## Links for reference

|  |  |
| --- | --- |
| Junit user guide | <https://junit.org/junit5/docs/current/user-guide/#writing-tests> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Gradle

Its very fast (it will keep a track of what is recently changed recently so it will compile only changed files ), incremental builds,

Ex:- if code is not changed and only if test cases are changed then it will just run the test cases , it wont compile the source code again

We can write the code of build.gradle in DSL (domain specific language) in either groovy or kotlin

we will write build.gradle in groovy syntax

note: - gradlew.bat will download the gradle software into our machine in “.gradle” folder, so we don’t need to install the gradle at all

in “gradle-wrapper.properties” we will mention which gradle version we want

Gradle wrapper is responsible to download & install the mentioned gradle versions

Each project will have separate gradle wrapper

Important points

1. Even if build.gradle is not there in current directory, settings.gradle file will be there, where we mention the project name to include, then gradle will go into that folder
2. “project” is an class in gradle same like java object, same like java we can declare string variables

subprojects(**{**

apply plugin "application"

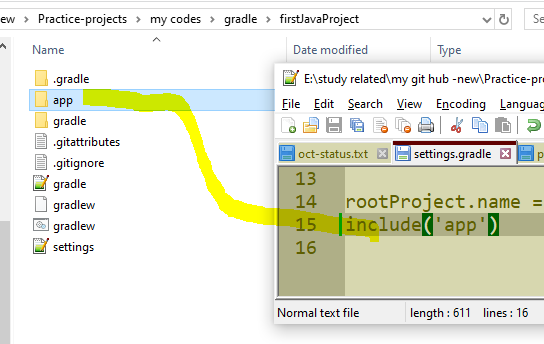
**version** ="1.0"  
**group** ="com.bharath"

1. This is also like java, here also we have methods , we can call those by passing arguments as from build.gradle file

allprojects(**{** task global **{** println("i am a global task star ramcharan")  
 **}  
}**)

Here for all projects method we passed a closure

1. dsf



See here in this folder we don’t have build.gradle, but in settings.gradle we included the project named “app” inside that folder we will have the build.gradle

Gradle Commands

|  |  |
| --- | --- |
| To list/see all tasks | gradle tasks --all |
| To execute a task | gradle <task name> ex:- gradle test, gradle clean |
| To create gradle project – we can create java project with this command | Gradle init |
| to build | gradle build |
| to clean build folder / to delete the target folder | gradle clean |
| to run spring boot app | gradle bootRun |
| To run the test cases | gradle test // this will run the test task and execute test cases |
| To create jar file | gradle jar |
| To run the jar file | gradle run |
| to see all dependencies | gradle dependencies |
| to see dependencies for a sub module | gradle :<module name>:dependencies |

Create gradle project using cmnd prompt

Gradle init

Select build script as groovy

By default command this will create 1 sub module project

By default the root project doesn’t have any build.gradle

Tasks

Note:- u can print the data in both single or double quotes

1. Creating all the tasks in build.gradle file
2. To run any task 🡪 gradle <task-name>

Here manicopy is the task name

|  |  |
| --- | --- |
| task manicopy{  println 'manideep'  } | task printer**{** print("i am from root build.gradle file") **}**  **to run gradle <Give your task-name>**  gradle printer |

|  |  |
| --- | --- |
| To compile | gradle app:compileJava - Compiles main Java source., here app is the project name |
| To run the test cases | gradle test // this will run the test task and execute test cases |
| To create jar file | gradle jar |
| To run the jar file | gradle run |
|  | Gradle build |
| to see all existing tasks – with this we can even see all tasks for all multi module projects | gradle tasks --all |

Plugin

A Gradle plugin is a powerful tool that extends Gradle's capabilities and automates complex build tasks.

Plugin will do lot of tasks like compiling java code, running test cases …

Think of it as a pre-packaged set of instructions that you can easily add to your project to perform specific actions.

These are like our spring libraries which will do lot of tasks for us

plugins **{** *// Apply the application plugin to add support for building a CLI application in Java.* id 'application'  
**}**

1 plugin can extend another plugin same like java classes and from 1 plugin we will get lot of tasks

**Here's a breakdown of what Gradle plugins do and why they're useful:**

**1. Automating Build Tasks**

* Plugins streamline common build tasks, such as compiling code, running tests, packaging applications, and deploying them to different environments.
* By using plugins, you avoid writing repetitive code for these tasks, saving time and effort.

**2. Integrating with External Tools**

* Many plugins integrate with external tools and services, such as code analysis tools, dependency management systems, and cloud platforms.
* This makes it easy to incorporate these tools into your build process.

**3. Tailoring the Build Process**

* Plugins allow you to customize your build process to meet your project's specific needs.
* You can configure plugins to perform tasks in a certain order, include or exclude specific files, and set various build properties.

**4. Organizing Build Logic**

* Plugins help organize build logic by encapsulating it into reusable components.
* This makes your build scripts more modular and easier to maintain.

**Types of Gradle Plugins**

* **Core Plugins:** These are included with Gradle and provide essential functionality, such as Java compilation and dependency management.
* **Community Plugins:** These are developed by third-party developers and shared on the Gradle Plugin Portal or other repositories. They offer a wide range of functionalities for various needs.
* **Custom Plugins:** You can create your own plugins to address specific requirements not covered by existing plugins.

Scopes

compileOnly(only for compilation purposes like lombok), runtimeOnly(use only if these jars are provided by server while deploying )

implementation, testImplementation

implementation – means for both compilation and runtime

Build phases

1. Initialization : - it will verify the settings.gradle and checks if it is a single or multi module project
2. Configuration : here it will check all tasks and creates DAG- direct acyclic graph of all tasks which one to execute first and which to execute later
3. Execution:- it will execute all tasks

Multi module project

Gradle init – not only for multi module to create any project u can use this

When u create using above, by default build.gradle will be absent for root module

1. Create a module – right click on main project—new -- module
2. When u

Configure child /sub project

1. every child project should have a build.gradle hence for every sub module 1 – jar file will be created
2. Include that child module name in main settings.gradle as

**rootProject**.**name** = 'firstJavaProject'  
include('app') //here app is the sub/new module name

Submodule tasks from main module

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | task collectCashInte**{** print("i am a task from integration sub module build.gradle ") **}**  **suppose if I placed this above sub task in intergration build.gradle file**  **u can still run it from main project**  **Executing commands only for certain sub module (most used in real time)**  gradle :<sub module name>:<task name>  gradle :integration:collectCashInte //inside integration sub module run a task called collectCashInte   |  |  | | --- | --- | | Bec in real time, main project will have bdd sub module, we may want to build only that module | gradle :bdd:build | |  |  |   to clean only web project  gradle :web:clean   * + Here web is one of the sub module name as per image, clean is the predefined task name   task only for 1 project  task global **{** println("i am a global task star ramcharan") **}**  if u keep like this in root build.gradle then this task will be available only for root project  gradle :integration:global //It says run this global task from integration sub module  the above will throw error, because that task is present only in root module, by default it will not be available for all its child |
| making a task visible for parent project + sub all modules | allprojects(**{** task global **{** println("i am a global task star ramcharan")  **} }**)  if u pass that task to allProjects closure then it will be available for all |
| making a task visible only for sub modules (not root module) | subprojects(**{  var** p=**project** task global **{** println("i am a global task star ramcharan --"+p.**name**)  **} }**) |
| if u want same build.gradle file for all sub modules then  we can keep all build.gradle contents in subprojects{} closure | subprojects(**{**  apply plugin "application"  **version** ="1.0" **group** ="com.bharath"  **plugins {   }** dependencies**{   }** repositories **{    }  var** p = **project** task global **{** println("i am a global task star ramcharan --" + p.**name**)  **} }**)  this subprojects() & allprojects() closures will accept all the content |
| adding one module as lib to another module  adding one project as lib to another project | Here for integration module, we are adding services module as library  // with this services module will be added as a lib to integration module   |  |  | | --- | --- | | so goto integration module build.gradle file and add below  dependencies **{** *// This dependency is used by the application.* implementation libs.**guava** implementation project(path: ':services') **}**  **here : indicates services is a sub projects** | to check whether that sub module “services” got added to integration module try below command  gradle :<module name>:dependencies  gradle :integration:dependencies // Here integration is my module name | | add root project to child project  implementation project(:) // Here : means root project |  |   the below entry in my root build.gradle didn’t work for me  project(":integration")**{** dependencies **{** implementation project(":services")  **} }** |
| to see dependencies for a sub module | gradle :<module name>:dependencies  gradle :integration:dependencies // Here integration is my module name  output of above command  compileClasspath - Compile classpath for source set 'main'. +--- com.google.guava:guava:32.1.3-jre | +--- com.google.guava:failureaccess:1.0.1 | +--- com.google.guava:listenablefuture:9999.0-empty-to-avoid-conflict-with-guava | +--- com.google.code.findbugs:jsr305:3.0.2 | +--- org.checkerframework:checker-qual:3.37.0 | +--- com.google.errorprone:error\_prone\_annotations:2.21.1 | \--- com.google.j2objc:j2objc-annotations:2.8 \--- project :services  implementation - Implementation dependencies for the 'main' feature. (n) +--- com.google.guava:guava:32.1.3-jre (n) \--- project services (n)  we can see that at both compilation & at run time stages we are seeing project was set as classpath for integration module |
|  |  |
|  |  |
|  |  |

FQA

1. how to make gradle to use certain version of java to build

keep this in build.gradle

java **{  
 sourceCompatibility** = JavaVersion.***VERSION\_11* targetCompatibility** = JavaVersion.***VERSION\_11*}**

1. we dont need to mention the version of jar then how gradle is exactly fetching that single jar??

it will intenally use spring boot dependencies bom bill of materials that bom contains all the compatible jar versions 16

in gradle How to reference a jar that Is present locally

To add a jar locally

* paste that jar in libs/new folder
* add that lib/jar to project "module settings" -->"libraries"--> add that library (i mean fol
* add this entry in build.gradle

implementation fileTree(dir: 'libs/new', include: ['\*.jar'])

settings.gradle

repositories {

flatDir {

dirs("libs")

}

When external Libraries are not visible

then execute a plugin called "idea" then it will generate a iml file with all dependencies, then and then all those ext lib will be visible

plugins {

id 'java-library'

id 'maven-publish'

id 'jacoco'

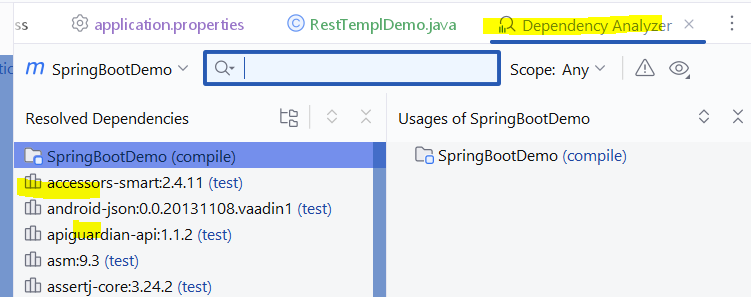
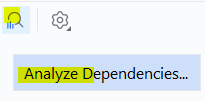
id 'idea'

}

command --> \gradlew.bat IDEA

How to see finalized dependencies in our project

1. u can see in left side Intellij External dependencies
2. using analyse dependencies icon in gradle tab and from which parent that jar came it is like



This is Intellij feature it seems

U can see all these dependencies / jars finally took into our project