CSE 331
Software Design & Implementation

Spring 2020

Section 2 – Development Tools

Course resources

- We can't cover everything in an hour

- Read documentation: cs.uw.edu/331 > "Resources" tab
  - "Project Software Setup"
  - "Editing, Compiling, Running, and Testing Java Programs"
  - "Version Control (Git) Reference"
  - "Assignment Submission"

- The resources page is a treasure trove of helpful information!

- And we've got videos.
  - Look for the Videos links on the resources page below Tools.
  - Helpful if you're still stuck after the section demo.

Software You Need

- Java 11
  - adoptopenjdk.net
  - Choose “OpenJDK 11” and “HotSpot”
  - Windows: Select ”Add to PATH” and “Fix Registry” during install
- IntelliJ
  - jetbrains.com/idea
  - Recommended: Ultimate version
    - Comes in handy later in the course
    - Free for students, see course website for link to license
  - **Install the latest version**
- Git
  - git-scm.com
  - (Slightly newer version than the XCode command line tools on macOS)
  - Comes with Git Bash on Windows – important!

Warning: You must use JDK 11+

- Must use JDK version 11 or later
  - Be sure that's what you have installed!
  - Download links in Resources webpage
  - Use the AdoptOpenJDK installers (only)

- An out-of-date JDK can lead to very confusing bugs
  - No fun for either of us!

IntelliJ

- The officially supported editor for this course
  - Full setup instructions in “Project Software Setup” handout

- A modern IDE, commonly used in industry
  - Get the “Ultimate” version – free license for education use

- IDE = “Integrated Development Environment”
  - Auto completion
  - Version-control (git) integration
  - Debugger integration
  - …and an assortment of other fun features

- Necessary functionality covered in course documentation
  - “Editing, Compiling, Running, and Testing Java Programs”

Version control

- Also called source control, revision control
- System to track changes in a project codebase
  - Unit of change ~ lines inserted/deleted across some files
- Essential for managing software projects
  - Maintain a history of code changes
  - Revert to an older project state
  - Merge changes from multiple sources

- We'll use **git** and **GitLab** in this course, but alternatives exist
  - Subversion, Mercurial, CVS
  - Email, dropbox, thumbdrives (don't even think of doing this!)

Version control concepts

- A **repository** (“repo”) stores a project’s entire codebase
  - Stored in multiple places and synchronized over the internet
  - Tracks the files themselves and changes to them over time

- Each developer **clones** her own **working copy** of the repo
  - Makes a local copy of the codebase, on her laptop/computer
  - She modifies these files directly, with her IDE or text editor

- Each developer **commits** changes to her working copy
  - Saves “a commit” to version control history
  - Affects only the local working copy
  - Must synchronize with remote repo to share commits each way

Essential git concepts

- **`commit`**
  - Saves (a subset of) the changes to the local repository
  - Has a brief message summarizing changes

- **`push`**
  - Sends local commits to the repository (on GitLab)
  - Allows other computers to then "pull" those commits/ changes, see below.

- **`pull`**
  - Synchronizes working copy to match the remote repository
  - **`clone`** = the first pull, also sets up the repository for the first time

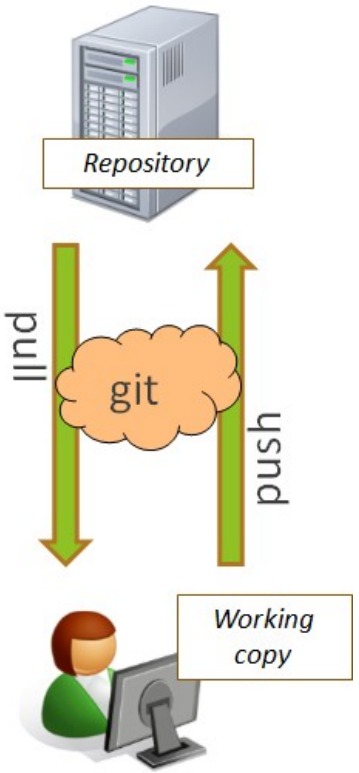
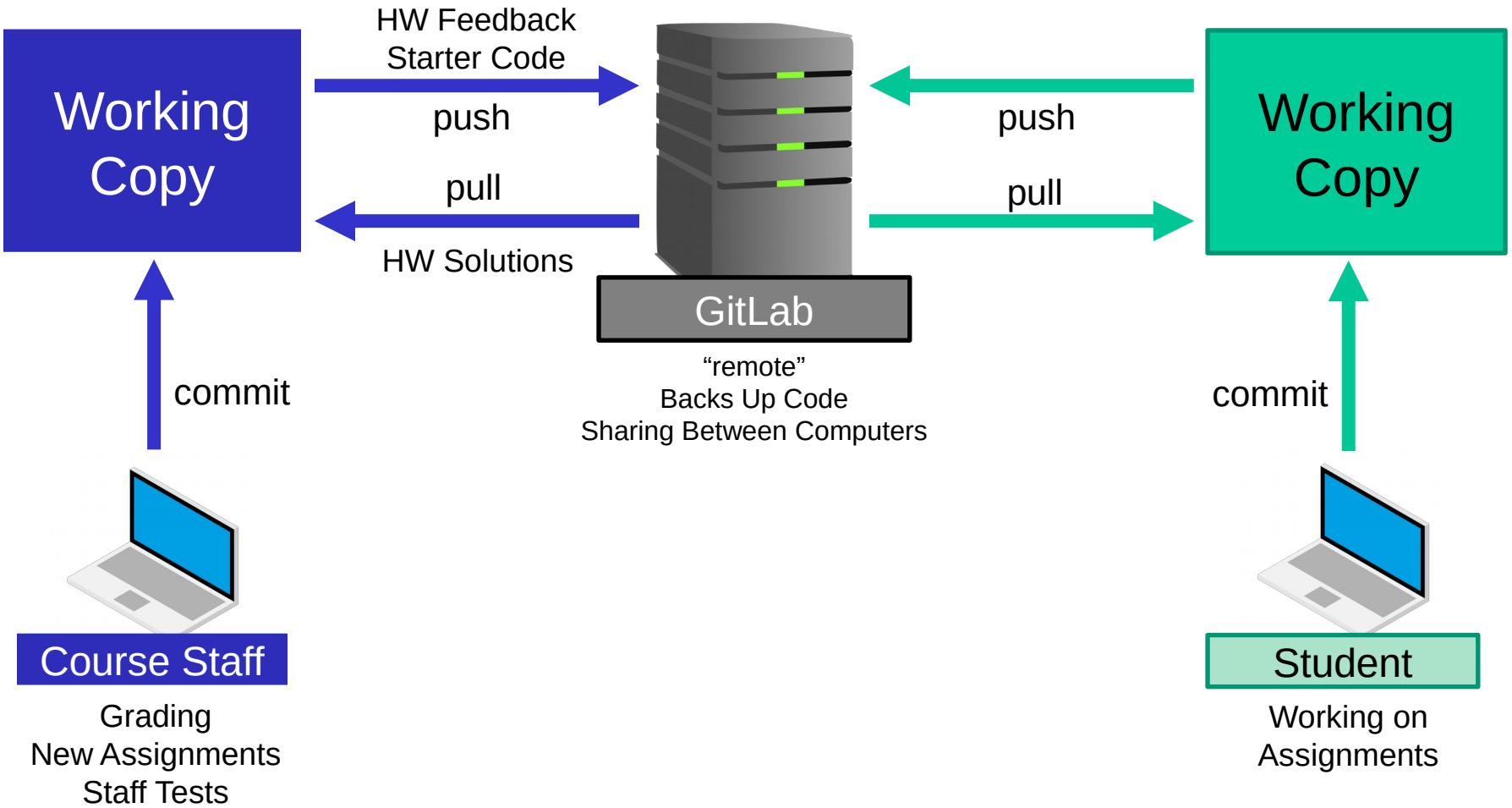


Diagram of git usage



Your GitLab repository

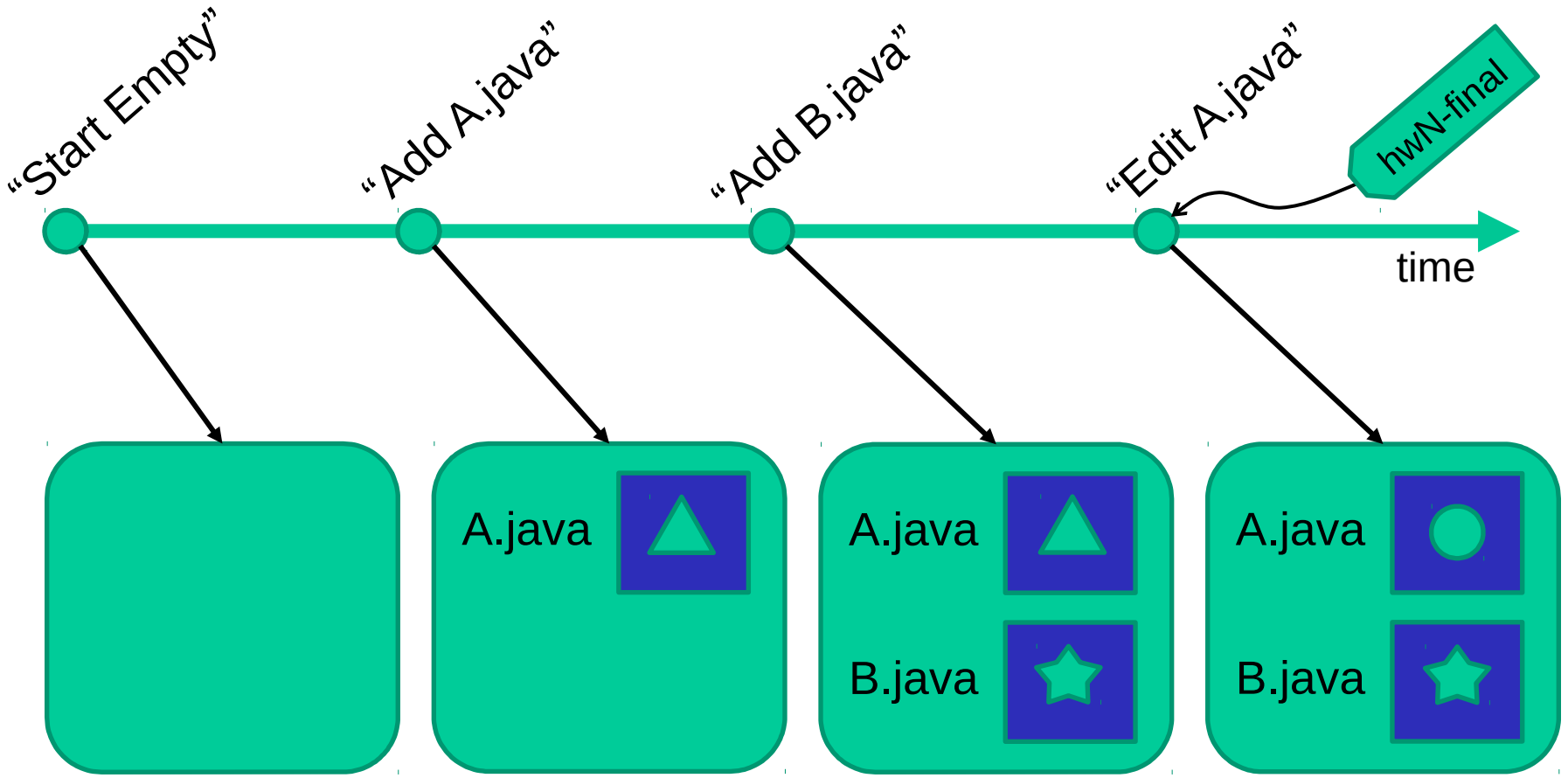
- We'll push starter code to your repo for each homework
  - After HW3, you'll get it by pulling

- Commit and push your code as you do the assignment
  - Recommended process: edit, test, pull, commit, push

- Submit homework ***N*** by creating a tag "**`hwN-final`**"
  - *Check that you've committed and pushed all your work!*
  - Do not attach a message with the tag
  - Example: "**`hw3-final`**" for HW3

- Without the right tag, your homework might not be graded!

Example commit history



Best practices when using git

- Pull/Commit/Push your code *early* and *often*!!
  - You really, really don't want to deal with merge conflicts
  - Best to pull before you commit (in 331, industry is more complex)
  - Keep your repo up-to-date as much as possible

- Do not rename files and folders that we gave you
  - That will mess up our grading process
  - It would be a silly reason to lose points!

- Use this repo just for homework

Gradle: what is it

- Gradle is a tool for build automation
  - Simplifies compiling, running, and testing a software project
  - No need to install: included in the starter code!

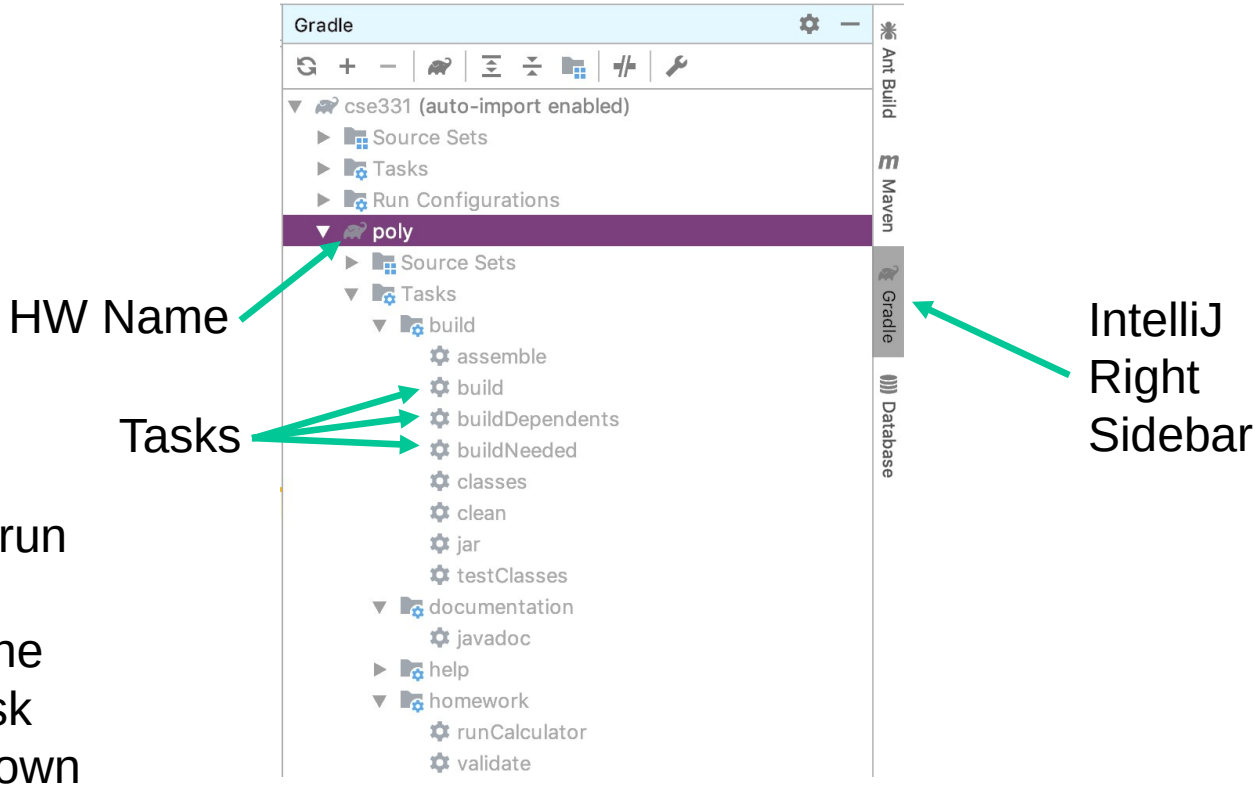
- Configured by the file **`build.gradle`** (and others) in your repo
  - You shouldn't modify this (can interfere with grading)!
  - Ask the course staff for help if it got messed up accidentally.

- IntelliJ has built-in support to work with Gradle

- Gradle is how you run/validate your code on attu

Gradle: how to use it

- You can use Gradle at the command line or in IntelliJ (recommended)
  - Every homework assignment has a “name” – HW3 is “hw-setup”



IntelliJ Gradle Panel

- Double-click tasks to run them.
- Make sure you’re in the right assignment’s task list, each one has its own tasks.

Let’s Try It!

Get your computers out and start up
Terminal (macOS) or Git Bash (Windows)

Getting Connected to GitLab

- Generate an RSA key pair:

  `ssh-keygen -o -t rsa -b 4096 -C "your@email.com"`

  - The (-C) comment can be any string, make it something you'll recognize.
  - Press enter when asked for a file name (use default)
  - No passphrase
  - You'll be told: "Your public key has been saved in (…)"

- Copy the generated public key (use the file name of the public key from above, if different)

  `cat ~/.ssh/id_rsa.pub | clip` (Windows)
  `cat ~/.ssh/id_rsa.pub` (macOS/linux)

- macOS/linux: Select and copy the output of running the **`cat`** command

Getting Connected to GitLab (2)

- Paste that into your GitLab account, under “Settings” > “SSH Key”
  - Sign in at: gitlab.cs.washington.edu
- In Terminal/Git Bash, type the following to check that you’re set up:

```
ssh -T git@gitlab.cs.washington.edu
```

- Getting “The authenticity of host (…) can’t be established”?
  - Type **`yes`** – only a one-time thing, the GitLab server is just unfamiliar to your computer.
- Should get a welcome message back!

Cloning Your Repo

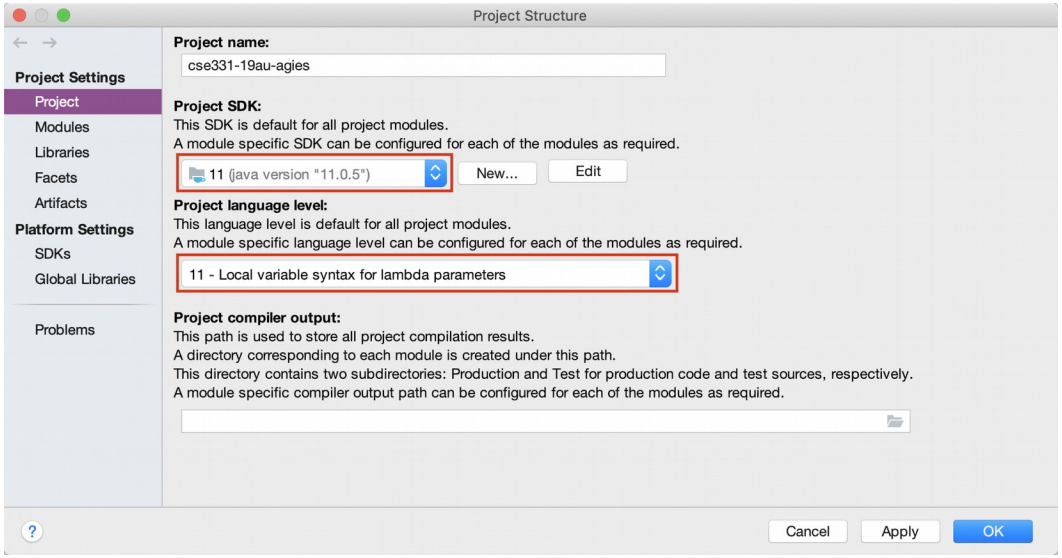
- In GitLab, open your project page and get the SSH clone URL

gitlab.cs.washington.edu/cse331-20wi-students/cse331-20wi-*NETID*

- Blue ”Clone” button in top right: copy the “Clone with SSH” URL
- Open IntelliJ
  - You don’t need any plugins or launcher scripts, skip those steps
- Choose “**Check out from Version Control** > **Git**”
- Paste the clone link from earlier, click “Test” and make sure it works.
  - Choose a place on your computer to set up your code.
- Click **Clone**
- “Would you like to open it?” **Yes**

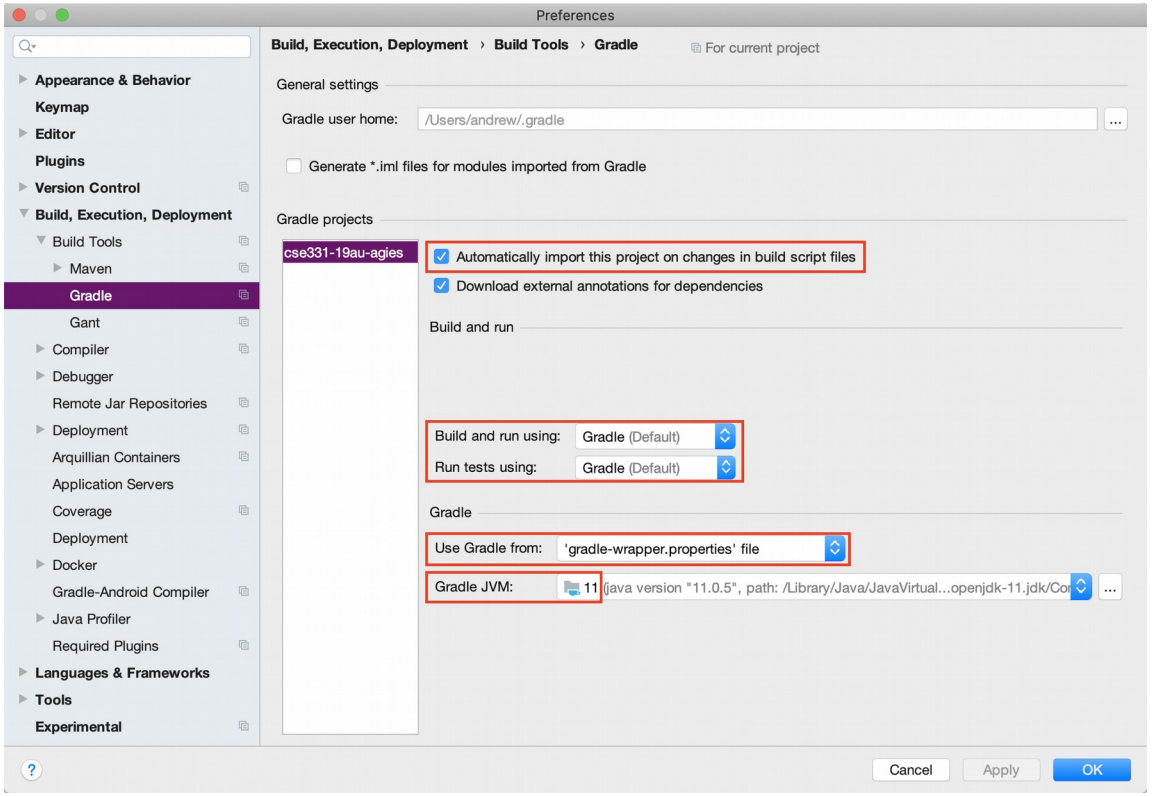
Importing Into IntelliJ

- Need to set up project SDK: Select Java 11
  - **File** > **Project Structure** > **Project**
- Missing?
  - Click **New** > **JDK**, IntelliJ should auto-find your Java 11 install
  - Can't find it? Check your Java installation and ask for help.



Importing Into IntelliJ (2)

- Also, need to check some Gradle settings in IntelliJ preferences:
  - **Build, Execution Deployment** > **Build Tools** > **Gradle**



Development Workflow Demo

1. Open the first part of the hw3 starter code:
   - **hw-setup/src/main/java/setup/HolaWorld.java**
2. Fix the two bugs in this code: Lines 35 & 43
3. Run the code using Gradle:
   - Open the Gradle panel on the right edge of IntelliJ
   - Provided a runHolaWorld Gradle task under the “homework” group
   - **cse331** > **hw-setup** > **Tasks** > **homework** > **runHolaWorld**
4. Double-click to run the task: see the output at the bottom!
   - Gradle automatically compiles your code and then runs it.

Development Workflow Demo (2)

We've finished part 3 of the assignment (!) – let's commit this code to save it.

1. "pull" to make sure we have any updates that happened while we were editing:
   - **VCS** > **Git** > **Pull**
2. "commit" the changes to save them to our local copy of the repository:
   - **VCS** > **Commit**
   - **Check the boxes for the file changes you want to include in the commit (usually all files)**
   - Uncheck everything under "Before Commit" (just some IntelliJ warnings, you can keep them but it adds extra steps to the commit)
   - Enter a short (< 15 words), **helpful** description of the changes in "Commit Message"
3. "push" the changes to tell GitLab about the new commit:
   - **VCS** > **Git** > **Push**

Development Workflow Demo (3)

In general, only do this at the end of an assignment, but let's see how it works with a practice tag.

1. Create the tag with the correct name. For now, use **section-demo**. See assignment specs for the tags to use for each assignment.
   - **VCS** > **Git** > **Tag**
   - Enter a tag name. (**Tags are case-sensitive.**)
   - **DO NOT** include a message. (This can make the tags difficult to move later, if you need to.)
   - Tags are attached to the current commit (usually the most recent one you created, so you need to create tags *after* creating the commit you want to tag).
2. "push" the changes to tell GitLab about the tag (so the staff can see it!)
   - **VCS** > **Git** > **Push**
   - Make sure "Push Tags" (bottom left) is checked. (Choose "All")

Development Workflow Demo (4)

Need to check that our assignment was submitted successfully. Checks happen in two places **(always do both checks**):

(1) attu

- Run your code in the same place we'll be grading it!
- Sign into attu: `ssh` *`NETID`*`@attu.cs.washington.edu`
  - Clone your repo, checkout tag, and run the gradle task
  - See Assignment Submission handout for instructions.
- Since you're testing on a new clone, it'll only have the files that are in git.
  - Makes sure you didn't miss any when making commits. (Common error in 331, can make assignments impossible to grade.)

Development Workflow Demo (5)

(2) Gitlab Runners:

- Triggered when you push the tag
  - Don’t see a runner? Make sure you have the right tag name! (Tags are case-sensitive)
- Runs some sanity checks (build, javadoc, and your tests) to look for common errors.
- If your runner fails, you should *definitely* fix it, then re-push a new tag and check the runner again.
- Open your gitlab project online, go to CI/CD Pipelines
- For **section-demo**, you’ll see a message and the pipeline should pass.
- For actual assignments, you’ll see it run checks on your assignment, then it’ll either pass or fail and print an error message on failure.

Important Handouts

https://cs.uw.edu/331/resources.html

- Project Software Setup handout
  - Important settings for IntelliJ (**you need to set these**)
  - Running your code on attu, in a virtual machine, or on remote desktop.
- Running/Compiling/Testing/Editing
  - How to use Gradle to run automated tests and see test results in IntelliJ.
  - [Optional] SpotBugs: A useful tool for finding bugs in your code
- Version Control handout
  - Git best practices, instructions, and more advanced usage
- Assignment Submission handout
  - Creating and moving tags, using late days, GitLab validation