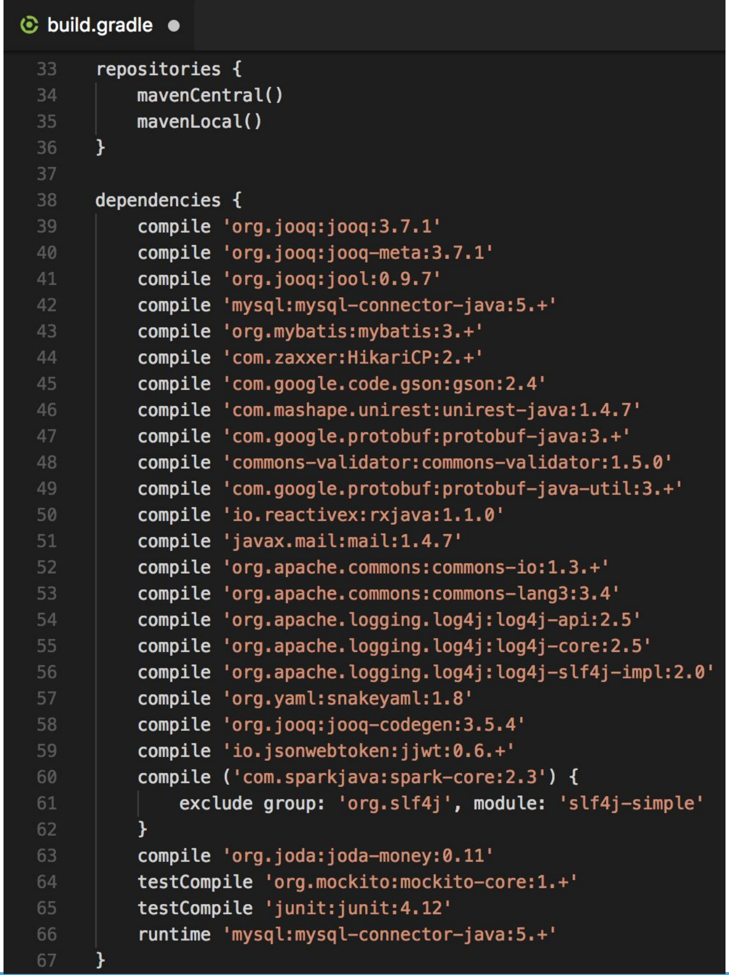
Package Managers & Build Tools

But.. why?

- Managing multiple dependencies is a *huge mess*
- Building across multiple IDEs & platforms is a *huge mess*



```
repositories {
    mavenCentral()
    mavenLocal()
}

dependencies {
    compile 'org.jooq:jooq:3.7.1'
    compile 'org.jooq:jooq-meta:3.7.1'
    compile 'org.jooq:jool:0.9.7'
    compile 'mysql:mysql-connector-java:5.+'
    compile 'org.mybatis:mybatis:3.+'
    compile 'com.zaxxer:HikariCP:2.+'
    compile 'com.google.code.gson:gson:2.4'
    compile 'com.mashape.unirest:unirest-java:1.4.7'
    compile 'com.google.protobuf:protobuf-java:3.+'
    compile 'commons-validator:commons-validator:1.5.0'
    compile 'com.google.protobuf:protobuf-java-util:3.+'
    compile 'io.reactivex:rxjava:1.1.0'
    compile 'javax.mail:mail:1.4.7'
    compile 'org.apache.commons:commons-io:1.3.+'
    compile 'org.apache.commons:commons-lang3:3.4'
    compile 'org.apache.logging.log4j:log4j-api:2.5'
    compile 'org.apache.logging.log4j:log4j-core:2.5'
    compile 'org.apache.logging.log4j:log4j-slf4j-impl:2.0'
    compile 'org.yaml:snakeyaml:1.8'
    compile 'org.jooq:jooq-codegen:3.5.4'
    compile 'io.jsonwebtoken:jjwt:0.6.+'
    compile ('com.sparkjava:spark-core:2.3') {
        exclude group: 'org.slf4j', module: 'slf4j-simple'
    }
    compile 'org.joda:joda-money:0.11'
    testCompile 'org.mockito:mockito-core:1.+'
    testCompile 'junit:junit:4.12'
    runtime 'mysql:mysql-connector-java:5.+'
}
```

So what does a build tool do?

Package manager:

- Dependency management

Build tool:

- Automated and system independent build lifecycles

- Testing
- Building
- Deploying
- ...

So what do these tools **exactly** do?

1. Downloading dependencies.
2. Compiling source code into binary code.
3. Packaging that binary code.
4. Running tests.
5. Deploying to server.

A quick example: Maven™

A few examples…

- Maven (Java)
- Gradle (Java)
- NPM (Javascript)
- CMake (C/C++)

Maven - 2004

“Convention is good.”

Maven - 2004

Pros:

- It works!
- Decent dependency management

Cons:

- XML (big and cumbersome)
- Huge overhead
- Not flexible

➡ **Need a programming language!**

Gradle - 2012

“Convention is good and so is flexibility.”

Gradle - 2012

- No XML, instead Groovy DSL:
  - Much less boilerplate code
  - Config file much shorter and clearer

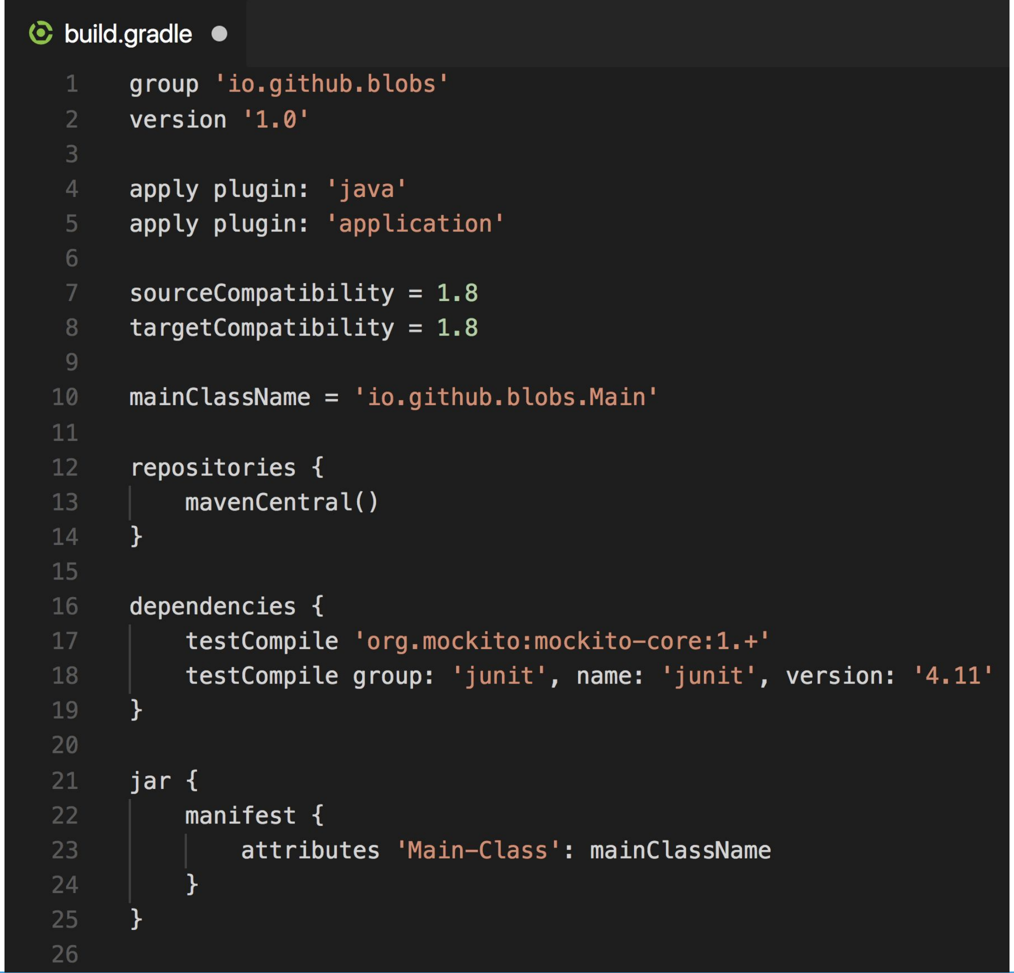
- Adopted by Google for Android

Gradle - 2012

- No XML, instead Groovy DSL:
  - Much less boilerplate code
  - Config file much shorter and clearer

- Adopted by Google for Android

Cheat Sheet



```
group 'io.github.blobs'
version '1.0'

apply plugin: 'java'
apply plugin: 'application'

sourceCompatibility = 1.8
targetCompatibility = 1.8

mainClassName = 'io.github.blobs.Main'

repositories {
    mavenCentral()
}

dependencies {
    testCompile 'org.mockito:mockito-core:1.+'
    testCompile group: 'junit', name: 'junit', version: '4.11'
}

jar {
    manifest {
        attributes 'Main-Class': mainClassName
    }
}

```

NPM

Show by doing: npm

CMake

Let’s play: Gradle

Let’s play: Gradle

- Groovy DSL basics
- Managing repositories and dependencies
- Configuring build tasks

Some basic Gradle commands

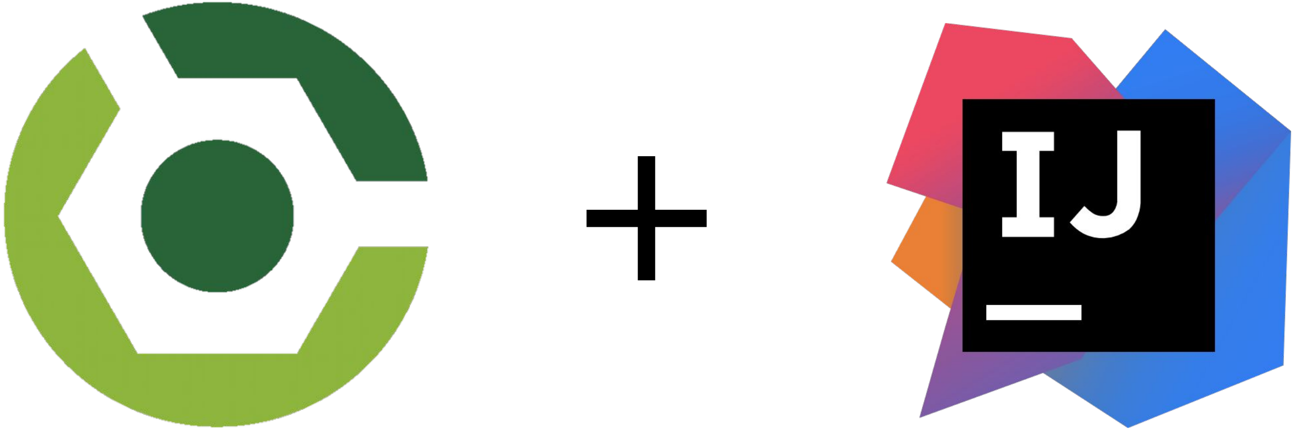
`gradle taskname:` Execute task ‘taskname’

`gradle tasks:` Show *most* available tasks

`gradle tasks -all:` Show *all* available tasks

`gradle -b anotherfile.gradle taskname:` Use another gradle file

Gradle IntelliJ Integration



Cheat Sheet

Overview of Package Managers & Build Tools

Java: Maven, Gradle

Javascript: NPM, Webpack, Rollup, Grunt, Gulp

Python: pip, PyBuilder

Ruby: RubyGem, Bundler, Rake

C/C++: Meson, CMake

C#: MsBuild

Haskell: Cabal, Stack