• Still grading your midterms
• Review Midterm at the end of next week’s lecture
• Was the test Hard? Easy?
HW #3

• Due today
HW #4 & 5

- Change of plans due to late HW and test scores and time
- Will be assigned this week (I am making some modifications – it was too hard)
- Will post HW#4 this week (by end of week, watch for a class email.
- NO HW#5
Lecture 9

Servlets and JSPs
WEB Application Modules

Web Module Root Directory

JSP, HTML, GIF

WEB-INF

classes

Java class files, including servlets, filters, and listeners

lib

JAR files, including tag libraries and database drivers

web.xml, the deployment descriptor for the web module
• Right click the mounted webmodule and create a new servlet
My Servlet Directory

[Diagram of My Servlet Directory with physical and mounted mounts]
You pressed Page 1 Button
You entered firstName []
You entered lastName []
doPost() invoked on = /servlet/ServletSessionManagementDemo

First Name [ ]
Last Name [Poelman]

Page 1  Page 2  Page 3  Reset

You pressed Page 1 Button
You entered firstName []
You entered lastName [Poelman]
doPost() invoked on = /servlet/ServletSessionManagementDemo

First Name [ ]
Last Name [ ]

Page 1  Page 2  Page 3  Reset

Copyright Poelman & Associates, Inc. 2002 - All Rights Reserved
first it does a GET

doGet() invoked on
= /servlet/ServletSessionManagementDemo

First Name

Last Name

Page 1 Page 2 Page 3 Reset
two browsers executing the same servlet
see that the instance variable has been changed but the session variables are indeed local to that user.
session

- session is private to a specific user of a page (html, servlet, JSP)
- It is shared across all the pages in a web application.
- It's basically a hashmap like object that is bound to a specific client's requests (usually via a unique id stored in a browser cookie)
Session attributes are:
creation time = 1068416347685
Session ID = B0667C419BFD37ACE3D15E76969F361B
getMaxInactiveInterval = 1800
getLastAccessedTime = 1068416884557

----------------------------------

----------------------------------
Session Management

• How do you save information between HTTP requests that relate to a specific user?
  – hidden HTML fields
  – cookies
  – URL rewriting
  – a database
  – Session object of the servlet container
The GVBooks in the Lecture9 directory

- Show a multiple view (page) servlet
- Shows accessing the servlet session object
- Shows read/write the parameters of the POST
You have to ask for a session object to be created by the container.
Filled in the form and pressed SAVE
Saved it to disk, then reloaded it and saved the customer info into the session
Fields are loaded from the customer object in memory
Spawn a new browser and change the info and save
2 DIFFERENT Browsers viewing the same page and sharing the session? (same machine)
But from a different machine the session is different
you pressed Save

Customer: Poelman 212-217-2487 2307 bevery road manhattan NY 11111

Created a customer file on the server
Added the fields to the resson
Reloaded the data from disk:
Customer: Poelman 212-217-2487 2307 bevery road manhattan NY 11111

Goto display resson attributes page

Session attributes are:
creation time = 1036631634946
Session ID = 5587b11
getMaxInactiveInterval = 15
getLastAccessedTime = 1036631634946

ActionToDo: BUTTON = Save
Phone = 212-217-2487
Zipcode = 11111
Email = fifff@erterte
Address = 2307 bevery road
LastName = Poelman
City = manhattan
State = NY
Then press the Get Session Info button
The same page visited twice (see lastAccessedTime)
After a period of activity longer than the MaxInactiveInterval

- the session has timed out
- All the attributes I set are gone
- A new session has been created (SessionId is changed)
Session points

- Session is shared across servlets but not across clients of the web application
- Session is not persisted to disk
- Sessions can timeout, write your code to deal with the timeouts appropriately
- You must create the session object, not automatically created for every servlet executed (scalability issue)
3 browsers on the same machine – 2 sharing a session one that does not. Started #1 & #2 by clicking on the ICON, #3 I pressed Ctrl-N
Internet Explorer 6.x

• Start browser from start menu – get a new session each time
• “Ctrl + N” while viewing a page clones the session and shares it between both browser windows.
• Can you share session in two browsers on different machines?
• Can two different users share session objects/values?
“scopes” of a web application

- **Request** – only lives for the life of a request to the HTTP Server and the response to the browser. Not shared across users. Uses no space in the data sent from/to browser. Objects automatically freed up by web server.
- **Page** – per servlet/JSP instance - *note some engines may instantiate multiple copies of a servlet, not just one means not shared by all instances of a servlet for a given page*. Automatically frees objects at end of page execution.
- **Session** – bound to a client and shared across all the servlets in a web app accessed by that client. Stored in cookies/URLs. Sent to browser/to server.
- **Context (Application)** – shared by all clients and shared by all pages in the web application – servlets, JSPs, etc. you need to free objects when not needed.
• **Application scope**
  Application scope is the broadest scope and should only be used when necessary. You can create objects bound at application level in JSPs that are not session-aware, so application scope is useful for storing information when using these types of JSPs. You can also use application-bound objects to share data among different sessions of the same application. When you no longer need objects bound to an application, you should explicitly remove them to free up memory resources.

• **Session scope**
  In my experience, session scope is more commonly used than application scope. Session scope allows you to create and bind objects to a session. You must create objects bound to the session in session-aware JSPs and make them available to all JSPs and servlets in the same session. Session scope is often used for managing security credentials and for managing state among multiple pages (such as in a Web-based wizard). As with application scope, objects bound to session scope should be explicitly removed when no longer needed. Also, I typically make classes serializable if I intend to bind their instantiations to the session scope.

• **Request scope**
  I use request scope most often for binding created objects. Objects created and bound to request scope are available to pages in that same request. These objects are automatically released from memory when request processing completes. This is advantageous because explicit cleanup is not required and there is less risk of burdening the system with needless memory consumption.

• **Page scope**
  You should use page scope for objects created only for the current page. Like request scope, the page scope does not require explicit removal of created and bound objects. I rarely use page scope in my JSP applications, but it is the `<jsp:useBean>` action's default scope.
Relationship of servlets, clients, sessions and the web application context

Client A
Session A

Client B
Session B

Client C
Session C

Servlet W
Servlet X
Servlet Y
Servlet Z
Different State Mechanisms

<table>
<thead>
<tr>
<th></th>
<th>Shared By Servlets in Web Module</th>
<th>Shared by clients (Browser)</th>
<th>Lifespan</th>
<th>Automatic Object Cleanup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>N</td>
<td>N</td>
<td>Very Short (while request is on server)</td>
<td>Y</td>
</tr>
<tr>
<td>Request</td>
<td>Y</td>
<td>N</td>
<td>Very Short (while request is on server)</td>
<td>Y</td>
</tr>
<tr>
<td>Session</td>
<td>Y</td>
<td>N</td>
<td>Long</td>
<td>Y</td>
</tr>
<tr>
<td>Application</td>
<td>Y</td>
<td>Y</td>
<td>Very Long</td>
<td>N</td>
</tr>
</tbody>
</table>
Cookies

- Stored on client machine (browser)
- (Name.Value) pair strings
- Can be persistent for while browser is open or written to disk and persistent for a specified time interval (in seconds)
- `public Cookie(String name, String value)`
- `Cookie.setMaxAge(int expiry)`
  - A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the maximum age when the cookie will expire, not the cookie's current age.
  - A negative value means that the cookie is not stored persistently and will be deleted when the Web browser exits.
  - A zero value causes the cookie to be deleted.
- Get all cookies by `HttpServletRequest.getCookies()`
- Sent by browser to server in HTTP headers, returned in headers to client.
• Questions
• Comments
JSPs a better servlet
(YAAWT -yet another actual WWW technology!)
Adding a JSP
Create a JSP

This wizard creates a JSP file in standard syntax. JSP files can only run as part of a web module. Enter the name, and optionally a folder.

Name: HelloWorld3
Folder: 
Web Module: C:\NYUCode\CodeSamples\Lecture6
Location: C:\NYUCode\CodeSamples\Lecture6\HelloWorld3.jsp
Add some text

```xml
<%@ page content-type="text/html" %>
<%@ page page-encoding="UTF-8" %>
<html>
<head><title>JSP Page</title></head>
<body>

<!-- jsp:useBean id="beanInstanceName" scope="session" class="beanPackage.BeanClassName" / -->
<!-- jsp:getProperty name="beanInstanceName" property="propertyName" / -->
Hello World JSP
</body>
</html>
```
Execute the JSP

- NetBeans will compile it and then load it into the servlet engine (Tomcat, is installed with Netbeans)
- And then spawns a browser
- The browser will request the JSP
Change the web.xml file so the HelloWorld3.jsp is the default page served

```xml
<welcome-file-list>
  <welcome-file>HelloWorld3.jsp</welcome-file>
</welcome-file-list>
```
Java Server Pages - JSP

- Intentionally created to mimic the format of Microsoft ASPs
- Translated into Java code that is inserted into a template servlet that is then compiled
- The servlet is then loaded into the servlet engine
- When a JSP is referenced the engine checks to see if the version in memory is older than the JSP file date, if it is it retranslates it, compiles it and loads it (unlike a servlet)
A simple JSP  

```jsp
<%@page contentType="text/html"%>
<%@page import = "java.util.*"%>

<html>
<head><title>JSP Page</title></head>
<body>

<%-- <jsp:useBean id="beanInstanceName" scope="session" class="package.class" /> --%>
<%-- <jsp:getProperty name="beanInstanceName"  property="propertyName" /> --%>

The current time is <%= (new Date()).toString() %>

</body>
</html>
```
A fragment of the servlet that gets created via the translation process

```java
out.print( (new Date()).toString() );
```

```html
The current time is 
```

```java
out.print( (new Date()).toString() );
```

```html
The current time is 
```

JSP “view” of the code
To see the generated servlet

• NetBeans will pop up the servlet code if your JSP has a translation/compilation error.

• You can’t directly edit that servlet!
Servlet “view” of the code
The browser view of the code

<table>
<thead>
<tr>
<th>Counting 0</th>
<th>Counting 1</th>
<th>Counting 2</th>
<th>Counting 3</th>
<th>Counting 4</th>
<th>Counting 5</th>
<th>Counting 6</th>
<th>Counting 7</th>
<th>Counting 8</th>
<th>Counting 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hello</td>
<td>yellow</td>
<td>0</td>
<td>hello</td>
<td>yellow</td>
<td>0</td>
<td>hello</td>
<td>yellow</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>hello</td>
<td>yellow</td>
<td>1</td>
<td>hello</td>
<td>yellow</td>
<td>1</td>
<td>hello</td>
<td>yellow</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>hello</td>
<td>yellow</td>
<td>2</td>
<td>hello</td>
<td>yellow</td>
<td>2</td>
<td>hello</td>
<td>yellow</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>hello</td>
<td>yellow</td>
<td>3</td>
<td>hello</td>
<td>yellow</td>
<td>3</td>
<td>hello</td>
<td>yellow</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>hello</td>
<td>yellow</td>
<td>4</td>
<td>hello</td>
<td>yellow</td>
<td>4</td>
<td>hello</td>
<td>yellow</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>hello</td>
<td>yellow</td>
<td>5</td>
<td>hello</td>
<td>yellow</td>
<td>5</td>
<td>hello</td>
<td>yellow</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>hello</td>
<td>yellow</td>
<td>6</td>
<td>hello</td>
<td>yellow</td>
<td>6</td>
<td>hello</td>
<td>yellow</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>hello</td>
<td>yellow</td>
<td>7</td>
<td>hello</td>
<td>yellow</td>
<td>7</td>
<td>hello</td>
<td>yellow</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>hello</td>
<td>yellow</td>
<td>8</td>
<td>hello</td>
<td>yellow</td>
<td>8</td>
<td>hello</td>
<td>yellow</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>hello</td>
<td>yellow</td>
<td>9</td>
<td>hello</td>
<td>yellow</td>
<td>9</td>
<td>hello</td>
<td>yellow</td>
<td>9</td>
</tr>
</tbody>
</table>
• Questions
• Comments

• 8.0
JSP

• **Directives** – affect the overall structure of the servlet generated
  - `<%!  //directive goes here%>`
  - Static variables, methods, etc.

• **Scripting Elements**
  - `<%  //java code goes here %>`
  - Code that is compiled into the service method

• **Actions** – special tags that affect the runtime – standard and custom
  - `<jsp:useBean>`
Directives

- `<%@ directivename attribute="value" attribute="value"%>`
  - page
  - include
  - taglib
Creating a JSP based page with a table

```html
<%@page contentType="text/html"%>
<html>
<head><title>JSP Page</title></head>
<body>

<p>Create a table in HTML</p>
<div align="left">
<table border="2" style="border-collapse: collapse" bordercolor="#111111" width="90%" id="AutoNumber1">
<tr>
<td width="33%">This is a cell</td>
<td width="33%">2</td>
<td width="34%">3</td>
</tr>
<tr>
<td width="33%">4</td>
<td width="33%">So is this</td>
<td width="34%">6</td>
</tr>
<tr>
<td width="33%">7</td>
<td width="33%">8</td>
<td width="34%">And this</td>
</tr>
<tr>
<td width="33%">10</td>
<td width="33%">11</td>
<td width="34%">12</td>
</tr>
</table>
</div>
</body>
</html>
```
Request object

- A RequestDispatcher allows one JSP/Servlet to forward you to another JSP/Servlet/HTML page in the same web application via `forward()`
- Or you can include the output from a different JSP/Servlet via `include()`
- The URL should be relative without the "HTTP://localhost:8080"
JSP code fragment showing a RequestDispatcher

<%  
  if (request.getMethod().equalsIgnoreCase("POST"))  
  {  
    String buttonUserPushed = (String)request.getParameter("ActionToDoBUTTON");  
    
    if (buttonUserPushed.equalsIgnoreCase("Add or Update Customer"))  
    {  
      String URLToReference = "/Lecture9/GVBookstoreCustomerForm.jsp";  
      RequestDispatcher currentDispatcher = request.getRequestDispatcher(URLToReference);  
      currentDispatcher.forward(request,response);  
      return;  
    }  
    
    if (buttonUserPushed.equalsIgnoreCase("Browse Customers"))  
    {  
      String URLToReference = "/Lecture9/GVBookstoreCustomerManager.jsp";  
      RequestDispatcher currentDispatcher = request.getRequestDispatcher(URLToReference);  
      currentDispatcher.forward(request,response);  
      return;  
    }  
    
    if (buttonUserPushed.equalsIgnoreCase("Browse Orders"))
• JSP Best Practices


• Questions
• Comments

• 8.5
WebUI Design Patterns
MVC – Model View Controller

• Design Pattern for dealing with User Interfaces (UIs)
• Handles changes over time very well
• “Separation of concerns” approach
• Separates the:
  – input handling (controller)
  – the display logic (view)
  – In memory representation / business rules/persistence (model)
MVC – View

- View is the code that creates a displayable representation of the model (generates the markup)
- View is the code that renders the model for the user to see it
- Shouldn’t process input
  - controller’s job
- Shouldn’t contain the domain object or its attributes
  - The model
- In Swing, was the widgets like a JPanel, JList, JText
  - The part that was visible in the running app
- In Web UI, what ever creates the HTML that gets passed back to the browser
  - Could be a servlet or a JSP (JSP is easier to create a view)
MVC – Controller

• Controller processes input from user interface
  – Button pushes, entered data, slider positions, …
• It manipulates the model, via accessor methods, to update/change attributes on the model
• Shouldn’t generate the displayable model
  – that’s the views job
• Shouldn’t contain the domain object or its attributes
  – that’s the model
• In Swing, the widgets like a JPanel, JList, JText have a controller
  – Swing widgets have both view and controller (UIDelgate pattern).
  – We added controller code by code we put into the event handler for navigation and manipulating the model, updating the view.
• In Web UI, what ever processed the Post/Gets and manages navigation between pages and updates/reads the domain model object(s).
MVC – Model

• This is the representation of the physical object in code. The DomainModel is a model.
• Contains methods for accessing/modifying its attributes
• Shouldn’t process input – controller does that
• Shouldn’t generate display code – view does that
• In Swing, was the DomainModel objects like LoanApplication
• A UI might have more than one model (different things that get modeled).
  – Remember each widget in the Swing UI has a model whether it’s a JPanel, JList or JButton
• In Web UI, would be that same DomainModel
MVC is cool!

- We are adding a UI of a completely different type in HW4/5 and you should be able to reuse the DomainModel without ANY refactoring (rewriting) of the model classes!
- Just adding controllers and views that are web oriented
- Avoids “spaghetti code” for UIs allows easier modification and expansion to the UI
- Advanced MVC
  - Suppose you also wanted to support a PDA browser and a Web Browser, you could just add different views and reuse the controllers and the models!!
  - less coding by using this design pattern to start with.
FrontController

• A pattern for web UIs that extends the MVC pattern
• A front controller is a single point that all requests go to
  – Can be a controller for multiple pages
  – Or can dispatches to individual page controllers
  – Also may invoke the appropriate view, based on what is passed in in the request using the Command Pattern for Posts/Gets from a posting page to determine the view to display next

• http://java.sun.com/blueprints/patterns/FrontController.html
**Value Object (VO)**

- A class that just contains a number of private data fields
- Public accessor/mutator methods `setXxx()` and `getXxx()` methods
- Used to pass around a data structure
- Reduces making lots of smaller method calls to pass data around, just pass a value object as a parameter
- Reduces network traffic if RMI is used
- Easier evolvability of classes:
  - Add a `middleName` field later to the `PersonIdInfo` class without breaking the users of the class because they just pass a `PersonIdInfo` object around.
- **Should be** `Serializable`
- Encapsulates business data
- Ex: Address, PersonIdInfo,
• Questions
• Comments

• 8.75
A few bug warnings
Shadow variable bug alert

```
//fragment from my class PersonIdInfo
private String     firstName;
public void setFirstName(String FirstName)
{
    this.firstName = firstName;
}

//this code compiles but is, in fact a bug, a very
//hard to find bug
//what is it?
//Would my JUnit tests have caught this bug?
```
Visibility of your classes in a web application bug alert

- Code that is not in the classes or lib directories are NOT visible by default in runtime to your web app. You will get loading errors.
Summary
Summary

• Servlets/JSPs have
  – page, session, request and application scope variables

• JSP are:
  – a markup friendly servlet format
  – `<% %>` scriptlets
    • intermingle HTML and Java code in JSP

• RequestDispatcher class can be used to invoke other servlets
  – forward() – “jump” without return
  – include() – method call
Summary

• MVC
  – Model – the business object that is in memory
    • POJO – Plain Old Java Object like the LoanApplication
    • EJB – Enterprise Java Bean
    • Database Table
  – View – generates the markup to display some/all of the attributes of the model
    • Swing Widget
    • JSP
  – Controller – Manages input from the user.
    • Swing Widget
    • POJO
    • Servlet

• Front Controller
  – Single controller that may dispatch/delegate to other Page Controllers
  – Servlet
  – Good: single entry point
  – Bad: could get bloated if many views

• Value Object
  – A Business Object ex: PersonIdInfo, Address,
  – Stores a set of attributes in a single object
  – Good: Saves net bandwidth and eases evolution of your code
  – Bad: Attributes could get “stale” as they are a copy of the actual object (some times),

• Command Pattern
  – Single interface method that accepts commands as values for the parameter
  – HTTP commands (Post, Get, …) are all sent to the same HTTP Port. The HTTP server looks at the data sent to determine what command was requested
  – Good: Avoids explosion of interface methods – one per command – easier evolution
  – Bad: what if a bad command is sent by requester?