Gradle

Gradle Project Overview

- Project can be small or big
- a project consist of
  - a project directory
  - a single file settings.gradle containing the project name
- Interact with project using gradle command

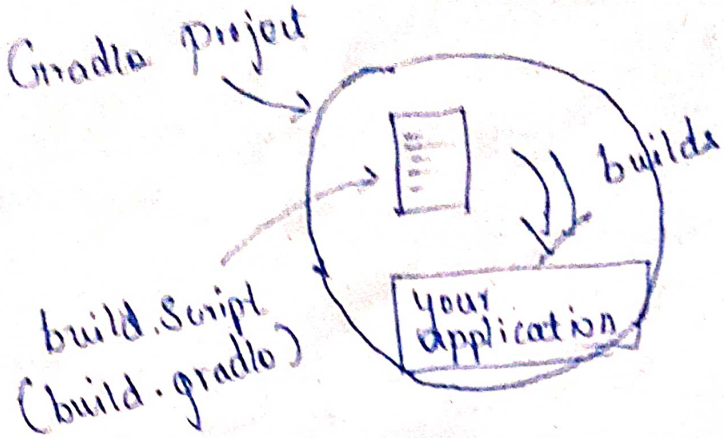
```
settings.gradle

rootProject.name = '30-second-project'
```

```
> gradle tasks

Task runnable from root project
'30-second-project'
```

Gradle project naming

- can name project anything you like (not tied to project directory name)
- don't have to specify a name in settings.gradle
- Its best practice because even if someone clones your project to a different named directory, the project name is fixed

The Gradle build Script



Build script languages

1) Groovy
2) Kotlin

Groovy

build.gradle

```
println 'This is the 30 second project'
def a=2
def b=3
println a+b
```

```
> gradle

configure project:
This is the 30 second project
5
```

Need to write lots of code in build Scripts ?

No, because normally we're using predefined:

- tasks

- plugins
- other elements

Which you configure together in your build script.

Gradle task Overview

- Unit of work to get done in your build
  - copying files
  - compiling code
  - running test
  - much more
- every task can be executed from command line
- Gradle has pre-packaged tasks you can add to build script
- can create custom tasks for anything else

How to use a task in a build script?

Here's an example, where in the build.gradle

1. define a task to copy a file
2. use Gradle's pre-packaged Copy task
3. pass in these 4 pieces of information
   a. task name
   b. task type (Copy)
   c. source file location
   d. destination file location

build.gradle

```
tasks.register('copyMessage', Copy) {
    from 'important-message.txt'
    into "$buildDir"
}
```

The Gradle Plugin

- Plugin automatically register tasks in project
- you can execute any tasks exposed by a plugin
- tasks exposed by a plugin may be pre-packaged or custom
- plugin allow us to add rich functionality to a project

How to apply a plugin?

Here's an example, where in the build.gradle

- use the simply apply plugin syntax
- apply the base plugin
- list new tasks added to the project
- execute a task added by the plugin

build.gradle

```
plugins {
    id 'base'
}
```

> gradle tasks

Build tasks

assemble - Assembles the output of this project
build - Assembles and tests this project.
clean - Delete the build directory

Small Project [Theme Park]

- Description
  - rollercoaster.txt
    - At @THEME_PARK_NAME@ thrill seekers ...
  - log-flume.txt
    - At @THEME_PARK_NAME@ take a ride on
  - teacups.txt
    - At @THEME_PARK_NAME@ if you want some

build.gradle

```
import org.apache.tools.ant.filters.ReplaceTokens

tasks.register('generateDescription', Copy)
{ from 'descriptions'
  into "$buildDir/description"
} filter(ReplaceTokens, tokens: [THEME_PARK_NAME: "SUE"])
```

Zip Task

build.gradle

```
tasks.register('zipDescriptions', Zip)
{
    from 'descriptions'
    into "$buildDir"
    archiveFileName = 'descriptions.zip'
}
```

Gradle init

```
> mkdir gradle-init-test
> cd gradle-init-test
> gradle init
```

Select type of project to generate:

1. basic
2. application
3. Library
4. Gradle plugin

: Enter selection (default: basic) [1..4] 1

Select build scripts DSL:

1. Groovy
2. Kotlin

Enter Selection (default: Groovy) [1..2] 1

DSL = Domain Specific Language

Project name (default: gradle-init-test): Enter

> gradle tasks

Task runnable from root project: 'gradle init --task'

> ls

build.gradle gradle gradlew
gradlew.bat settings.gradle

Gradle Wrapper Introduction

Script used to execute tasks, which means:

1. no Gradle installation required
2. forces Gradle version
3. can easily update version

> tree

```
├ build.gradle
├ gradle
│  └ wrapper
│     ├ gradle-wrapper.jar
│     └ gradle-wrapper.properties
├ gradlew  - (shell cmd Linux)
├ gradlew.bat - (shell cmd windows)
└ settings.gradle
```

> cat gradle/wrapper/gradle-wrapper.properties

distributionBase = GRADLE_USER_HOME
distributionPath = wrapper/dists
distributionUrl = https\://services.gradle.org/distributions/gradle-7.1.1-bin.zip
zipStoreBase = GRADLE_USER_HOME
zipStorePath = wrapper/dists

Command line Interaction

You'll learn how to:

1. Use built-in Gradle tasks
2. run multiple tasks
3. Specify a default task
4. Use abbreviated task names
5. Use command line options

1. Use built-in Gradle tasks

```
> ./gradlew tasks

Build Setup Task          ← group
-----------
init
wrapper

Help tasks                ← group
----------
buildEnvironment
```

```
> ./gradlew tasks --all
    :
Other tasks
-----------
generateDescriptions   } These are not belongs
zipDescriptions        } to any group
```

build.gradle

```
tasks.register('generateDescription', Copy) {
    group 'Theme park'
    description 'Generate descriptions'
    :
}
```

> ./gradlow tasks

⋮

Theme park tasks

generateDescriptions - Generate descriptions

2. run multiple tasks

>./gradlew projects help

1 2

3. specify a default tasks

>./gradlew help is default task

Task: help

build.gradle

```
plugins {
    id 'baso'
}
defaultTasks 'generateDescriptions'
tasks.~~project~~register('generateDescriptions', Copy) {
    ⋮
}
```

> ./gradlew

Task: generateDescriptions

Abbreviated task names

- Only provide enough of task name to be unique ex. ./gradlow h for ./gradlow help

- for camelCase, use first letter from each word
  e.g ./gradlew gD for ./gradlew generateDescr
- Gradle errors on task name clash

Build Lifecycle — 3 Phases

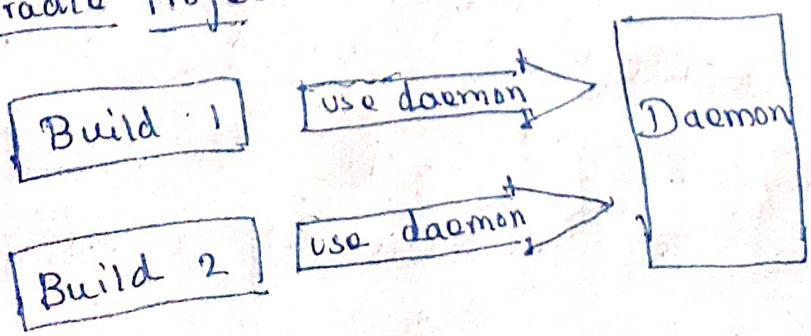
1) Initialization Phase:
   - Gradle figures out projects to build
   - executes settings.gradle

2) Configuration Phase:
   - Gradle executes all build.gradle files
   - build in-memory representation of your project

3) Execution
   - Gradle figures out what tasks to execute
   - may decide not to execute task again
   - tasks are actually executed

Gradle daemon

Gradle Project



benefits

- faster startup
- in-memory caches
- JVM runtime optimisations

> ./gradlew --status

PID	STATUS	INFO
8186	IDLE	7.1.1.
7858	STOPPED	(stop command received)

> ./gradlew --stop
Stopping Daemon(s)
1 Daemon stopped

> ./gradlew --status

PID	STATUS	INFO
8186	STOPPED	
7858	STOPPED	

> ./gradlew
starting...

--no-daemon Do not use the Gradle daemon to run the build. Useful occasionally if you have configured Gradle to always run with the daemon by default

> ./gradlew --no-daemon (slow performance)

Gradle

Configuring Project

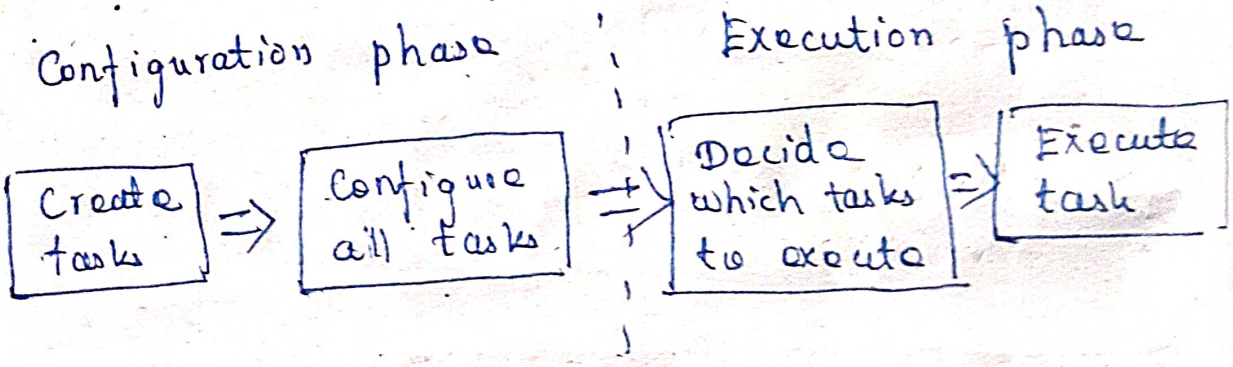
```
build.gradle
description 'Just a demo'
group 'com.tomgregory'
version '1.0-SNAPSHOT'
```

Locating Tasks

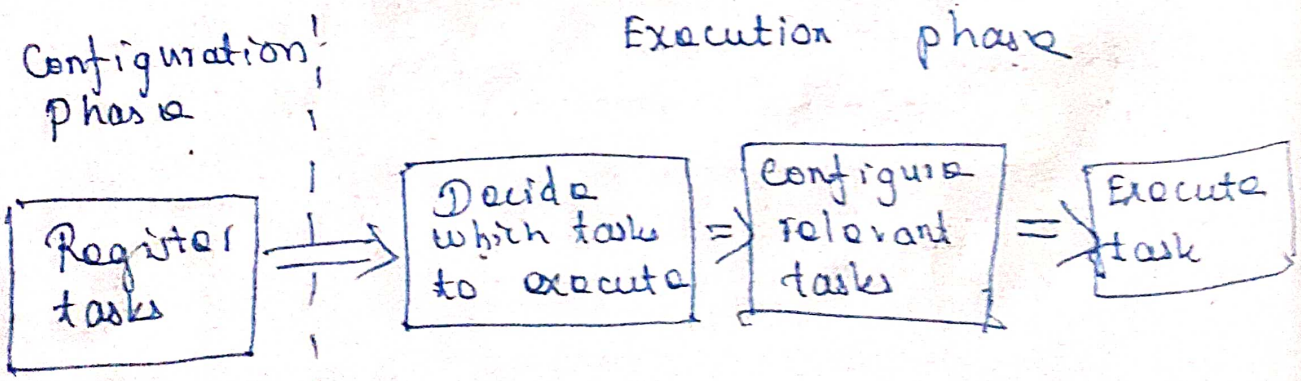
You'll learn

- importance of configuration avoidance
- 4 ways to locate tasks
- all about TaskProvider
- how to configure task once located

Task configuration option 1



Task configuration option 2 [best performance]



Registering tasks

=> tasks.register

Ensures tasks are only created if needed

Locating tasks

Approach 1 (task not created - recommended)

=> TaskProvider <T> named (String name)

build.gradle

```
plugin {
    id 'base'
}
tasks.register ('generateDescriptions', Copy) {
    from 'descriptions'
    into "$buildDir/descriptions"
} ~~ditto~~

tasks.named ('generateDescriptions') {
    into "$buildDir/descriptions - renamed"
}
```

This will not create new task.

Approach 2 (task created immediately)
(best avoided)

=> T getByName (String name)

build.gradle

```
tasks.getByName ('generateDescriptions') {
    into "$buildDir/descriptions - renamed"
}
```

Approach 3 (Shorthand syntax)

⇒ tasks.clean

Only works pre builded tasks and plugins

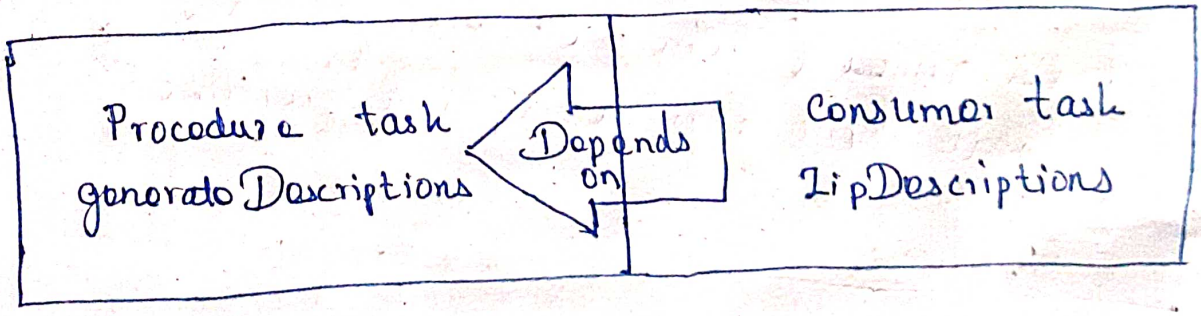
build.gradle

```
tasks.clean {
    doLast {
        println 'squeaky clean'
    }
}
```

Approach 4 (Groovy DSL shorthand syntax)

⇒ clean (best avoided)

Task dependency



build.gradle

```
tasks.register('generateDescriptions', Copy) {
    from 'descriptions'
    into "$buildDir/descriptions"
}
tasks.register('zipDescription', Zip) {
    from "$buildDir/descriptions/"
```

```
    destinationDirectory = buildDir
    archiveFileName = 'descriptions.zip'
(1) => dependsOn tasks.named('generateDescriptions')
(2) => dependsOn generateDescriptions
```

```
>./gradlew zipDescriptions
    Task: generateDescriptions
    Task: zipDescriptions
```

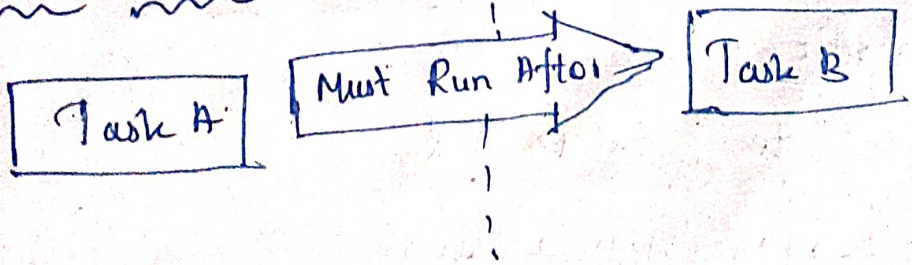
build.gradle

```
TaskProvider<Copy> generateDescriptionsTask
    = tasks.register('generateDescription', Copy){
    :
}
tasks.register('zipDescriptions', Zip) {
    :
    dependsOn generateDescriptionsTask
}
```

We can also set two or more dependicies

```
dependsOn generateTaskA, generateTaskB
```

Force task order,



build.gradle

```
tasks.register('generateDescriptionsA')
{
    println 'A'
    mustRunAfter generateDescriptions.B
}
tasks.register('generateDescriptionB') {
    println 'B'
}
```

> ./gradlew TaskA TaskB

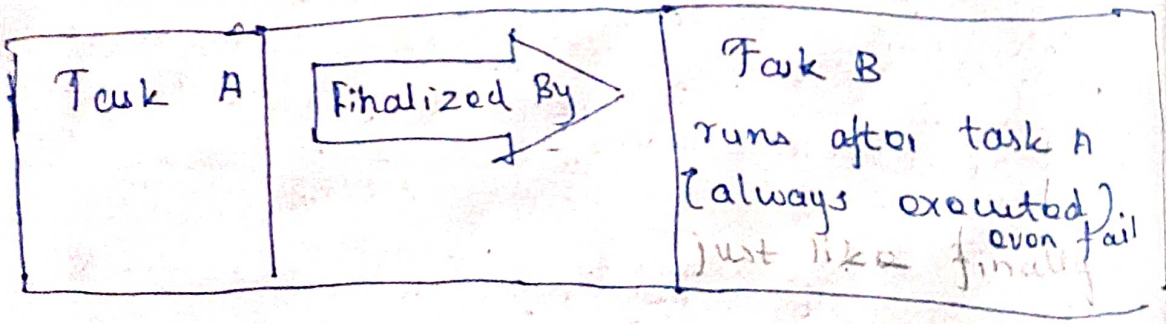
Task: PrintB (generateDescriptionB)
Task: PrintA (generateDescriptionsA)

> ~~tigr~~ ./gradlew TaskA

Task: PrintA (generateDescriptionsA

mustRunAfter Task is only get executed if we execute or part of the dependency

FinalizedBy task



build.gradle

```
tasks.register('PrintA') {
    doLast {
        println 'A is printed'
```

```
    finalizedBy confirmFinished }
}

tasks.register('TriggerA') {
    dependsOn PrintA
}
tasks.register('confirmFinished')
{
    doLast {
        println 'Finished'
    }
}
```

> ./gradlew TriggerA

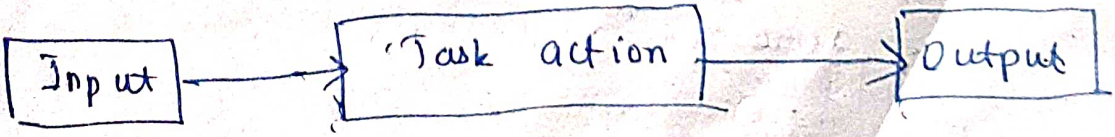
Task: PrintA
A is printed

Task: confirmFinished
Finished

Remember that

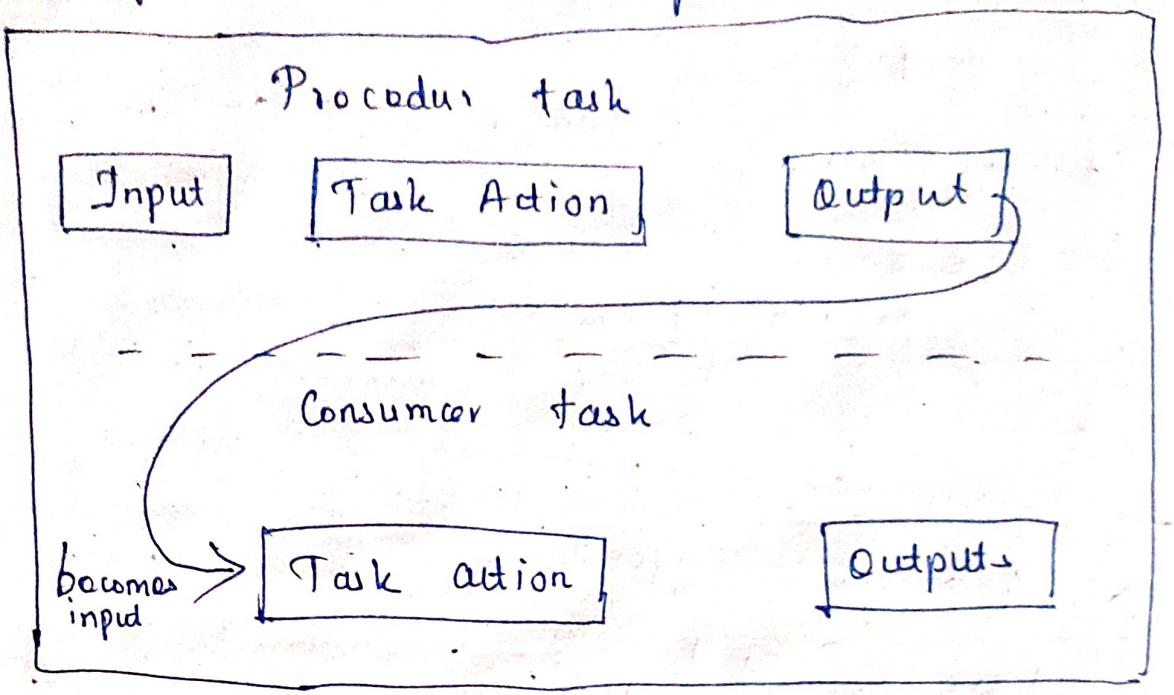
- if task A dependsOn B, B always runs before A
- use mustRunAfter to force order between tasks (whenever both are executed)
- if task A is finalizedBy task B, B always runs after A

Task inputs and outputs



Benefits 1 : Incremental build

Benefits 2 : Link task input & output



build.gradle

```
tasks.register('generateDescriptions', Copy) {
    from 'descriptions'
    into "$buildDir/descriptions"
}

tasks.register('zipDescriptions', Zip) {
    from generateDescriptions
    destinationDirectory = buildDir
    archiveFileName : 'descriptions.zip'
}
```

```
> ./gradlew zipDescriptions
  Task : generateDescriptions
  Task : zipDescriptions
```

gradle automatically adds the dependency so dependsOn is not required

Advantage

1. reduce duplication
2. Improve maintainability
3. Keeps build. scripts more concise

Applying Plugins

Core Plugins

```
plugins {
    id 'base'
}
```

3rd Party Plugins

```
plugins {
    id 'org.barfuin.gradle.taskinfo'
        version '1.3.0'
}
```

URL : plugins.gradle.org

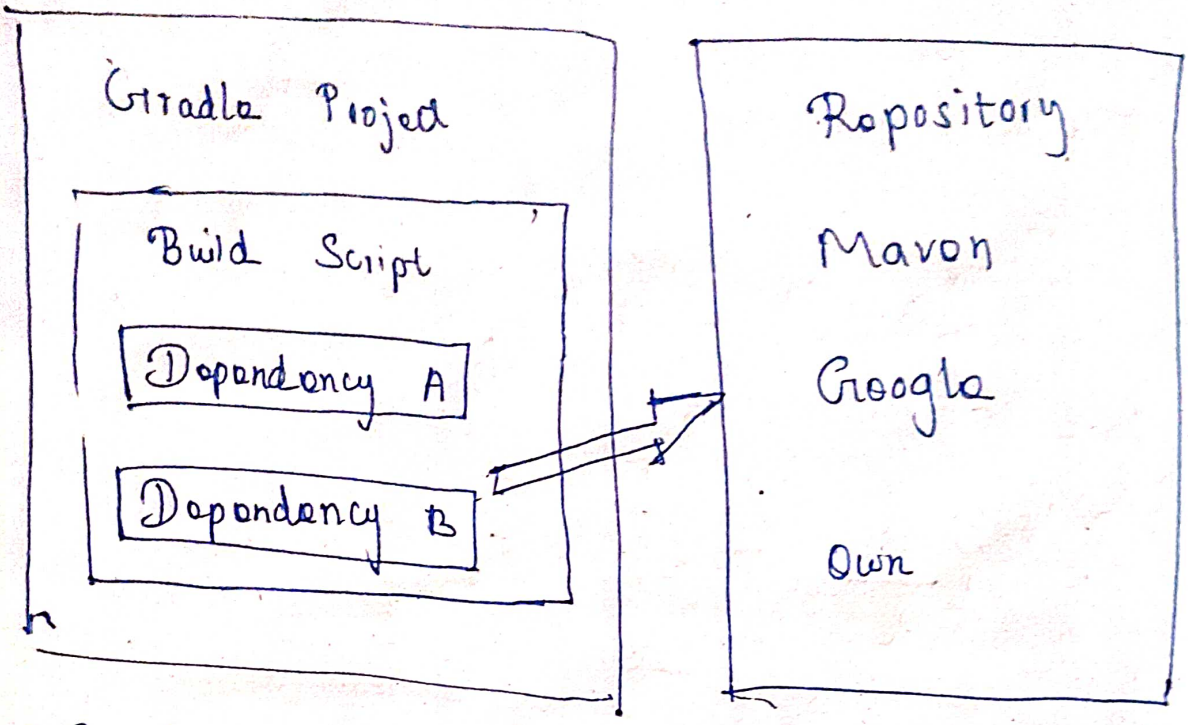
~~Configuration~~

Configuring Plugins

Configuration depends on the plugins e.g for the base plugin

```
base {
    archivesName = "gradle"
}
```

Repositories & Dependencies



Gradle repository Format

1) Maven
2) Ivy
3) Flat directory (Local)

Build in Gradle repositories

(No need configure the URL)

1) Maven Central (mvnrepository.com)
2) Google Maven

```
repositories {
    maven Central()   1st priority
    google            2nd priority
    mavend            3rd priority
        url 'https://example OwnRepo.com'
    }
}
```

Multiple stages to build application

1) Compiling
2) Testing
3) Runtime

dependicies requires for multiple stages might be differed

---

build.gradle

```
plugins {
    id 'java'
}
repositories {
    mavenCentral()
}
dependencies {
1st way  implementation  group: 'commons-beanutils',
                         name: 'common-beanutils',
                         version: '1.9.4'
}
2nd way  implementation  'commons-beanutils:comons-
                          beanutils:1.9.4'
}
```

Remember

- declare repositories from which to fetch dependency
- repository order is important
- dependencies grouped in configuration e.g implementation
- dependencies declared with group, name and version

Java Plugin

- It's a core plugin

```
plugins {
    id 'java'
}
```

- adds additional tasks
- configures projects

Java Plugin: Compile classes

- compileJava task

```
./gradlew compileJava
```

- uses same Java version used for Gradle by checking environment variable
- generates .class files in build directory

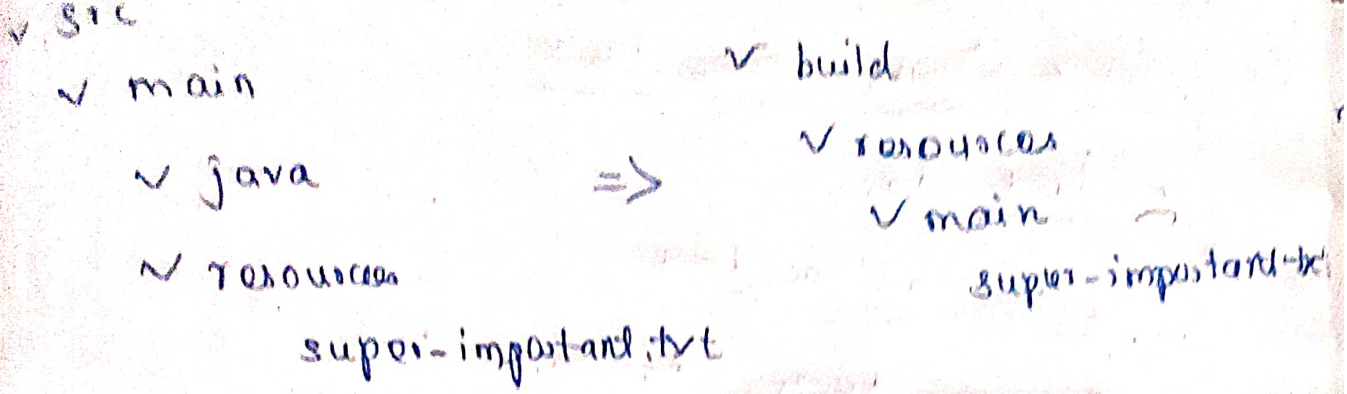
```
v src
  v main
    v java
      v com.tomgregory
          MyFirstClass

v build
  v classes
    v java
      v main
        v com
          v tomgregory
              MyFirstClass
```

Java Plugin: manage resources

- ~~Procss~~ processResources task

```
./gradlew processResources
```

- copies contents of resources directories into build directory



java plugin: define dependencies

- ~~dependencies~~ defined in dependencies in build.gradle

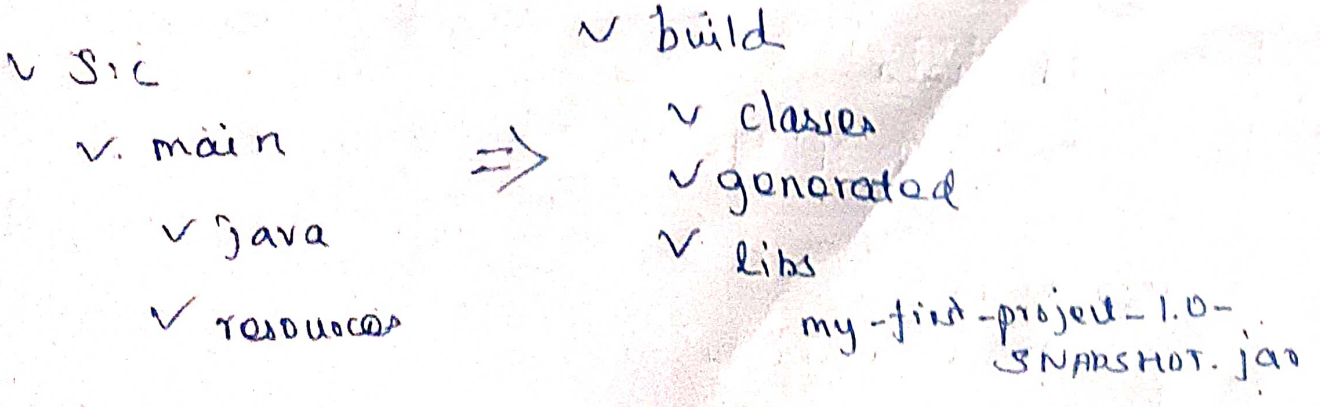
```
dependencies {
    implementation 'org.apache.commons:commons-lang3:3.12.0'
}
```

(group, name, version)

- dependency configuration (e.g implementation)
- dependency configuration used to generate classpath

Java Plugin: package

- Java plugin: add jar task
  - ./gradlew jar
- add compiled classes and resources to jar archive
- jar file named <project-name>-<version>.jar



Java plugin: run tests

- test task

  ./gradlew test

- compiles tests, processes test resources, run tests
- creates test report

Java plugin: project layout

		Destination
• src/main/java	>>	build/classes/java/main
• src/main/resources	>>	build/resources/main
• src/test/java	>>	build/classes/java/test
• src/test/resources	>>	build/resources/test

everything is configurable
But try to use the default

Sping Boot API Application

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.5.3'
}
repositories {
    mavenCentral()
}
```

```
dependencies {
    implementation 'org.springframework.boot:
                    spring-boot-starter-web:2.5.3'
}
java {
    toolchain {
        LanguageVersion = JavaLanguageVersion.of(16)
    }
}
```

> ./gradlew tasks

Tasks

bootRun — whatever tasks start with 'boot' that is part of spring boot task

> ./gradlew bootRun - Run the application and automatically deploy it to tomcat server

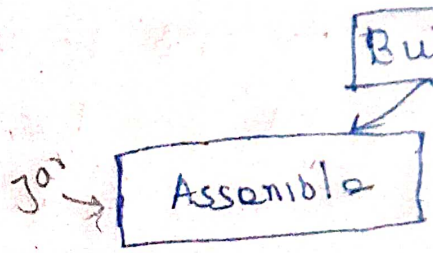
> ./gradlew assemble - generate jar file

=> build/lib

> java -jar build/libs/*.jar

Tasks in Java Projects

- tasks can depend on other tasks



- build aggregates assemble and check tasks

Java Plugin task graph

- build ✧
  - assemble ✧
    - jar ✧
      - classes
        - compileJava
        - processResources
  - check ✧
    - test ✧
      - classes
        - compileJava
        - processResources
      - testClasses
        - compileTestJava
        - classes
          - compileJava
          - processResources
        - processTestResources

▮ = ~~task which perform~~ aggregate tasks

▮ = tasks which perform an action

✧ = important tasks to remember

- can choose specific task to save time
  - check (just need to test project)
  - assemble (just need to build project)

Gradle

Run Jar File

```
> java -jar gradlejaval.jar
```

Java

build.gradle

```
plugins {
    id 'java'
}

project.group = 'com.donofprogramming'
project.version = '0.0.1-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.apache.commons:commons-math3:3.6.1'

    testImplementation 'junit:junit:4.12'
}
```

```
jar {
    baseName = "${project.name}-all"
    println ">>> baseName: ${baseName}"

    manifest {
        attributes
            'Implementation-Title': 'Gradle jar',
            'Created-By': 'donOf programming'
            'Implementation-Version': project.version,
            'Main-Class': 'com.donofprogramming.random.App'
    }
```

Change jar file name

VS Code Plugin

Maven: Extension Pack for Java

Gradle: Gradle for Java

Note:
Both two extensions are required for Gradle.