Java Servlets

IS2470 Interactive Systems Design (Fall 2011)

Agenda

- Overview (HTTP, URL, …)
- Servlets
- Servlet Examples
- Create and Run Servlets in Tomcat

Overview

- [Java] Servlets
  - Pieces of code (like applets, ASP, PHP, CGI)
  - Reside on server, receive requests from / send output to client browser or another servlet
  - Applets are downloaded to the client, servlets run on server
- HTTP - HyperText Transfer Protocol
  - Operations: GET, POST, PUT, DELETE
  - GET - typing web address and hitting enter
  - POST - submitting a web form

Overview (cont'd)

- URL - Uniform Resource Locators

- Protocols: http, ftp, e2dk
- Server: localhost
- Ports: 80 (browsers), 8080 (Tomcat's default), 21 (ftp)

Overview (cont'd)

- URL as viewed from a Servlet

Agenda

- Overview (HTTP, URL, …)
- **Servlets**
- Servlet Examples
- Create and Run Servlets in Tomcat

Servlet API

javax.servlet.http.HttpServlet

- `init(ServletConfig config)`
  - Initializes servlet in server's Java machine
- `getServletConfig()`
  - Access to init parameters and context
- `service(ServletRequest request, ServletResponse response)`
  - Key method, processing of request and assembling response
  - Would handle all requests: GET, POST, …

Servlet API (cont'd)

javax.servlet.http.HttpServlet

- `doGet(ServletRequest request, ServletResponse response)`
  - Handles GET requests
- `doPost(ServletRequest request, ServletResponse response)`
  - Handles POST requests
- `destroy()`
  - Called in the end of servlet lifecycle

Servlet Lifecycle

- `init`
  - runs once, when servlet class is (re)loaded
- `service` or `doGet` or `doPost`
  - run multiple times in a threaded mode, up to 150 concurrent threads (Tomcat default, could be changed)
- `destroy`
  - runs once, when servlet class is unloaded

Servlet Request
javax.servlet.http.HttpServletRequest

- `HttpServletRequest`
  - Object implementing this interface is passed to `service/doGet/doPost` methods (by Tomcat)
  - Contains information about client request
- `getParameter(String name)`
  - Returns value of a specific parameter (or `null`)

http://www.pitt.edu:6500/accnts/is2470?cls=090507&obj=stdlst

```
String s = request.getParameter("cls");
s.equals("090507")==true
```


Servlet Request (cont'd)
javax.servlet.http.HttpServletRequest

- `getParameterNames()`
  - Returns all passed parameter names

http://www.pitt.edu:6500/accnts/is2470?cls=090507&obj=stdlst

- `getParameterValue()`
  - Returns all parameter values

http://www.pitt.edu:6500/accnts/is2470?cls=090507&obj=stdlst

- `getCookies()`
  - Returns cookies stored on client side

Servlet Response
javax.servlet.http.HttpServletResponse

- `HttpServletResponse`
  - Object implementing this class is passed to `service/doGet/doPost` methods
  - Provides functionality to assembly a response to the client
- `addCookie(Cookie cookie)`
  - Stores cookie on client side
- `setContentType(String type)`
  - Sets the type of servlet output.
  - E.g. in case of HTML `text/html` is usually used
- `getWriter()`
  - Returns `PrintWriter` object used for text output to client

Agenda

- Overview (HTTP, URL, …)
- Servlets
- **Servlet Examples**
- Create and Run Servlets in Tomcat

Difference between GET & POST

- GET
  - All parameters are visible
  - URL length is limited to 255 characters
- POST
  - All parameters are hidden
  - No limit for URL and parameters length
  - Non-character data (Java objects, audio, video)

Servlet Handling GET request

- Demo - Knowledge Tree→IS2470…→Lecture 3→doGet Servlet Method example

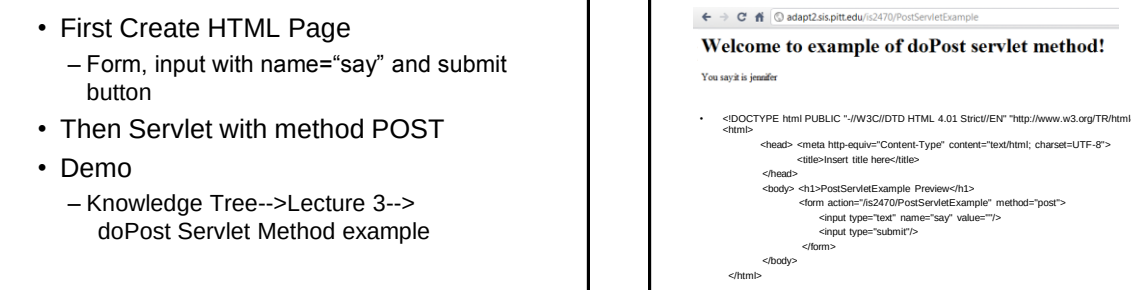
```
// Handling HTTP GET Requests by Servlets
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GetServletExample extends javax.servlet.http.HttpServlet
{
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException
   {
      String say_param = request.getParameter("say");
      response.setContentType( "text/html" );
      PrintWriter out = response.getWriter();
      out.println("<HTML><HEAD><TITLE>Get Servlet Example</TITLE></HEAD>");
      out.println("<BODY>");
      out.println("<H1>Welcome to example of doGet servlet method!</H1>");
      out.println(((say_param != null && say_param.length() != 0)?"<p>You say:" + say_param + "</p>":""));
      out.println("</BODY>");
      out.println("</HTML>");
      out.close();
   }
}
```

Servlet Handling GET request

Servlet Handling POST request

- First Create HTML Page
  - Form, input with name="say" and submit button
- Then Servlet with method POST
- Demo
  - Knowledge Tree-->Lecture 3--> doPost Servlet Method example

Servlet Handling POST request




```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
    </head>
    <body> <h1>PostServletExample Preview</h1>
        <form action="/is2470/PostServletExample" method="post">
            <input type="text" name="say" value=""/>
            <input type="submit"/>
        </form>
    </body>
</html>
```


Servlet Handling POST request

```
package edu.pitt.sis.paws.is2470;
// Handling HTTP GET Requests by Servlets
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class PostServletExample extends javax.servlet.http.HttpServlet
 {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException
    {
        String say_param = request.getParameter("say");
        response.setContentType( "text/html" );
        PrintWriter out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>Get Servlet Example</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Welcome to example of doPost servlet method!</H1>");
        out.println(((say_param != null && say_param.length() != 0)?"<p>You say:" + say_param + "</p>":""));
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

JSPs and Servlets

- JSP - Java Server Page – (X)HTML+servlet code snippets
- In the end JSPs compile into a server (Jasper library on Tomcat)
- When appropriate to use
  - A lot of static HTML, limited servlet functionality necessary

Agenda

- Overview (HTTP, URL, …)
- Servlets
- Servlet Examples
- **Create and Run Servlets in Tomcat**

Before You Begin

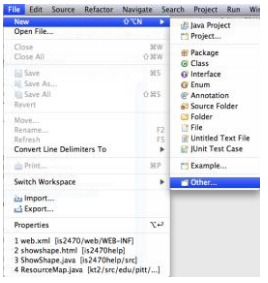
- Prerequisite
  - Java & Tomcat Installation & Configuration
    - CourseWeb→IS2470→Links→Java Servlet Resources→Java and Tomcat. Install and Configure

Anatomy of a Web Application

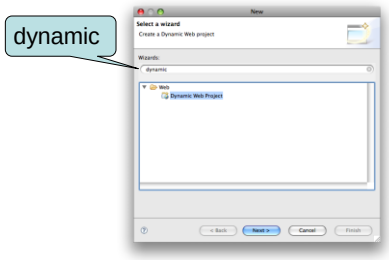
- %TOMCAT_HOME% (e.g. c:\tomcat)
  - **/webapps**
    - **/%app_folder%**
      - **/WEB-INF** - protected content
        - **/classes** - compiled classes (servlets go here)
        - **/lib** - libraries (jar files)
        - web.xml - deployment descriptor
      - **/META-INF**
        - context.xml - advanced configuration, e.g. database connection pooling, user authentication realms

Eclipse: New Project

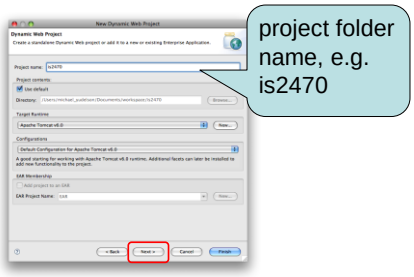
If you have NetBeans, the procedure & the things to pay attention to are virtually the same



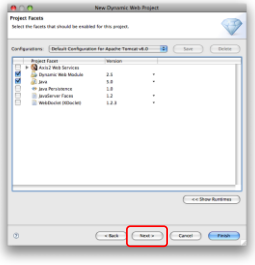
Eclipse: Dynamic Web Project



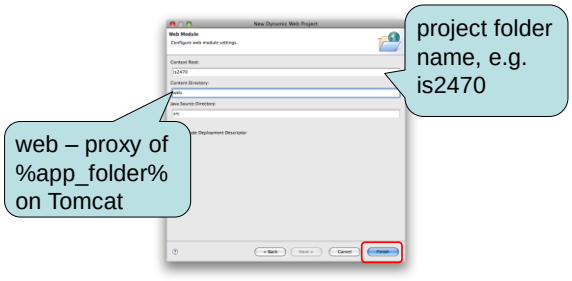
Eclipse: Project Folder Name



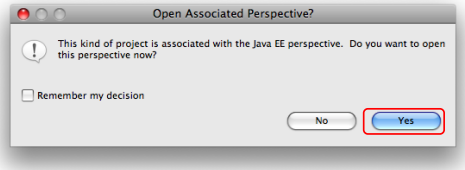
Eclipse: Accept defaults



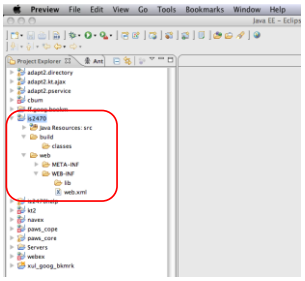
Eclipse: src, web Folders



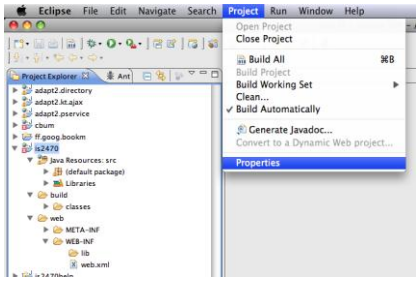
Eclipse: Switch to J2EE Perspective



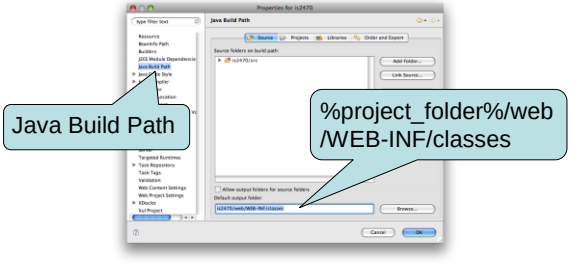
Eclipse: Project Stub Ready



Eclipse: Change Project Properties



Eclipse: Change Project Properties (cont'd)

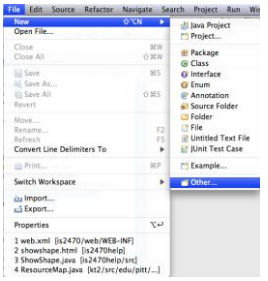


Anatomy of the Eclipse Project

- %Eclipse_workspace%
  /%project_folder%
  /build – compiled classes (we changed it)
  /src – source code
  /WebContent or **/web** – model of Tomcat's %app_folder%
- Good idea to have
  %project_folder% = %app_folder%

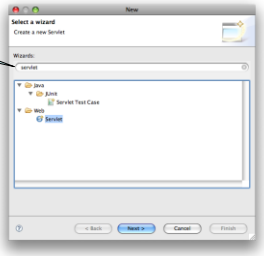
Eclipse: Creating a Servlet

If you have NetBeans, the procedure & the things to pay attention to are virtually the same



Eclipse: Search for Servlet

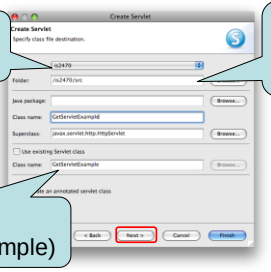
servlet



Eclipse: Name Servlet

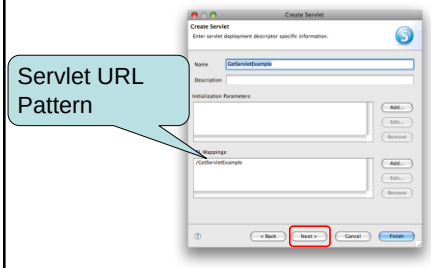
project [folder] name (is2470)

where servlet class is saved (/is2470/src)

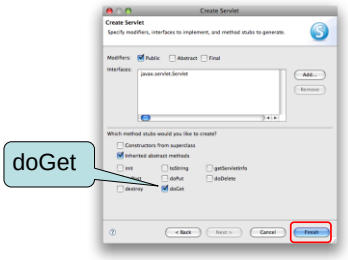


class name (GetServletExample)

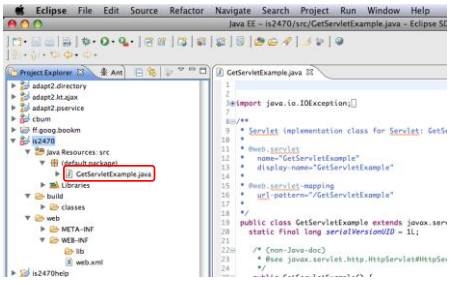
Eclipse: URL Pattern/Mapping



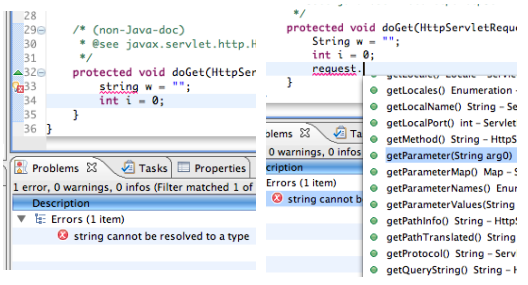
Eclipse: Stub Methods



Eclipse: Stub Servlet Class



Eclipse: Code Editor



Deploying Servlet App to Tomcat

- Copy files from Eclipse project's web folder to Tomcat application folder

  `%project_folder%/web/*.*` ➔

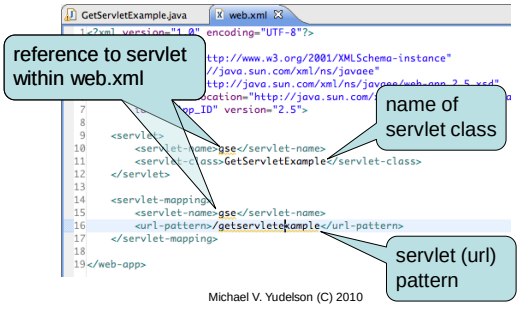
  `%TOMCAT_HOME%/webapps/%app_folder%/*.*`

  Restart Tomcat

Deployment Descriptor

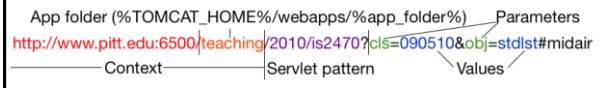
- /%app_folder%/WEB-INF/web.xml
  - Prescribes how to deploy your app to Tomcat (or other servlet servers, e.g. IBM's WebSphere)
  - Eclipse creates stub web.xml, and
  - updates web.xml for newly created servlets

Deployment Descriptor (cont'd)



Anatomy of Servlet's URL Revisited

- Servlet URL



Linking Servlets

- Your tomcat application is located at
  - `http://localhost:8080/myHW3`
- It has servlets First, Second with patterns
  - `http://localhost:8080/myHW3/doit`
  - `http://localhost:8080/myHW3/two/doit2`
- Tomcat application has the following files
  - `…/myHW3/index.html – HTML`
  - `…/myHW3/doit – mapped servlet First`
  - `…/myHW3/two/doit2 – mapped servlet Second`

Linking Servlets (cont'd)

- First prints out HTML links to Second
  - (i) `<a href="two/doit2">Link</a>`
  - (ii) `<a href="http://localhost:8080/myHW3/two/doit2">Link</a>`
- Both options will work for you, but
  - Once the app is moved elsewhere, (ii) will stop working (`http://localhost:8080…`)
  - If mapping of One changes, (i) might stop to work

Linking Servlets (cont'd)

- To properly obtain context path Servlets should use special function `request.getContextPath()`
  … is automatically mapped to `http://localhost:8080/myHW3/`
  or other location where application is deployed
- Best option to refer to Second from First
  `request.getContextPath() + "/two/doit2"`

Linking Servlets Done Right (cont'd)

- Tomcat application has the following files
  - `…/myHW3/index.html – HTML`
  - `…/myHW3/doit – mapped servlet First`
  - `…/myHW3/two/doit2 – mapped servlet Second`
- Link to servlet Second from HTML ?
  - `<a href="two/doit2">Link</a>`
  - `<a href="myHW3/two/doit2">Link</a>`
  - `<a href="/myHW3/two/doit2">Link</a>`
  - `<a href="http://localhost:8080/myHW3/two/doit2">Link</a>`

Linking Servlets Done Right (cont'd)

- Tomcat application has the following files
  - …/myHW3/index.html – HTML
  - …/myHW3/doit – mapped servlet First
  - …/myHW3/two/doit2 – mapped servlet Second
- Link to servlet Second from servlet First?
  - "Link"
  - "Link"
  - "Link"
  - "Link"

Context Parameters

- You have a set of constants
  - That [could] change frequently
  - Require you to recompile your app a lot
- For example
  - Database connection (URL, user, passw'd)
  - URLs of external servers your servlet calls
  - Other constants

Context Parameters (cont'd)

- `web.xml` – add the following lines

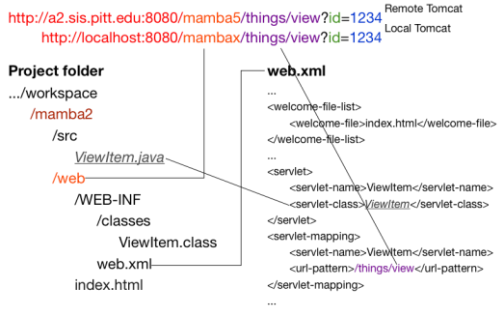
```
<context-param>
  <param-name>key23</param-name>
  <param-value>some_value34</param-value>
</context-param>
```

- In the servlet

```
getServletContext().getInitParameter("key23")
```

- Every time `web.xml` is changed Tomcat reloads the application

The Big Picture



Working with Sessions

- When sessions are useful
  - You have several interlinked servlets
  - You want to share some information between them
    - User input
    - Data retrieved from the database
    - Values obtained from other servlets

Working with Sessions (cont'd)

- New session is open
  - When a first request arrives from the client browser (identified by IP of client machine)
- If more requests arrive from same client
  - They are attributed to the same session
- Session has "time to live"
  - Tomcat default – 30 min, can be changed

Working with Sessions (cont'd)

- You can interact with sessions
  - To get some information about the session itself (e.g. creation time)
  - Store application specific data – arbitrary **objects** (values) associated with string names (keys)

Working with Sessions (cont'd)

- Getting session object
  - `HttpSession session = request.getSession()`
- Getting info about session itself
  - `session.getCreationTime()` (ms since 1-Jan-1900)
  - `session.getLastAccessedTime()` (ms since 1-Jan-1900)
  - `session.getMaxInactiveInterval()` (sec)

Working with Sessions (cont'd)

- Managing saved objects
  - Set an attribute `setAttribute("attr_name", value)`
  - Read attribute `getAttribute("attr_name")`
  - Get all attribute names `getAttributeNames()`
  - Remove attribute `removeAttribute("attr_name")`
- Invalidate session (clear all information)
  - `invalidate()`
  - If the user was logged in s/he would have to re-login

Working with Sessions (cont'd)

- Some facts about sessions
  - Only servlets of the same tomcat application can "share" a session
  - Servlets from different applications (let alone from different servers) – don't
  - There's no way to share sessions across servers (but there is a possibility to setup single-sign-on for apps of one server)

Questions?