Customizing and Refactoring Gradle Builds

Marc Philipp, Gradle Inc.





Marc Philipp

Software Engineer at Gradle, Inc.

JUnit 5 team lead

Twitter: @marcphilipp

Web: marcphilipp.de







What is Gradle?



What is Gradle?

Gradle is an open-source build automation tool

- based on the Java Virtual Machine (JVM)
- implemented in Java
- focused on flexibility and performance
- 100% open-source (Apache 2.0) and free





Versatile

- Java ecosystem: Java, Groovy, Kotlin, Scala, ...
- Official build tool for Android
- Native projects: C, C++, Swift, ...
- And more: Go, Asciidoctor, ...

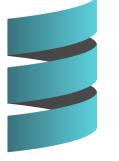


















Gradle Inc.

- Vision: Build Happiness
- Mission: Accelerate Developer Productivity
- Products:
 - Gradle Build Tool
 - Gradle Enterprise
- more than 60 employees including over 40 engineers

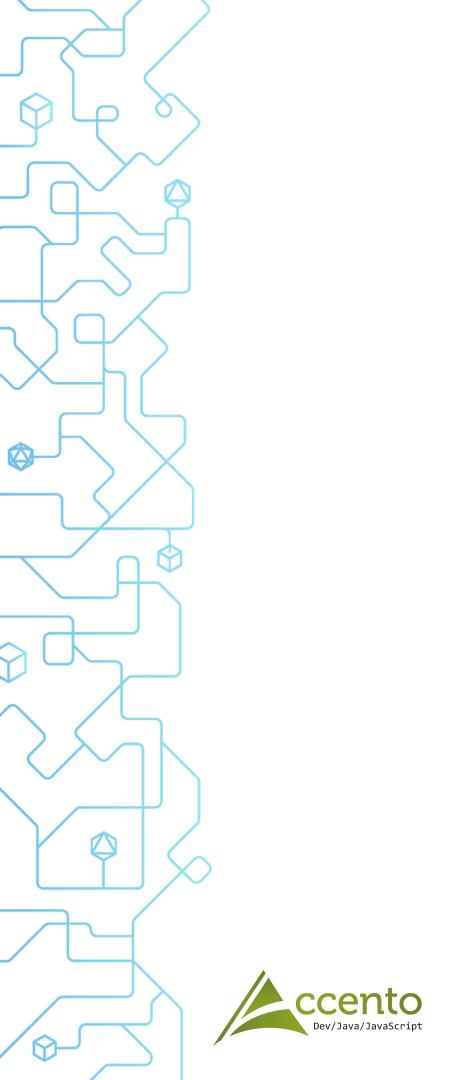




Agenda

- Basic concepts
- From Quick & Dirty to Safe & Sound
 - dependency management
 - custom tasks
 - custom configuration







Show of Hands



Basic Concepts



Tasks

- a Gradle build executes tasks
- tasks can depend on other tasks
- tasks have inputs and outputs



Hello World

```
tasks.register("helloWorld") { // in build.gradle
  doLast {
    println("Hello World!")
  }
}
```

```
$ gradle helloWorld
> Task :helloWorld
Hello World!

BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
```





Build Scripts

A Gradle project is configured in build scripts:

- settings.gradle[.kts]: configures the subprojects that comprise the build
- build.gradle[.kts]: configures the used plugins and tasks





settings.gradle[.kts]

```
rootProject.name = "new-project"
include("subproject-a")
include("subproject-b")
```





build.gradle[.kts]

```
plugins {
    java // to compile Java sources
    application // to generate startup scripts
repositories {
   jcenter() // to resolve dependencies
dependencies {
    implementation("com.google.guava:guava:28.0-jre")
    testImplementation("org.junit.jupiter:junit-jupiter:5.5.2")
application { // extension of the 'application' plugin
    mainClassName = "com.example.App"
```





Groovy vs. Kotlin DSL

- build scripts use a Domain Specific Language (DSL)
- initially Gradle only supported *Groovy*
 - dynamically typed
 - limited IDE support
- Kotlin DSL is stable since Gradle 5.0

Build scripts should be declarative – complex logic does not belong here.





Gradle Wrapper

- ./gradlew <tasks>instead of gradle <tasks>
- execute builds with prior installation of Gradle
- downloads required version
- caches already downloaded versions locally
- everyone uses the same version





Anatomy of a Gradle project

```
build.gradle.kts  // build script
gradle/wrapper  // wrapper jar and configuration
gradlew  // wrapper script for Linux/macOS
gradlew.bat  // wrapper script for Windows
settings.gradle.kts // settings script
src  // Java source tree
main
java
resources
test
java
resources
```





Incremental Builds

- only execute tasks that are affected by changes in between two subsequent builds
 - inputs have changed
 - outputs are present and unchanged
 - task implementation has changed (e.g. different plugin version)
- keep outputs of all tasks that are up-to-date





First Build

```
$ ./gradlew --console=plain build
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :jar
[...]
> Task :compileTestJava
> Task :testClasses
> Task :test
> Task :check
> Task :build
BUILD SUCCESSFUL in 5s
7 actionable tasks: 7 executed
```





Subsequent Build

```
$ ./gradlew --console=plain build
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :jar UP-TO-DATE
[...]
> Task :compileTestJava UP-TO-DATE
> Task :testClasses UP-TO-DATE
> Task :test UP-TO-DATE
> Task :check UP-TO-DATE
> Task :build UP-TO-DATE
BUILD SUCCESSFUL in 0s
7 actionable tasks: 7 up-to-date
```





Build Scans

- Accelerate debugging of build problems
- Private but shareable link
- Free to use on scans.gradle.com

```
$ ./gradlew build --scan

BUILD SUCCESSFUL in 1s
7 actionable tasks: 5 executed, 2 up-to-date

Publishing build scan...
https://gradle.com/s/lu7dxy7quyoju
```

> https://gradle.com/s/lu7dxy7quyoju





Build Cache

- allows reusing task outputs of any previous build
- local and remote cache

```
$ git pull
[...]
185 files changed, 4320 insertions(+), 1755 deletions(-)
```

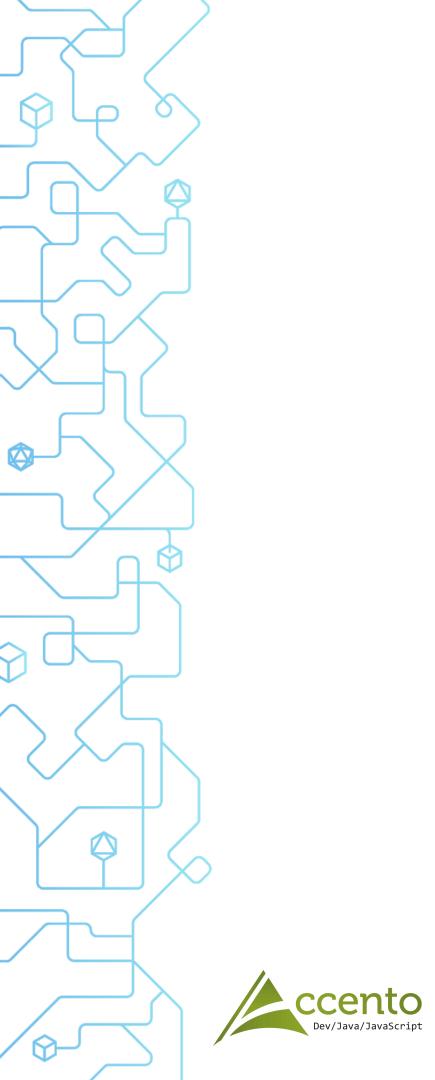
```
$ ./gradlew --build-cache sanityCheck
BUILD SUCCESSFUL in 1m 11s
1338 actionable tasks: 238 executed, 1100 from cache
```





Dependency Management





Demo



Recap

- Don't duplicate dependency version
- Prefer api or implementation over compile
- Use buildSrc to collect dependency versions
- Use a java-platform plugin to streamline dependency management





More on Dependency Management

Free webinars:

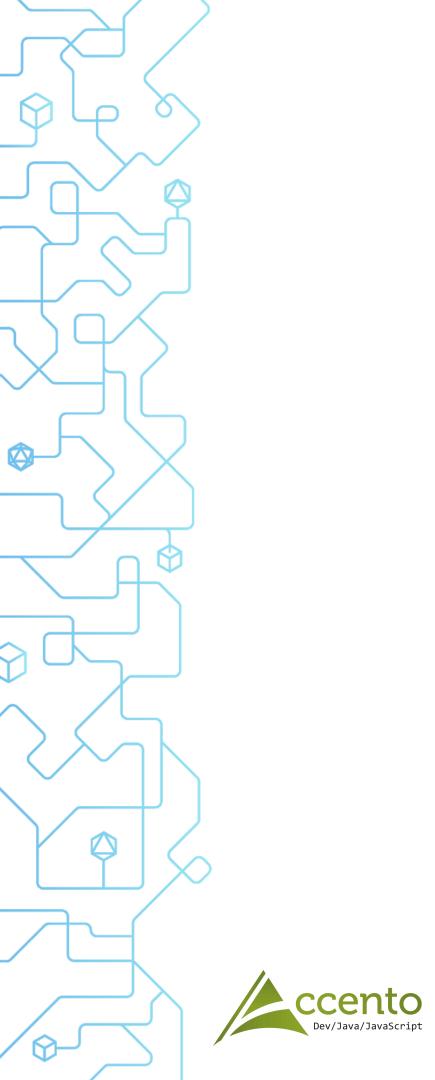
- https://gradle.com/blog/dependency-managementfundamentals/
- https://gradle.com/blog/dependency-managementpart-2-handling-conflicts/





Custom Tasks





Demo



Recap

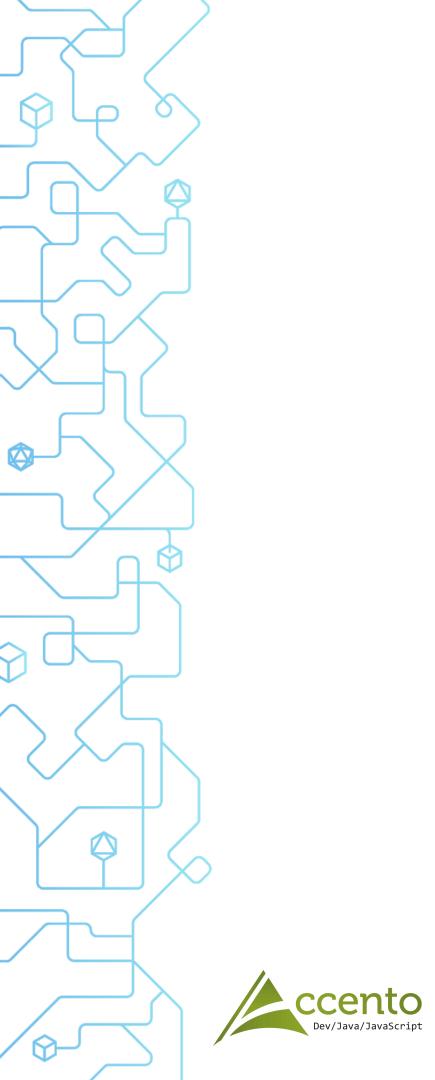
- Don't define complex tasks directly in the build script
- Define them in the buildSrc project
- Allows for testing and reuse in subprojects





Custom Configuration





Demo



Recap

- Extract custom logic into separate build scripts
- Even better: Extract your custom logic into a precompiled script plugin in **buildSrc**
- Next step: Move it to a separate plugin to use it in independent projects





Summary





Summary

- Keep your build scripts declarative
- Use buildSrc to share logic



Links

Demo code:

https://github.com/marcphilipp/gradle-refactorings

My talks on Gradle and JUnit:

https://www.marcphilipp.de/en/talks/







Thank you!

@marcphilipp

