Development Tools

IDEs

- Integrated development environments (IDEs), e.g. BlueJ, Dr.Java, VisualStudio, …
  - help programmers focus on programming
  - by hiding details of underlying tools
- But
  - important to know differences between e.g. compile-time & run-time errors
  - important to know what details are being managed, e.g. make dependencies
  - want to gain better control sometimes
  - want to support additional tools

Manual development tools

- Alternatively, can make programmers know about and use all the tools that were packaged up in the IDE
  - more knowledge, understanding
  - more power (e.g. adding new tools)

  - more work on programmer's part

Structure of an IDE

Unix tool suite

Main Java development tools

- Your favorite text editor
- `javac` *file*`.java`…
  - compile one or more `.java` source files into corresponding `.class` compiled files
- `java` *Class arg*…
  - run compiled Java program
  - start in class *Class* with method `public static void main(String[] `*`args`*`)`
    - typically, there's a *Class*`.class` compiled file
    - *args* array initialized with *arg*… from command line
- http://java.sun.com/j2se/1.4.2/docs/
  - "API & Language Documentation"
  - "SDK Tools Documentation"

Handling references to other classes

- One Java class can refer to many other Java classes
  - When compiling the first class, how does `javac` find the other classes, e.g. to check their types?
  - When running the main class, how does `java` find the other classes that the program references?
- Can give them as extra `javac` arguments
  - What about standard Java library classes?
  - Don't want to have to recompile every time
- Can specify a *classpath* argument to `javac`

The classpath

- `javac –classpath` *dirs file*`.java`…
- `java –classpath` *dirs Class arg*…
  - Specifies a series of directories in which to search for precompiled classes
- *dirs* has the form *path1*`:`*path2*`:`…`:`*pathN*
  - on Cygwin, use "`;`" instead of "`:`" and "`\\`" instead of "`/`"
- (A class named *Foo* is compiled into a file named *Foo*`.class`)

CLASSPATH

- Instead of specifying `–classpath` to every `javac` and `java` command, can set the `CLASSPATH` environment variable instead
  - `setenv CLASSPATH \`
    `$HOME/myClasses:$HOME/yourClasses`
- Do this in your `.cshrc` to "configure" your Java compilation and execution environment

Packages

- Java organizes classes into packages
  - E.g., `java.lang`, `myApp.UI.windows`
- Each Java source file declares its package
  - E.g., "`package myApp.UI.windows;` …"
- Packages correspond to directory hierarchies
  - E.g. the `myApp/UI/windows` directory contains the above `.java` source file
  - `myApp` should be found inside some directory in `CLASSPATH`

Archives

- Often want to put a collection of files together into a single file
  - `tar` is the standard Unix command to do this for regular files
- Collections of compiled files are libraries
  - `ar` is the command that builds `.a` library files from `.o` compiled source files

Java archives/libraries

- `jar` is the command for building Java `.jar` archives
  - can contain `.class` files, `.java` files, and anything else
- E.g.:
  `jar cvf myStuff.jar *.{java,class}`
  `jar cvf myApp.jar myApp` *(myApp is a dir)*
- Can put a `.jar` file in the classpath
  - Will search the `.jar` file's contents for matches
- (Can make "executable jar files" on Windows)

Standard libraries

- Every language has a set of standard things that every program should be able to access
  - Often called standard libraries
- In Java, there's a `.jar` file that contains all the `.class` files for the `java` package
  - Implicitly added to the classpath

Debugging

- `jdb`
  - Starts up a Java debugger
  - Works best if used "`javac -g` …" before
- Inside can run a program, set breakpoints, single-step through execution, and print out program state
  - If run under emacs, then emacs will show corresponding source lines where you are
  - Java's multiple threads makes this complicated

Debugger commands

- `run` *Class arg*…
  - run class *Class*'s `main` method, on args
  - good to set breakpoints first, if want to stop somewhere
- `stop in` *Class*`.`*method*
- `stop at` *Class*`:`*lineNumber*
  - set a break point at the start of a method or at a particular line in a source file
- `catch` *Exn*
  (e.g. `java.lang.NullPointerException`)
  - stop if an instance of *Exn* is thrown but not caught

More debugger commands

- `cont`
  - continue from a breakpoint
- `next`
  - continue to the next line in the current method
- `step`
  - continue to the next line, possibly in the callee or caller method

More debugger commands

- `where`
  - print out the current stack
- `print` *expr*
- `dump` *expr*
  - print out (short or long) description of result of evaluating *expr*
    - *expr* often a simple variable name, but can be as complex as a method call, too