Professional Java Projects

CS1331

Professional Java Projects

You know the basics of Java. Today you'll learn the basic elements of professional Java projects, including

- the classpath,
- separating source and compiler output,
- project directory layout,
- packages,
- jar files, and
- using an IDE.

The Classpath

Just as your operating system shell looks in the PATH environment variable for executable files, JDK tools (such as javac and java) look in the CLASSPATH for Java classes. To specify a classpath:

- set an environment variable named CLASSAPTH, or
- specify a classpath on a per-application basis by using the -cp switch. The classpath set with -cp overrides the CLASSPATH environment variable.

Don't use the CLASSPATH environment variable. If it's already set, clear it with (on Windows):

```
C:> set CLASSPATH=
```

```
$ unset CLASSPATH
```

Specifying a Classpath

A classpath specification is a list of places to find .class files and other resources. Two kinds of elements in this list:

- directories in which to find .class files on the filesystem, or
- .jar files that contain archives of directory trees containing .class files and other files (more later).

To compile and run a program with compiler output (.class files) in the current directory and a library Jar file in the lib directory called util.jar, you'd specify the classpath like this:

Notice that you include the entire classpath in the -cp, which includes the current directory (. means "current directory").

Separating Source and Compiler Output

To reduce clutter, you can compile classes to another directory with `-d` option to `javac`

```
$ mkdir classes
$ javac -d classes HelloWorld.java
$ ls classes/
HelloWorld.class
```

Specify classpath for an application with the -cp option to java.

```
$ java -cp ./classes HelloWorld
Hello, world!
```

If you really want to keep your project's root directory clean (and you do), you can put your source code in another directory too, like src.

```
$ mkdir src
$ mv HelloWorld.java src/
$ javac -d ./classes src/HelloWorld.java
```

Project Directory Layout

Source Directories

- `src/main/java` for Java source files
- `src/main/resources` for resources that will go on the classpath, like image files
- `src/test/java` for unit tests. Note that unit tests will be in the same packages as the classes they test.

Output Directories

- `target/classes` for compiled Java .class files and resources copied from src/main/resources

There's more, but this is enough for now. More details on the de-facto standard Java project directory layout can be found at Maven Standard Directory Layout (Note: Maven has fallen out of favor as a build tool, but the directory layout and dependency management system are still used.)

Packages

All professional Java projects organize their code in packages. The standard package naming scheme is to use reverse domain name, followed by project specific packages.

- Since we're in CS1331 at Georgia Tech, we'll use `edu.gatech.cs1331` as a base package name.

A "company" application with one package would contain the following package statement at the top of all source files:

```
package edu.gatech.cs1331.company;
```

Packages and Directory Layout

- Compiler output is organized according to packages
- Convention is to organize source directories by packages

So if we have a package `edu.gatech.cs1331.company`

- Source would go in `src/main/java/edu/gatech/cs1331/company`
- Compiled classes would go in `src/target/classes/edu/gatech/cs1331/company`

You can specify the compiler output (`src/target/clases`) with the `-d` switch to `javac`, or configure your IDE to do that, or use a build tool.

Example Application

Clone the repository at git@gitlab.com:cs1331/company.git

Compiling and Running

Use the command line:

Use the build tool:

Use the IDE:

Jar Files

A jar archive, or jar file, is a Zip-formatted archive of a directory tree. Java uses jar files as a distribution format for libraries.

- To create a JAR file: `jar cf jar-file input-file(s)`
- To view the contents of a JAR file `jar tf jar-file`
- To extract the contents of a JAR file: `jar xf jar-file` or `unzip jar-file`
- To extract specific files from a JAR file: `jar xf jar-file archived-file(s)`
- To run an application packaged as a JAR file (requires the Main-class manifest header): `java -jar app.jar`

See http://docs.oracle.com/javase/tutorial/deployment/jar/index.html for more details.