Gradle Concepts And Best Practices

Contact Info

Ken Kousen

Kousen IT, Inc.

ken.kousen@kousenit.com

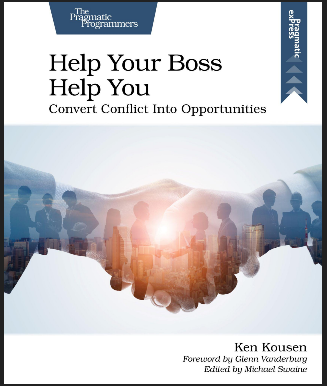
http://www.kousenit.com

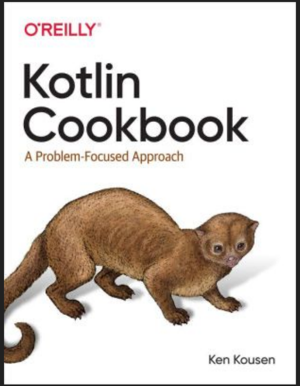
http://kousenit.org (blog)

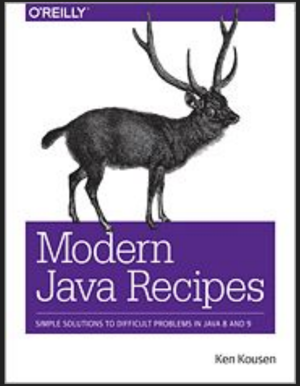
@kenkousen (twitter)

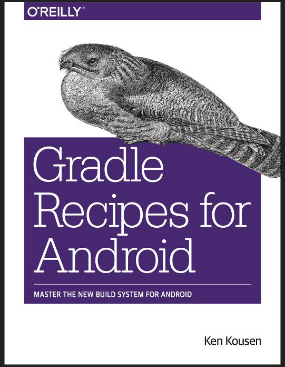
Tales from the jar side (free newsletter)

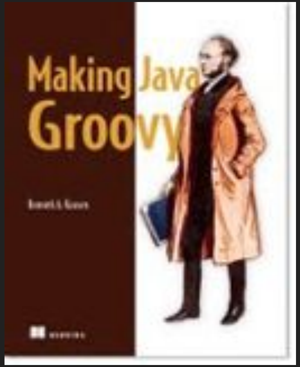
https://kenkousen.substack.com


The Pragmatic Programmers
Pragmatic exPress
Help Your Boss Help You
Convert Conflict Into Opportunities
Ken Kousen
Foreword by Glenn Vanderburg
Edited by Michael Swaine






O'REILLY
Gradle Recipes for Android
MASTER THE NEW BUILD SYSTEM FOR ANDROID
Ken Kousen



Gradle

- Build tool
- Defines DSL (domain specific language) for builds
- DSL in either Groovy or Kotlin
    - `build.gradle` → Groovy
    - `build.gradle.kts` → Kotlin

Know the Gradle Lifecycle

Phases

Initialization time

Configuration time

Execution time

Initialization

- Determines which build files to process
- Reads gradle.properties file
- Runs any "init" tasks

Configuration

- Gradle processes build files
- Instantiates and configures all tasks

Remember: ALL tasks are configured before ANY are executed

Anything outside doFirst or doLast happens at configuration time

Execution

- Runs `@TaskAction` method in requested tasks
- All tasks (other than DefaultTask) have a task action
- `doFirst` is pre-processing before task action
- `doLast` is post-processing after task action

From Eager to Lazy

- Instead of `task myTask` use `tasks.register("myTask")`
- Instead of `myTask { … }` use `tasks.named("myTask") { … }`

Arrange Tasks In The DAG

DAG

Gradle builds Directed Acyclic Graph

Graph based on task dependencies

- `dependsOn`
- `mustRunAfter`
- `shouldRunAfter`
- `finalizedBy`

See execution path with `--dry-run` (or `-m` for short)

Each task executed only once

Start with settings.gradle

Multi-project builds

`settings.gradle` → uses `include`

Specifies which subdirectories are also Gradle projects

Can also modify project dir, build dir, other properties

Extra properties

`ext.myprop` → avoid conflicting with existing project properties

myprop now available

- throughout project
- in any subproject

Use the Gradle API

Gradle API

- Gradle is written in (primarily) Java
- Each task in the DSL backed by Java class
- Those classes are heavily tested and maintained
- Use these typed tasks whenever possible

Ad-Hoc vs Typed tasks

Ad-hoc:

```
task hello {
    doLast { println 'What up?' }
}
```

Typed:

```
task copy(type: Copy) {
    from 'mydir'
    into 'otherDir'
}
```

Create a Custom Task

Custom Task

- Extend `DefaultTask` or other from API
- Annotate exactly one method with `@TaskAction`
- Can create class in `build.gradle`
- Eventually move it to `buildSrc` directory
- Eventually make a plugin

Creating Custom Tasks

Extend `DefaultTask`

Use `@org.gradle.api.tasks.TaskAction`

Ad hoc tasks have no defined action

Ad hoc task

- Basis of most Gradle tutorials (unfortunately)
- Have no `type:` parameter
- Are instances of `DefaultTask`
- Need `doFirst` or `doLast`, or they don't do anything

Useful command line flags

Command line flags

Exclude task with `-x`

Continuous build `-t`

Keep going `--continue`

Dry run `-m` or `--dry-run`

Parallel `--parallel` (for multi-project builds)

Manage project dependencies

Dependency Resolution

`jcenter()` or `mavenCentral()`

Any Ivy or Maven url

Local file system

Note: jcenter() is now deprecated and is going away

Configuration

A configuration is a collection of dependencies

Constraints DSL

- version block
- Specify:
    - strictly
    - prefer
    - require
    - reject

https://docs.gradle.org/current/userguide/rich_versions.html

Checking dependencies

```
> gradle dependencies [--configuration <name>]

> gradle dependencyInsight --dependency <name>
    --configuration <name>
```

Incremental builds (Inputs and Outputs)

Inputs and Outputs

files, directories, properties

In task class, annotate:

- `@InputFile, @InputFiles`
- `@InputDirectory`
- `@Input (for a String property)`
- `@OutputFile`
- `@OutputDirectory`

Inputs and Outputs

In the build file,

```
task mytask {
    inputs.file file('myfile.txt')
    outputs.file file('output.txt')
    doLast {
        // … whatever …
    }
}
```

Inputs and Outputs

Task is up to date when:

- Inputs haven't changed
- Outputs still present and unchanged

Input/Output files are hashed

Contents of directories are hashed

Values of properties are serialized

Miscellaneous

Build Scans

Add plugin

Add license agreement

Run with `--scan` (3.4) or `-Dscan`

Check out results

- Self-hosted version part of Gradle Enterprise

Logger

In build.gradle, use logger

```
logger.info "An info message"
logger.warn 'A warning'
```

```
> gradle -i hello
```

Ant integration

Gradle includes "ant" object → AntBuilder

Import Ant build files → Ant tasks become Gradle tasks

```
ant.importBuild 'build.xml'
```

IDE Support

Eclipse → Buildship

IntelliJ → Gradle support built-in

`apply plugin: 'eclipse'` or `apply plugin: 'eclipse-wtp'`

`apply plugin: 'idea'`

Script Plugins

Like an import

```
apply from: 'other.gradle'
```

"from" value can be a full URL

- Not cached -- if unavailable, build will fail

Binary Plugins

Class that implements `Plugin<Project>`

```
void apply(Project project) {
    // ...
}
```

```
apply plugin: 'org.foo.my-plugin'

apply plugin: org.foo.plugins.MyPlugin
```

Plugin ID

```
apply plugin: 'org.foo.my-plugin'
```

META-INF/gradle-plugins/org.foo.my-plugin.properties:

```
    implementation-class=org.foo.plugins.MyPlugin
```

Plugin Repository

- Gradle Plugins Repository
  - http://plugins.gradle.org

Key Concepts

- Know the Gradle lifecycle
- Start with settings.gradle
- Use the DSL where possible
- Then use Gradle API
- Try to create a custom task
- Only use ad hoc tasks when necessary
- Incremental builds
- Useful keyboard shortcuts
- Miscellaneous

Inside Joke

This is an inside joke, but if you know, you know…

Best Practice

Remember, kids! Andres says:

```
> mvn clean install
```

(Who cares about performance, anyway?)

Bestest Practice

Remember, kids! Andres says:

```
> mvn verify
```

(Who cares about performance, anyway?)