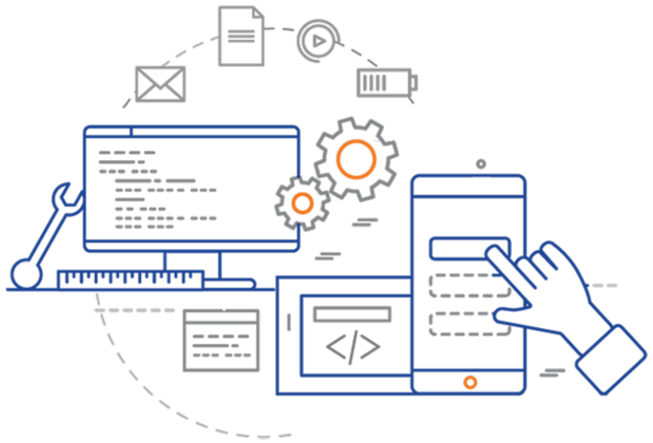


CS 398: Application Development

Week 05 Lecture: Infrastructure

Gradle and build systems

This Week

Goal: Get everything setup for Sprint 1 kickoff (next Mon Feb 7th in-class)

Lectures This Week

- **Mon:** Git, Branching, Collaboration
- **Wed:** Gradle, Build Systems
- **Fri:** Building desktop applications OR Building mobile applications **<— NOT on Quiz**

Lectures Next Week

- Unit testing, Refactoring

Review

Gradle

There are a number of build systems on the market that attempt to address these problems. e.g. cmake, scons for C++, Ant or Maven for Java.

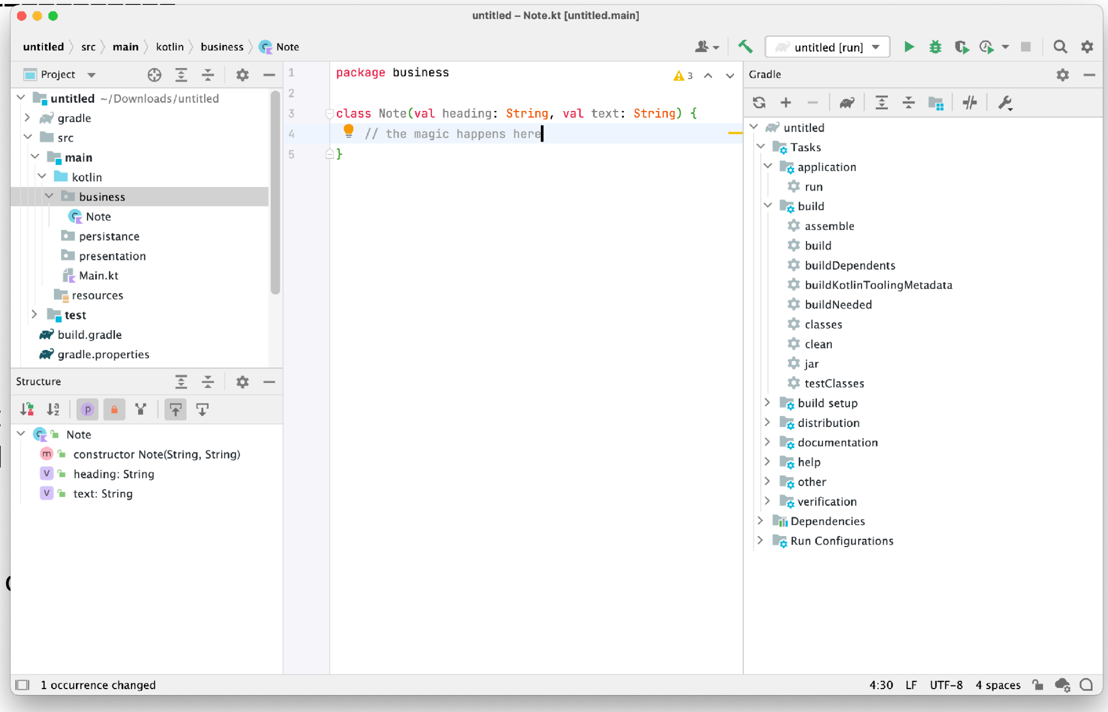
We’re going to use **Gradle** with Kotlin.

- It’s commonly used for large, complex Java and Kotlin projects.
- It handles all of our requirements, which is frankly, pretty impressive.
- It’s the official build tool for Android builds, so you will need it for Android applications.
- It’s declarative. You write Gradle build scripts in a DSL (Groovy or Kotlin), describing tasks to perform. Gradle figures out how to perform them.
- It handles multi-step builds and complex dependencies automatically! i.e. it tracks your libraries for you.

You can use gradle tasks to see all supported actions. The available tasks will vary based on the type of project you create.

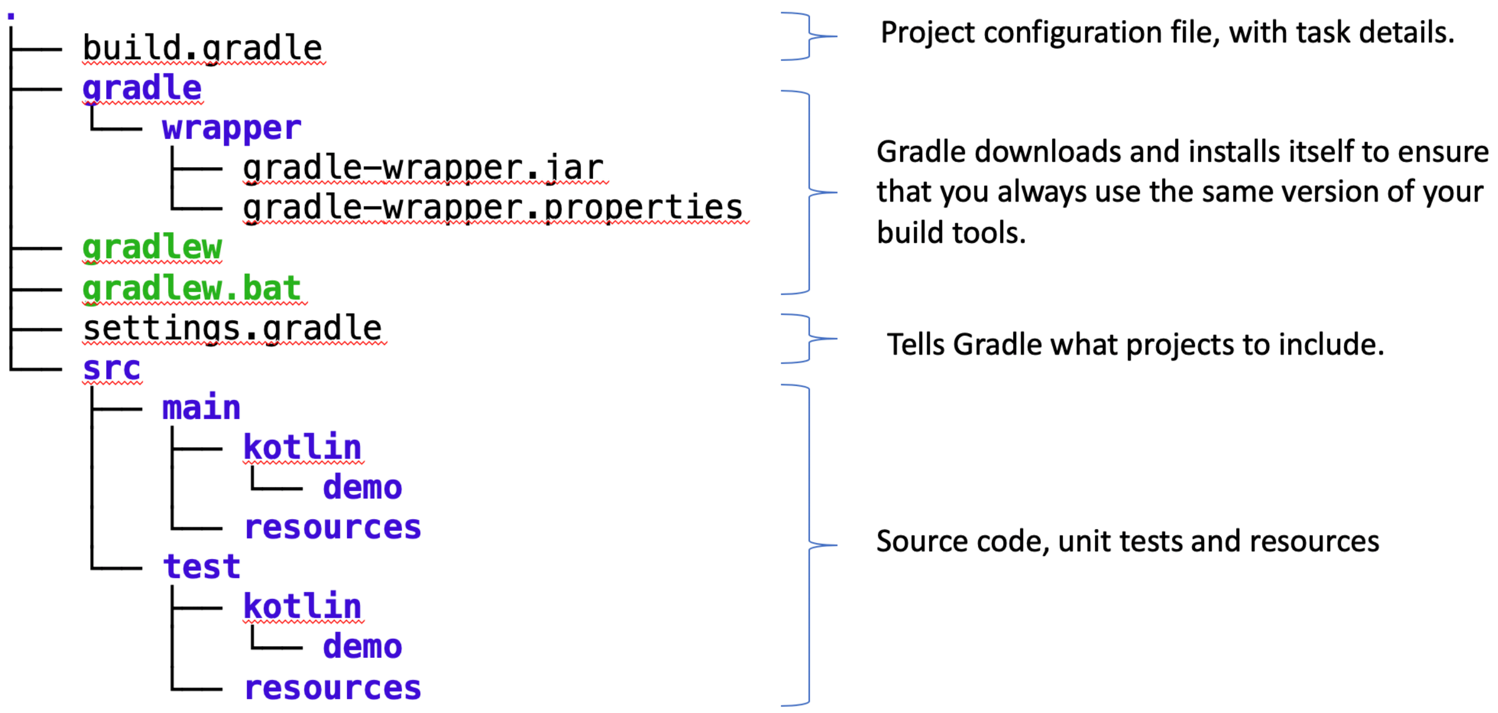
A “standard” Gradle project has about 30 tasks. Many of them are called infrequently, or called by other tasks.

```
$ gradle tasks

> Task :tasks

------------------------------------------------------------
Tasks runnable from root project
------------------------------------------------------------

Application tasks
-----------------
run - Runs this project as a JVM application

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project
buildNeeded - Assembles and tests this project and
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main c

....
```



```
package business

class Note(val heading: String, val text: String) {
    // the magic happens here
}
```

A standard Gradle project directory is structured like this:



```
.
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   ├── kotlin
    │   │   └── demo
    │   └── resources
    └── test
        ├── kotlin
        │   └── demo
        └── resources
```

Project configuration file, with task details.
Gradle downloads and installs itself to ensure that you always use the same version of your build tools.
Tells Gradle what projects to include.
Source code, unit tests and resources

Gradle project structure

Structuring source code

You can further divide source code up by package, where a package represents a logical grouping of your files.

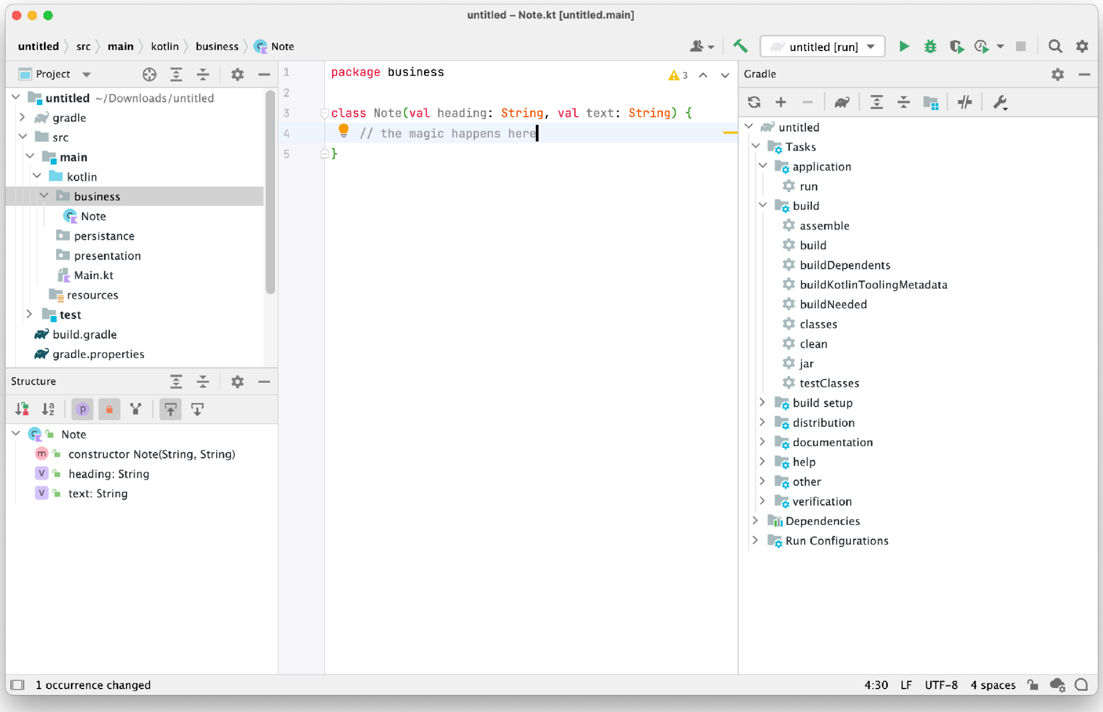
/persistance (/models)

/business (/controllers)

/presentation (/views)

e.g. you are using a layered architecture, so you can further subdivide your source code by layer.

This also aids in testing, since you can write tests that target a specific layer (and focus on testing its interface).



```kotlin
package business

class Note(val heading: String, val text: String) {
    // the magic happens here
}
```

1 occurrence changed

Packages further group presentation, business and persistence layers.

Activities

Activities This Week ✓

Setup GitLab

- All requirements logged in Git and unassigned.
- Milestones (sprints) setup.
- Infrastructure tasks moved to Sprint 0, closed as appropriate.

Source code

- Starting project committed to Git repo.
- Git works across all machines. Everyone has a git client, and knows how to git pull/push.
- IntelliJ is setup for everyone, and the starting project builds.

Technical Investigation

- Choose toolkits; investigate libraries
- Think about data format! How will you store, represent this data?