Recitation 2: IDEs, Build systems, Libraries, CI

During this course, you will become familiar with several popular and industry-relevant software engineering tools, including an IDE, build systems, Travis CI, and automated checkers for the programming language of your choice. In this recitation, you will set up your environment and explore the tools with the assistance of your recitation TA. If you encounter problems along the way, please alert your recitation TA, post on Piazza, or attend office hours.

Setting Up Your Repo

Each student in 17-214/514 is assigned a GitHub repo. If you have not done so already, you'll set up your repo and clone it to your local system.

- If you do not have a GitHub account, create one.
- Follow the GitHub classroom link on Canvas/Piazza to create your own GitHub repository with the starter code.
- Clone the repository to your local drive and check out the Java or TypeScript branch (see previous recitation)

Installing the Language Compiler

Java

Download and install Java 16. The language used in this class is Java 16. Several vendors provide implementations of Java 16, if in doubt choose the open-source OpenJDK 16. If you already use a package manager for your platform (homebrew, apt, snap, scoop, etc) installing Java with that tool is probably the easiest.
Alternatively, you can install Java manually

- MacOS. Download the MacOS tar.gz archive from the OpenJDK website. Untar the archive, and move the contained directory (named something like jdk-16.0.2.jdk) to the /Library/Java/JavaVirtualMachines/ directory
- Linux. If possible, please really use the package manager of your distribution. It will be much easier.
- 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 "16.0.2" 2021-07-20

TypeScript

If you choose to use TypeScript, you need to install Node.js and the TypeScript compiler. Follow the instructions here for more information.

To install TypeScript you can run "`npm install -g typescript`" and afterward "`tsc`" should be available on your command line. (The project we provide already sets up TypeScript, so "`npm install`" should already install TypeScript locally).

Checkpoint: Confirm that "`tsc --version`" works in your working directory (or "npx `tsc --version`" if you installed it only locally).

Integrated Development Environments (IDEs)

IDEs provide a more comprehensive set of features than classical text editors

Java

Install IntelliJ or Eclipse. You are required to use an IDE for development in this class. There are several IDEs available, but the most common ones (and the ones recommended 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.

Open the project in the IDE by locating the folder you just cloned, either via File -> Open, or (upon the first launch) when it prompts you for a folder. IntelliJ will take some time to load this project; expect about a minute of indexing and resolving dependencies.

- You need to have Java 16 installed for this project. If you have an older version, IntelliJ may "helpfully" suggest you downgrade the version in pom.xml, which will immediately break your build. Do not do this.
- If you have done this before in another folder, or e.g. before resetting your project, you may need to prompt it to "Invalidate Caches".

Checkpoint: You can explore the source files in the IDE.

TypeScript

We recommend also using an IDE for TypeScript. We recommend VSCode which supports TypeScript out of the box.

Open the project directory in the IDE using “Open Folder”.

Checkpoint: You can explore the source files in the IDE.

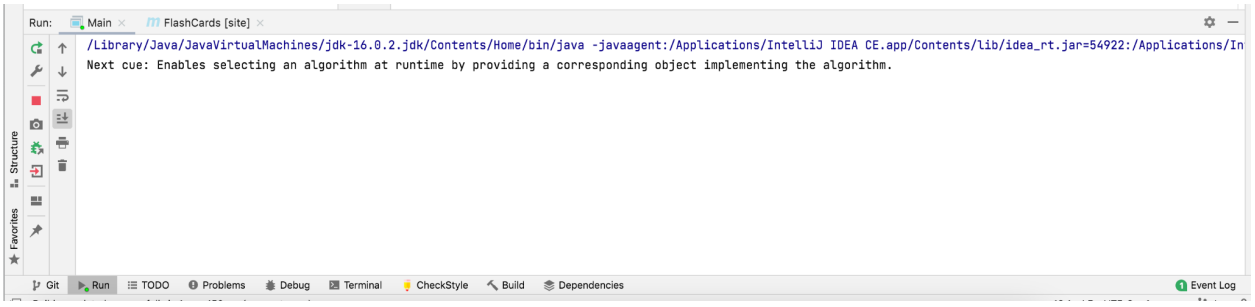
Running a Program in your IDE

Java (IntelliJ)

Once you have successfully opened the project, you should be able to view all the starter source files. Try building the program and run the main class.

To run a program:

Go to Main class, click the green triangle on line 12 (or on the top-right bar) and click “Run Main.main()”. You will be able to view the run result at the bottom:



You can interact with this program by replying on the console and hitting “enter”.

If you want to pass an argument to the program, click “Edit Configuration” and provide program arguments for the configuration of the “Main” program.

To start the Debugger, click the Debug instead of the Run icon.

TypeScript (VSCode)

VSCode executes code in the debugger by default. Select “Start debugging” from the menu and confirm to use node.js as the runtime. Note that the program will fail because the debugger does not support command line input, but everything until this part will execute.

If you just want to run the code use a terminal or the built-in terminal in VSCode (“New Terminal”) and run “`node dist/index.js`” to start the compiled program (or “`npm run start`” for the preconfigured run script in package.json)

Build and Test Automation

Java

Install a recent version of Maven. Maven is a build automation tool used primarily for Java projects. Follow the instructions in this link and verify that Maven 3.8.2 is installed. More information about Maven can be found here.

Explore the pom.xml file for how the project is currently configured.

Checkpoint: In your local directory, try to run “`mvn install`” which will download dependencies and compile the code.

TypeScript

We are using npm, which comes bundled with Node.js. npm does package management similar to maven, but works a bit differently:

- The package.json file is the configuration file, which lists dependencies and commands
- “`npm install`” installs all dependencies and puts them in a local directory node_modules, which should not be committed to version control and is automatically in the search path for imports from JS/TS code
- The package.json file has a dev-dependencies section, which are tools not needed at runtime. typically test execution, compilers, style checkers etc.

- "`npm install -g X`" installs package X globally, so not just for the current directory. This is often useful for tools like TypeScript that are installed through npm. `npx` is a useful tool for executing a tool installed locally and hence not otherwise in the path, like "`npx tsc`".
- The package.json file has a scripts section, which is just a way to describe command line instructions that can be easily run with "`npm run X`" from the command line. These are just convenient shorthands.

Checkpoint: Run `npm install` to install dependencies and tools locally. You can find them in the node_modules folder if you are interested.
Run "`tsc`" to compile the TypeScript code, you can find the output in the `dist/` directory.

Travis CI

Familiarize yourself with Travis CI. Travis CI is a web service that can execute commands on your GitHub repo each time you push. It can send you an email and/or update status trackers if the build fails, and provides detailed logs to find issues in. These tools enable you to maintain the integrity of your code as you extend it. Then, you can check the output of the latest build at app.travis-ci.com. The first time you go there, you can link your GitHub account to see builds of your repositories.

Exercise: Look into the .travis.yml file to see what Travis CI will be doing. Then push any code change to your GitHub repository and observe what Travis CI is executing (this might take a while to start if many students are working at the same time).

Linters

Check your code against a style-guide that specifies some good coding conventions (and sometimes annoying nitpicky things).

Checkstyle for Java

Checkstyle automates the process of checking for common style flaws such as the use of magic numbers. For this course, we have provided a custom, fairly minimal checkstyle.xml file in the Java branch. Checkstyle is set up to run automatically via Maven and will cause your build to fail when the guidelines are not followed.

Maven contains the checkstyle build artifact by default. You can just run "mvn site". In IntelliJ you can use View -> Tool windows -> Maven; then under FlashCards ->

Lifecycle, double-click “site” and you will be able to view the checkstyle result. The starter code will break by default, and if you read the associated error message, you will see that this is because: “Comment matches to-do format ‘TODO:’”. This is triggered by a checkstyle rule, which you may find in the style-file under the rule name “TodoComment”. Removing the TODO comment will resolve the error and pass checkstyle.

Checkpoint: Remove the TODO to see the linter pass or introduce a style violation to see it reported within your IDE.

TS-Standard for TypeScript

Similar to checkstyle, there is a (very picky) style checker called ts-standard. It comes preconfigured in the project with “`npm run lint`” or just run it with “`npx ts-standard`” (or just “`ts-standard`” if the package is installed globally). It has a fix option “`npx ts-standard --fix`” that will automatically fix many issues, which can be very useful. To integrate ts-standard into vscode, install the *StandardJS* extension and in the settings pick ts-standard as the engine.

Checkpoint: Install and configure the StandardJS extension. Introduce some style violation to see what it reports (like extra or missing whitespace). Try to fix it automatically with “`npx ts-standard --fix`”.

Explore IDE Capabilities

It is very useful to get to know your IDE and what it can do for you. In the remaining time we recommend to explore the following:

- Search for any commands in the IDE with *Shift+Ctrl+a* in IntelliJ or *Shift+Ctrl+p* in VSCode (may differ slightly on MacOS) -- a very powerful way of finding the right options quickly. Use it to start the debugger.

- Set breakpoints for the debugger and explore what information the debugger can show you at the breakpoint. Step through the program, into and over functions.
- Explore code navigation feature. Typically *Ctrl+Click* gets you to the declaration of code, typically there is a feature to find all places that call a method (“Find Usage”/”Go to References”).
- Commit code to Git directly from within your IDE. Most IDEs have Git support directly built in and provide nice user interfaces for previewing what to commit or selecting which files to commit.
- Explore automated fixes and code completion. In Java, write a field and let it generate setter and getter methods; write a variable without a type declaration and let it complete it. In TypeScript assign a variable without “let” or “const” or “var” and let it generate it, explore other “Quick Fixes” for coding problems. In either write “/**” above a method and see it complete a template for documentation. Write some code without the correct imports and see whether it can import those automatically.
- Explore refactorings, automated rewrites of code. See what refactorings are available and try them. Try at least renaming and “extract method”.
- Get familiar with common keyboard shortcuts for compiling, debugging (start debugger, step over, …), automated formatting, commenting out blocks of code, refactoring...

Show useful functionality you discover to the group.