Gradle

Project Layout of a Java Project

The default project layout of a Java project is following:

- The src/main/java directory contains the source code of our project.
- The src/main/resources directory contains the resources (such as properties files) of our project.
- The src/test/java directory contains the test classes.
- The src/test/resources directory contains the test resources.

All output files of our build are created under the build directory. This directory contains the following subdirectories which are relevant to this blog post (there are other subdirectories too, but we will talk about them in the future):

- The classes directory contains the compiled .class files.
- The libs directory contains the jar or war files created by the build.

Basic

```
plugins {
    id 'java'
}

sourceCompatibility = '1.8'
targetCompatibility = '1.8'
version = '1.2.1'
```

By applying the Java Plugin, you get a whole host of features:

- A `compileJava` task that compiles all the Java source files under *src/main/java*
- A `compileTestJava` task for source files under *src/test/java*
- A `test` task that runs the tests from *src/test/java*
- A `jar` task that packages the `main` compiled classes and resources from *src/main/resources* into a single JAR named *<project>-<version>.jar*
- A `javadoc` task that generates Javadoc for the `main` classes

Managing Dependencies

```
repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.hibernate:hibernate-core:3.6.7.Final'
}
```

The Gradle terminology for the three elements is as follows:

- *Repository* (ex: `mavenCentral()`) — where to look for the modules you declare as dependencies
- *Configuration* (ex: `implementation`) - a named collection of dependencies, grouped together for a specific goal such as compiling or running a module — a more flexible form of Maven scopes
- *Module coordinate* (ex: `org.hibernate:hibernate-core-3.6.7.Final`) — the ID of the dependency, usually in the form '*<group>*:*<module>*:*<version>*' (or '*<groupId>*:*<artifactId>*:*<version>*' in Maven terminology)

You can find a more comprehensive glossary of dependency management terms here.

As far as configurations go, the main ones of interest are:

- `compileOnly` — for dependencies that are necessary to compile your production code but shouldn't be part of the runtime classpath
- `implementation` (supersedes `compile`) — used for compilation and runtime
- `runtimeOnly` (supersedes `runtime`) — only used at runtime, not for compilation
- `testCompileOnly` — same as `compileOnly` except it's for the tests

- `testImplementation` — test equivalent of `implementation`
- `testRuntimeOnly` — test equivalent of `runtimeOnly`

You can learn more about these and how they relate to one another in the plugin reference chapter.

Source Sets

Imagine you have a legacy project that uses an *src* directory for the production code and *test* for the test code. The conventional directory structure won't work, so you need to tell Gradle where to find the source files. You do that via source set configuration.

Each source set defines where its source code resides, along with the resources and the output directory for the class files. You can override the convention values by using the following syntax:

```
sourceSets {
    main {
         java {
            srcDirs = ['src']
         }
    }

    test {
        java {
            srcDirs = ['test']
        }
    }
}
```

Samples

```
version '1.0-SNAPSHOT'

apply plugin: 'java'

sourceCompatibility = 1.11
targetCompatibility = 1.11

repositories {
    jcenter()
}

dependencies {
    testCompile project(":cipher")
    testCompile project(":is")
    testCompile group: 'javax.validation', name: 'validation-api', version: '2.0.1.Final'
    testCompile 'com.google.protobuf:protobuf-java:3.6.1'
    testCompile group: 'junit', name: 'junit', version: '4.12'
    testCompile('org.testcontainers:testcontainers:1.10.5')
    testCompile('org.assertj:assertj-core:3.8.0')
    testCompile('io.rest-assured:rest-assured:3.0.3')
    testCompile('io.rest-assured:json-schema-validator:3.0.3')
    testCompile('io.rest-assured:json-path:3.0.3')
    testCompile('ch.qos.logback:logback-classic:1.2.3')
    testCompile('io.cucumber:cucumber-java8:4.7.1')
    testCompile('io.cucumber:cucumber-junit:4.7.1')
    testCompile('io.cucumber:cucumber-picocontainer:3.0.2')
    testCompile('org.bouncycastle:bcprov-jdk15on:1.60')
    testCompile('com.auth0:java-jwt:3.4.0')
    testCompile('org.apache.commons:commons-lang3:3.7')
    testCompile('org.mock-server:mockserver-client-java:5.5.1')
    testCompile group: 'com.rabbitmq', name: 'amqp-client', version: '5.5.2'
    testCompile group: 'org.json', name: 'json', version: '20180813'
    testCompile group: 'commons-cli', name: 'commons-cli', version: '1.4'
}

task testJar(type: Jar) {
    from { configurations.testCompile.collect { it.isDirectory() ? it : zipTree(it) } }
    from sourceSets.test.output.classesDirs
    from sourceSets.test.resources
    jar

    exclude "META-INF/*.SF"
    exclude "META-INF/*.DSA"
    exclude "META-INF/*.RSA"

}

task copyBVTDockerfile(type: Copy) {
    from 'src/main/docker'
    into 'build/libs'
}


task buildBVTDockerImage(type: Exec) {
    workingDir "build/libs"
    commandLine "docker", "build", "--build-arg", "VERSION=${project.version}", "--tag", "${rootProject.name}
/${project.name}:${project.version}", "./"
}

testJar.dependsOn(testClasses)

buildBVTDockerImage.dependsOn(copyBVTDockerfile,testJar)
build.dependsOn(buildBVTDockerImage)
```

References

Reference	URL
Building Java & JVM projects	https://docs.gradle.org/current/userguide/building_java_projects.html