Creating Modules with an IDE

In chapter 17 we showed you how to create and use modules from the command line. As we explained there, modules can also be created with an IDE, although the process can be a little fiddly. Here we show you how to create a modular program using Intellij®. Other IDEs can be used in a similar way and instructions can be found online.

In this guide we will show you how to create the modules discussed in chapter 17.

Creating the `calculations` module

Figure 1 shows the structure of the `calculations` module that we developed in chapter 17.

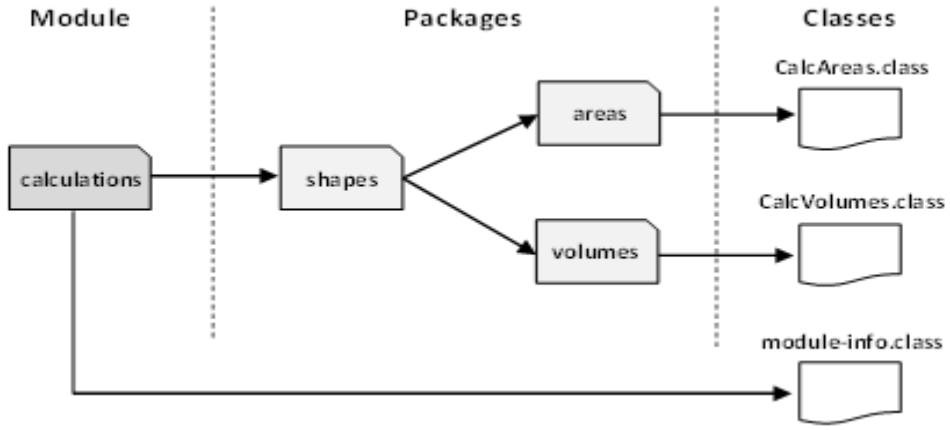

Figure 1

We will now create this module in IntelliJ.

Start as usual by creating a new project. Then click on **File > New > Module** (Figure 2).

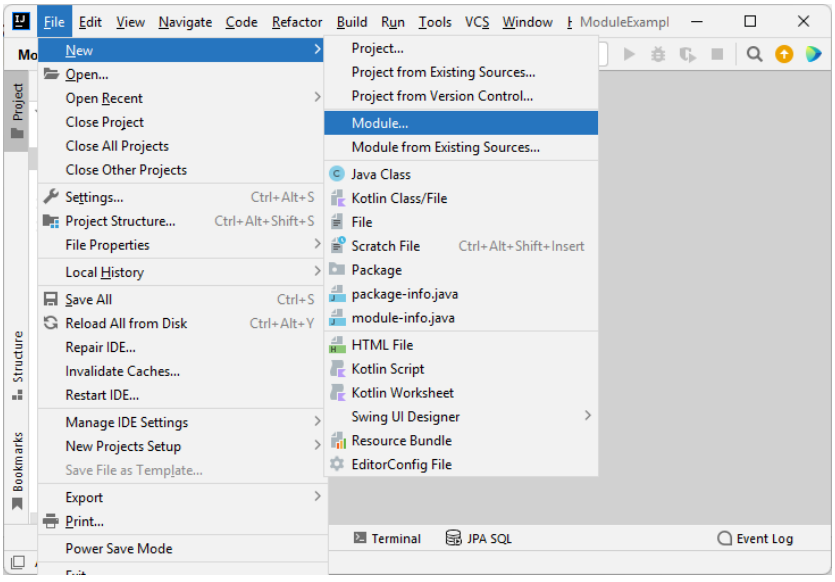

Figure 2

You will see the screen shown in Figure 3.

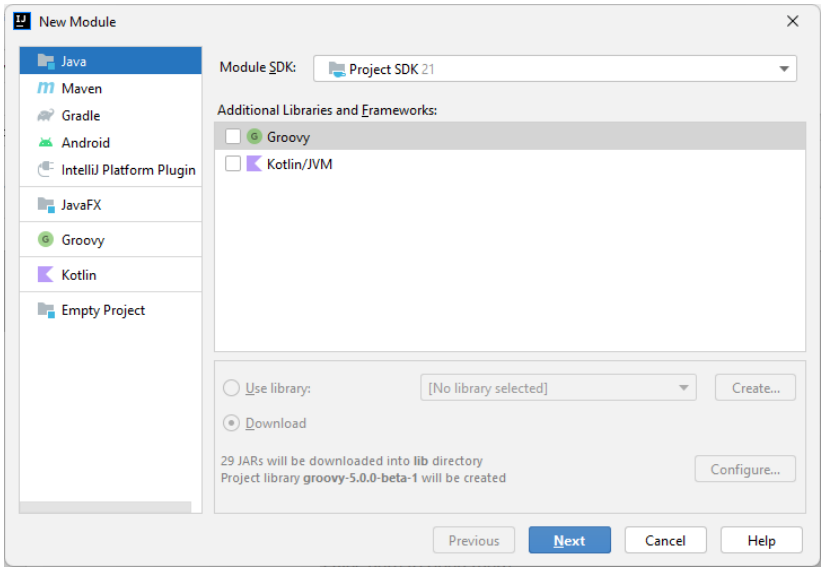

Figure 3

Select the SDK(JDK), making sure that this is the same for the SDK chosen for the project itself, and that the same one is chosen for each module.

You will now be able to choose the name for the module - in this case `calculations` (Figure 4).

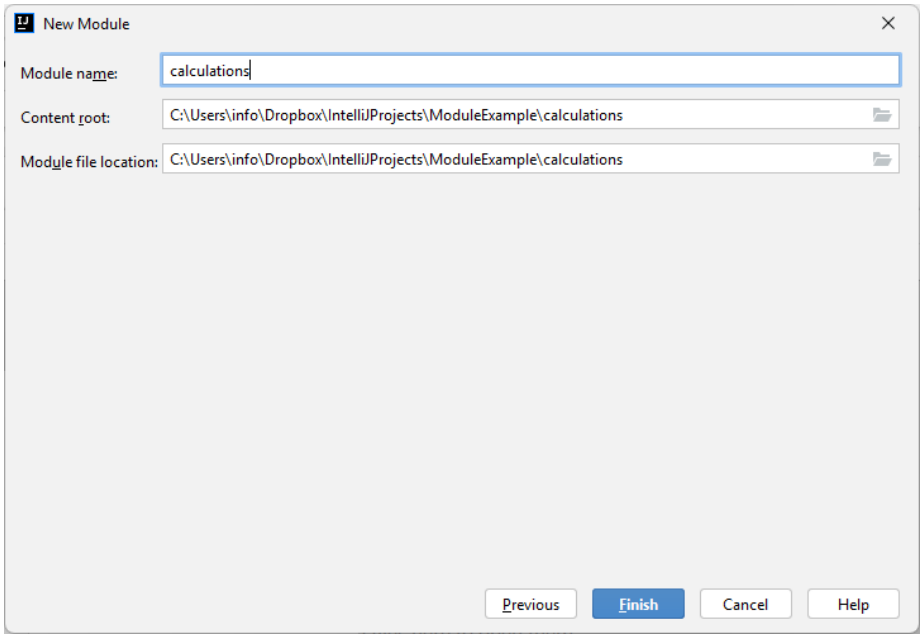

Figure 4

The calculations module will now be listed in the project structure as shown in Figure 5.

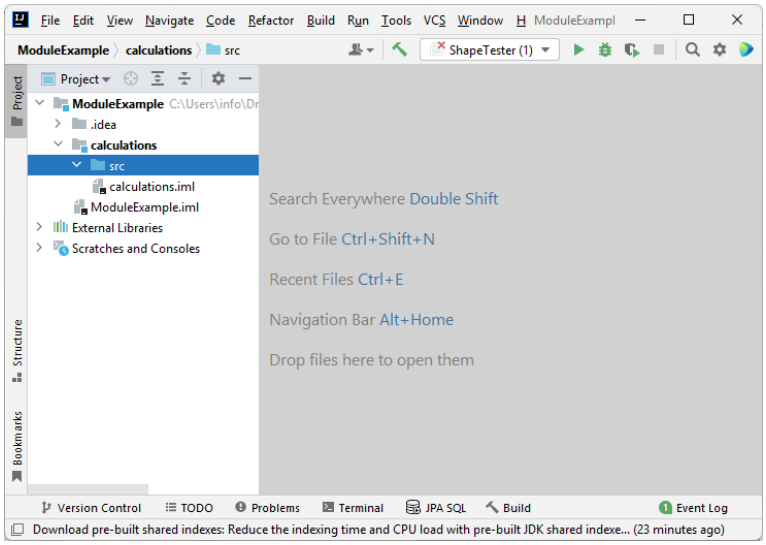

Figure 5

Right-click on the source folder (*src*) - and select **New > Package**. You will be asked to supply a name for your package. As shown in Figure 1, the first package should be named `shapes.areas`. See Figure 6.

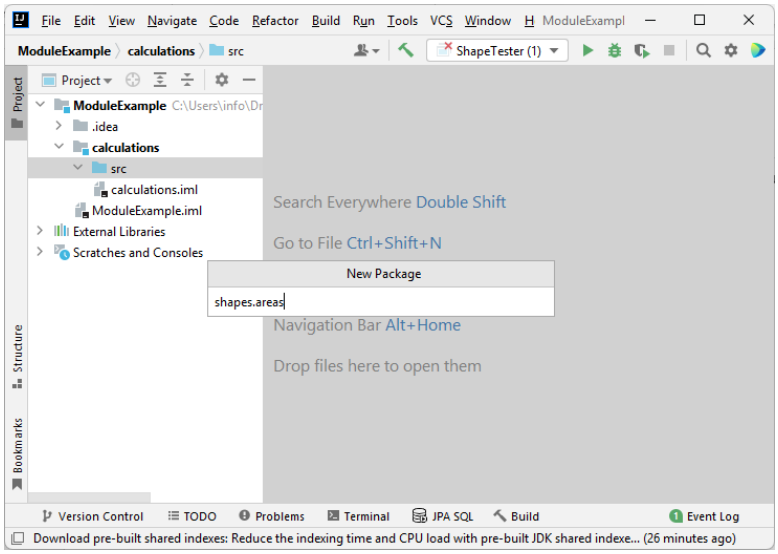

Figure 6

The `shapes.areas` package will be listed in the *src* folder, as shown in Figure 7.

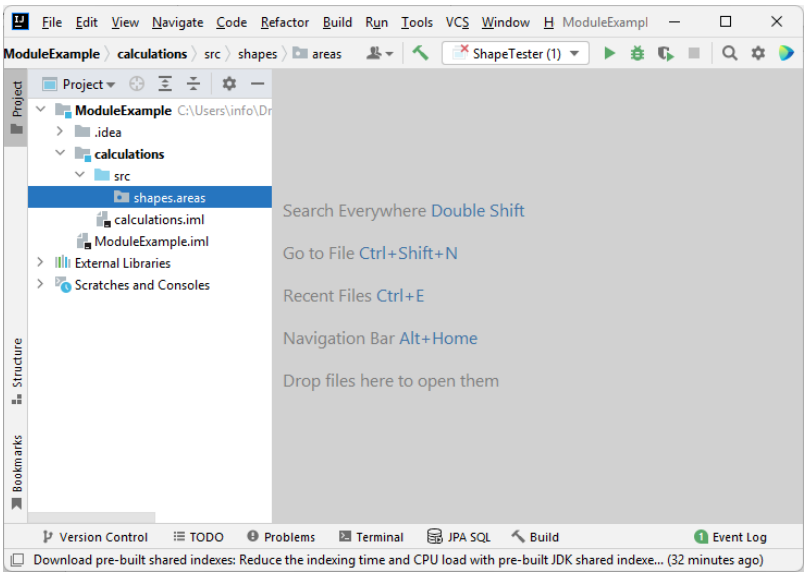

Figure 7

Right-click on *src* again, and add another package, this time calling it `shapes.volumes`.

You will now see both `areas` and `volumes` listed as sub-packages of `shapes` (Figure 8).

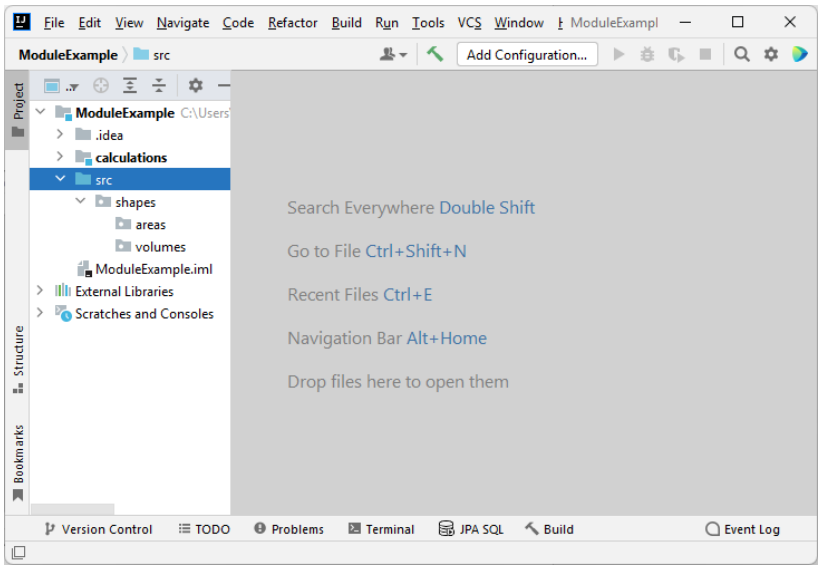

Figure 8

You can now go ahead and add the `CalcAreas` class and `CalcVolumes` class to the `areas` and `volumes` packages respectively (Figure 9)

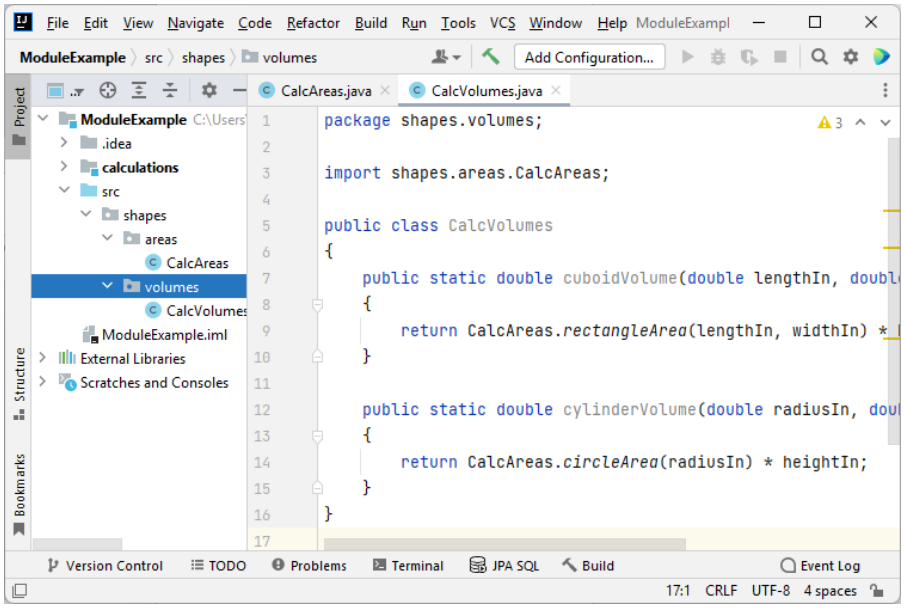

Figure 9

You now need to create the module descriptor. Right-click on `src`, select **new > module-info.java**. Write the code for `module-info` as shown in Figure 10.

You can see that IntelliJ displays this file as `module-info.java(calculations)` to distinguish it from any other module descriptors that may be present in the project.

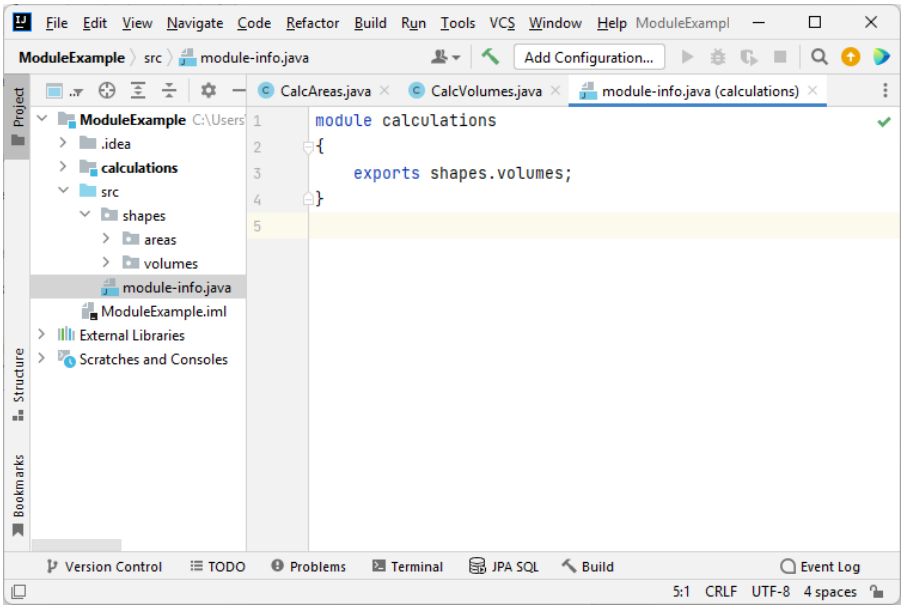

Figure 10

To compile the classes and the `module-info` file we need to choose **Build** from the top menu, and then select - in this case - **Build Module 'calculations'** as shown in Figure 11. Alternatively you can simply choose **Build Project** to build the whole project.

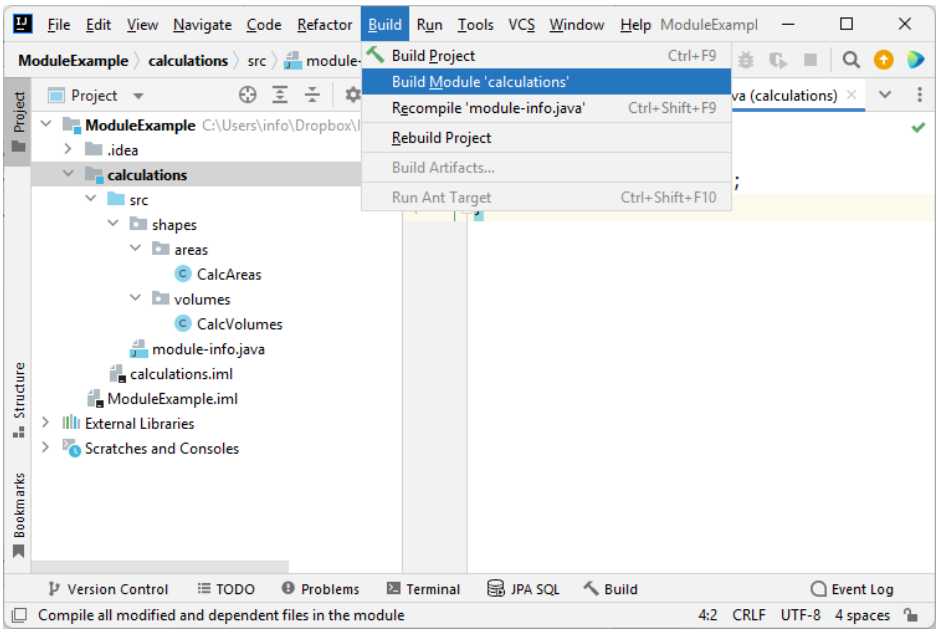

Figure 11

The compiled files now appear, in the correct locations, in the **out > production** folder as shown in Figure 12.

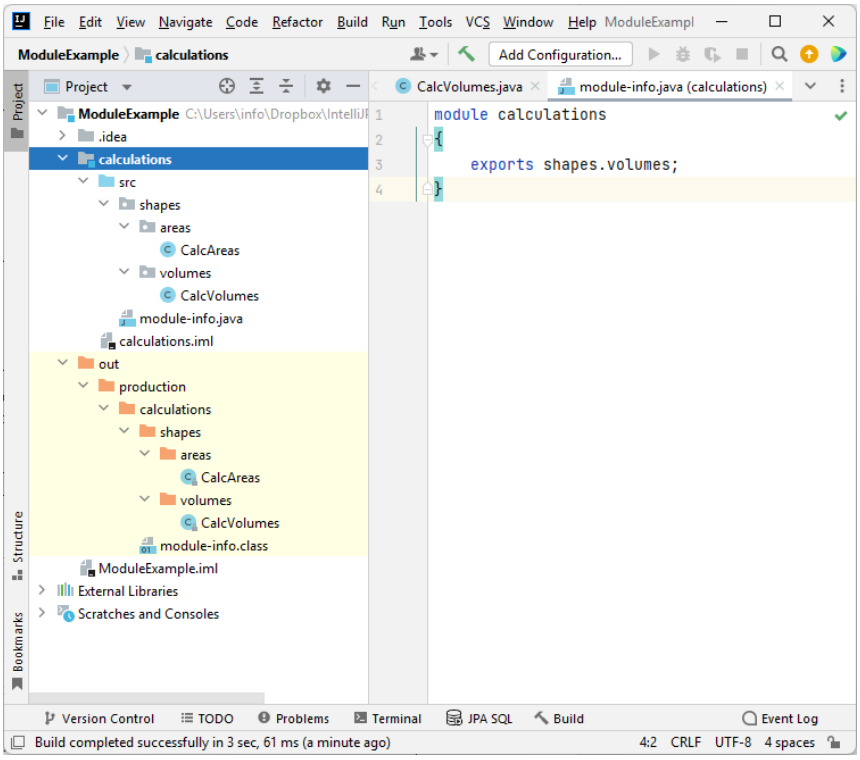

Figure 12

Creating a JAR file from your module

You can now, if you wish, create a JAR file with which to deploy your module.

Select **File > Project Structure > Artifacts**.

Press the **+** sign, then highlight *JAR*, and then *From modules with dependencies…* (See Figure 13)

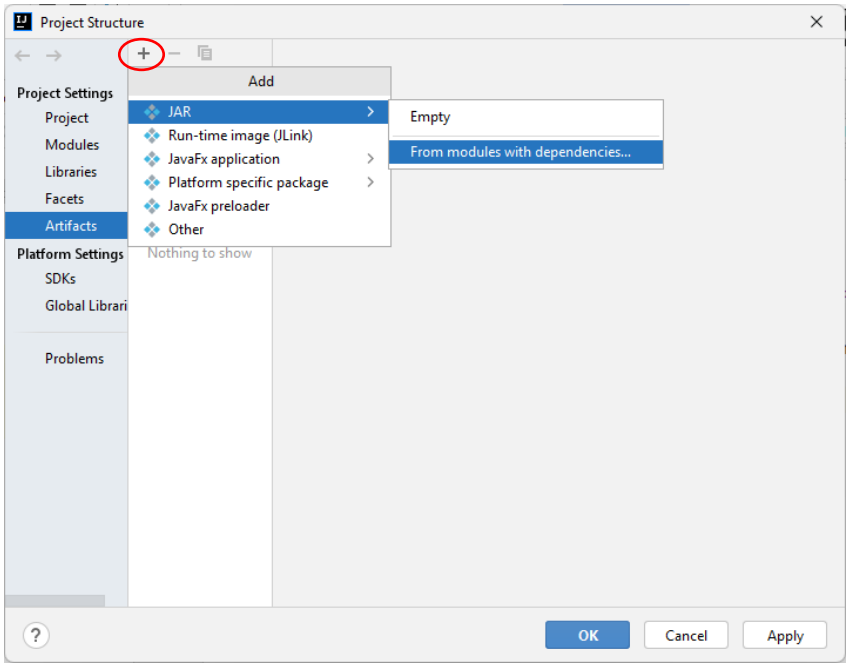

Figure 13

You will be presented with the dialogue shown in Figure 14.

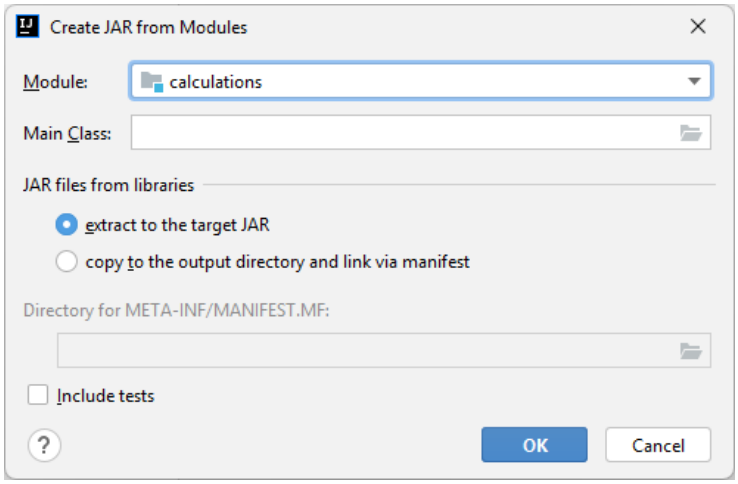

Figure 14

Select `calculations` as the module. You do not need to indicate a main class, as this is not intended to be an executable JAR.

You will now see the screen shown in Figure 15.

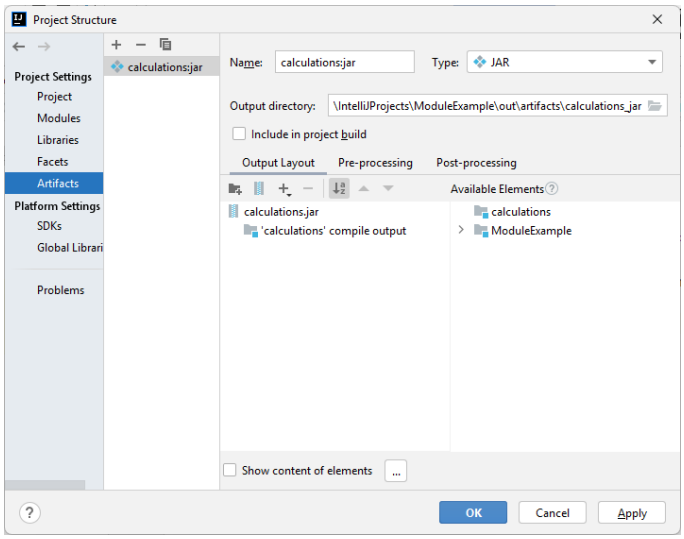

Figure 15

You can now press *OK* to finish.

You now have to build the JAR file. Select **Build > Build Artifacts…** from the top menu as shown in Figure 16.

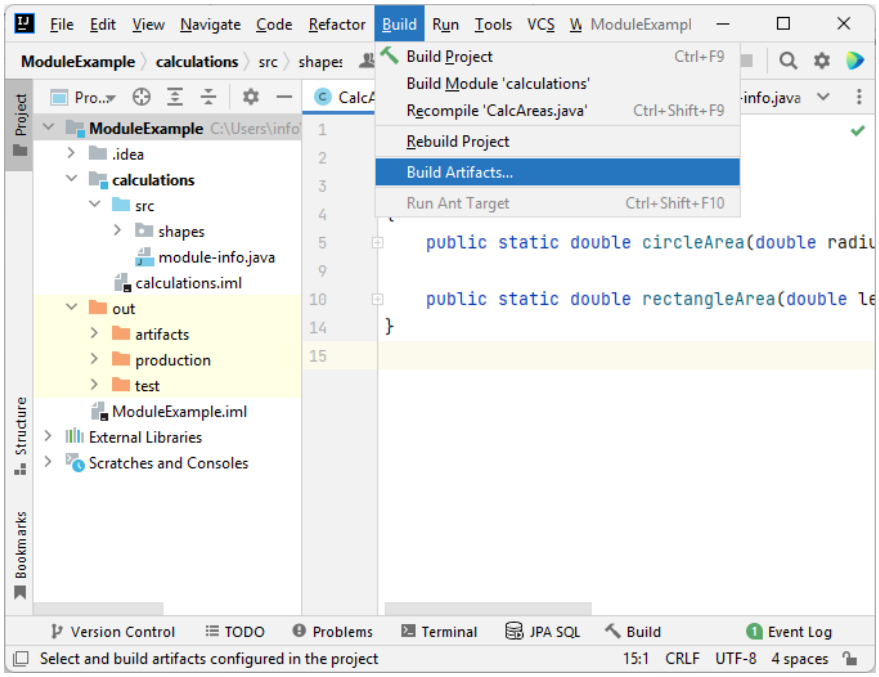

Figure 16

Your JAR file will now be found in a directory called *calculations_jar*, in a folder called *artifacts* which is a sub directory of *out* - see Figure 17.

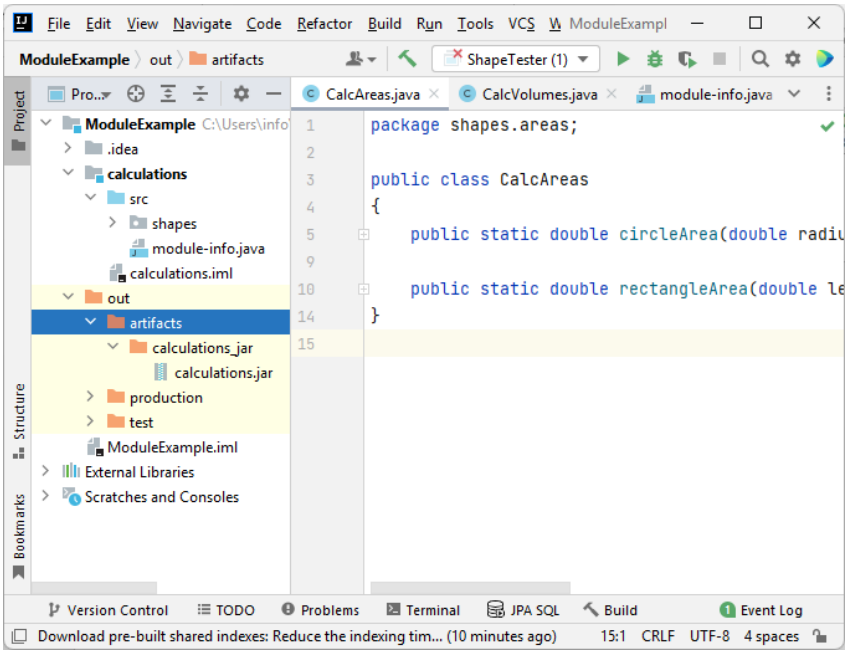

Figure 17

Adding the `moduletester` module

We can now add other modules to our project. In chapter 17 we developed a module called `moduletester` which made use of the `calculations` module. The structure of this module is shown in Figure 18.

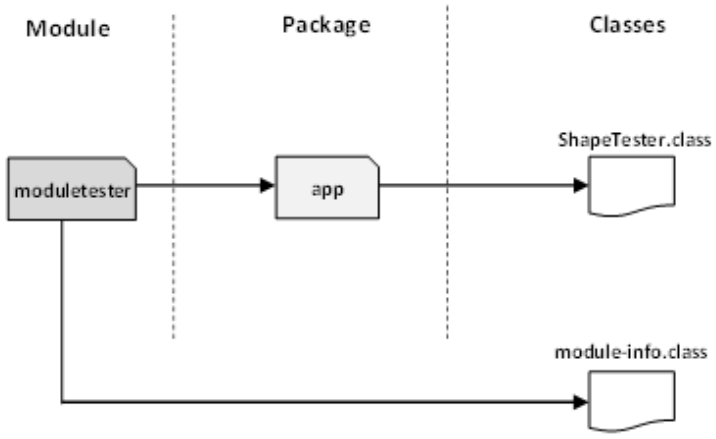

Figure 18

We can proceed in the same way as before - right click on **ModuleExample** and create the new module, `moduletester`. This will then appear in the project structure as shown in Figure 19.

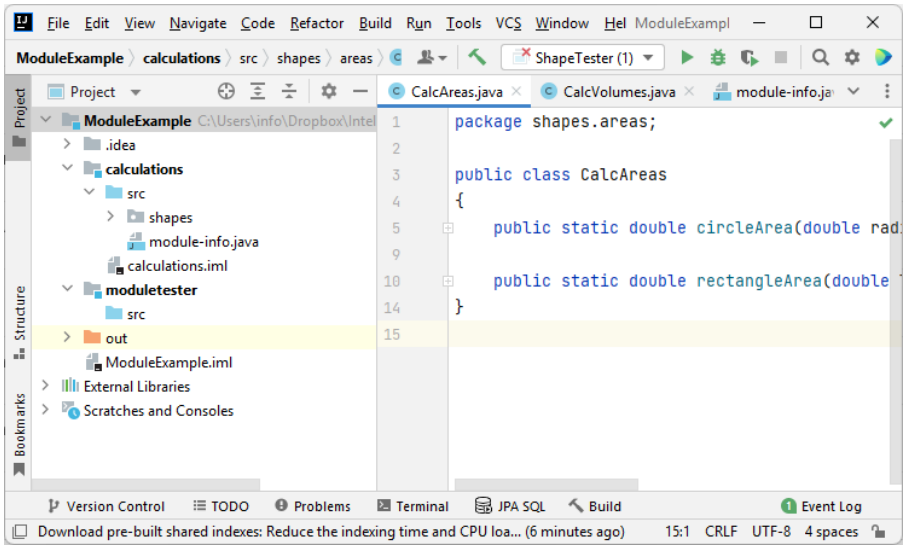

Figure 19

You can now add the `app` package and the `ShapeTester` class in the same way as you did previously with the `calculations` module - however, at this stage you will see that you have some compiler errors, as shown in Figure 20.

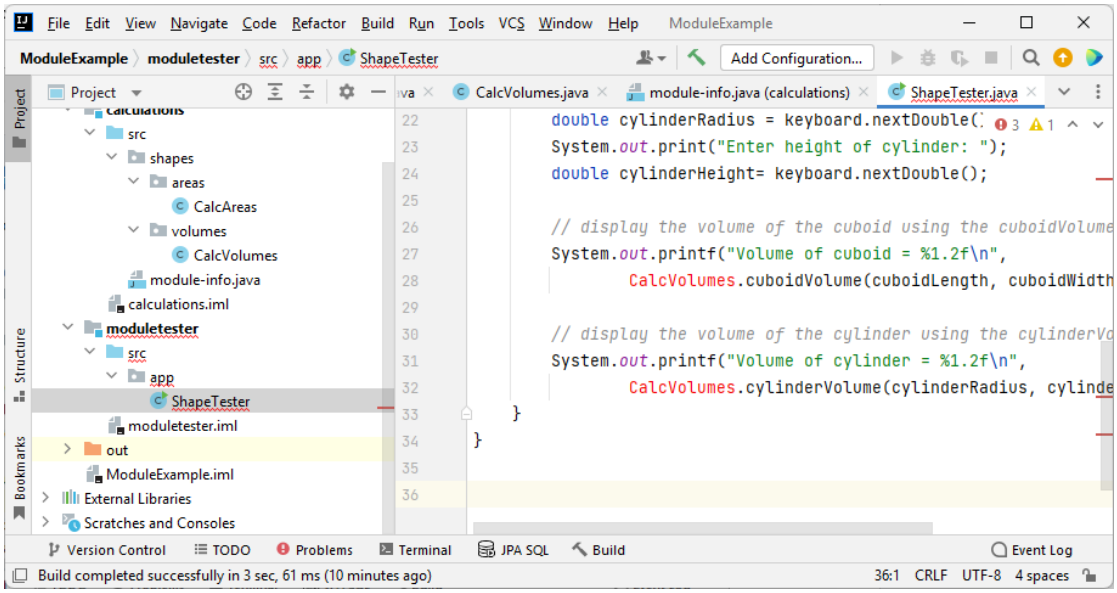

Figure 20

The reason for the compiler errors is that the `moduletester` module requires the presence of `calculations` module, so the latter has to be added to its list of dependencies. To do this proceed as follows:

Select **File > Project Structure > Modules** and highlight the module in question, in this case `moduletester`. Select the *Dependencies* tab, click on the **+** sign as shown in Figure 21, and select *Module Dependency…*

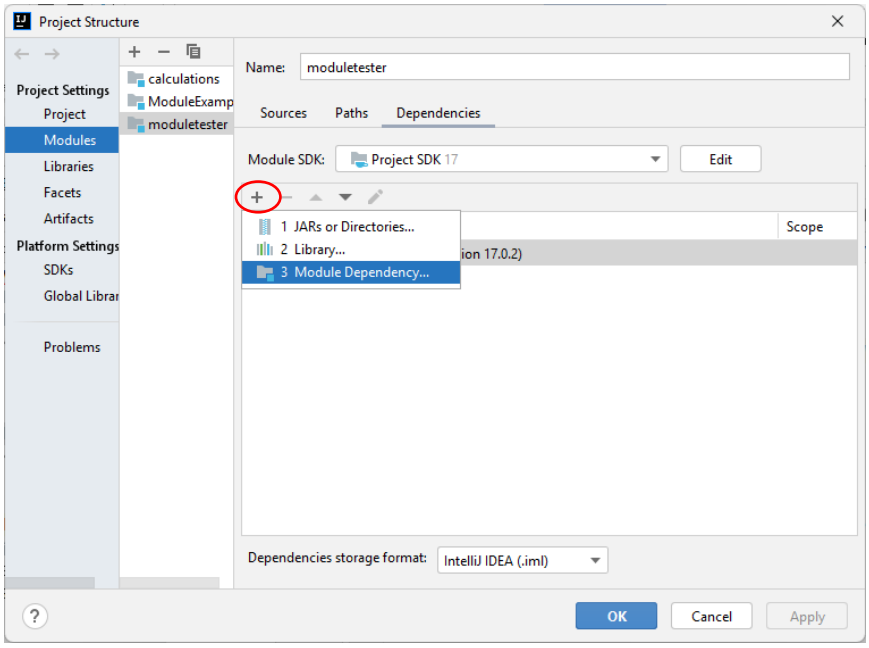

Figure 21

You will then be able to select the module on which your module should depend (`calculations` in this case). See figure 22.

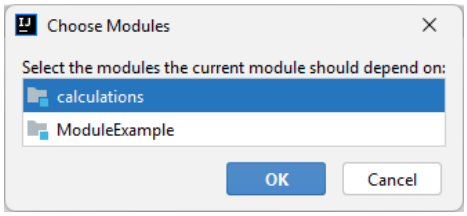

Figure 22

The files will now compile successfully. All that remains is to add the module descriptor. Right-click on source folder (*src*) of `moduletester`, and choose new `module-info.java`. Add the code as shown - the file will be referred to by IntelliJ as `module-info.java (moduletester)` - see Figure 23.

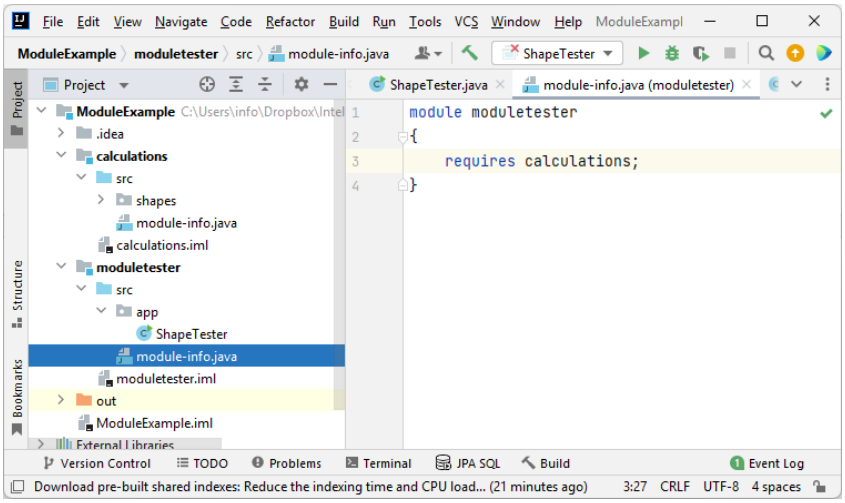

Figure 23

As this module contains an executable class (`ShapeTester`) you can now run this class, and the build will take place automatically as shown in Figure 24.

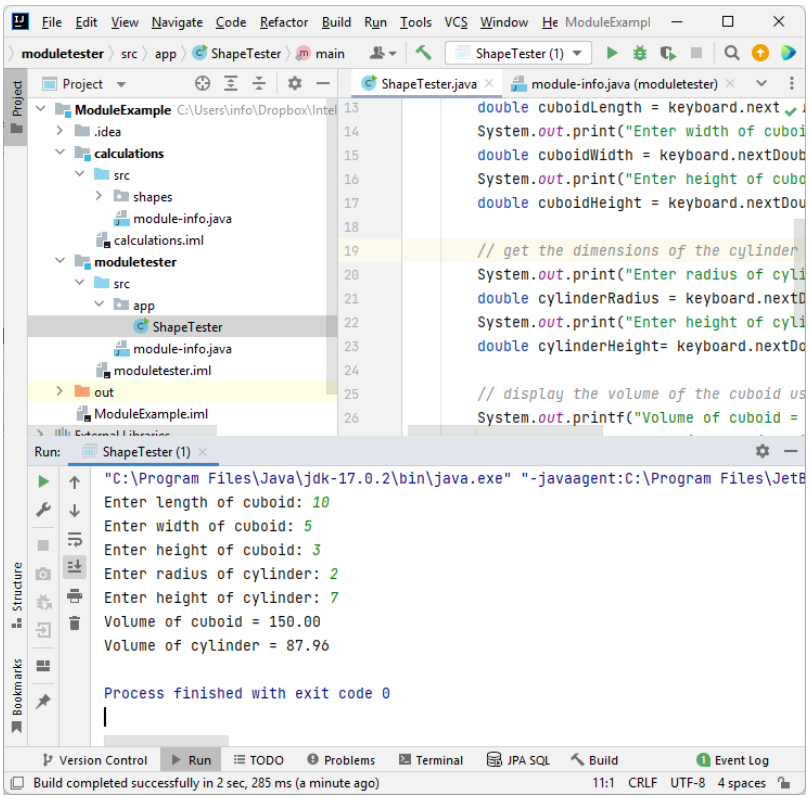

Figure 24

Summary

You've now created two Java modules in IntelliJ, added one as a dependency of the other, written module descriptors, and compiled both. You also created a JAR file for a module, which can now be reused or deployed.