JAR Files and JAVA Extensions

How To Make
Your JAVA Software
Feel Professional

The perception of (dis)order…

- When we create a JAVA application, it is typically implemented as a set of JAVA objects with some global POE
  - Each object has it's own JAVA file…
  - … which compiles into it's own CLASS file
- Most users are used to applications that are a single file or button click away

Enter the JAR file

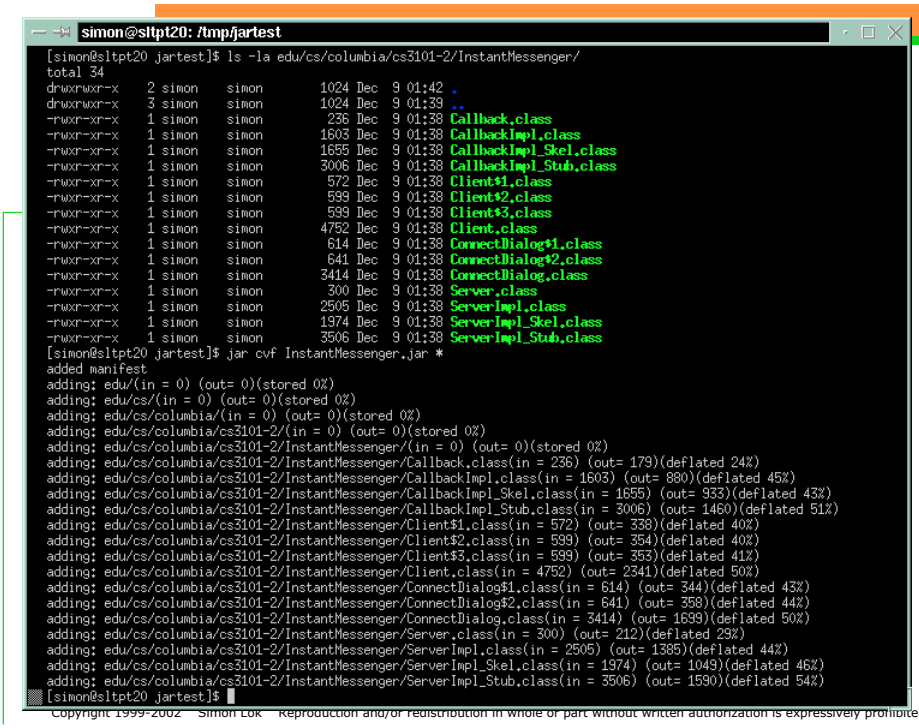
- JAR files are archive files, like ZIP files
  - They can contain files and folders
  - They are automatically “compressed”
- There are also some additional features
  - JAR files can be digitally signed to verify the origin of the code in the file
  - JAR files can be “executed” as if they were a full fledged application

The First Step…

- First thing you have to do is make sure that you have got your Package right
- Remember to invert your domain name to make globally unique package names
- package edu.columbia.cs.cgui.mars
- This will lead to a mess of directories that… luckily RAD tools like Forte deals with this auto-magically

Creating a JAR

- The jar tool works just like UNIX tar
  - jar cvf MyApplication.jar file1 file2 …
- Typically you would change directory to the root of your development
  - jar cvf MyApplication.jar *
- The –C option allows more flexibility
  - jar cvf MyApplication.jar –C /devel/root *
- Be careful of source code inclusion!



Viewing/Extracting a JAR

- JAR file contents can be viewed by:
  - jar tvf MyApplication.jar
- The “t” stands for Table of Contents
- A JAR can also be extracted by using the “x” command
  - jar xvf MyApplication.jar
  - This automatically overwrites existing files!

The Manifest

- If you create a JAR and view it, you will see a file called MANIFEST.MF in the table of contents
- This file is automatically created by the JAR tool and contains special information for the JAR
- To alter the manifest:
  - jar cmvf manifest-addition MyApp.jar *

Manifest Manipulations

- Set the class with the point of entry (main) for this application:
  - Main-Class: PrimaryPointOfEntry
  - You can execute a JAR file by using the command “java –jar MyApplication.jar”
  - The Win32 JRE allows “double clicking”
- Import other JAR files:
  - Class-Path: place/OtherStuff.jar

Version Control

- Name: ThePackage/MyApplication/
  Sealed: true
  Implementation-Title: “My Stuff”
  Implementation-Version: “build57”
  Implementation-Vendor: “My Company”
- Requires all classes of this application to be present in this JAR and saves some descriptive information with it

Signed JARS

- Digital signatures verify the integrity and source of the JAR
- Use the “keytool” to manipulate your public and private keys
- Use “jarsigner” to actually sign the jar
  - jarsigner will overwrite your existing JAR file and replace it with the signed version

Remote JARs

- You can run remote JARs with the JarRunner class…
  - java JarRunner http://server/MyJar.jar
- The java.util.jar package has utilities to download and use JAR files into your current VM
  - JarClassLoader(http://server/MyJar.jar);
  - invokeClass(“TheClassName”, args);

JAVA Extensions

- The JRE can be extended using code that you write in JAVA…
  - A number of core technologies started out as extensions to the JRE
  - Commonly used extensions from Sun include JAVAMail (to access email via IMAP), JAVA3D (for 3D graphcs) and JAXP (XML parsing and generation)

Your Own Extensions

- Create a JAR that contains your code with the proper package statements setup for a global namespace
- Put the JAR in JRE_HOME/lib/ext
  - Win32 - c:\jdk1.3\jre\lib\ext\
  - Linux - /opt/jdk1.3/jre/lib/ext/
  - Solaris - /usr/java/jre/lib/ext/

Good Practices

- Globalize your namespace!
- Separate your executable into libraries that can be reused and code for an application (front end)
  - Install your reusable code in jre/lib/ext
  - Put your front-end code into a JAR with a Main-Class attribute in the manifest
  - Sign and seal all of your JARs