Introduction to Course Infrastructure & Java

During this course you will become familiar with several popular and industry-relevant software engineering tools, including IntelliJ (or Eclipse), Git, GitHub, Gradle, Travis CI, SpotBugs, and Checkstyle in addition to Java. In this recitation you will set up your environment with the assistance of your recitation TA.

This recitation document is structured as a series of steps that will lead to a fully-fledged Java setup. If you encounter problems along the way, please alert your recitation TA, post on Piazza, or attend office hours.

Course Information

- **Ensure you are enrolled in the course Piazza.** This course uses Piazza as a discussion forum. We encourage you to make public posts whenever possible and to answer your classmates' questions. On Piazza you will find a version of this document with clickable links.
- **Familiarize yourself with the course website.** The course website is your reference for course information, including the syllabus, links to reading assignments, and the lecture schedule.

Shell preliminaries. At several points in this guide, we will list a series of commands for you to run in your shell. On Mac and Linux, you can use the built-in terminal as a shell; on Windows, it is sufficient to use Command Prompt. By convention, we prefix these commands with a dollar sign (`$`) to indicate that they are to be entered into the shell; however, when entering the command, you should not include the dollar sign.

If you use Mac or Linux, we instruct you at several points in this guide to add a directory to your `PATH` environment variable (sometimes just called your "path"). This involves three steps:

1. Open your `~/.bashrc` or `~/.bash_profile` file in a text editor.
2. Add a line to the bottom of the file that appends the desired directory to the end of the `PATH` environment variable. For example, if you wanted to add `/opt/gradle/gradle-6.8.1/bin` to your path, you would add the line:

   ```
   PATH="${PATH}:/opt/gradle/gradle-6.8.1/bin"
   ```

3. Ensure your session of the shell is in sync with the file's contents by running:

   ```
   $ source ~/.bashrc
   ```
   (or `~/.bash_profile`)

Java Platform

- **Download and install Java 11.** The language used in this class is Java 11. Several vendors provide implementations of Java 11; we recommend installing the open-source OpenJDK 11. The easiest way to do this varies by OS.
  - **MacOS.** Download the MacOS tar.gz archive from the OpenJDK website. Untar the archive, and move the contained directory (named something like `jdk-11.0.1.jdk`) to the `/Library/Java/JavaVirtualMachines/` directory.
  - **Linux.** Download the Linux tar.gz archive from the OpenJDK website, untar the archive, and add both the untarred archive and its `bin` directory to your path. For example, if the untarred contents of the archive are placed at `/usr/lib/jdk-11.0.1`, you would append the paths `/usr/lib/jdk-11.0.1` and `/usr/lib/jdk-11.0.1/bin` to your `PATH` environment variable.
  - **Windows.** To install OpenJDK, download the Windows zip file from the OpenJDK website, and follow the instructions at this StackOverflow post to correctly set Windows environment variables. Alternatively, if you wish to just use an install wizard, download and install the Oracle JDK.

Checkpoint. Confirm your Java installation by inspecting the output of the command `java -version`. You should see something like:

```
openjdk version "11.0.1" 2018-10-16
```

Integrated Development Environments (IDEs)

- **Install IntelliJ or Eclipse.** IDEs provide a more comprehensive set of features than classical text editors for Java development. You are required to use an IDE for development in this class. There are several IDEs available, but the most common ones (and the only ones supported by the course staff) are IntelliJ IDEA Community Edition and Eclipse. If you haven't already, you should install one of these. Ask your recitation TAs which IDE they prefer.

Checkpoint. Launch your IDE and try to create a new Java project. The flow for creating a new project varies somewhat between versions of Eclipse and IntelliJ, so try your best to follow the prompts onscreen.

Git and GitHub Git is a distributed version control system popular for large software projects, and GitHub is a hosting service for Git repositories ("repos"). We will be using GitHub and Git to distribute homework assignments, for you to turn in your homework, and for us to give you grades and other feedback on your work. The basic idea is that you will *clone* (make a copy of) your GitHub repo on your local computer. You can then *pull* changes from GitHub to receive our feedback and any new homework assignments, make local *commits* on your own computer to complete your homework, and *push* your completed homework back to GitHub for us to grade.

- **Download Git, if you don't have it already.** A command line interface (CLI) for Git comes pre-installed on many non-Windows systems. If you don't already have Git, you can download either the CLI or a graphical user interface (GUI) such as GitHub Desktop.

Checkpoint. To see the version of Git installed on your system, run:

```
$ git --version
```

Setting Up Your Repo Each student in 17-214 is assigned a GitHub repo. In this section, you'll set up your repo and clone it to your local system.

- **Create a GitHub account.** You may use an existing account if you wish.
- **Fill out the web form at this link.** After filling out that form, confirm your email. Upon confirmation, a GitHub repo will be set up for you and you will receive an email from GitHub asking you to join the 17-214 organization. If you do not get this email, go to `https://github.com/CMU-17-214/` and join the organization.
- **Clone your course repo.** From the directory where you want to place your course materials, run the command:

  ```
  $ git clone https://github.com/CMU-17-214/your-andrew-id
  ```

  where *your-andrew-id* is your Andrew ID. This will create a local copy of your Git repo in a directory with your Andrew ID as its name. In this directory you will do all of your work for 17-214.

- **Practice pulling changes to your local copy.** When changes are pushed to GitHub from another copy of your repo, you must, in turn, pull those changes to your local copy. You will need to do this when the course staff pushes assignments, recitations, and grades, or when you have pushed work from another clone of your repo. From any directory within your Git repo, run the command:

  ```
  $ git pull
  ```

  This command won't do anything yet—you just made a fresh clone of your repo, after all. It will come in handy once the course staff adds more materials to your repo, though.

Build and Test Automation

- **Install Gradle 6.8.1.** Gradle is a build tool that automates many aspects of the development process, allowing you to build, analyze, and execute your code from the command line. It will later be useful to automatically manage dependencies.

  The linked guide contains instructions for either installing with a package manager like Homebrew or installing the binary-only version manually. If you install manually, you must likewise manually add the unzipped archive's `bin` directory to your path. For example, if you place the unzipped directory at `/usr/local/gradle-6.8.1`, you will append `/usr/local/gradle-6.8.1/bin` to your `PATH` environment variable.

Checkpoint. Confirm Gradle's version to be 6.8.1 by running `gradle -v`.

- **Import Recitation 1 into your IDE.** Recitations and homeworks are distributed as individual Eclipse projects, which may be imported into Eclipse or IntelliJ.
  - **Eclipse.** From `File > Import > Existing Files into Workplace`, select the directory `recitation/01`. The appropriate project will appear to import.
  - **IntelliJ.** From `File > New > Project from Existing Sources` (or after choosing "Import Project" from the launch screen), select the directory you wish to import as a project (for example, `recitation/01`). Then, choose "Gradle" as the external module from which you're importing. Click through the prompts to finish importing.

    IntelliJ may require you to enter the location of Gradle before proceeding. Here are some things you can try.
    - If you manually installed Gradle to a directory like `/usr/local/gradle-6.8.1`, you can enter that path directly.
    - If running the command `which gradle` prints a path ending in `/bin/gradle` (like `/usr/lib/gradle-6.8.1/bin/gradle`), you can truncate that suffix to find the Gradle location (like `/usr/lib/gradle-6.8.1`).
    - If you installed Gradle with Homebrew, you can run `brew info gradle` to find the path where Gradle is located. Sometimes, you will need to append `/libexec` to the end of this path for the Gradle location.
    - Try this StackOverflow answer for more avenues of attack.
- **Familiarize yourself with what Gradle can do.** When you are in a directory with a `build.gradle` or `settings.gradle` file, you may use the `gradle` command to run *tasks*, which are actions to perform on a project, such as compiling and testing code, generating Javadoc, and generating Eclipse project files. Furthermore, multiple tasks can be composed into a single task.

  You'll get to know a few tasks very well:
  - `build`: Compile the Java source code in this project.
  - `clean`: Delete build artifacts.
  - `javadoc`: Generate Javadoc API documentation for the source code.
  - `test`: Run unit tests.
  - `check`: Run unit tests, then perform additional static analysis on the source code as specified in the `build.gradle` file.

Checkpoint. Navigate to `recitation/01`, and inspect the output of running:

```
$ gradle build
```

- **Familiarize yourself with Travis CI.** Travis CI is a web service that can execute commands on your GitHub repo each time you push, sending you an email if the commands fail. In particular, we have Travis CI run Gradle when you push to confirm that your unit tests succeed. These tools enable you to maintain the integrity of your code even as you add new features.

  We will be connecting your GitHub repository to Travis CI soon. You will receive an email saying "The following SSH key was added..." when Travis CI is ready for you. Then, you can check the output of the latest build at `https://travis-ci.com`.

- **Install Checkstyle in your IDE.** Checkstyle is a development tool used to identify where Java code does not adhere to style conventions. It automates the process of checking for common style flaws such as the use of magic numbers. For this course, we will be following a subset of the Sun Code Conventions. Checkstyle is set up to run automatically with Gradle and will cause your build to fail when the guidelines are not followed. For each homework assignment, the course staff distributes a configuration file that Checkstyle uses to determine what constitutes a style violation.

  We recommend installing the Checkstyle plugin in IntelliJ or Eclipse to receive immediate feedback in the IDE.

  - **Eclipse.** From `Help > Eclipse Marketplace`, search for "Checkstyle" and choose *Install*. Follow the prompts to finish installation.
  - **IntelliJ.** From `File > Settings > Plugins`, search for "Checkstyle" and install it.

  For Eclipse, we provide a `.checkstyle` file with each assignment such that the course configuration file is automatically loaded. In IntelliJ, you will have to load the configuration file manually once you have imported a project. Go to `File > Settings > Checkstyle` and use the plus sign in the "Configuration File" dialog box to browse and select the configuration file `config/checkstyle/checkstyle.xml`. If necessary, change the Checkstyle version to 8.7 in `Settings > Checkstyle` before loading the configuration file.

Checkpoint. From your IDE, run a Checkstyle scan to confirm that the TODO comment in `Example.java` is a flagged as a style violation.

Exercise: Java Practice After you have imported the `recitation/01` project to your preferred IDE, examine the `Example` and `Main` classes. Complete the `printList` method for the `Example` class.

> **Checkpoint.** Confirm that running the `main` method from the `Main` class produces the desired result.

Turning in your work

After you are done working within a clone of your repository, you can turn in your work with

```
$ git add file1 file2 dir1
$ git commit -m "Completed recitation 01"
$ git push
```

where *file1*, *file2*, and *dir1* are the names of files and directories you have added or changed, and the commit message (after the `-m`) is an arbitrary message describing your work.

1. The command `git add` instructs Git to track changes to a set of files in your clone; this is called adding the files to your *staging area*. If you pass a directory name to `git add` then all the files added or changed in that directory and all subdirectories (recursively) will be tracked and staged.[1] You can check what files are staged with the `git status` command.
2. The command `git commit` records all the locally-staged changes as a new version of the repository, along with a message that describes the new version. You can view recent commits with the `git log` command.
3. The command `git push` records the most recent committed version to the remote server, your repository on GitHub. GitHub will then automatically trigger a build on Travis CI.

Your work is not turned in unless you have completed all three steps. Check on GitHub to confirm that all your files are correctly pushed.

Each commit is a *local* checkpoint of your work which you can turn in when you push your repository to GitHub. If you push your repository to GitHub but have not staged

[1] Remember that `.` denotes the current directory. So, `git add .` will add all files in the current directory to the staging area.

and committed your changes, those changes will not be pushed to GitHub. Commits also provide a way of rolling back to an earlier version of your work.

When you have finished a unit of work, it is a good practice to commit it to your repository by doing the above. You should commit often and write useful commit messages. It is common to commit multiple versions locally before pushing your work, although you might want to periodically push your work to GitHub before you finish your homework to back up your work.

If you attempt to push your repository to GitHub but the GitHub repository has changed since you last ran `git pull`, your push will fail. To fix this, pull the other changes from GitHub (using `git pull`) and attempt your push again.

When you are done pushing your work to GitHub, you should use GitHub's web interface to confirm that your files are in the expected state. Alternatively, you can create a new clone of your repository (using `git clone`) in a new location on your computer, and test your solution in that new location. This method allows you to test exactly what the TAs will test when they clone your repository from GitHub.

Don't push changes to a homework subfolder after the deadline (unless you intend to spend part of your "late" budget). We will use the GitHub timestamp of the latest `git push` event to determine if you were on time.

Additional Resources

Here are some resources if you are interested in learning more about Git. Apart from being used heavily in this class, Git is popular in both the industrial software development and open source communities. Version control is a powerful tool in software development, and mastering it will remove many headaches that you would have encountered otherwise.

- 15-Minute Tutorial: https://try.github.io/
- Pro-Git Book (free): http://git-scm.com/book/en/v2/
- GitHub Game (free): http://pcottle.github.io/learnGitBranching/