

# Classpaths, packages, jarfiles, and all that gunk.

Ch 4, pp 135--144

[http://java.sun.com/j2se/\(1.4.2|1.5.0\)/docs/  
tooldocs/findingclasses.html](http://java.sun.com/j2se/(1.4.2|1.5.0)/docs/tooldocs/findingclasses.html)

# Administrivia

---

- P1 rollout due today (!)
  - Assessments?
- P2 assigned next Monday
  - Happy full moon
- Midterm exam: Mar 9, in class
  - Interfaces, abstract classes, concrete classes
  - Swing
  - Parsing
  - Misc Java syntax/semantics

# UNM CS Research day

---

- Fri, Mar 4 (next Fri)
- UNM CS research presentations and posters
- Undergrad and grad research work
- You're all invited!
- See <http://cs.unm.edu/~csgsa/conference/> for details and registration

# News blurb o' the day

---

- “Cabir” cell phone worm discovered in US
- Spreads via Bluetooth to phones running Symbian Series 60 OS
- Phone must be set to “discoverable”, not “hidden”
- Prints “Caribe” when phone is activated
  
- So...
- What are the design flaws here?

# The Java class loader

- Every *class* you use must be available to the JVM
- JVM dynamically loads classes on demand
- First reference of a class in a program (direct or indirect)
  - `new`, deserialization, etc.
- When explicitly requested via  
`java.lang.ClassLoader.loadClass()`
- But how does it find the `.class` file?
  - Where does it look for the file?
  - What steps take place?

# Watching the class loader

---

- `java -verbose:` prints out all class loads
- Worth trying yourself -- get some of the “behind the scenes view”

# CLASSPATH

---

- Java uses a “search path” named `CLASSPATH`
- Only looks for files on the `CLASSPATH`
- Looks in order of elements in `CLASSPATH`
  - Directories
  - Jar files
  - Zip files

# Contents of CLASSPATH

- CLASSPATH is a colon-separated list of elements:
  - CLASSPATH=\$HOME/lib/java/java-getopt-1.0.9.jar:\$HOME/lib/java/libtdrl.jar:\$HOME/private/src/projects/p1/:
  - \$HOME/lib/java/java-getopt-1.0.9.jar
  - \$HOME/lib/java/libtdrl.jar
  - \$HOME/private/src/projects/p1/
  - .
- Can be given as command-line option to javac/java (-classpath) or via environment var (\$CLASSPATH)

# Caveat

- CLASSPATH also contains some “hidden” elements
  - Java “bootstrap” classes (`rt.jar`, etc.)
    - Includes all “standard” Java stuff -- `java.lang`, `java.util`, `javax.swing`, etc.
  - “Extension” classes
    - Contains system-specific add-ons

# Coolness of CLASSPATH

- System-specific (or user-specific)
  - Easy to customize which classes are available
  - Every user/site can have diff classes/packages available
- Code designer doesn't have to know anything about the arrangement of the target system
  - Doesn't even have to know that it has directories!
  - Can load class files over the net, from memory stick on a PDA, etc.
- Can play clever tricks w/ class loading

# Pain of CLASSPATH

- Can have subtle (or not subtle) bugs when CLASSPATH is set up wrong
  - Can't find something that "is right there!"
  - Can get wrong (out of date) class
- User can sabotage your program
  - Can deliberately substitute a class for yours
  - Can subvert your security mechanisms
- Java provides a `SecurityManager` API to help you control such subversions
  - Getting it right not trivial

# Seems simple enough...

- Easy, right? Just make sure that all your `.class` files appear on the `CLASSPATH`, and life is good
- But...
- What does it mean for a jar file to be on the classpath?
- What about packages?

# Jar files

- Simple version:
  - Jar file is just a Zip archive containing a bunch of `.class` files
  - Class loader treats *entire jar file* as a directory
  - If `CLASSPATH=foo.jar:bar.jar:.`
  - `java HelloWorld` will look for
    - `HelloWorld.class` in `foo.jar`
    - `HelloWorld.class` in `bar.jar`
    - `HelloWorld.class` in current directory (`.`)

# Building jar files

- Very convenient for shipping around large amounts of code/class files
- Create jar file w/

```
jar cf [jarfilename] [files/dirs to put in jar]
```

# Jar files, reprise

- More complicated version:
  - Jar files can have a “default main class”
  - Can “run” an entire jar file w/
    - `java -jar [jarfilename]`
- *But:* jar file must contain “manifest” file that specifies default main class
- If you don’t specify a manifest, jar creates an empty one for you
- Create a jar file w/ manifest:
  - `jar cfm [jarname] [manifestfile] [targetfiles]`

# Packages

- Nastier problem: packages
- Every Java class exists in a “package”
- Essentially a namespace
- Allows multiple developers to have classes w/ same base name
  - `java.util.Date` **diff from** `java.sql.Date`
- “True” name of a class is  
[packagename].[classname]
- Java class loader expects class file for  
`java.util.Date` **to be**
- `[CLASSPATH elt]/java/util/Date.class`