Java 9 introduced the *Java Platform Module* system. The Java libraries of classes that came with Java (the Java Platform APIs, or *Application Programmer* Interfaces) was getting too big, and most applications don’t need them all. Whether one wanted them or not, when creating a Java application for others to use (in a JAR file), it included all those classes. Especially if an application were to be run on a small device —a phone, for example— this could be a problem.

In Java 9 and later versions of Java, one has the ability to split the Java APIs into *modules* and to use only those modules that the application requires. This can make applications a lot smaller. Another advantage: one now has the ability to say which packages inside a module should be visible to other modules and which should not. The module system has other features and benefits, too.

**The default module**

However, for backward compatibility and for ease of use in IDEs like Eclipse, in which one generally doesn’t create applications for others to use, one can use a single *unnamed module*, the *default module*. Just as there is a default package, there is a default module. Therefore, many people using Eclipse or another IDE don’t have to be concerned at all with modules.

In Eclipse, all the classes found in the *classpath* are included in the unnamed module, and everything works as it did in earlier versions of Java.

**Should you create a module-info.java file**

Starting in version 11 of Java, in Eclipse, when creating a new project, a window pops up asking whether a new “module-info.java” file should be created. It is not necessary as long as you are using Eclipse to run programs, and we suggest not creating it. Hit button *Don’t Create*.