Docs

[User Manual](http://docs.google.com/userguide/userguide.html)

[Guides and Tutorials](https://guides.gradle.org)

[DSL Reference](http://docs.google.com/dsl/)

[Javadoc](http://docs.google.com/javadoc/)

[Release Notes](http://docs.google.com/release-notes.html)

[Forums](https://discuss.gradle.org/)

[Training](https://gradle.org/training/)

[Try Gradle Enterprise](https://gradle.com/enterprise)

[PDF](http://docs.google.com/userguide/userguide.pdf)

* [User Manual Home](http://docs.google.com/userguide/userguide.html)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Installing Gradle](http://docs.google.com/userguide/installation.html)
* [Tutorials](https://guides.gradle.org/)

### Reference

* [Groovy DSL Reference](http://docs.google.com/dsl/)
* [Gradle API Javadoc](http://docs.google.com/javadoc/)
* [Core Plugins](http://docs.google.com/userguide/plugin_reference.html)
* [Gradle & Third-party Tools](http://docs.google.com/userguide/third_party_integration.html)

### Getting Started

* [Creating New Gradle Builds](https://guides.gradle.org/creating-new-gradle-builds/)
* [Creating Build Scans](https://guides.gradle.org/creating-build-scans/)
* [Migrating From Maven](https://guides.gradle.org/migrating-from-maven/)

### Running Gradle Builds

* [Command-Line Interface](http://docs.google.com/userguide/command_line_interface.html)
* [Customizing Execution](#gjdgxs)
  + [Configuring the Build Environment](http://docs.google.com/userguide/build_environment.html)
  + [Configuring the Gradle Daemon](http://docs.google.com/userguide/gradle_daemon.html)
  + [Initialization Scripts](http://docs.google.com/userguide/init_scripts.html)
* [Directory Layout](http://docs.google.com/userguide/directory_layout.html)
* [Executing Multi-Project Builds](http://docs.google.com/userguide/intro_multi_project_builds.html)
* [Gradle Wrapper](http://docs.google.com/userguide/gradle_wrapper.html)
* [Troubleshooting](http://docs.google.com/userguide/troubleshooting.html)
* [Using Build Scans](https://docs.gradle.com/build-scan-plugin)
* [Enabling and Configuring the Build Cache](http://docs.google.com/userguide/build_cache.html)
* [Integrating Separate Gradle Builds (Composite Builds)](http://docs.google.com/userguide/composite_builds.html)

### Authoring Gradle Builds

* [Fundamentals](#30j0zll)
  + [Introducing the Basics of Build Scripts](http://docs.google.com/userguide/tutorial_using_tasks.html)
  + [Working with Tasks](http://docs.google.com/userguide/more_about_tasks.html)
  + [Learning More About Build Scripts](http://docs.google.com/userguide/writing_build_scripts.html)
  + [Working with Files](http://docs.google.com/userguide/working_with_files.html)
  + [Creating Custom Task Types](http://docs.google.com/userguide/custom_tasks.html)
  + [Using Gradle Plugins](http://docs.google.com/userguide/plugins.html)
  + [The Standard Gradle Plugins](http://docs.google.com/userguide/standard_plugins.html)
  + [Understanding the Build Lifecycle](http://docs.google.com/userguide/build_lifecycle.html)
  + [Working with Logging](http://docs.google.com/userguide/logging.html)
  + [Configuring Multi-Project Builds](http://docs.google.com/userguide/multi_project_builds.html)
* [Best Practices](#1fob9te)
  + [Authoring Maintainable Build Scripts](http://docs.google.com/userguide/authoring_maintainable_build_scripts.html)
  + [Organizing Gradle Projects](http://docs.google.com/userguide/organizing_gradle_projects.html)
  + [Optimizing Build Performance](https://guides.gradle.org/performance/)
  + [Using the Build Cache](https://guides.gradle.org/using-build-cache/)
* [Dependency Management](#3znysh7)
  + [Introduction to Dependency Management](http://docs.google.com/userguide/introduction_dependency_management.html)
  + [Dependency Management Terminology](http://docs.google.com/userguide/dependency_management_terminology.html)
  + [Dependency Types](http://docs.google.com/userguide/dependency_types.html)
  + [Repository Types](http://docs.google.com/userguide/repository_types.html)
  + [Declaring Dependencies](http://docs.google.com/userguide/declaring_dependencies.html)
  + [Declaring Repositories](http://docs.google.com/userguide/declaring_repositories.html)
  + [Inspecting Dependencies](http://docs.google.com/userguide/inspecting_dependencies.html)
  + [Managing Dependency Configurations](http://docs.google.com/userguide/managing_dependency_configurations.html)
  + [Managing Transitive Dependencies](http://docs.google.com/userguide/managing_transitive_dependencies.html)
  + [Dependency Locking](http://docs.google.com/userguide/dependency_locking.html)
  + [Troubleshooting Dependency Resolution](http://docs.google.com/userguide/troubleshooting_dependency_resolution.html)
  + [Customizing Dependency Resolution Behavior](http://docs.google.com/userguide/customizing_dependency_resolution_behavior.html)
  + [Dependency Cache Internals](http://docs.google.com/userguide/dependency_cache.html)
  + [Working with Dependencies](http://docs.google.com/userguide/working_with_dependencies.html)
* [Publishing Artifacts](http://docs.google.com/userguide/artifact_management.html)
* [C++ Projects](#2et92p0)
  + [Building Native Software](http://docs.google.com/userguide/native_software.html)
  + [Software Model Concepts](http://docs.google.com/userguide/software_model_concepts.html)
  + [Rule-based Model Configuration](http://docs.google.com/userguide/software_model.html)
  + [Implementing Model Rules in a Plugin](http://docs.google.com/userguide/rule_source.html)
  + [Extending the Software Model](http://docs.google.com/userguide/software_model_extend.html)
* [Java Projects](#tyjcwt)
  + [Building Java & JVM projects](http://docs.google.com/userguide/building_java_projects.html)
  + [Testing Java & JVM projects](http://docs.google.com/userguide/java_testing.html)
* [Advanced Techniques](#3dy6vkm)
  + [Configuring Tasks Lazily](http://docs.google.com/userguide/lazy_configuration.html)
  + [Developing Parallel Tasks](https://guides.gradle.org/using-the-worker-api/)
  + [Testing Your Build with TestKit](http://docs.google.com/userguide/test_kit.html)
  + [Using Ant from Gradle](http://docs.google.com/userguide/ant.html)
* [Sample Gradle builds](#1t3h5sf)
  + [Groovy DSL Samples](https://github.com/gradle/gradle/tree/master/subprojects/docs/src/samples)
  + [Kotlin DSL Samples](https://github.com/gradle/kotlin-dsl/tree/master/samples)

### Extending Gradle

* [Writing Custom Plugins](http://docs.google.com/userguide/custom_plugins.html)
* [Plugin Development Guides](https://gradle.org/guides/?q=Plugin+Development)

[Edit this page](https://github.com/gradle/gradle/edit/master/subprojects/docs/src/docs/userguide/)

# Java Quickstart

Contents

[The Java plugin](#4d34og8)

[A basic Java project](#2s8eyo1)

[Multi-project Java build](#17dp8vu)

[Where to next?](#3rdcrjn)

[The Java plugin](#4d34og8)

As we have seen, Gradle is a general-purpose build tool. It can build pretty much anything you care to implement in your build script. Out-of-the-box, however, it doesn’t build anything unless you add code to your build script to do so.

Most Java projects are pretty similar as far as the basics go: you need to compile your Java source files, run some unit tests, and create a JAR file containing your classes. It would be nice if you didn’t have to code all this up for every project. Luckily, you don’t have to. Gradle solves this problem through the use of *plugins*. A plugin is an extension to Gradle which configures your project in some way, typically by adding some pre-configured tasks which together do something useful. Gradle ships with a number of plugins, and you can easily write your own and share them with others. One such plugin is the *Java plugin*. This plugin adds some tasks to your project which will compile and unit test your Java source code, and bundle it into a JAR file.

The Java plugin is convention based. This means that the plugin defines default values for many aspects of the project, such as where the Java source files are located. If you follow the convention in your project, you generally don’t need to do much in your build script to get a useful build. Gradle allows you to customize your project if you don’t want to or cannot follow the convention in some way. In fact, because support for Java projects is implemented as a plugin, you don’t have to use the plugin at all to build a Java project, if you don’t want to.

We have in-depth coverage with many examples about the Java plugin, dependency management and multi-project builds in later chapters. In this chapter we want to give you an initial idea of how to use the Java plugin to build a Java project.

[A basic Java project](#2s8eyo1)

Let’s look at a simple example. To use the Java plugin, add the following to your build file:

[Example: Using the Java plugin](#26in1rg)

**build.gradle**

apply plugin: 'java'

| **✨** | The code for this example can be found at samples/java/quickstart in the ‘-all’ distribution of Gradle. |
| --- | --- |

This is all you need to define a Java project. This will apply the Java plugin to your project, which adds a number of tasks to your project.

| **💡** | What tasks are available?  You can use gradle tasks to list the tasks of a project. This will let you see the tasks that the Java plugin has added to your project. |
| --- | --- |

Gradle expects to find your production source code under src/main/java and your test source code under src/test/java. In addition, any files under src/main/resources will be included in the JAR file as resources, and any files under src/test/resources will be included in the classpath used to run the tests. All output files are created under the build directory, with the JAR file ending up in the build/libs directory.

[Building the project](#lnxbz9)

The Java plugin adds quite a few tasks to your project. However, there are only a handful of tasks that you will need to use to build the project. The most commonly used task is the build task, which does a full build of the project. When you run gradle build, Gradle will compile and test your code, and create a JAR file containing your main classes and resources:

[Example: Building a Java project](#35nkun2)

**Output of** gradle build

> gradle build  
> Task :compileJava  
> Task :processResources  
> Task :classes  
> Task :jar  
> Task :assemble  
> Task :compileTestJava  
> Task :processTestResources  
> Task :testClasses  
> Task :test  
> Task :check  
> Task :build  
  
BUILD SUCCESSFUL in 0s  
6 actionable tasks: 6 executed

Some other useful tasks are:

clean

Deletes the build directory, removing all built files.

assemble

Compiles and jars your code, but does not run the unit tests. Other plugins add more artifacts to this task. For example, if you use the War plugin, this task will also build the WAR file for your project.

check

Compiles and tests your code. Other plugins add more checks to this task. For example, if you use the checkstyle plugin, this task will also run Checkstyle against your source code.

[External dependencies](#1ksv4uv)

Usually, a Java project will have some dependencies on external JAR files. To reference these JAR files in the project, you need to tell Gradle where to find them. In Gradle, artifacts such as JAR files, are located in a *repository*. A repository can be used for fetching the dependencies of a project, or for publishing the artifacts of a project, or both. For this example, we will use the public Maven repository:

[Example: Adding Maven repository](#44sinio)

**build.gradle**

repositories {  
 mavenCentral()  
}

Let’s add some dependencies. Here, we will declare that our production classes have a compile-time dependency on commons collections, and that our test classes have a compile-time dependency on junit:

[Example: Adding dependencies](#2jxsxqh)

**build.gradle**

dependencies {  
 compile group: 'commons-collections', name: 'commons-collections', version: '3.2.2'  
 testCompile group: 'junit', name: 'junit', version: '4.+'  
}

You can find out more in [Dependency Management for Java Projects](http://docs.google.com/dependency_management_for_java_projects.html#dependency_management_for_java_projects).

[Customizing the project](#z337ya)

The Java plugin adds a number of properties to your project. These properties have default values which are usually sufficient to get started. It’s easy to change these values if they don’t suit. Let’s look at this for our sample. Here we will specify the version number for our Java project, along with some attributes to the JAR manifest.

[Example: Customization of MANIFEST.MF](#3j2qqm3)

**build.gradle**

version = '1.0'  
jar {  
 manifest {  
 attributes 'Implementation-Title': 'Gradle Quickstart',  
 'Implementation-Version': version  
 }  
}

| **💡** | What properties are available?  You can use gradle properties to list the properties of a project. This will allow you to see the properties added by the Java plugin, and their default values. |
| --- | --- |

The tasks which the Java plugin adds are regular tasks, exactly the same as if they were declared in the build file. This means you can use any of the mechanisms shown in earlier chapters to customize these tasks. For example, you can set the properties of a task, add behaviour to a task, change the dependencies of a task, or replace a task entirely. In our sample, we will configure the test task, which is of type [Test](http://docs.google.com/dsl/org.gradle.api.tasks.testing.Test.html), to add a system property when the tests are executed:

[Example: Adding a test system property](#1y810tw)

**build.gradle**

test {  
 systemProperties 'property': 'value'  
}

[Publishing the JAR file](#4i7ojhp)

Usually the JAR file needs to be published somewhere. To do this, you need to tell Gradle where to publish the JAR file. In Gradle, artifacts such as JAR files are published to repositories. In our sample, we will publish to a local directory. You can also publish to a remote location, or multiple locations.

[Example: Publishing the JAR file](#2xcytpi)

**build.gradle**

uploadArchives {  
 repositories {  
 flatDir {  
 dirs 'repos'  
 }  
 }  
}

To publish the JAR file, run gradle uploadArchives.

[Creating an Eclipse project](#1ci93xb)

To create the Eclipse-specific descriptor files, like .project, you need to add another plugin to your build file:

[Example: Eclipse plugin](#3whwml4)

**build.gradle**

apply plugin: 'eclipse'

Now execute gradle eclipse command to generate Eclipse project files. More information about the eclipse task can be found in the [Eclipse Plugin](http://docs.google.com/eclipse_plugin.html#eclipse_plugin) chapter.

[Summary](#2bn6wsx)

Here’s the complete build file for our sample:

[Example: Java example - complete build file](#qsh70q)

**build.gradle**

apply plugin: 'java'  
apply plugin: 'eclipse'  
version = '1.0'  
jar {  
 manifest {  
 attributes 'Implementation-Title': 'Gradle Quickstart',  
 'Implementation-Version': version  
 }  
}  
repositories {  
 mavenCentral()  
}  
dependencies {  
 compile group: 'commons-collections', name: 'commons-collections', version: '3.2.2'  
 testCompile group: 'junit', name: 'junit', version: '4.+'  
}  
test {  
 systemProperties 'property': 'value'  
}  
uploadArchives {  
 repositories {  
 flatDir {  
 dirs 'repos'  
 }  
 }  
}

[Multi-project Java build](#17dp8vu)

Now let’s look at a typical multi-project build. Below is the layout for the project:

**Multi-project build - hierarchical layout**

.  
├── api  
├── services  
│   ├── shared  
│   └── webservice  
└── shared

Here we have four projects. Project api produces a JAR file which is shipped to the client to provide them a Java client for your XML webservice. Project webservice is a webapp which returns XML. Project shared contains code used both by api and webservice. Project services/shared has code that depends on the shared project.

[Defining a multi-project build](#3as4poj)

To define a multi-project build, you need to create a *settings file*. The settings file lives in the root directory of the source tree, and specifies which projects to include in the build. It must be called settings.gradle. For this example, we are using a simple hierarchical layout. Here is the corresponding settings file:

[Example: Multi-project build - settings.gradle file](#1pxezwc)

**settings.gradle**

include "shared", "api", "services:webservice", "services:shared"

You can find out more about the settings file in [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds).

[Common configuration](#49x2ik5)

For most multi-project builds, there is some configuration which is common to all projects. In our sample, we will define this common configuration in the root project, using a technique called *configuration injection*. Here, the root project is like a container and the subprojects method iterates over the elements of this container - the projects in this instance - and injects the specified configuration. This way we can easily define the manifest content for all archives, and some common dependencies:

[Example: Multi-project build - common configuration](#2p2csry)

**build.gradle**

subprojects {  
 apply plugin: 'java'  
 apply plugin: 'eclipse-wtp'  
  
 repositories {  
 mavenCentral()  
 }  
  
 dependencies {  
 testCompile 'junit:junit:4.12'  
 }  
  
 version = '1.0'  
  
 jar {  
 manifest.attributes provider: 'gradle'  
 }  
}

Notice that our sample applies the Java plugin to each subproject. This means the tasks and configuration properties we have seen in the previous section are available in each subproject. So, you can compile, test, and JAR all the projects by running gradle build from the root project directory.

Also note that these plugins are only applied within the subprojects section, not at the root level, so the root build will not expect to find Java source files in the root project, only in the subprojects.

[Dependencies between projects](#147n2zr)

You can add dependencies between projects in the same build, so that, for example, the JAR file of one project is used to compile another project. In the api build file we will add a dependency on the shared project. Due to this dependency, Gradle will ensure that project shared always gets built before project api.

[Example: Multi-project build - dependencies between projects](#3o7alnk)

**api/build.gradle**

dependencies {  
 compile project(':shared')  
}

[Creating a distribution](#23ckvvd)

We also add a distribution, that gets shipped to the client:

[Example: Multi-project build - distribution file](#ihv636)

**api/build.gradle**

task dist(type: Zip) {  
 dependsOn spiJar  
 from 'src/dist'  
 into('libs') {  
 from spiJar.archivePath  
 from configurations.runtime  
 }  
}  
  
artifacts {  
 archives dist  
}

[Where to next?](#3rdcrjn)

In this chapter, you have seen how to do some of the things you commonly need to build a Java based project. This chapter is not exhaustive, and there are many other things you can do with Java projects in Gradle. You can find out more about the [Java plugin](http://docs.google.com/java_plugin.html#java_plugin), and you can find more sample Java projects in the samples/java directory in the Gradle distribution.

Otherwise, continue on to [Dependency Management for Java Projects](http://docs.google.com/dependency_management_for_java_projects.html#dependency_management_for_java_projects).

Docs

* [User Manual](http://docs.google.com/userguide/userguide.html)
* [DSL Reference](http://docs.google.com/dsl/)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Javadoc](http://docs.google.com/javadoc/)

News

* [Blog](https://blog.gradle.org/)
* [Newsletter](https://newsletter.gradle.com/)
* [Twitter](https://twitter.com/gradle)

Products

* [Build Scans](https://gradle.com/build-scans)
* [Build Cache](https://gradle.com/build-cache)
* [Enterprise Docs](https://gradle.com/enterprise/resources)

Get Help

* [Forums](https://discuss.gradle.org/c/help-discuss)
* [GitHub](https://github.com/gradle/)
* [Training](https://gradle.org/training/)
* [Services](https://gradle.org/services/)

Subscribe for important Gradle updates and news

Subscribe

By entering your email, you agree to our [Terms](https://gradle.org/terms/) and [Privacy Policy](https://gradle.org/privacy/), including receipt of emails. You can unsubscribe at any time.

© [Gradle Inc.](https://gradle.com) 2018 All rights reserved.

[Careers](https://gradle.com/careers) | [Privacy](https://gradle.org/privacy) | [Terms of Service](https://gradle.org/terms) | [Contact](https://gradle.org/contact/)