Java Build Tools

App Design Lecture 9 Part 1

(presented by sam !!)

Maven™

What’s a build tool ? Why’s a build tool ??

Tools for managing a java project.

- Managing and downloading dependencies
- Building/running from source code
- Building releasable artifacts (like jars and javadocs)
- Managing large projects with multiple modules

“What’s wrong with javac and bash scripts”

- Manually managing dependencies is a pain
- Faster to use existing tools.
- Standardization is better for interop and collaboration

Popular Build Tools

- Maven
  - Older, well known
  - XML based
- Gradle *(our focus for today)*
  - More flexible
  - Groovy DSL or Kotlin DSL based
- Ant
  - Even older, not very commonly used.
  - Also XML based

Maven: A build tool

- Project Object Model (POM) defines the project
  - XML based
- Relies on plugins for Doing Things
  - Compiling, building docs, publishing, etc
- “Convention Over Configuration”
  - not the most flexible
- Documentation: https://maven.apache.org/index.html

Maven: An artifact publishing system

- **Maven Repositories:** decentralized servers that host published java modules and artifacts
- POM format describes project dependencies
- Modules identified by coordinates: `group.id:artifact-id:version.id`
  - ex: `com.google.guava:guava:33.2.1-jre` hosted at https://repo1.maven.org/maven2/com/google/guava/guava/33.2.1-jre/
  - Group id usually corresponds to project root package
  - Can usually find repo and coordinate on project readme or homepage
- Standard format supported by most java build tools.
- A simpler example: https://maven.blamejared.com/com/samsthenerd/inline/
  - (go look at the POM)

Activity: Maven Repo Hunting

- Find the maven repo and coordinates where some Java project is published.
  - Try looking for libraries or other projects meant to be depended on.
  - Try to find a project not covered so far in this lecture
- Take 5-10ish mins

Gradle: A better build tool

- A code based build tool
  - Build files written in Groovy DSL or Kotlin DSL - both similar-ish to Java
- Do Things with Tasks
  - Can be added by plugins or written in the build script
  - Easier to configure, customize, and extend.
- Documentation: https://docs.gradle.org/
- VSCode setup guide:
  - https://github.com/SamsTheNerd/psoft-vscode-gradle-lore
  - Originally written for psoft, but should be helpful here too

Gradle: Setup – Installing

- Gradle runs through a Gradle Wrapper that sits in your project directory. If you don't have one:
  - Generate it through an IDE
  - Grab it from another project
  - Or actually install gradle and generate the wrapper yourself.
- You can run tasks through the gradle wrapper: `./gradlew [taskname]`
  - This may vary by OS/Terminal. You just need to run the `gradlew` (linux/mac) or `gradlew.bat` (windows) script.
- You can see all tasks with `./gradlew tasks`
  - Some commonly used ones are `run`, `build`, and `test`

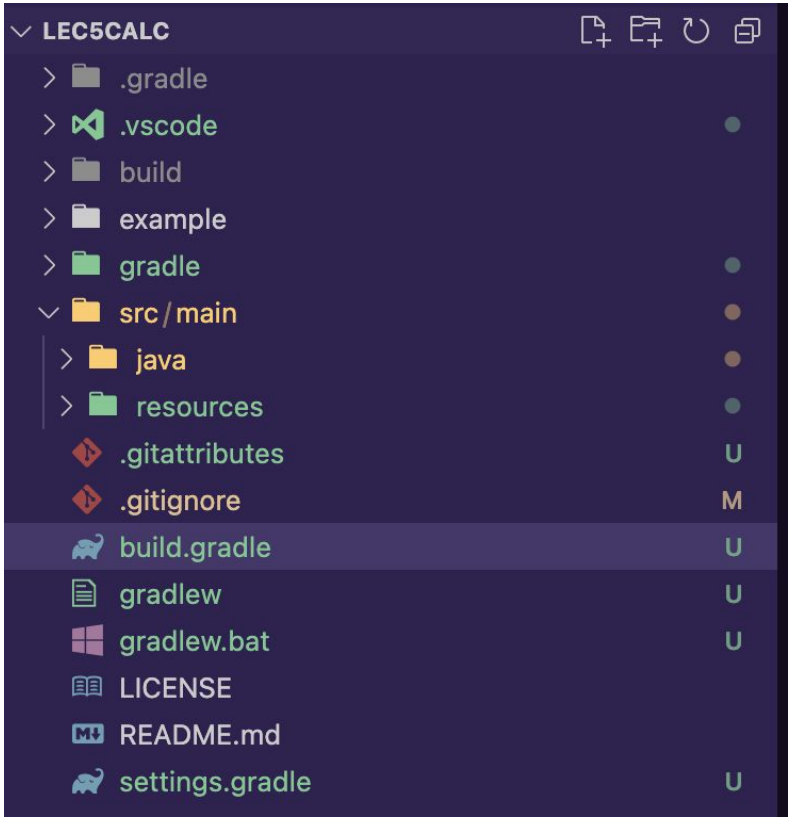
Gradle: Setup – Project Types

Either single or multi-module project setup.

- Multi module allows configuring, building, and publishing modules/subprojects separately. ex:
  - server/client/common
  - api/lib/gui
- Focusing on single project setup today.
  - See https://docs.gradle.org/current/userguide/intro_multi_project_builds.html for multi-module project info. Very similar just more build files and directories

Gradle: Setup – Structure

- `gradle/` – the gradle wrapper
- `gradlew` & `gradlew.bat` – gradle wrapper scripts
- `settings.gradle` – defines the gradle project
- `build.gradle` – project/module build configuration file. Most stuff happens here.
- `src/main/` – default source directory



Gradle: Setup – build.gradle

build.gradle > ...

```
1
2   plugins {
3       id 'application' // allows us to run our code
4       id 'org.openjfx.javafxplugin' version '0.1.0' // adds javafx to the project
5   }
6
7   // list maven repos to check for dependencies
8   repositories {
9       mavenCentral() // everything we need is hosted in maven central
10  }
11
12  // add dependencies using their artifact coordinates. These will be automatically downloaded.
13  dependencies {
14      // junit for testing
15      testImplementation 'org.junit.jupiter:junit-jupiter:5.9.1'
16
17      implementation 'com.google.guava:guava:31.1-jre'
18  }
19
20  application {
21      // Define the main class for the application.
22      mainClass = 'calculator.Main'
23  }
24
25  // configure javafx
26  javafx {
27      version = "22" // use version 22 of javafx
28      modules = [ 'javafx.controls', 'javafx.fxml' ] // tell it what modules we need
29  }
```

A sample build file for the calculator project

Automation & CI

Builds tools help you build from other computers too:

- Build/publish snapshots with each commit
- Publish releases automatically
- Using tools like Jenkins and GitHub actions
  - https://docs.gradle.org/current/userguide/jenkins.html
  - https://docs.gradle.org/current/userguide/github-actions.html

Jenkins