# Getting Started with Java Modules

## Overview

In this lab you’ll create, build, and run a simple modularized Java application in IntelliJ. The application will contain just a single module at this stage, so you understand what a module really is.

The word “module” crops up twice when we discuss modules in the context of IntelliJ:

* In IntelliJ, a “module” is effectively just a sub-project.
* In Java 9, a “module” is a Java Platform Module System JPMS unit of software (as we discussed in the PowerPoint chapter).

If you want to create a JPMS module in IntelliJ, there are two steps:

1. Create an IntelliJ module (i.e., an IntelliJ sub-project)
2. Convert it into a proper JPMS module, by adding a module-info.java file

You will perform these two steps in this lab, and see what difference the module-info.java file makes to the way IntelliJ builds and runs the application.

## Step 1: Creating an IntelliJ module

In IntelliJ, create a new module as follows:

* On the *File* menu, click *New | Module*
* In the *New Module* dialog box, select *Java* in the left-hand list, then click *Next*
* In the next dialog box, give the module a name such as student.module.helloworld

The module contains an empty src folder initially. In this folder, create a new package named main (for example), and in this package define a simple Java class named Main (for example) to print a hello-world message on the console.

Build and run the application as it stands. Take a close look in the IntelliJ *Run* panel; notice that IntelliJ runs the application as follows (simplified):

java.exe

-classpath C:\ModernJavaDev\out\production\student.module.helloworld

main.Main

In other words, it uses the traditional -classpath option to tell the JVM where to pick up Java classes. This is how things have always worked in Java before the advent of JPMS modules – all classes were just on the classpath without any additional modularization involved.

## Step 2: Using a JPMS module approach

You will now convert your IntelliJ module into a proper JPMS module. To do this, add a module-info.java file in the *src* folder.IntelliJ creates the file as follows (notice that the JPMS module name is the same as the IntelliJ module name):

module student.module.helloworld {

}

Congratulations, you now have a simple JPMS module!

Build and run the application again. Take a look in the IntelliJ *Run* panel; notice that IntelliJ now runs the application in modular fashion as follows (simplified):

java.exe

-p C:\ModernJavaDev\out\production\student.module.helloworld

-m solution.module.helloworld/main.Main

Note the -p and -m options here:

* -p is shorthand for --module-path. It specifies a semicolon-separated list of directories where modules are located. There’s only one directory mentioned in our case, because our simple application only comprises a single module.
* -m is shorthand for --module. It specifies which particular module/mainclass is the entrypoint to execute.

In other words, just the mere presence of the module-info.java file tells IntelliJ to run the application according to the JPMS modular approach. Going forward, all Java applications will eventually be like this!