Applets and Applications

- Application run by user, double-clickable/command-line
  - No restrictions on access, reads files, URLS, ...
  - GUI applications typically include a JFrame
    - Has title, menus, closeable, resizeable

- Applet is downloaded via the web
  - Runs in browser, not trusted (but see policy later)
  - Can't read files on local machine (but see policy)
  - Can't be resized within browser
  - Uses jar file to get all classes at once
    - Alternative? Establish several connections to server

Developing Applets and Applications

- Create a JPanel with the guts of the GUI/logic
  - What will be in the content pane of both deployments
  - Makes GUI very simple, see code examples
  - Use JPanel in both Applet and Application

- Test with application first, easier to read files/resources
  - Migrate to Applet, test first with appletviewer
  - Migrate to web, may need to clear cache/reload

- Ideally first cleanly into OOGA architecture
  - Gui isn't the view, what about interfaces?

Packages, JAR files, deployment

http://java.sun.com/docs/books/tutorial/jar/basics/index.html

- **Java packages correspond semantically to modules (related classes) and syntactically to a directory structure**
  - **Class names correspond to file names**
  - **Package names correspond to directories**
  - **Related classes belong together, easier to develop, easier to deploy**
  - **Leverage default/package access, use properties of protected which is subclass *and package* access**

Packages, javac, java, javadoc

- In moderately big programs packages are essential
  - Can’t easily live in a directory with 50 .java files
  - Can’t easily co-exist in such a directory
  - Harder to use tools like Make and Ant

- Each of javac, java, javadoc is slightly different with packages, all must co-exist with CLASSPATH
  - File system vs. compiler vs. runtime
  - Source of confusion and problems
  - IDEs can manage Make/CLASSPATH issues

CLASSPATH and related concepts

- The default CLASSPATH is . current directory
  - Works fine with default/unnamed packages
  - Will not work with named packages

- Set CLASSPATH to directory in which packages live also include current dir
  - `setenv CLASSPATH "~ola:."`
  - `` setenv CLASSPATH "`pwd`:." ``
  - On windows machines change registry variable, separator is semi-colon rather than colon

- All problems are CLASSPATH problems

More package details

- To compile
  - Can cd into directory and type javac *.java
  - Can also type javac ooga/*.java from one level up
  - If CLASSPATH isn't set, the second won't work
- To run
  - `java ooga.TicTac` will work, you must specify the "real" name of the class being used.
  - Reading files requires full-paths or run from directory in which file lives
- To document
  - http://java.sun.com/j2se/javadoc/faq.html
  - Don't need to use –sourcepath, but can
  - `javadoc -d doc ooga ooga.timer ooga.game` …

javadoc for packages

- See the javadoc faq
  http://java.sun.com/j2se/javadoc/faq.html
  - For each package create a package.html file
    - Not in /** */ javadoc format, strictly html
    - First *sentence* after <body> is main description; a sentence ends with a period.
    - The package.html file should provide complete instructions on how to use the package. All programmer documentation should be accessible or part of this file, e.g., in the file or linked to the file
  - Use the `{@link foo.bar bar}` tag appropriately.
    - See the FAQ, or the source for the elan package online

- You may want to keep .java and .class files separate, see sourcepath and classpath as commandline args to java

From JITs to Deoptimization

- JITs compile bytecodes when first executed
  - If we can cache translated code we can avoid re-translating the same bytecode sequence
  - Spend time compiling things that aren't frequently executed (optimistic optimization?)
  - Errors indicate "compiled code" rather than line number
- Sun's HotSpot VM uses a different strategy for performance
  - Adaptive compilation: save time over JIT, compile "hotspots" rather than everything, uses less memory, starts program faster, http://java.sun.com/products/hotspot/
  - No method inlining, but uses *dynamic deoptimization*
    - Program loads new subclass, compiled code invalid, so ...?
- What does the class loader do?

Loading .class files

- **The bytecode verifier "proves theorems" about the bytecodes being loaded into the JVM**
  - **These bytecodes may come from a non-Java source, e.g., compile Ada into bytecodes (why?)**

- **This verification is a *static* analysis of properties such as:**
  - **.class file format (including magic number 0xCAFEBABE)**
  - **Methods/instances used properly, parameters correct**
  - **Stack doesn't underflow/overflow**

- **Verification is done by the JVM, not changeable**
  - **Contrast ClassLoader, which is changeable, can modify classes before they're loaded into the JVM**

http://securingjava.com

http://java.sun.com/sfaq/verifier.html

The ClassLoader

- The "boot strap" loader is built-in to the JVM
  - Sometimes called the "default" loader, but it's not extensible or customizable the way other loaders are
  - Loads classes from the platform on which the JVM runs (what are loader and JVM written in?)

- Applet class loader, RMI class loader, user loaders
  - Load .class files from URLs, from other areas of platform on which JVM runs
  - A class knows how it was loaded and new instances will use the same loader

- Why implement a custom loader?
  - Work at Duke with JOIE

Applications and Applets

- An applet is a program delivered via the web
  - security issues, sandbox model
  - where does code/images/etc come from? How is it delivered?
  - what browsers support JDK1.2 out-of-the box?
  - Use IE/Netscape with plugin, use Opera as is, use appletviewer for debugging, testing

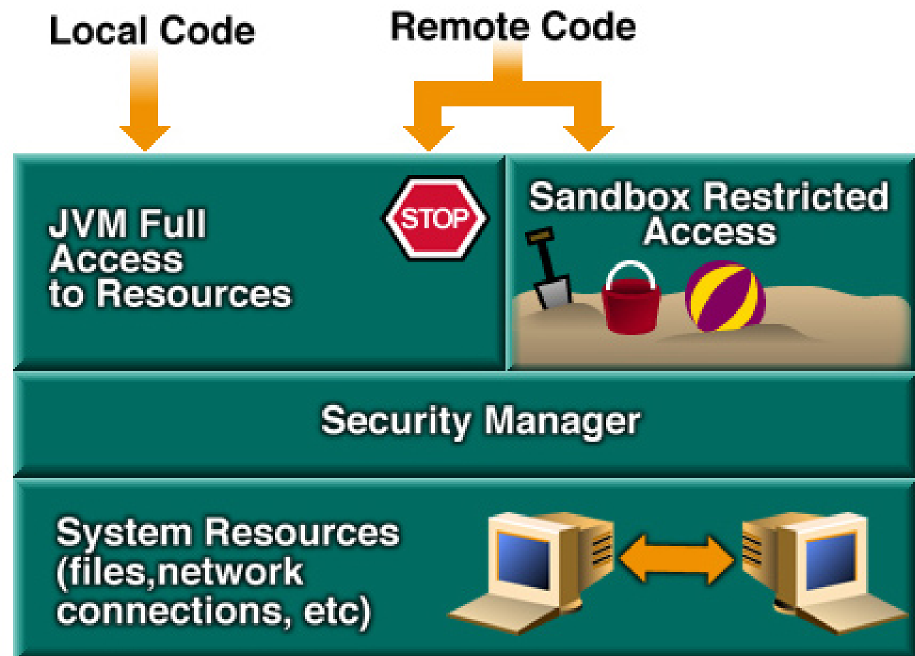
- Possible to wrap up lots of classes in a .jar file
  - java archive, similar to a tar file, possible to include .class files, .au, .gif, etc, so all code transferred at once

Running an Applet

- An applet has an init() method
  - similar to constructor, called only once, when the applet is first loaded
- An applet has a start() method
  - called each time the applet becomes “active”, run the first time, or revisited e.g., via the back button in a browser
- An applet has a stop() method
  - called when applet is invisible, e.g., user scrolls or goes to another web page
- other methods in an applet
  - destroy, getAppletInfo, getParameterInfo
- Applet subclasses Panel, so it is an Container/Component

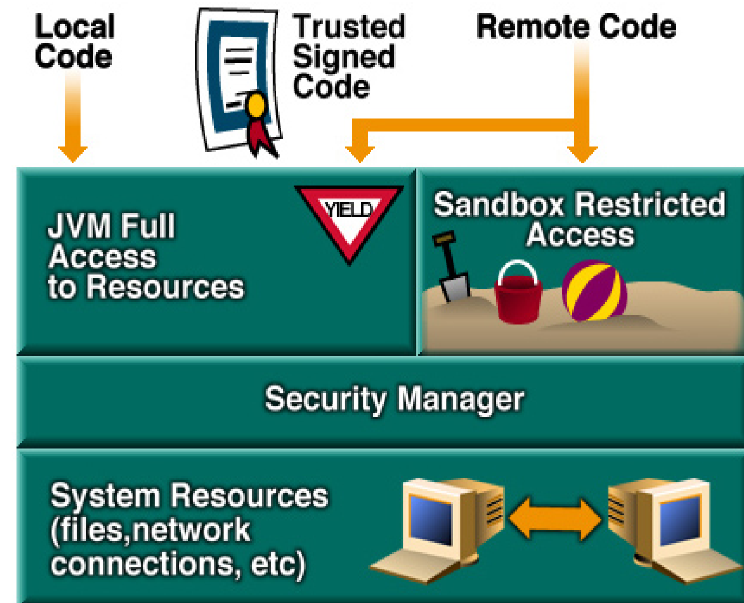
Security Manager

- Applets use a SecurityManager
  - Query for permissions
  - Supported by browsers by convention (would you use an “untrusted” browser)
- The picture shows JDK 1.0 model, “sandbox” restrictions supported by SecurityManager
  - Untrusted code restricted to the sandbox
  - All downloaded/applets are untrusted
  - Severely limits what a downloaded program can do



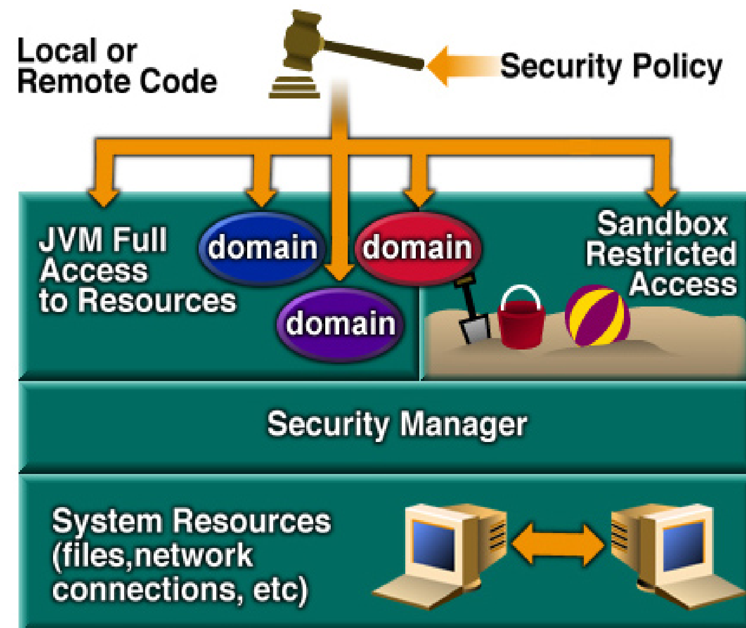
SecurityManager changes in JDK 1.1

- Applets support signing using digital signatures
  - Signature stored with code in JAR file that's downloaded
  - Clients support open/full access to "trusted" applets, some signatures ok
- Still "all-or-nothing", an applet is untrusted or completely trusted
  - What might be preferable?



SecurityManager changes in JDK 1.2

- Policies are now supported
  - Allow more fine-grained control of access, permission
  - Based on location (URL) and/or digital signatures
  - Uses public/private key, applets don't need to be signed, can be from a trusted location
- Set policies on a system wide basis using `policytool`
  - What about user-level permissions?



Applet codebase

- JVM executing in browser has different capabilities than “regular” JVM
  - Looks in codebase as its CLASSPATH, also uses client/browser side CLASSPATH
  - Codebase is relative to location of web page originating the applet for security reasons
  - Implications for downloading foo.jar?