**Why we need a build tool?**

* We need to compile our source code
* We need to add *jar* files to the project
* We need to test our projects functionality
* We need to package our files

We need build tools to fulfil all the above mentioned tasks.

***Gradle*** is a build tool.

**How does gradle know how to build the project?**

We need to describe:

* Application type
* Libraries our application depends on
* Configuration about related to packaging and testing

We specify all these in a file named ***build script***.

***What is incremental build?***

An optimization in gradle that skips running tasks which have been executed previously with the same inputs.

|  |
| --- |
|  |

***Maven vs Gradle***

Gradle was designed to solve some of maven’s shortcomings:

* Code-based script rather than xml
* Writing plugins is easier
* It is faster

***Why gradle is said to be faster than maven:***

* Incremental build: only re-executes tasks whose inputs/outputs have changed
* Build cache: it caches the outputs and use the same outputs for the subsequent tasks with the same inputs even on different environments
* Parallel execution: gradle can run tasks in parallel

***Gradle installation***

We need java (8 or above)

Download gradle

Unzip it

Set the environment variable to the bin folder

Open *cmd*  and type ***gradle –version***

***Create a simple project***

* Create a folder and navigate to it
* Type ***gradle init***
* follow the remaining steps

|  |
| --- |
|  |

***Project files:***

* setting.gradle: sets up high level configurations like the project name
* build.gradle: the build script
* gradlew(.bat): build an app without having to download *gradle*
* .gitignore: configures git so that .gradle and build directories are not committed

***Gradle fundamental components***

|  |
| --- |
|  |

* Project: a container for everything *gradle* knows about your project
* Build script: a file containing information about how to build the application
* Task: individual build actions one can run from cmd. E.g. compile task, test task… we could also define our own custom task in build script
* Plugins: when applied, it adds tasks to the project to achieve some particular outcome

***groovy***

it also runs on JVM.

|  |
| --- |
|  |

It is scripting so you can write code out of a class and execute it.

It is dynamically typed so you can declare variables using ***def*.** semicolon at the end of the line and () are optional.

We can define **closure** using brackets. Closures are blocks of code which could be passed around.

|  |
| --- |
|  |

If we are calling a method with multiple arguments using round brackets, if the last argument is a closure, then it can go outside of the brackets.

|  |
| --- |
|  |

***Build a java project***

First apply the java plugin

|  |
| --- |
|  |

This is equal to:

Plugins(closure) 🡪 Plugins(Id(‘java’))

This plugin has added a few tasks. The most important one is ***build***.

Let’s run the build task 🡪 *gradlew build*

Then the ***build*** folder would be added to the project.

The jar file is in ***build/lib*** but it is not executable since the **manifest** file does not contain *Main-Class*. We fix this issue writing:

|  |
| --- |
|  |

Jar(

manifest(attributes(map.of(“Main-Class”, ”com….”)))

)

**Declaring repositories**

|  |
| --- |
|  |

**Adding dependencies**

|  |
| --- |
|  |

***Types of dependency configuration***

*api*: the dependency is accessible to other modules in the project

*implementation:* needed to compile the source code and purely internal

*compileOnly:* needed only at compile time (dependencies don’t appear in the package)

*runtimeOnly:* required only at runtime

*testImplementation, testCompileOnly, testRuntimeOnly*