Pro Java

Christopher Simpkins
chris.simpkins@gatech.edu

Pro Java

You know the basics of Java. Today you'll learn a few baics properties 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=
```

or (on Unix):

```
$ 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:

```
$ ls -R # -R means recursive (show subdirectory listings)
MyProgram.java      AnotherClass.java

./lib:
util.jar
$ javac -cp .:lib/util.jar *.java # : separates classpath elements
$ java -cp .:lib/util.jar MyProgram # would be ; on Windows
```

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

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

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 http://maven.apache.org/guides/introduction/introduction-to-the-standard-

Organizing your Code in 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. So if you're writing a zombie game and you're in the Lab for Interactive AI your application's base package would be specified like this (first line of source files):

```
package edu.gatech.iai.zombie;
```

and it would be located in a directory under your `src/main/java` directory as follows

```
src/main/java/edu/gatech/iai/zombie
```

And if you tell `javac` to put compiler output in `target/classes` then the compiled `.class` file would end up in:

```
target/classes/edu/gatech/iai/zombie
```

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.

Reorganizing Blackjack

Let's apply these organizational practices to an existing application.

- Create a directory somewhere on your hard disk called, say, `blackjack`
- Download a zip file of the Blackjack source to your newly created project directory. If you have `wget` installed on your computer[1] you can go to your project directory on the command line and do:

```
$ wget
    www.cc.gatech.edu/~simpkins/teaching/gatech/cs2340/code/blackjack
```

[1] On a mac with Homebrew you can just do `brew install wget`.

Dealing with Zip/Jar Files

Before we unzip a zip file it's a good idea to see what's in it, which is easy with `jar`:

```
$ jar tf blackjack.zip
Blackjack.java
BlackjackHand.java
BlackjackPlayer.java
Deck.java
HumanPlayer.java
PlayingCard.java
RandomPlayer.java
```

Notice that unarchiving this zip file will not create a subdirectory, so we need to do that ourselves (which we already have – `blackjack/`)

```
$ mkdir blackjack
$ mv blackjack.zip blackjack/
$ cd blackjack/
$ unzip blackjack.zip
Archive:  blackjack.zip
  inflating: Blackjack.java
  inflating: BlackjackHand.java
```

Package Statements

Before we move these soruce files into the project directory structure (which is somewhat annoyingly deeply nested) we can add package statements. First, decide on a name:

- We're part of Georgia Tech, so our package name should begin with `edu.gatech`
- Our "sub-organization" within GT is CS 2340, so we'll add `cs2340` to the package name
- Our application name is Blackjack, so that can be the final piece of our package name.

All of this yields a package name of

```
edu.gatech.cs2340.blackjack
```

Updating Source Files

We need to add a package statement to the top of each of our source files (we'll put everythign in one package).
We can do this with a Unix one-liner:

```
$ for file in `ls *.java`; \
    do printf "package edu.gatech.cs2340.blackjack;\n\n" > $file.new; \
    cat $file >> $file.new; \
    mv $file.new `basename $file .new`; \
  done
```

OK, it's a bit of a stretch to call that a one-liner, and I'm sure a real Unix geek could do it more elegantly, but it works.

```
$ head -n 5 Blackjack.java
package edu.gatech.cs2340.blackjack;

import java.util.Scanner;

public class Blackjack {
```

Creating the Directory Structure

Remember:

- The root of the source directory is `src/main/java`
- The source files go in a directory structure that matches the package name *under* the source root

Use `mkdir -p` to create the whole nested directory structure:

```
$ mkdir -p src/main/java/edu/gatech/cs2340/blackjack
$ mv *.java src/main/java/edu/gatech/cs2340/blackjack/
[chris@nijinsky ~/scratch/blackjack]
$ ls -R
blackjack.zip src

[empty intermediate directories elided]

./src/main/java/edu/gatech/cs2340/blackjack:
Blackjack.java    Deck.java   RandomPlayer.java
BlackjackHand.java  HumanPlayer.java
BlackjackPlayer.java  PlayingCard.java
```

Compiling and Running

We want compiler output to go into `target/classes`, so we must create that directory:

Then we can compile and run from the command line:

```
$ javac -d target/classes/ -cp target/classes/
    src/main/java/edu/gatech/cs2340/blackjack/*.java
[chris@nijinsky ~/scratch/blackjack]
$ java -cp target/classes/ edu.gatech.cs2340.blackjack.Blackjack
What's your name? Chris
...
```

It's useful to know how to do this so you can debug problem with your IDE or build script, but you'll normally set up an IDE and an automated biuld.

Using an IDE

Here are a few basic things you need to configrue when using an IDE:

- Editor settings for non-awful source code
- Source Directory
- Classpath
- Libraries

The best approach to most of this is to generate an IDE project configuration from your build specification, e.g., `build.xml`. Let's see how to do these things with Eclipse.

Closing Thoughts

- There's more "Pro Java" to learn, like Junit and Checkstyle, but these are the basics.
- Speaking of Checkstyle, follow the Java code conventions at http://www.oracle.com/technetwork/java/codeconv-138413.html.
- We'll learn much more about build automation and Ant.