SDKs should work. Don't forget setting your JAVA_HOME environment variable on your Windows PC. (If it happens that you don't know how to set an env variable on Windows, please google it).

The next step is to create a Maven/Eclipse project for the MapReduce programs presented throughout this book.

1.2.2 Setting up a Maven/Eclipse Hadoop Development Environment

Note: On Windows, I always create a personal folder like *C:\mspc* to hold my own stuff like *myapp*, *mydev*, etc., in one place. (Incidentally, "*mspc*" means "*my space*.") The instructions given below are based on such an assumption. Also, I assume that the reader is familiar with how to search online and install open source software, how to create environment variables and how to add the "*bin*" folder of a software package to the PATH environment variable of the underlying OS, and so on. If not, please search online to take care of such nuances on your own.

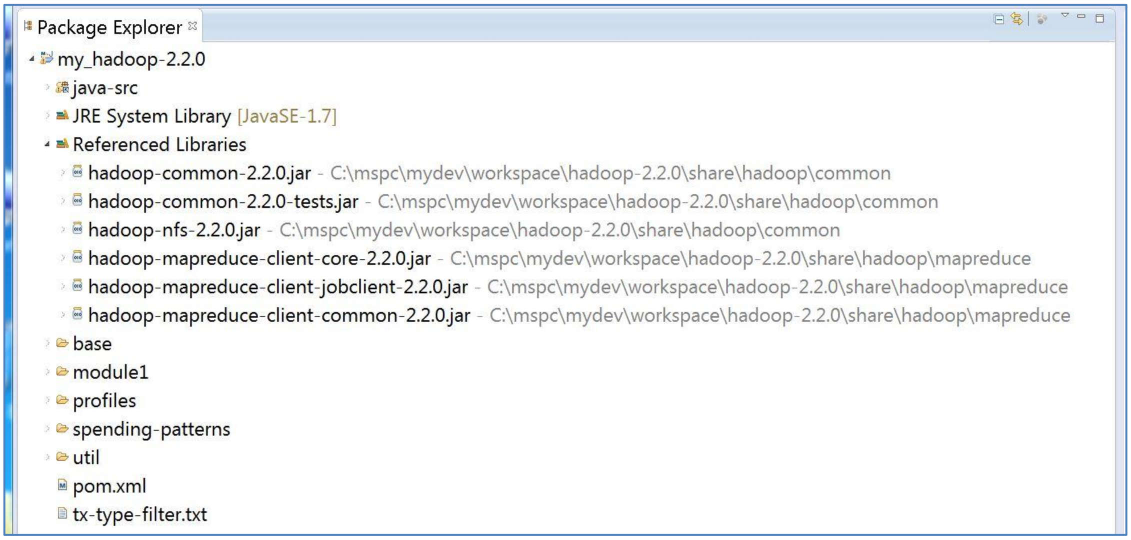
This section describes how to set up a Maven/Eclipse project for our Hadoop 2 samples. Please follow the below steps:

1) **JDK**: If you do not have a JDK installed, install a latest Java 7 JDK, e.g., JDK1.7.0_45 or newer. Set the JAVA_HOME environment variable and add %JAVA_HOME%\bin to your PATH environment variable.
2) **Maven**: If you do not have Maven installed, download and unzip it to a folder on your PC. For example, I installed Maven at `C:\mspc\myapp\apache-maven-3.1.1` on my PC. Also, for convenience, set the M2_HOME environment variable and add %M2_HOME%\bin to your PATH environment variable.
3) **Eclipse**: If you do not have Eclipse Juno for Java Developers installed, download and install it. For example, I installed Eclipse at `C:\mspc\myapp\eclipse_juno` on my PC. If Maven is not enabled on your Eclipse, you need to enable it as described in Appendix B.
4) **Hadoop 2.2.0**: You need to add all Hadoop core jar files to your Hadoop sample project. Refer to Figure 1.3 and download the *hadoop-2.2.0.tar.gz* package onto a Linux machine. Unzip it and copy the *hadoop-2.2.0* folder to the *workspace* folder of your Eclipse.
5) **The my_hadoop-2.2.0 project**: Follow the below steps:
    a. Download the `my_hadoop-2.2.0` project from this book's website at http://www.perfmath.com/download.htm, unzip it and copy the `my_hadoop-2.2.0` folder to the `workspace` folder of your Eclipse.
    b. Create a new Java project with the same name of `my_hadoop-2.2.0` and finish it with all default settings. Now the `my_hadoop-2.2.0` project on your Eclipse should look similar to Figure 1.5. Note that the folder named *java-src* is a linked source that is linked to the sample programs contained in the `module1` module. Since it's a source folder, automatic error checking is enabled. (If you see a folder named `perfmath` under the project, delete it as it's created by mistake by Eclipse – Eclipse has problems in importing linked sources).

6) **Build the sample project with Maven**: Open up an MS-DOS command prompt, change to your project directory and execute the following command and verify that your build process is successful with similar output as shown in Listing 1.1:

```
cmd>mvn clean package –DskipTests –Dhadoop.version=2.2.0
```

You can jump to Section 1.2.3 now.



```
Package Explorer
my_hadoop-2.2.0
  java-src
  JRE System Library [JavaSE-1.7]
  Referenced Libraries
    hadoop-common-2.2.0.jar - C:\mspc\mydev\workspace\hadoop-2.2.0\share\hadoop\common
    hadoop-common-2.2.0-tests.jar - C:\mspc\mydev\workspace\hadoop-2.2.0\share\hadoop\common
    hadoop-nfs-2.2.0.jar - C:\mspc\mydev\workspace\hadoop-2.2.0\share\hadoop\common
    hadoop-mapreduce-client-core-2.2.0.jar - C:\mspc\mydev\workspace\hadoop-2.2.0\share\hadoop\mapreduce
    hadoop-mapreduce-client-jobclient-2.2.0.jar - C:\mspc\mydev\workspace\hadoop-2.2.0\share\hadoop\mapreduce
    hadoop-mapreduce-client-common-2.2.0.jar - C:\mspc\mydev\workspace\hadoop-2.2.0\share\hadoop\mapreduce
  base
  module1
  profiles
  spending-patterns
  util
  pom.xml
  tx-type-filter.txt
```

Figure 1.5 The my_hadoop-2.2.0 project on Eclipse

Listing 1.1 Output of building the my_hadoop_2.2.0 sample project

```
C:\mspc\mydev\workspace\my_hadoop-2.2.0>mvn package -DskipTests -Dhadoop.version=2.2.0
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Base
[INFO] Module1
[INFO] Util
[INFO] Spending Patterns
[INFO] Root
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Base 1.0
[INFO] ------------------------------------------------------------------------
.......
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Root 1.0
```

```
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Base ............................................ SUCCESS [0.479s]
[INFO] Module1 .......................................... SUCCESS [2.801s]
[INFO] Util ............................................ SUCCESS [0.093s]
[INFO] Spending Patterns ................................ SUCCESS [0.673s]
[INFO] Root ............................................ SUCCESS [0.006s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.184s
[INFO] Finished at: Wed Feb 19 18:46:31 PST 2014
[INFO] Final Memory: 23M/316M
[INFO] ------------------------------------------------------------------------
C:\mspc\mydev\workspace\my_hadoop-2.2.0>
```

1.2.3 Maven pom.xml Files for the Hadoop 2 Project

After importing the Hadoop 2 sample project for this book, notice the five modules as shown in Figure 1.5. Each of the modules is described as follows:

- **base**: This module defines all Maven plug-ins and all non-Hadoop dependencies, which are shared among all other modules. See the `pom.xml` file for this module by expanding this module on Eclipse. Out of those Maven plug-ins, the compiler plug-in is for compiling Java sources, the failsafe and surefire are related to running JUnit tests, while the enforcer plug-in is for environmental constraint checking such as Maven and JDK versions. The dependencies include logging and apache commons.
- **profiles**: This module defines all Hadoop dependencies required by all Hadoop programs. These dependencies are specified in terms of *profiles*. A profile defines a group of dependencies that can be activated at build time through an element of `<activation>` inside the `pom.xml` file. See the `pom.xml` file for this module by expanding this module on Eclipse, which contains two profiles of `hadoop-2.2.0` and `hadoop-1.0.3`. By passing in the profile in the form of `–Dhadoop.version=<version-number>` at build time when an `mvn` command is issued, for example, `mvn package –Dhadoop.version=2.2.0`, the designated profile will be used. This is convenient when you want to build a package using different versions of the same framework.
- **util**: This module centralizes all utility programs for all Hadoop programs. As is shown in the `pom.xml` file for this module, its parent is specified with a `<relativePath>` tag pointing to the `profiles`' `pom.xml` file, which is because the `profiles` module is at the same containing directory as the `util` module rather than one level higher. Also note that `util`'s packaging is specified as `jar` since it is for containing Java sources.
- **module1**: We use this module to demonstrate how to develop our first Hadoop MapReduce program, which processes customer credit card spending activities. As is shown in the `pom.xml` file for this module, its parent is also specified with a `<relativePath>` tag pointing to the `profiles`' `pom.xml` file. The packaging format for this module is specified as `jar` as well since it contains Java sources.