure it mentions the Java8 JDK that you installed earlier as the Gradle JVM. If so, you're good to go. Hit Finish and it will then do all the work for you.

Now that it's all done, you'll have a directory that lives inside IdeaProjects *in your home directory. You'll create a separate Git repository every week as part of Comp215. Please make sure they're in separate directories. Don't try to open one on top or inside of another.*

8.3 Make sure it's working

If everything is working, you should be able to scroll through the "Project" tab on the left, into the test directory, which will have a green background. You can dig your way down to and right-click on `StringThingTest`, selecting "Run StringThingTest". Everything should compile but one of the tests will fail. If so, then you're almost done. If you get compiler errors, then you need to back up and revisit the steps above to get Java8 configured properly. (Or, if you have problems, ask for help.)

9 Do the "lab assignment"

There's an actual assignment in here, but it's easy.

- There's a file called `README.md` at the very top-level of your week's files. Right now, it's got your instructor's name in it. Please edit it to have your own name, your Rice NetID, etc. In future weeks, we'll have other questions for you here as well.

  This file is an a format called "Markdown", which lets you do all sorts of fancy formatting without having to learn HTML. You're free to read up on how it works: https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf. When you visit the webpage at GitHub.com corresponding to your project, you'll see this Markdown turned into beautiful HTML.

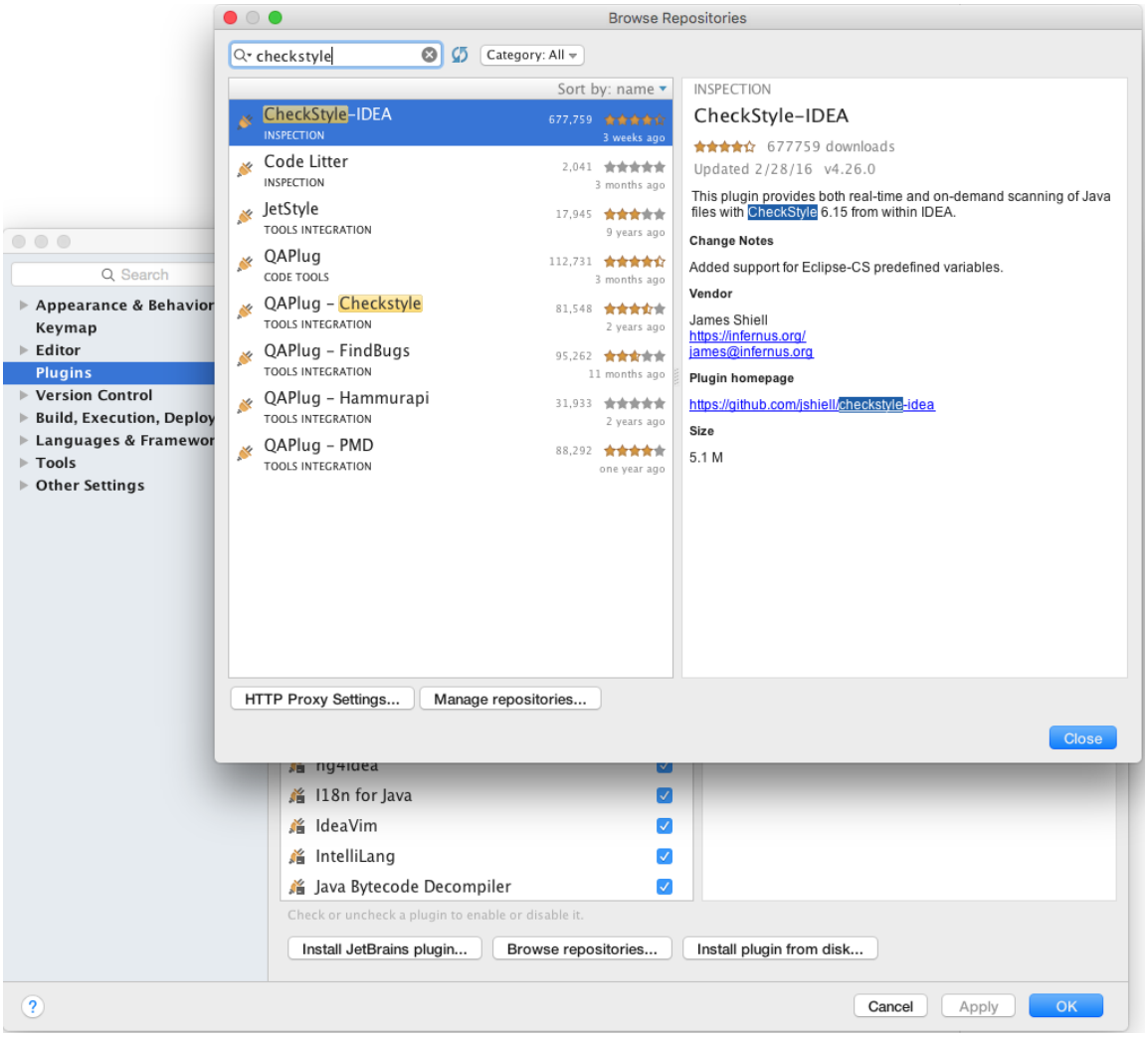
- You will edit `StringThing.java`. There's a static class instance, defined on top, that's meant to represent your student information. Change it to instead have your own name, expected graduation year, Rice NetID, and preferred email.

- Run `StringThingTest`, and you'll notice that one of the tests fails. There's a bug in StringThing's `getNetID()` method. Fix it. Verify that `StringThingTest` works.

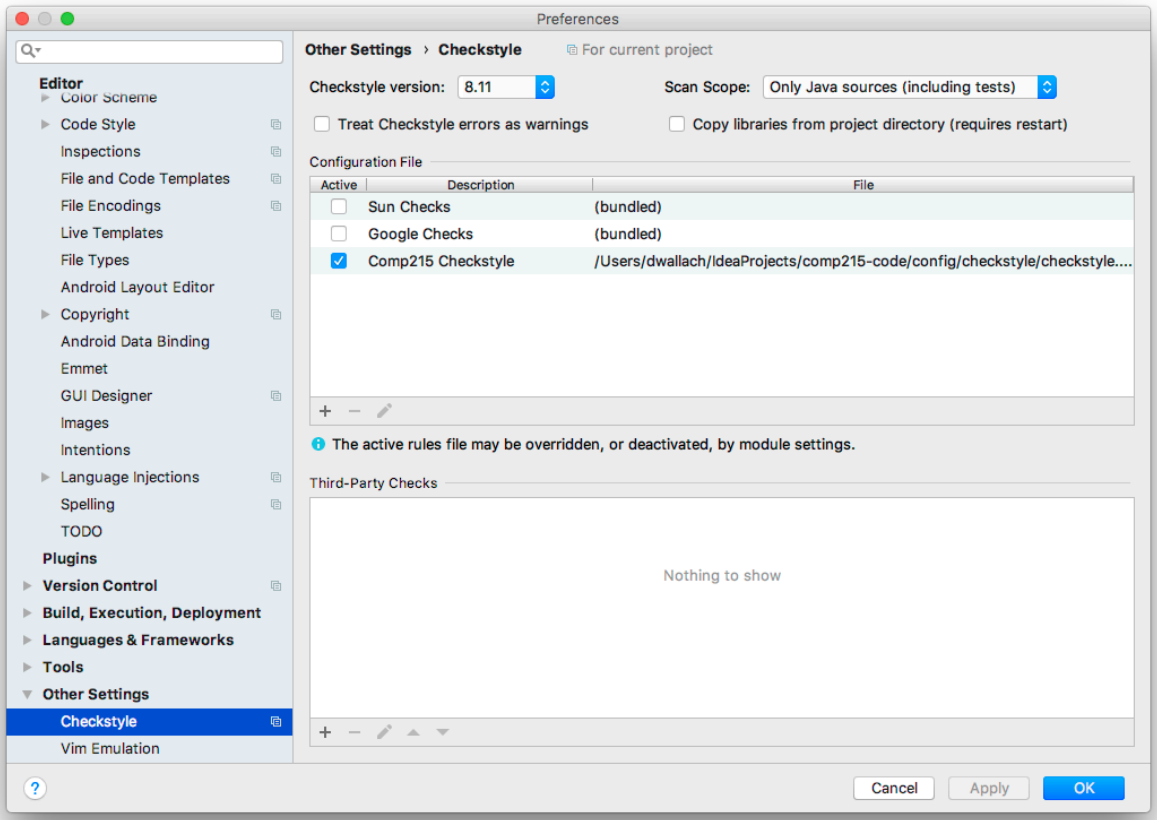
A quick warning: GitHub has a bunch of features that you should absolutely not use. Notably, you can directly edit files on the web in GitHub. Don't use GitHub's editing features. We're trying to ensure that you make all your changes in IntelliJ, so you never have to worry about merge conflicts and other problems that can happen when you're editing in more than one placed at a time.

10 Install a few plugins

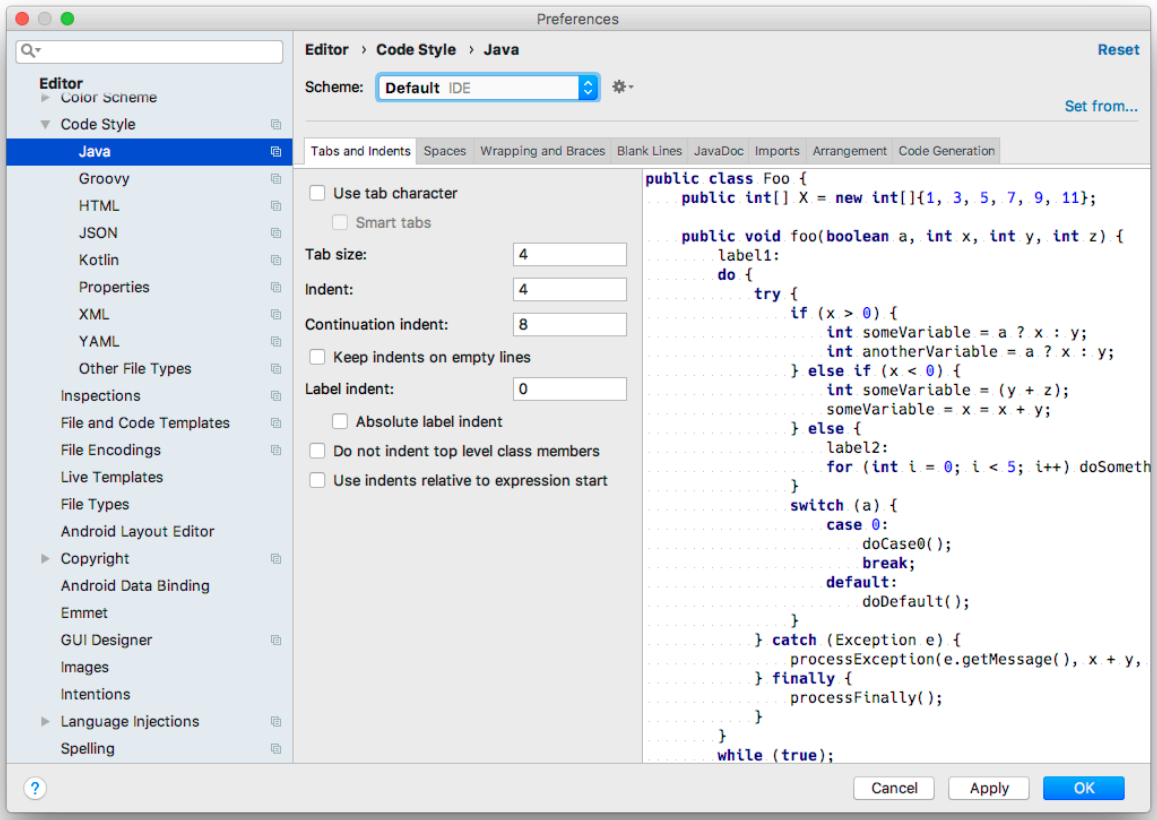
- We care about your coding style. Please install the CheckStyle-IDEA plugin. This makes CheckStyle run all the time, as you type, highlighting a variety of style issues that we require: indentation, variable naming and capitalization, maximum line lengths, and many other such things. You need to go to the plugins page and search the repository like so:



After installation, you'll have to restart IntelliJ. Thereafter, you'll be able to configure the CheckStyle-IDEA plugin to use the Comp215 style:



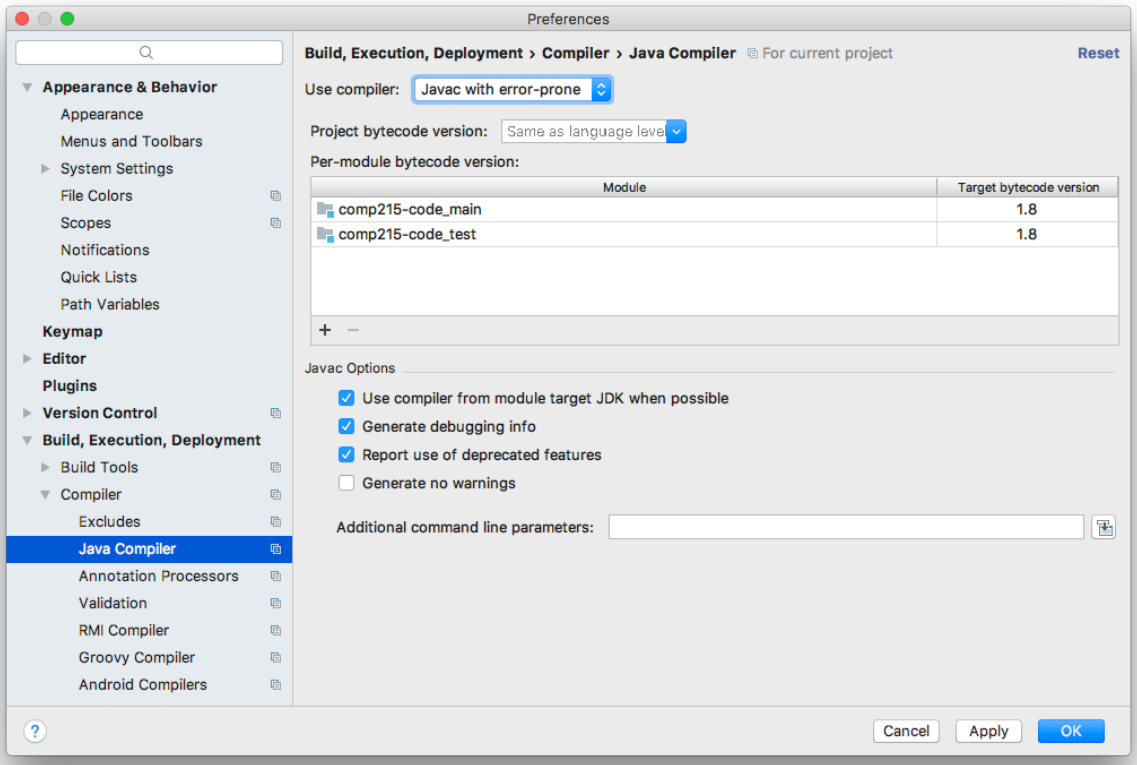
- Tell CheckStyle about the Comp215 configuration by hitting the + button and leading it to config/checkstyle/checkstyle.xml in your project.
- Run CheckStyle from its tab on the bottom of the window. Oops! CheckStyle is unhappy that one of the lines is too long. Fix it (insert a new line before the +). Run CheckStyle again and make sure it comes up clean.
- While you're at it, you should configure IntelliJ's Java style settings. This will help you write code that passes CheckStyle without errors.



Above, you can see the dialog that lets you configure all of the Java indentation and other stylistic features. Hit the gear icon at the top next to the Default box, then select Import Scheme → IntelliJ IDEA code style XML. You can then navigate the file selector to config/codestyles/comp215-intellij-style.xml, which we've built for you. Lastly, select the Default box and you'll now have Comp215-IntelliJ-Style as one of your choices. Select it.

- Install the "Error-prone compiler integration" plugin, using the same process you used to install CheckStyle-IDEA. After yet another restart of IntelliJ, you'll need to head back to the configuration panel again and enable Error-prone[6], developed by Google and used extensively in industry. Choose Settings → Compiler → Java Compiler → Use compiler: Javac with error-prone. This will add additional checks to help you ensure your code is bug-free.

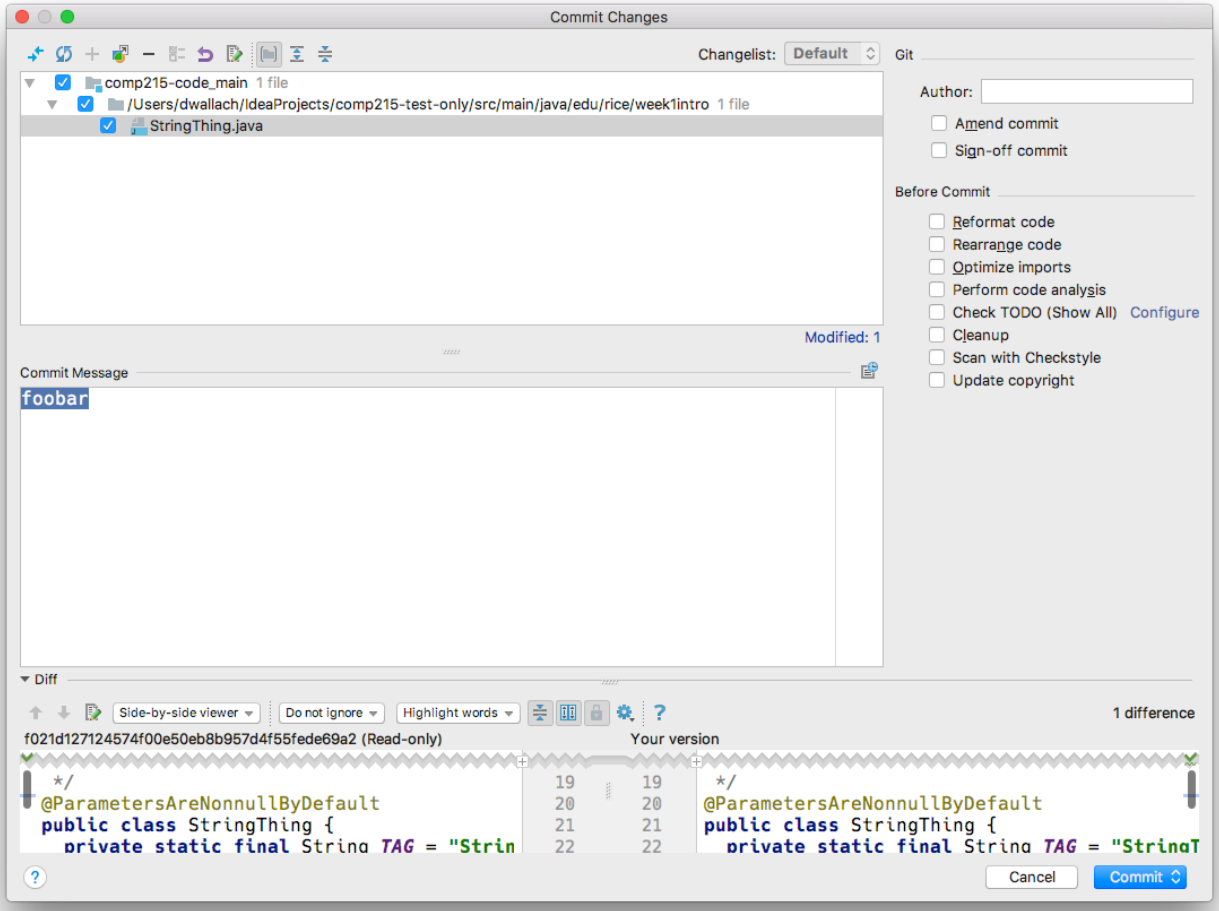
[6]Error-Prone is a sophisticated bug-finding tool. Lots of details here: http://errorprone.info/



- Every week, when you get a new Git repository to do your work, you won't have to reinstall the plugins, but you will have to switch to the Comp215 code style and Error-prone Java compiler. Unfortunately, we cannot automate those two selections for you. What we will do, however, is run the same checks "in the cloud" to help make sure you don't forget to do them locally. More on that coming soon.

11 Commit and push your changes to GitHub

- Now that you're passing the tests, it's time to *commit* your changes and *push* them to GitHub. You'll do VCS → Commit Changes (or just hit Command-K or whatever it is on your computer). At that point, you'll see something like this:



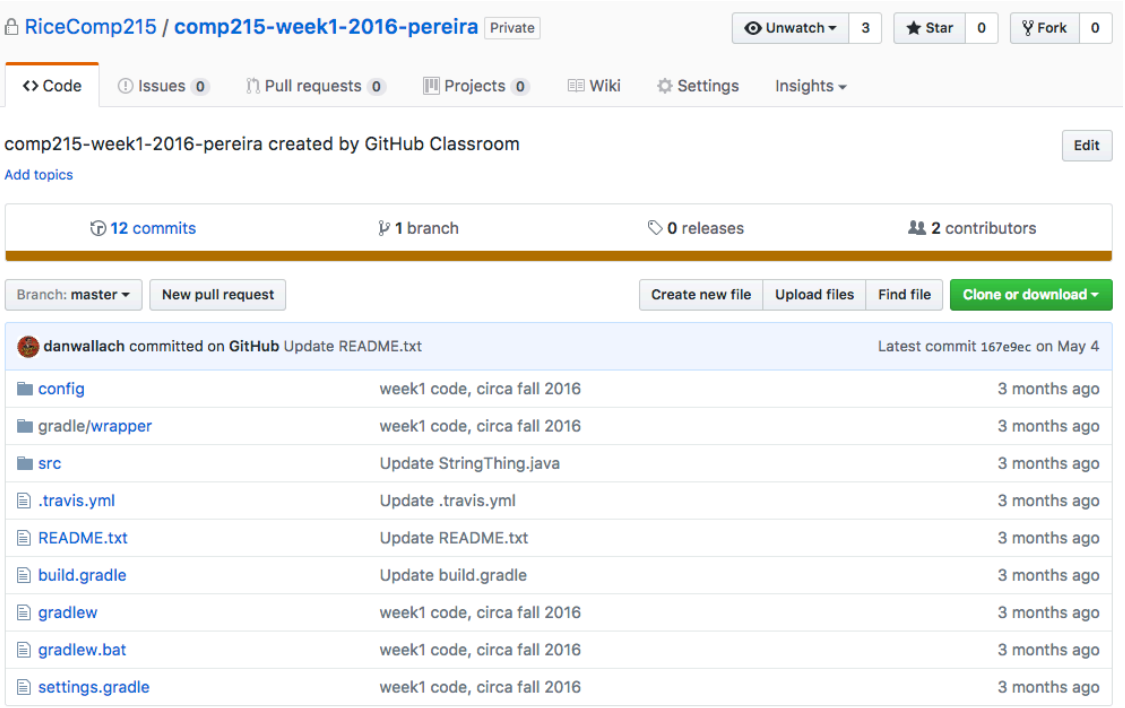
Get in the habit of putting something more useful in the "commit message" than just "foobar", since you'll later be looking at your own history of changes and it's invaluable to have meaningful text here. Also, uncheck the "perform code analysis" and "check TODO" options. You don't need those.

The Commit button is actually a pop-up menu, and what you want is Commit and push. The former is like a fancy "save" button for your own computer and the latter "pushes" your changes from your computer to GitHub where your graders will be able to see them.

If you've never used Git before, IntelliJ will ask you for your name and email. This could be anything, but it's helpful for us if you use your Rice email address and the name the registrar uses. You can always change this later, but GitHub will show us whatever you enter.

If you forget to push at the same time as you commit, there's a separate menu item: VCS → Git → Push... which will do the right thing. **You should always push your changes immediately. This protects you if your computer breaks!**

- Finally, **let's verify that your commit worked.** Visit https://github.com and click to your repository. You should see something like this:



  Click the button on the left side that says "12 commits" (or however many commits there are), and it will show your commit history including those commit messages and the dates when you committed them. If it's missing from GitHub, perhaps because you "committed" but didn't "push" your code, then all you have to do is run VCS → Git → Push....

- Your lab assignment, which is worth all of one point, is to make the necessary edits to `StringThing.java`, as above, verify that the tests run without error, and then commit/push the changes. **You'll demonstrate to your labbie that the tests pass locally and that your changes are visible on GitHub. The labbie will check you off and you're good to go.**

- We'll have a separate "project" for you this week, due Sunday night. You'll do that work in the same Git repository that you checked out for your lab, although it will be in a different Java package. (Java packages are kind of like file directories to separate and organize your code.)

12 On your own time

At this point, you're perhaps wondering why we're doing all of this checking stuff on week 1. Seems a bit extreme, right? In subsequent weeks, we're actually going to crank it up even more. Comp215 isn't just about writing code you *think* is correct. It's about writing code that passes a battery of

tests: unit tests (that verify your output is correct), coverage tests (that verify your unit tests exercise enough of your code), and a variety of compile-time tests, like CheckStyle, Error-Prone, and other tests that are built into IntelliJ, which identify common bug patterns.

Ask yourself the question, "what does it mean to say my code is *correct*?" Is your code correct because you got full points on the assignment? Is it correct because you wrote a bunch of tests and it passed them? Is it correct because you anticipated some future need correctly and it worked for you a month later? Is it correct because you came back to your code a year later and your comments and coding style were good enough to remind you what you were thinking?

Our goal is that you write code that works in all of these ways, and more.

Meanwhile, here are a few useful things for you to do.

12.1 Fonts

We recommend you install the *Hack* font (http://sourcefoundry.org/hack/) and configure IntelliJ to use Hack as your code-editing font. Hack is carefully engineered to make it easy to read for programmers, using decidedly different glyphs for zero-0 vs. capital-O as well as the number one-1, the lower-case-l, and a capital-I. Here are Times Roman, Helvetica, and Hack, side-by-side, with these letters: 0O1lI vs. 0O1lI vs. 0O1lI. Hack's extra legibility will help you avoid typos that can be painful to debug. You should be able to find font-installation instruction online with your favorite search engine.

12.2 Advanced: Using the command-line

As a Comp215 student, you will normally never use the command-line, but it's there and it's fun to be able to run your tests and interact with Git, all without moving your mouse.

If you've never used the command-line before, then this is something you'll benefit from learning on your own time. CodeAcademy has an entire course on this: https://www.codecademy.com/learn/learn-the-command-line. See also: https://www.learnenough.com/command-line-tutorial. Windows users: you will have installed the `bash` shell as a side-effect of installing Git, above. You'll have everything you need.

Note that using the `git` command creates several username/password headaches for you that IntelliJ normally hides. The solution is to set up Git to work well with `ssh` (secure shell). GitHub has a page walking you through the process: https://help.github.com/articles/connecting-to-github-with-ssh/. You don't have to do this as part of this week's lab. See also Section 12.3, below, for a discussion on more advanced Git usage within IntelliJ.

12.3 Advanced: Digging into Git

IntelliJ turns out to have a huge set of Git-related features. You might enjoy this guided tour: https://www.jetbrains.com/help/idea/using-git-integration.html. Again, you're not expected to understand *any* of this, but you may find several specific things in here to be useful, such as how to undo a commit, or revert a file back to a previously committed version.

For Comp215, we strongly advise you to stay away from the fancy features of Git, such as working on multiple branches at the same time. At some point, when you may find yourself working at a real company, you'll have to deal with all of these things and every company will have specific local instructions for what you should and shouldn't do. For example, many large projects follow some version of GitFlow[7]. For now, keep your life simple.

If you feel that you *really* want to understand Git, there's a free book you can read: (https://git-scm.com/book/en/v2). Furthermore, the Internet can be remarkably helpful. If you're trying to do something fancy with Git and it's not working, almost any question you might ask of Google will lead you to a StackOverflow page with a useful answer.

[7]GitFlow is a standardized way of managing a commercial software development and release cycle as applied in Git. There's been a lot written on this. One example: https://datasift.github.io/gitflow/IntroducingGitFlow.html