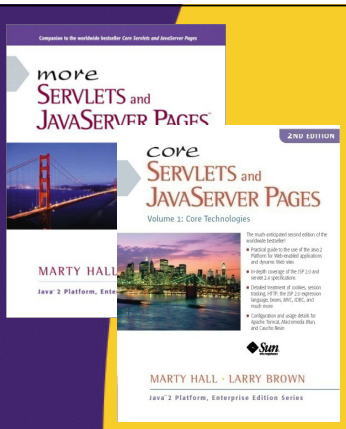
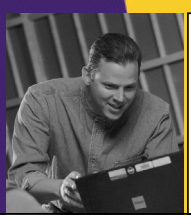
Using and Deploying Web Applications

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/msajsp.html

© 2009 Marty Hall





Agenda

- Purpose of Web applications
- Structure of Web applications
- Setting up Web applications with Tomcat
- Sharing data among Web applications

Idea of Web Applications

- Single directory or file
  - Servlets, JSP pages, HTML files, utility classes, beans, tag libraries, etc. are bundled together in a single directory hierarchy or file
- Common URL prefix
  - Access to content in the Web app is always through a URL that has a common prefix
  - http://host/webAppPrefix/blah/blah
- web.xml controls many things
  - Many aspects of Web application behavior controlled through deployment descriptor (web.xml)
    - The deployment descriptor is covered in detail in the next section.

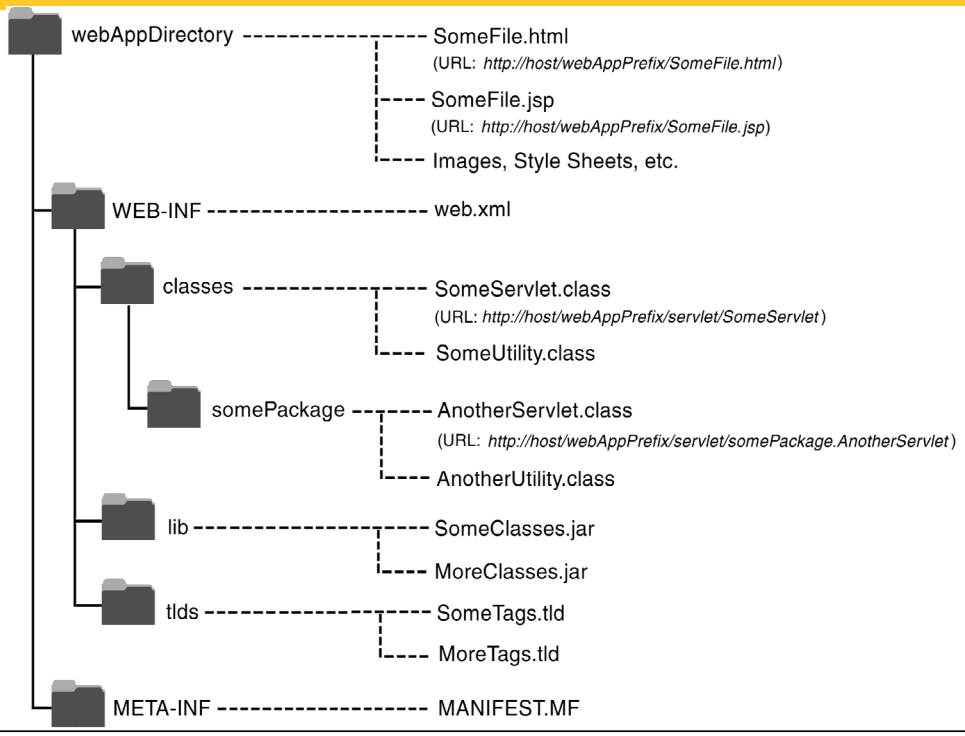
Purposes of Web Applications

- **Organization**
  - Related files grouped together in a single file or directory hierarchy.
    - HTML files, JSP pages, servlets, beans, images, etc.
- **Portability**
  - All compliant servers support Web apps.
  - Can redeploy on new server by moving a *single* file.
- **Separation**
  - Each Web app has its own:
    - ServletContext
    - Class loader
    - Sessions
    - URL prefix
    - Directory structure

Structure of a Web Application

- **JSP and regular Web content (HTML, style sheets, images, etc.):**
  - Main directory or a subdirectory thereof.
- **Servlets:**
  - WEB-INF/classes (if servlet is unpackaged – i.e. in default package)
  - A subdirectory thereof that matches the package name.
- **Unjarred beans and utility classes:**
  - Same place as servlets (but *always* use packages!)
- **JAR files:**
  - WEB-INF/lib.
- **web.xml:**
  - WEB-INF
- **Tag Library Descriptor files:**
  - WEB-INF or subdirectory thereof
- **Files in WEB-INF not directly accessible to clients**
  - Server can use RequestDispatcher to forward to pages in WEB-INF

Example Deployment Structure

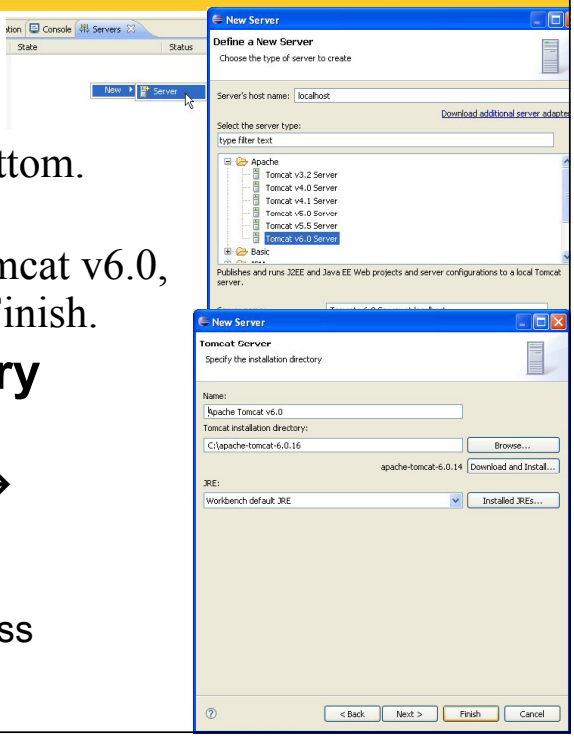


Installing Eclipse

- **Overview**
  - Eclipse is a free open-source development environment with support for Java and many other languages
- **Downloading**
  - http://www.eclipse.org/downloads/
    - Choose "Eclipse IDE for Java EE Developers"
    - As of 8/2008, version 3.4, called Eclipse Ganymede
- **Installing**
  - Unzip into directory of your choice
  - Put shortcut to eclipse.exe on your desktop
- **Integrating Tomcat in Eclipse**
  - http://www.coreservlets.com/Apache-Tomcat-Tutorial/eclipse.html

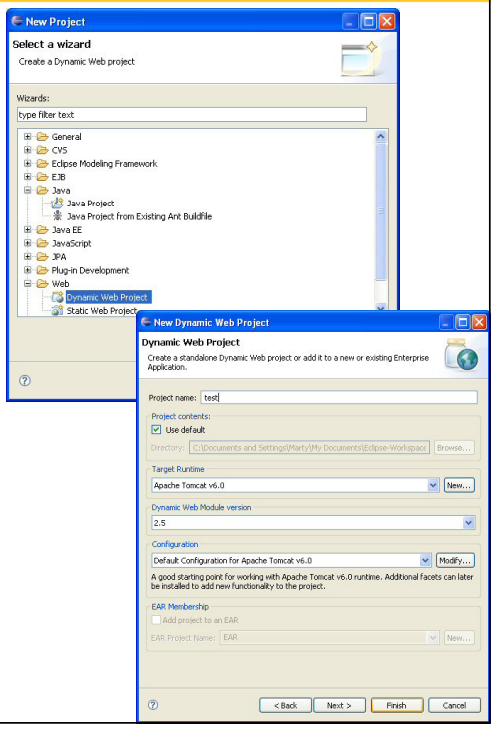
Configuring Eclipse

- **Make sure Eclipse knows about Tomcat**
  - Click on Servers tab at bottom. R-click in window.
  - New, Server, Apache, Tomcat v6.0, Next, navigate to folder, Finish.
- **Suppress unnecessary compiler warnings**
  - Window → Preferences → Java → Compiler → Errors/Warnings
    - Change "Serializable class without ..." to "Ignore"



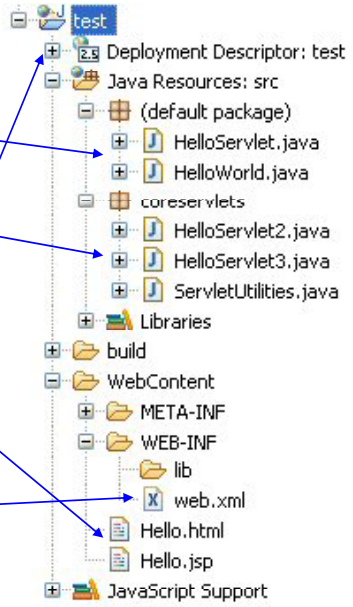
Making Web Apps in Eclipse

- **Make empty project**
  - File → New → Project → Web → Dynamic Web Project
  - Give it a name (e.g., "test")
  - Accept all other defaults
- **Shortcut**
  - If you have made Dynamic Web Project recently in workspace, you can just do File → New → Dynamic Web Project



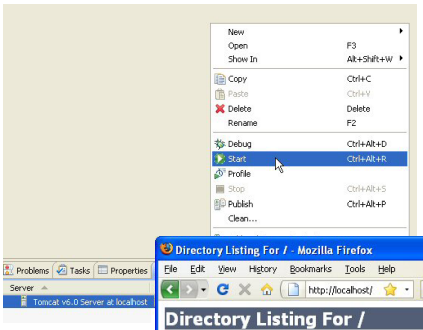
Adding Code to Eclipse Projects

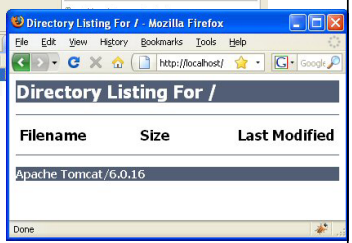
- **Locations**
  - src
    - Unpackaged Java code
    - Packages strongly recommended
  - src/somePackage
    - Java code in somePackage package
  - WebContent
    - Web files (HTML, JavaScript, CSS, JSP, images, etc.)
  - WebContent/some-subdirectory
    - Web content in subdirectory
  - WebContent/WEB-INF
    - web.xml (will be discussed later)
    - Can also click on "Deployment Descriptor"
- **Note**
  - Can cut/paste or drag/drop files into appropriate locations

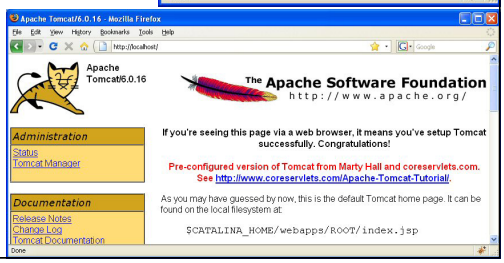


Starting Server in Eclipse

- **Start Tomcat**
  - Select "Servers" tab at bottom
  - R-click on Tomcat
  - Choose "Start"
- **Verify server startup**
  - Open browser
  - Enter http://localhost/
    - You should see blank directory listing
      - If you want pretty Tomcat welcome page, search for a folder called ROOT in your Eclipse workspace. Copy files from C:*tomcat-dir*\webapps\ROOT to that folder

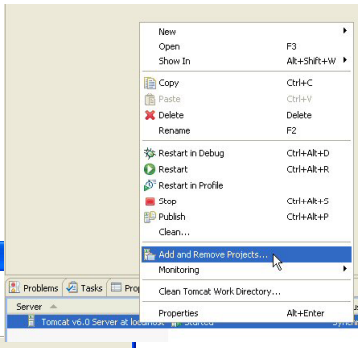


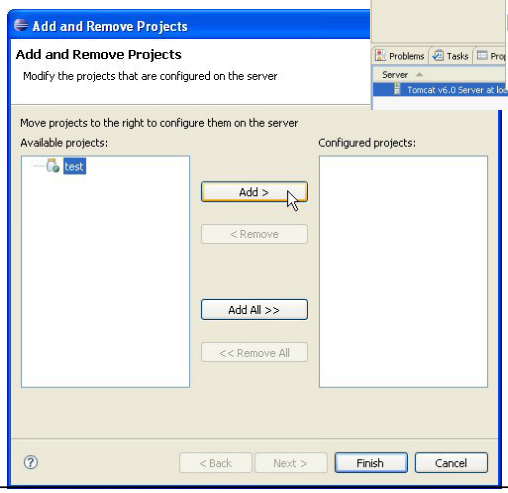




Deploying App in Eclipse

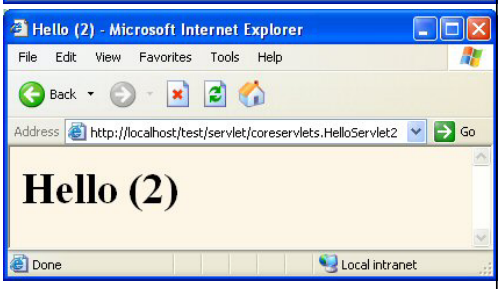
- **Deploy project**
  - Select "Servers" tab at bottom
  - R-click on Tomcat
  - Choose "Add and Remove Projects"
  - Choose project
  - Press Add
  - Click "Finish"
- **Restart Server**
  - R-click Tomcat at bottom
  - Restart





Testing Deployed Apps in Eclipse

- **Start a browser**
  - Eclipse also has builtin browser, but I prefer to use Firefox or Internet Explorer
- **Test base URL**
  - http://localhost/test/
- **Test Web content**
  - http://localhost/test/Hello.html (case sensitive!)
  - http://localhost/test/Hello.jsp
  - If you used subdirectories
    - http://localhost/test/some-subdirectory/blah.html
- **Test servlets**
  - http://localhost/test/servlet/HelloServlet
  - http://localhost/test/servlet/coreservlets.HelloServlet2
    - Note: custom URLs discussed in next section



Eclipse Structure (IDE-specific) vs. Deployment Structure (Standard)

Eclipse

- **Java code**
  - src/subDirMatchingPackage
- **HTML, JSP, Images**
  - WebContent
  - WebContent/randomDir
- **web.xml**
  - WebContent/WEB-INF

Deployed

- **Java code**
  - deployDir/webAppName/WEB-INF/classes/subDirMatchingPackage
- **HTML, JSP, Images**
  - deployDir/webAppName
  - deployDir/webAppName/randomDir
- **web.xml**
  - deployDir/webAppName/WEB-INF
- **Note**
  - On Tomcat, deployDir is tomcat_installdir/webapps

Making Custom Web Apps Manually

1. **Make a directory called app-blank**
   - app-blank/WEB-INF/web.xml (copy from mine)
   - app-blank/WEB-INF/classes (empty)
2. **Copy/rename**
   - E.g., copy app-blank and call it myApp
3. **Put code in proper place in myApp**
   - Web content (HTML , JSP, images, etc.) goes in the top-level directory (myApp) or any subdirectory other than WEB-INF (e.g., myApp/someDir)
   - Servlets and other classes go in a subdirectory of WEB-INF/classes that matches the package name.
4. **Copy app to deployment directory**
   - On Tomcat, entire directory goes in *install_dir*/webapps
5. **Update your CLASSPATH.**
   - Add *webAppDir*/WEB-INF/classes to it.
   - Not usually needed if you have ".." in the CLASSPATH

Manual Web App Development Strategy with Tomcat

- **Development**
  - Keep the original of your Web app directory in your development directory. Have all the files in the proper location within that Web app directory.
- **Deployment**
  - Copy the entire Web app directory to the server's deployment location (e.g., to *install_dir*/webapps).
    - I keep a shortcut to webapps and drag the Web app dir onto the shortcut with the R mouse and then say "Copy".
- **CLASSPATH**
  - Must include the top-level development directory
    - That now means WEB-INF/classes dir of your Web app
    - If your CLASSPATH has "..", you can leave CLASSPATH unchanged as long as you avoid nested packages

Changing the Web App Prefix

- **Eclipse default: project name is Web App prefix**
  - So, if project is named foo, when you deploy locally the URL is http://localhost/foo/whatever
- **Tomcat default: folder name is Web App prefix**
  - So, if you deploy the folder bar to tomcat_dir/webapps, the URL is http://localhost/bar/whatever.
- **Custom prefix in Eclipse**
  - R-click on project, then Properties → Web Project Settings → Context Root
- **Custom prefix in Tomcat**
  - Edit tomcat_dir/conf/server.xml

Defining Custom URLs

- **Java code**

```
package myPackage;  ...
public class MyServlet extends HttpServlet { ... }
```

- **web.xml entry (in <web-app...>...</web-app>)**
  - Give name to servlet

```
<servlet>
  <servlet-name>MyName</servlet-name>
  <servlet-class>myPackage.MyServlet</servlet-class>
</servlet>
```

  - Give address (URL mapping) to servlet

```
<servlet-mapping>
  <servlet-name>MyName</servlet-name>
  <url-pattern>/MyAddress</url-pattern>
</servlet-mapping>
```

- **Resultant URL**
  - http://*hostname*/*webappPrefix*/MyAddress

Defining Custom URLs: Example (Assume Eclipse Project is "test")

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    ...
    version="2.5">

  <!-- Use the URL http://hostname/intro/hi instead
       of http://hostname/intro/servlet/HelloServlet -->
  <servlet>
    <servlet-name>Second Hello Servlet</servlet-name>
    <servlet-class>coreservlets.HelloServlet2</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Second Hello Servlet</servlet-name>
    <url-pattern>/hi2</url-pattern>
  </servlet-mapping>
</web-app>
```

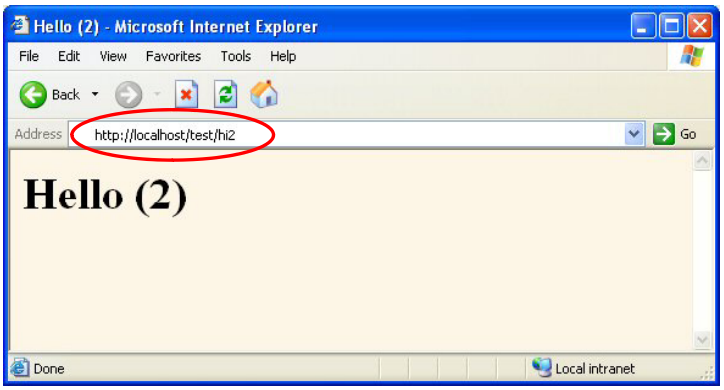
Don't edit this manually. Should refer to version 2.4 or 2.5 (Tomcat 6 only).

Fully qualified classname.

Any arbitrary name. But must be the same both times.

The part of the URL that comes after the app (project) name. Should start with a slash.

Defining Custom URLs: Result

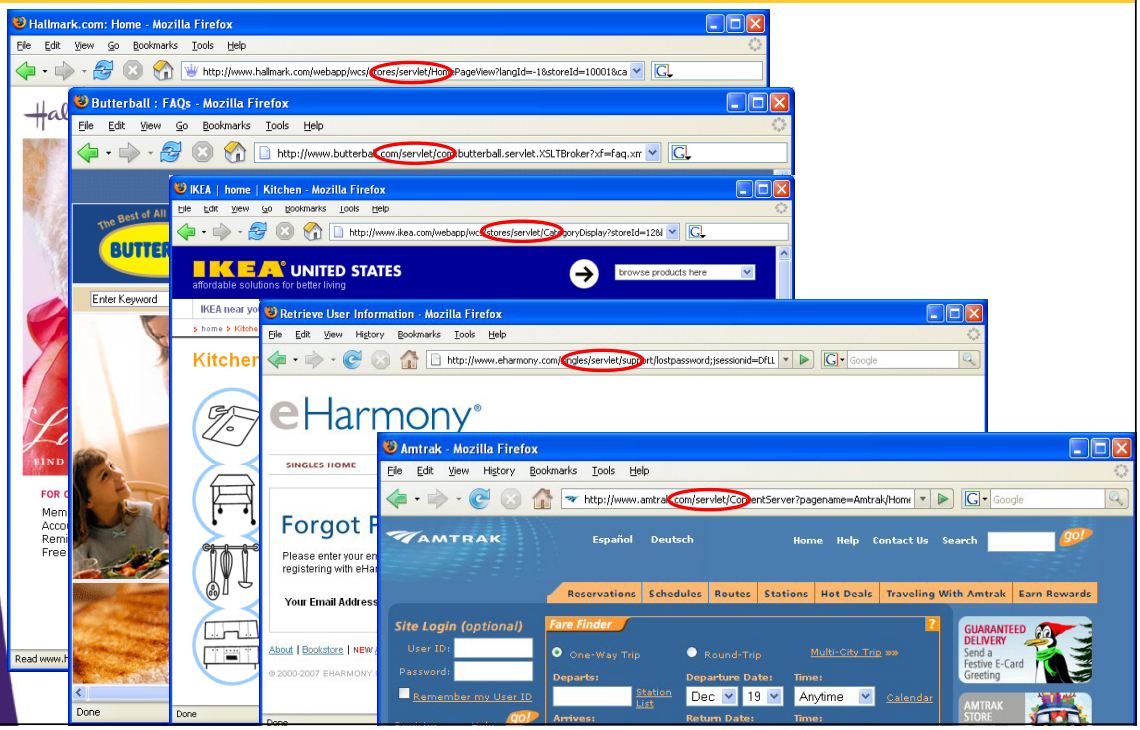


- **Eclipse details**
  - Name of Eclipse project is "test"
  - Servlet is in src/coreservlets/HelloServlet2.java
  - Deployed by right-clicking on Tomcat, Add and Remove Projects, Add, choosing test project, Finish, right-clicking again, Start

Failing to Define Custom URLs

- **You should <u>always</u> use custom URLs on deployed projects**
  - URLs look cleaner and simpler and shorter
  - URLs have more meaningful names
  - You don't expose possibly proprietary class file names
  - You can use web.xml to assign init params later
    - Does not work with …/servlet/myPackage.MyServlet
  - You can apply filters and security settings later (via web.xml) in a more predictable and controllable manner
  - Most importantly of all, you can avoid being added to Marty's "Hall of Shame"
    - The kiss of death for any self-respecting Java EE developer

The Hall of Shame (Deployed Sites with Ugly …/servlet/… URLs)



The Art of WAR (Files)

- **WAR files are simply JAR files with a different file extension**
  - And JAR files are simply ZIP files
- **All servers are required to support Web apps that are in WAR files**
  - Technically, they are not absolutely required to support unbundled Web apps.
- **To create a WAR file, change directory to top-level Web app directory and do:**
  - `jar cvf webAppName.war *`
  - Or use WinZip (or "Create Compressed Folder" on XP)
  - Eclipse can build WAR files automatically
    - R-click project, Export → WAR file
- **Registering is still server-specific**
  - Tomcat: just drop WAR file in *install_dir*/webapps
  - *webAppName* becomes Web application URL prefix

Handling Relative URLs: Problem

- **Individual JSP or HTML page: easy to load image from relative location**
  - <IMG SRC="EvilEmpire.gif" …>
  - <IMG SRC="redmond/EvilEmpire.jpg" …>
- **What about servlets?**
  - Same strategy doesn't work
  - Default servlet URL: http://host/prefix/servlet/Name
  - *Browser*, not *server*, resolves relative URL
- **What if same image is used by JSP or HTML pages scattered throughout app?**
  - Same problem
- **Also same problem:**
  - Style sheets, applets, even regular hypertext links

Handling Relative URLs: Solutions

- **Use the Web application name in the URL.**
  - <IMG SRC="/webAppPrefix/redmond/Borg.gif" ...>
- **Use web.xml to assign URLs that are at the top level of the Web application**
  - Change http://host/webAppPrefix/servlet/SomeName to just http://host/webAppPrefix/SomeName
  - More useful for servlets than for JSP
- **Use getContextPath**
  - Call request.getContextPath() and add result to URLs by hand

Velocity, WebMacro, and Other Alternatives to JSP Technology

- **Issues**
  - Standardization
  - Portability
  - Integration
  - Industry support
  - Technical features
- **Arguments for alternatives focus almost exclusively on last issue**
  - Even if proponents were right about all their technical arguments, would that matter?

Alternatives to JSP Technology: Integration Issues

- **Web apps give standard location for:**
  - Servlets, JSP pages, and regular Web content
  - Not for Velocity or WebMacro pages
- **Security settings apply to**
  - Servlets, JSP pages, and regular Web content
  - Not Velocity or WebMacro pages
- **Initialization parameters defined for**
  - Servlets and JSP pages
  - Not Velocity or WebMacro pages
- **Filters apply to**
  - Servlets, JSP pages, and regular Web content
  - Not Velocity or WebMacro pages
- **Listeners apply to**
  - Servlets, JSP pages, and regular Web content
  - Not Velocity or WebMacro pages

Sharing Data Among Web Applications

- **Failure:**
  - **Sessions.** Each Web app has its own set of sessions.
  - **Standard ServletContext.** Each Web app has a separate one.
  - **Static methods or fields.** Each Web app uses a different ClassLoader.
- **Success:**
  - **Explicit cookies.** Cookies are shared by the whole site (even the whole top-level domain if set appropriately).
    - Be sure to do cookie.setPath("/"), however.
  - **ServletContext associated with a specific URL.**

```
ServletContext myContext =
  getServletContext();
String url = "/someWebAppPrefix";
ServletContext otherContext =
  myContext.getContext(url);
Object someData =
  otherContext.getAttribute("someKey");
```


Setting Shared Data: Example

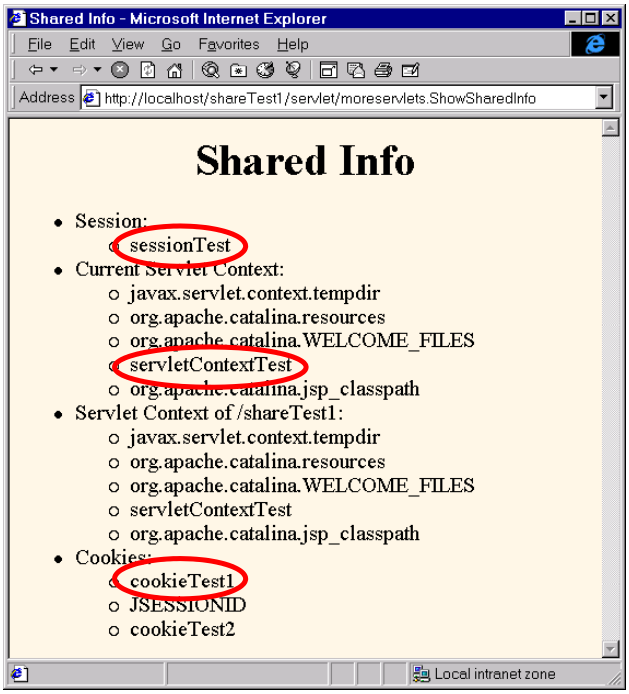
```
public class SetSharedInfo extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession(true);
    session.setAttribute("sessionTest","Session Entry One");
    ServletContext context = getServletContext();
    context.setAttribute("servletContextTest",
                         "Servlet Context Entry One");
    Cookie c1 = new Cookie("cookieTest1", "Cookie One");
    c1.setMaxAge(3600);      // One hour
    response.addCookie(c1); // Default path
    Cookie c2 = new Cookie("cookieTest2", "Cookie Two");
    c2.setMaxAge(3600);      // One hour
    c2.setPath("/");         // Explicit path: all URLs
    response.addCookie(c2);
    String url = request.getContextPath() +
                 "/servlet/moreservlets.ShowSharedInfo";
    // In case session tracking is based on URL rewriting.
    url = response.encodeRedirectURL(url);
    response.sendRedirect(url);
  }}
```

Displaying Shared Data: Example

```
public class ShowSharedInfo extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Shared Info";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n"+
                "…");
    HttpSession session = request.getSession(true);
    …
    ServletContext application = getServletContext();
    …
    application = application.getContext("/shareTest1");
    …
    Cookie[] cookies = request.getCookies();
```

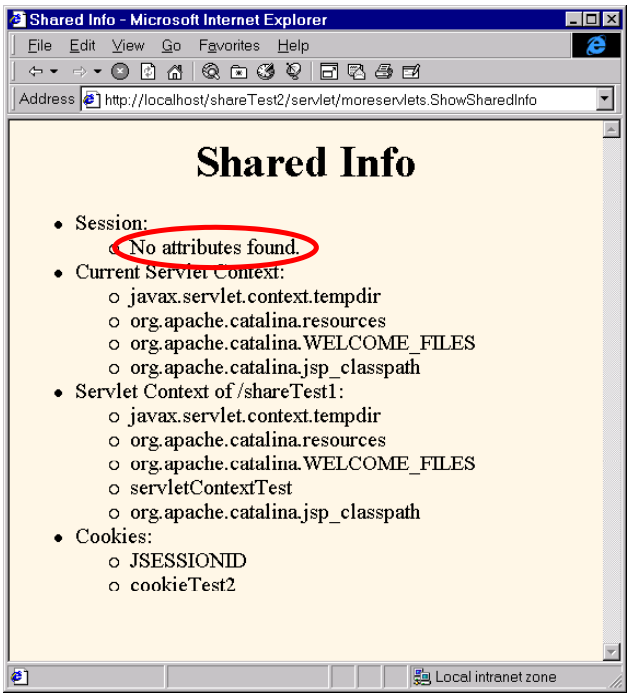

Accessing Web App Data: Case 1

- SetSharedInfo run from shareTest1
- ShowSharedInfo also run from shareTest1
- Results
  - **Found:** session data
  - **Found:** servlet context data from normal servlet context
  - **Found:** servlet context data when explicitly requesting servlet context from shareTest1
  - **Found:** all cookies



Accessing Web App Data: Case 2

- SetSharedInfo run from shareTest1
- ShowSharedInfo run from shareTest2
- Results
  - **Not found:** session data
  - **Not found:** servlet context data from normal servlet context
  - **Found:** servlet context data when explicitly requesting servlet context from shareTest1
  - **Not found:** cookies that had default path
  - **Found:** cookies with / as path



Summary

- **Web application benefits**
  - Easy organization and deployment
  - Isolation from other applications
- **Structure**
  - Top-level directory or subdirectory other than WEB-INF:
    - JSP, HTML, other Web content
  - WEB-INF
    - web.xml
  - WEB-INF/classes/directoryMatchingPackage
    - Servlets, beans, utilities
- **Creating a Web app in Eclipse**
  - Make a new Dynamic Web project.
  - Eclipse will create deployment structure automatically.
- **Creating a Web app in Tomcat**
  - Make a directory with proper structure (e.g. WEB-INF and WEB-INF/classes subdirectories)
  - Copy to tomcat_dir/webapps.

Questions?