https://technologyconversations.com/2014/06/18/build-tools/

<http://www.vogella.com/tutorials/ApacheMaven/article.html>

<https://spring.io/guides/gs/maven/>

<https://spring.io/guides/gs/gradle/>

Gradle is a build tool similar to Maven and Ant. If you are a Java developer you will most likely have used either Ant or Maven already. Gradle is the latest of these three build tools, and Gradle is starting to get wider adoption (at the time of writing in 2015).

Some of the advantages Gradle has over Maven and Ant is more concise build scripts and a more flexible build script language. One disadvantage Gradle has is that it is incredibly slow! (come on, Gradle devs, you can do better than that!).

**Gradle Core Concepts**

To better understand Gradle I will briefly explain the core concepts of Gradle. The Gradle core concepts are:

* Projects
* Tasks
* Build scripts

A project is typically some software you want to build. Tasks are actions needed to build the project. A task could be compiling the source code, generating JavaDoc, zipping the compiled classes into a JAR file etc. A Gradle project contains one or more tasks to execute in order to build the project.

A project typically has a build script in which its tasks are defined. The build script is used by the gradlecommand so it can know what tasks are defined for the project.

The build script is typically called build.gradle and is normally located in the root directory of the project you are building. When the gradle command is executed it looks in the directory which the command is executed in, for the build script to execute.

You will see projects, tasks and build scripts mentioned again and again in the following texts in this Gradle tutorial.

**Gradle Requirements**

Gradle requires a Java SDK 6 or higher on the computer Gradle is to run on. Before you install Gradle, make sure you have a Java SDK 6 or higher installed. Gradle will use whatever Java SDK it finds on your system's path (environment.

If you want to use a specific Java version (if you have more than one Java SDK installed) you can set theJAVA\_HOME environment variable to point to the install directory of the Java SDK you want to use.

**Gradle Install**

To install Gradle, simply download the version of Gradle you want to install, and unzip it into the directory you want Gradle installed in.

When Gradle is unzipped into the install directory, add the INSTALL-DIR/bin to the path environment variable. The INSTALL-DIR part of the added path must be the full path to the Gradle install directory (and not "INSTALL-DIR" as text).

To test if Gradle is properly installed, open a command prompt and execute this command:

gradle -version

If Gradle is correctly installed, you should see an output similar to this:

C:\Users\jjenkov>gradle -version

------------------------------------------------------------

Gradle 2.5

------------------------------------------------------------

Build time: 2015-07-08 07:38:37 UTC

Build number: none

Revision: 093765bccd3ee722ed5310583e5ed140688a8c2b

Groovy: 2.3.10

Ant: Apache Ant(TM) version 1.9.3 compiled on December 23 2013

JVM: 1.8.0\_45 (Oracle Corporation 25.45-b02)

OS: Windows 7 6.1 x86

**Gradle commands:**

**Build Setup tasks:**

**init - Initializes a new Gradle build.**

**wrapper - Generates Gradle wrapper files.**

**Help tasks:**

**buildEnvironment - Displays all buildscript dependencies declared in root project 'Gradle'.**

**components - Displays the components produced by root project 'Gradle'. [incubating]**

**dependencies - Displays all dependencies declared in root project 'Gradle'.**

**dependencyInsight - Displays the insight into a specific dependency in root project 'Gradle'.**

**dependentComponents - Displays the dependent components of components in root project 'Gradle'. [incubating]**

**help - Displays a help message.**

**model - Displays the configuration model of root project 'Gradle'. [incubating]**

**projects - Displays the sub-projects of root project 'Gradle'.**

**properties - Displays the properties of root project 'Gradle'.**

**tasks - Displays the tasks runnable from root project 'Gradle'.**

**To see all tasks and more detail, run gradle tasks --all**

**To see more detail about a task, run gradle help --task <task>**

*Gradle tasks* define the actions needed to build your project. Without tasks your build script will not build anything when Gradle executes it. In this tutorial we will see how to define tasks in Gradle build scripts.

## Defining Gradle Tasks

You define a Gradle task inside the Gradle build script. You can define the task pretty much anywhere in the build script.

A task definition consists of the keyword task and then the name of the task, like this:

task myTask

This line defines a task named myTask. The task doesn't do anything yet - it has only been defined.

## Task Execution Phases

When a Gradle task is executed the execution passes through two phases:

* first
* last

A task is represented internally in Gradle as an object. To hook into the two build phases you must call one of the corresponding methods, passing a closure as parameter (a closure is an anonymous function). The corresponding build phase methods are:

* doFirst()
* doLast()

Here is an example of how to call these two methods in a task:

task myTask

myTask.doFirst( {

println "This is done first"

} );

myTask.doLast( {

println "This is done last"

} );

First, this example defines a task named myTask. This you have already seen earlier in this tutorial.

Second, this example calls the myTask doFirst() method, passing along a closure as parameter. The closure is contained within the { and } characters. Everything inside the curly brackets ({ and }) is interpreted as a separate function.

Third, this example calls the myTask doLast() method, passing along another closure as parameter.

When the task myTask is executed Gradle will first execute all closures passed as parameters the doFirst()method, and then all closures passed as parameters the doLast() method.

### Shorter Syntax

A Gradle script is a Groovy script and Groovy allows a few syntax optimizations to make your Groovy scripts shorter.

First of all you do not need the semicolon ; after the doFirst() and doLast() method calls, if the code line with the method calls ends with a new line.

Second, you do not need the parentheses around the closure when calling doFirst() or doLast().

With these two syntax optimizations you can write the doFirst() and doLast() method calls like this:

myTask.doFirst {

println "This is done first"

}

myTask.doLast {

println "This is done last"

}

This shorter syntax is what you will most often see in the official Gradle documentation and in other Gradle tutorials.

### doLast() Shortcut Notation

Often you will not need to pass closures to both doFirst() and doLast() . It is quite common to pass just a single closure to the doLast() method. Because this is such a common use case, Gradle has a shortcut for passing a closure to the doLast() method. Here is how the shortcut doLast() call notation looks:

myTask <<{

println "This is done last"

}

The two characters << corresponds to calling the doLast() method like this:

myTask.doLast {

println "This is done last"

}

You can even define the task on the same line as you pass the closure to the doLast() method. Here is an example of how that looks:

task myTask << {

println "This is done last"

}

Defining the task and last closure in the same Groovy statement makes the build script more concise.

## Multiple Closures For the Same Execution Phase

It is actually possible to add multiple closures for the same task execution phases. Here is an example:

myTask.doLast {

println "This is done 2nd last"

}

myTask.doLast {

println "This is done last"

}

This examples adds two closures to the last execution phase.

Multiple closures added to the same execution phase will be executed in the sequence they were added to the execution phase. In the example above, first the closure printing the text "This is done 2nd last" would be executed, then the closure printing the "This is done last" text would be executed.

All closures added to the first execution phase will of course be executed before any of the closures added to the last execution phase.

You can also add multiple closures using the doLast() shortcut notation. Here is an example:

myTask << {

println "This is done 2nd last"

}

myTask << {

println "This is done last"

}

## Default Tasks

A Gradle build script can contain one or more default tasks. A default task is a task that is executed if you do not provide any task name to execute to the gradle command. If you open a command prompt and change directory into the directory where your project (incl. gradle build script) is located, and execute this command:

gradle

Then Gradle will execute the default tasks listed in the Gradle build script for that project.

You specify the default task for a Gradle build script like this:

defaultTasks 'firstTask'

This example sets the task named firstTask as default task for the gradle build script.

If you need more than one default task, you can list the extra tasks after the first task, with the task names separated by comma. Here is an example:

defaultTasks 'firstTask', 'secondTask'

## Task Dependencies

A task can depend on another task, just like in Ant. If task B depends on task A, that means that task A must have been executed before task B is executed. If you execute task B then Gradle will detect the dependency and execute task A first, then task B.

Gradle task dependencies are listed inside parentheses after the task name. Here is a Gradle task dependency example:

task firstTask << {

println "This is first task"

}

task myTask (dependsOn:firstTask) << {

println "This is myTask"

}

This example defines two tasks called firstTask and myTask. The myTask task depends on the firstTasktask. This is stated inside the parentheses after the name of myTask. More specifically, by this code:

(dependsOn:firstTask)

When you execute myTask Gradle will first execute firstTask and then myTask because myTask depends on firstTask .

## Real Build Actions

The tasks you have seen so far only printed out text. I kept the tasks simple so that you could more easily understand what was going on. In later texts in this Gradle tutorial trail you will see how to actually compile Java code, run unit tests, create JAR files and similar build actions.