

# Course Material Usage Rules

- **PowerPoint slides for use only in for-credit courses at degree-granting institutions**
  - Slides not permitted for commercial training courses except when taught by coreservlets.com (see <http://courses.coreservlets.com>).
- **Slides may be modified by instructor**
  - But please retain reference to coreservlets.com
- **Instructor may give PDF or hardcopy to students**
  - But should protect the PowerPoint files

*This slide is suppressed in Slide Show mode*



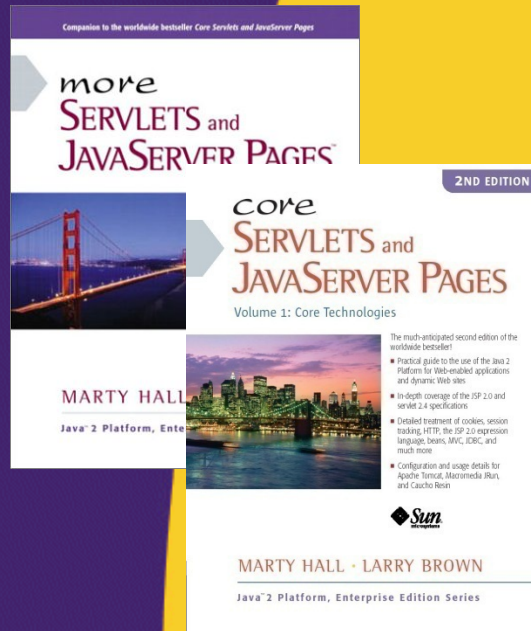
# Integrating Servlets and JSP: The Model View Controller (MVC) Architecture

Originals of Slides and Source Code for Examples:  
<http://courses.coreservlets.com/Course-Materials/csajsp2.html>

**Customized Java EE Training:** <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live Java EE training, please see training courses at <http://courses.coreservlets.com/>.**

**JSF 2, PrimeFaces, Servlets, JSP, Ajax (with jQuery), GWT, Android development, Java 6 and 7 programming, SOAP-based and RESTful Web Services, Spring, Hibernate/JPA, XML, Hadoop, and customized combinations of topics.**

**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact [hall@coreservlets.com](mailto:hall@coreservlets.com) for details.**



# Agenda

- **Understanding the benefits of MVC**
- **Using RequestDispatcher to implement MVC**
- **Forwarding requests from servlets to JSP pages**
- **Handling relative URLs**
- **Choosing among different display options**
- **Comparing data-sharing strategies**



# MVC Motivation

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



# Uses of JSP Constructs

Simple  
Application



Complex  
Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- **Servlet/JSP combo (MVC)**
- **MVC with JSP expression language**
- Custom tags
- MVC with beans, custom tags, and a framework like JSF 2.0

# Why Combine Servlets & JSP?

- **Typical picture: use JSP to make it easier to develop and maintain the HTML content**
  - For simple dynamic code, call servlet code from scripting elements
  - For slightly more complex applications, use custom classes called from scripting elements
  - For moderately complex applications, use beans and custom tags
- **But, that's not enough**
  - For complex processing, starting with JSP is awkward
  - Despite the ease of separating the real code into separate classes, beans, and custom tags, the assumption behind JSP is that a *single* page gives a *single* basic look

# Possibilities for Handling a Single Request

- **Servlet only. Works well when:**
  - Output is a binary type. E.g.: an image
  - There is *no* output. E.g.: you are doing forwarding or redirection as in Search Engine example.
  - Format/layout of page is highly variable. E.g.: portal.
- **JSP only. Works well when:**
  - Output is mostly character data. E.g.: HTML
  - Format/layout mostly fixed.
- **Combination (MVC architecture). Needed when:**
  - A single request will result in multiple substantially different-looking results.
  - You have a large development team with different team members doing the Web development and the business logic.
  - You perform complicated data processing, but have a relatively fixed layout.



# MVC Misconceptions

- **An elaborate framework is necessary**
  - Frameworks are often useful
    - JSF (JavaServer Faces)
      - You should *strongly* consider JSF 2.0 for medium/large projects!
    - Struts
  - They are *not* required!
    - Implementing MVC with the builtin RequestDispatcher works very well for most simple and even moderately complex applications
- **MVC totally changes your system design**
  - You can use MVC for individual requests
  - Think of it as the MVC *approach*, not the MVC *architecture*
    - Also called the *Model 2* approach

# MVC-Based Alternative to Servlets and JSP: JSF 2

- **Servlets and JSP**
  - Well-established standard
  - Used by google.com, ebay.com, walmart.com, and thousands of other popular sites
  - Relatively low level by today's standards
  - Covered in this tutorial
- **JSF (JavaServer Faces) Version 2**
  - Now an official part of Java EE 6
    - But runs in any recent Java-enabled server, including Tomcat 6+
  - Higher-level features: integrated Ajax support, field validation, page templating, rich third-party component libraries, etc. *Designed around the MVC approach.*
  - Not yet as widely used, but recommended for many or most new projects
  - Covered at <http://www.coreservlets.com/JSF-Tutorial/jsf2/>



# Beans

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Review: Beans

- **Java classes that follow certain conventions**
  - (Must have a zero-argument (empty) constructor)
    - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
    - In this version of MVC, it is not required to have zero arg constructor if you only instantiate from Java code
  - Should have no public instance variables (fields)
    - I hope you already follow this practice and use accessor methods instead of allowing direct access to fields
  - Persistent values should be accessed through methods called `getXxx` and `setXxx`
    - If class has method `getTitle` that returns a String, class is said to have a String *property* named `title`
    - Boolean properties can use `isXxx` instead of `getXxx`

# Bean Properties: Examples

Method Names	Property Name	Example JSP Usage
getFirstName setFirstName	firstName	<jsp:getProperty ... property="firstName"/> <jsp:setProperty ... property="firstName"/> \${customer.firstName}
isExecutive setExecutive (boolean property)	executive	<jsp:getProperty ... property="executive"/> <jsp:setProperty ... property="executive"/> \${customer.executive}
getExecutive setExecutive (boolean property)	executive	<jsp:getProperty ... property="executive"/> <jsp:setProperty ... property="executive"/> \${customer.executive}
getZIP setZIP	ZIP	<jsp:getProperty ... property="ZIP"/> <jsp:setProperty ... property="ZIP"/> \${address.ZIP}

Note 1: property name does not exist anywhere in your code. It is just a shortcut for the method name.

Note 2: property name is derived only from method name. Instance variable name is irrelevant.



# Example: StringBean

```
package coreservlets;  
  
public class StringBean {  
    private String message = "No message specified";  
  
    public String getMessage() {  
        return(message);  
    }  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
}
```

- **Beans installed in normal Java directory**
  - Eclipse: *src/folderMatchingPackage*
  - Deployed: *.../WEB-INF/classes/folderMatchingPackage*
    - Beans (and utility classes) must *always* be in packages!



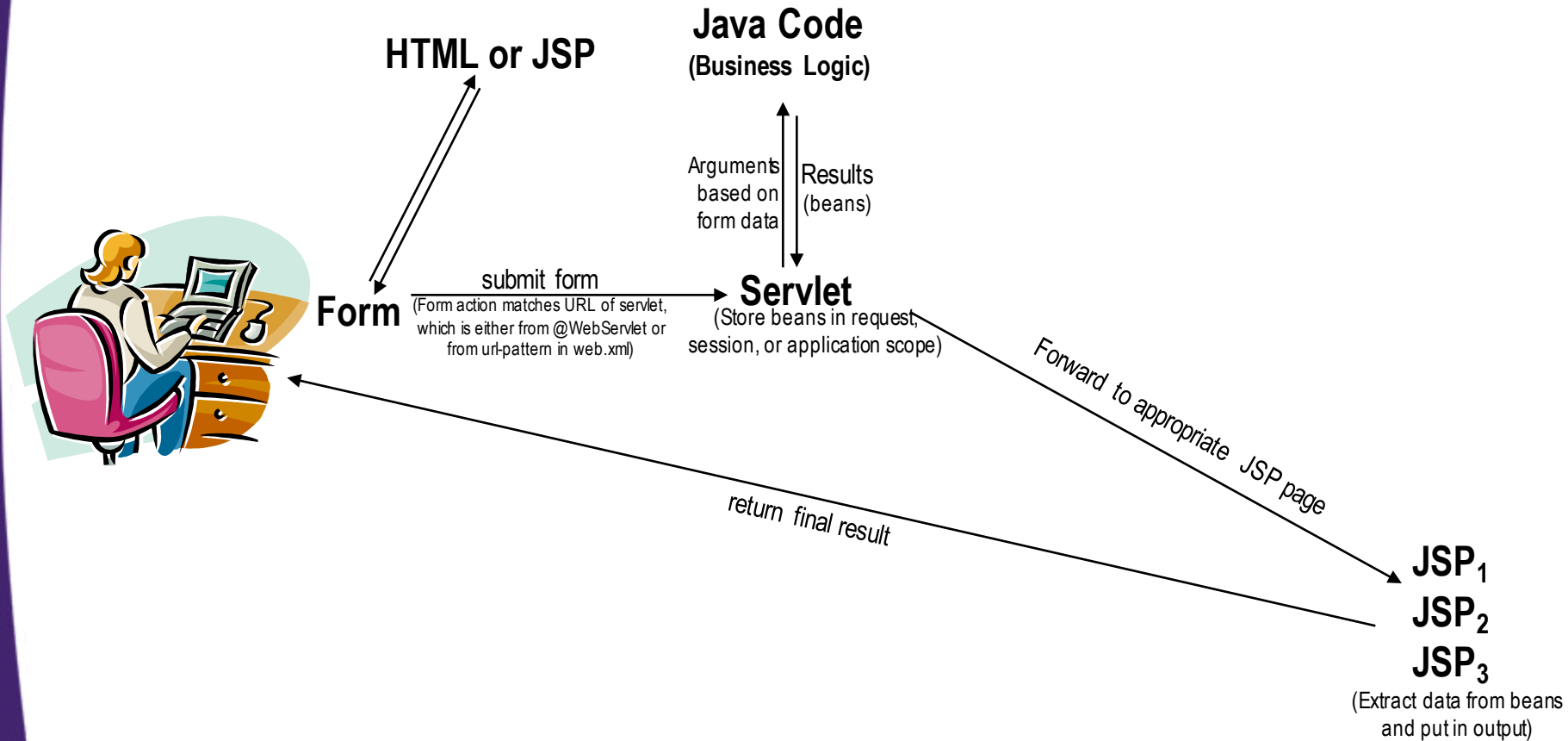
# Basic MVC Design

**Customized Java EE Training: <http://courses.coreservlets.com/>**

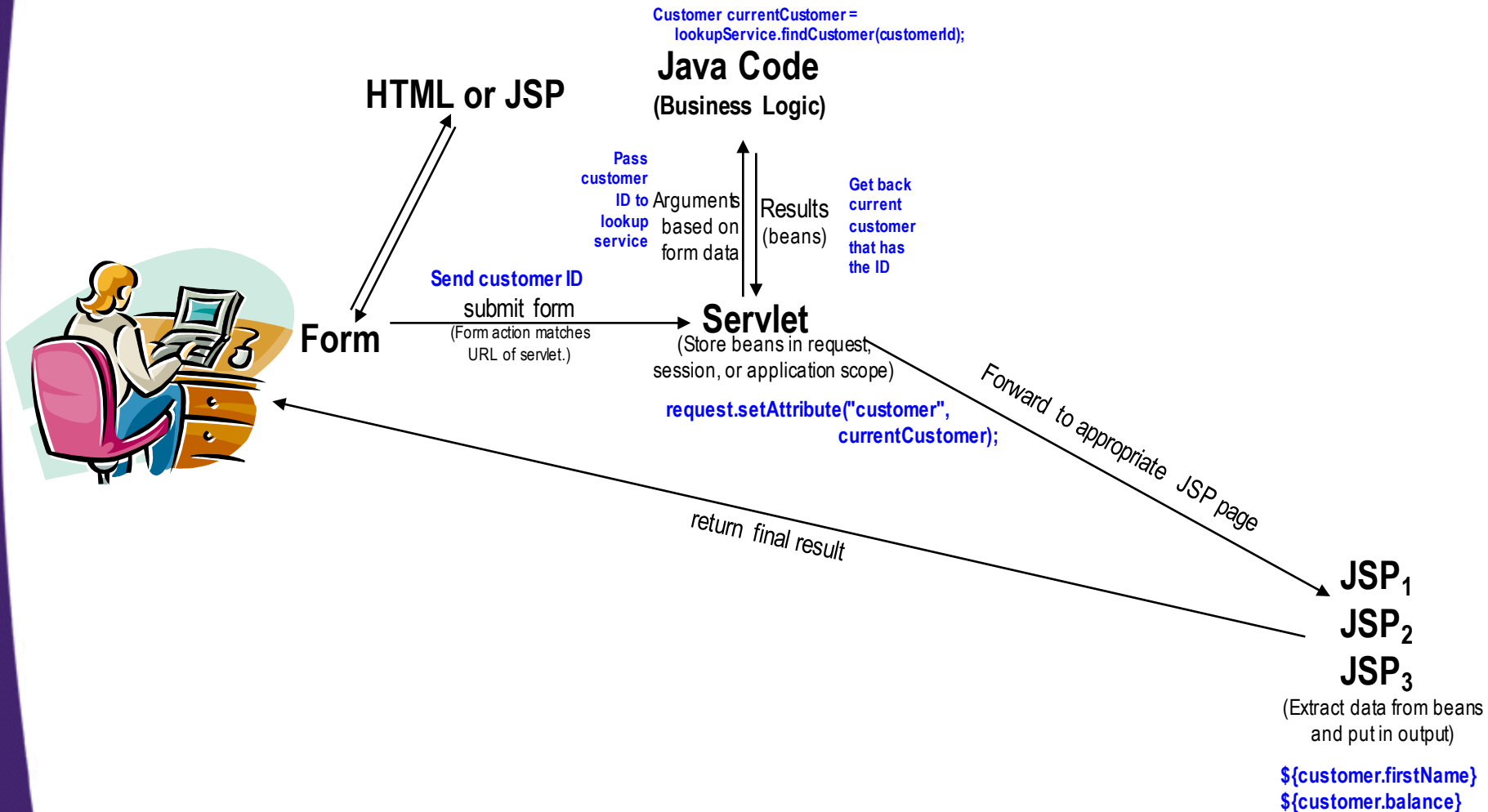
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# MVC Flow of Control



# MVC Flow of Control (Annotated)



Parts in blue are examples for a banking application.

# Implementing MVC with RequestDispatcher

## 1. Define beans to represent result data

- Ordinary Java classes with at least one *getBlah* method

## 2. Use a servlet to handle requests

- Servlet reads request parameters, checks for missing and malformed data, calls business logic, etc.

## 3. Obtain bean instances

- The servlet invokes business logic (application-specific code) or data-access code to obtain the results.

## 4. Store the bean in the request, session, or servlet context

- The servlet calls `setAttribute` on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.



# Implementing MVC with RequestDispatcher (Continued)

## 5. Forward the request to a JSP page.

- The servlet determines which JSP page is appropriate to the situation and uses the forward method of RequestDispatcher to transfer control to that page.

## 6. Extract the data from the beans.

- JSP 1.2 (Old!)
  - The JSP page accesses beans with jsp:useBean and a scope matching the location of step 4. The page then uses jsp:getProperty to output the bean properties.
- JSP 2.0 (Preferred!)
  - The JSP page uses `${nameFromServlet.property}` to output bean properties
- Either way, JSP page does not create or modify bean; it merely extracts and displays data that servlet created.

# Request Forwarding Example

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    ... // Do business logic and get data
    String operation = request.getParameter("operation");
    if (operation == null) {
        operation = "unknown";
    }
    String address;
    if (operation.equals("order")) {
        address = "/WEB-INF/Order.jsp";
    } else if (operation.equals("cancel")) {
        address = "/WEB-INF/Cancel.jsp";
    } else {
        address = "/WEB-INF/UnknownOperation.jsp";
    }
    RequestDispatcher dispatcher =
        request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

# jsp:useBean in MVC vs. in Standalone JSP Pages

- **The JSP page should not create the objects**
  - The servlet, not the JSP page, should create all the data objects. So, to guarantee that the JSP page will not create objects, you should use  
`<jsp:useBean ... type="package.Class" />`  
instead of  
`<jsp:useBean ... class="package.Class" />`
- **The JSP page should not modify the objects**
  - So, you should use `jsp:getProperty` but not `jsp:setProperty`.



# Scopes: request, session, and application (ServletContext)

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Scopes

- **Idea**

- A “scope” is a place that the bean is stored. This place controls where and for how long the bean is visible.

- **Three choices**

- Request
    - Data stored in the request is visible to the servlet and to the page the servlet forwards to. Data cannot be seen by other users or on other pages. Most common scope.
  - Session
    - Data stored in the request is visible to the servlet and to the page the servlet forwards to. Data can be seen on other pages or later in time if it is the same user. Data cannot be seen by other users. Moderately common.
  - Application (Servlet Context)
    - Data stored in the servlet context is visible to all users and all pages in the application. Rarely used.



# Request-Based Data Sharing

- **Servlet**

```
SomeBean value = LookupService.findResult(...);  
request.setAttribute("key", value);  
RequestDispatcher dispatcher =  
    request.getRequestDispatcher  
        ("/WEB-INF/SomePage.jsp");  
dispatcher.forward(request, response);
```

- **JSP 2.0**

```
${key.someProperty}
```

Name chosen by the servlet.

Name of accessor method, minus the word "get", with next letter changed to lower case.

- **JSP 1.2 (Old!)**

```
<jsp:useBean id="key" type="somePackage.SomeBean"  
    scope="request" />  
<jsp:getProperty name="key" property="someProperty" />
```

# Request-Based Data Sharing: Simplified Example

- **Servlet**

Assume that the findCust method  
handles missing/malformed data.

```
Customer myCustomer =  
    Lookup.findCust(request.getParameter("customerID"));  
request.setAttribute("customer", myCustomer);  
RequestDispatcher dispatcher =  
    request.getRequestDispatcher  
        ("/WEB-INF/SomePage.jsp");  
dispatcher.forward(request, response);
```

- **JSP 2.0**

```
${customer.firstName}
```

Note: the Customer class must  
have a method called "getFirstName".

- **JSP 1.2**

```
<jsp:useBean id="customer" type="somePackage.Customer"  
            scope="request" />  
<jsp:getProperty name="customer" property="firstName"/>
```

# Session-Based Data Sharing

- **Servlet**

```
SomeBean value = LookupService.findResult(...);  
HttpSession session = request.getSession();  
session.setAttribute("key", value);  
RequestDispatcher dispatcher =  
    request.getRequestDispatcher  
        ("/WEB-INF/SomePage.jsp");  
dispatcher.forward(request, response);
```

- **JSP 2.0**

```
${key.someProperty}
```

- **JSP 1.2**

```
<jsp:useBean id="key" type="somePackage.SomeBean"  
            scope="session" />  
<jsp:getProperty name="key" property="someProperty" />
```

# Session-Based Data Sharing: Variation

- **Redirect to page instead of forwarding to it**
  - Use `response.sendRedirect` instead of `RequestDispatcher.forward`
- **Distinctions: with `sendRedirect`:**
  - User sees JSP URL (user sees only servlet URL with `RequestDispatcher.forward`)
  - Two round trips to client (only one with forward)
- **Advantage of `sendRedirect`**
  - User can visit JSP page separately
    - User can bookmark JSP page
- **Disadvantages of `sendRedirect`**
  - Two round trips to server is more expensive
  - Since user can visit JSP page without going through servlet first, bean data might not be available
    - So, JSP page needs code to detect this situation

# ServletContext-Based Data Sharing (Rare)

- **Servlet**

```
synchronized(this) {  
    SomeBean value = SomeLookup.findResult(...);  
    getServletContext().setAttribute("key", value);  
    RequestDispatcher dispatcher =  
        request.getRequestDispatcher  
            ("/WEB-INF/SomePage.jsp");  
    dispatcher.forward(request, response);  
}
```

- **JSP 2.0**

```
${key.someProperty}
```

- **JSP 1.2**

```
<jsp:useBean id="key" type="somePackage.SomeBean"  
             scope="application" />  
<jsp:getProperty name="key" property="someProperty" />
```





# Example: Bank Balance Lookup

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Applying MVC: Bank Account Balances

- **Bean**
  - BankCustomer
- **Business Logic**
  - BankCustomerLookup
- **Servlet that populates bean and forwards to appropriate JSP page**
  - Reads customer ID, calls BankCustomerLookup's data-access code to obtain BankCustomer
  - Uses current balance to decide on appropriate result page
- **JSP pages to display results**
  - Negative balance: warning page
  - Regular balance: standard page
  - High balance: page with advertisements added
  - Unknown customer ID: error page

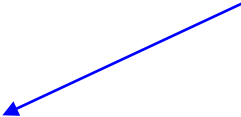
# Bank Account Balances: Servlet Code

```
@WebServlet("/show-balance")
public class ShowBalance extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        String customerId = request.getParameter("customerId");
        CustomerLookupService service = new CustomerSimpleMap();
        Customer customer = service.findCustomer(customerId);
        request.setAttribute("customer", customer);
        String address;
        if (customer == null) {
            request.setAttribute("badId", customerId);
            address = "/WEB-INF/results/unknown-customer.jsp";
        } else if (customer.getBalance() < 0) {
            address = "/WEB-INF/results/negative-balance.jsp";
        } ... /* normal-balance and high-balance cases*/ ...
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}
```

# Bank Account Balances: Bean

```
public class Customer {  
    private final String id, firstName, lastName;  
    private final double balance;  
  
    public Customer(String id,  
                    String firstName,  
                    String lastName,  
                    double balance) {  
        this.id = id;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.balance = balance;  
    }  
  
    // getId, getFirstName, getLastName, getBalance. No setters.  
  
    public double getBalanceNoSign() {  
        return(Math.abs(balance));  
    }  
}
```

Since the constructor is called from Java only (never from JSP), the requirement for a zero-arg constructor is eliminated. Also, since bean state is set only with constructor, rather than with `jsp:setProperty`, we can eliminate setter methods and make the class immutable.



# Bank Account Balances: Business Logic Interface

```
public interface CustomerLookupService {  
    public Customer findCustomer(String id) ;  
}
```

# Bank Account Balances: Business Logic Implementation

```
public class CustomerSimpleMap
    implements CustomerLookupService {
    private Map<String, Customer> customers;

    public CustomerSimpleMap() {
        // Populate Map with some sample customers
    }

    public Customer findCustomer(String id) {
        if (id != null) {
            return customers.get(id.toLowerCase());
        } else {
            return null;
        }
    }
    ...
}
```



# Bank Account Balances: Input Form

...

```
<fieldset>
```

```
  <legend>Bank Account Balance</legend>
```

```
  <form action="show-balance">
```

```
    Customer ID (id001, id002, id003):
```

```
    <input type="text" name="customerId"/><br/>
```

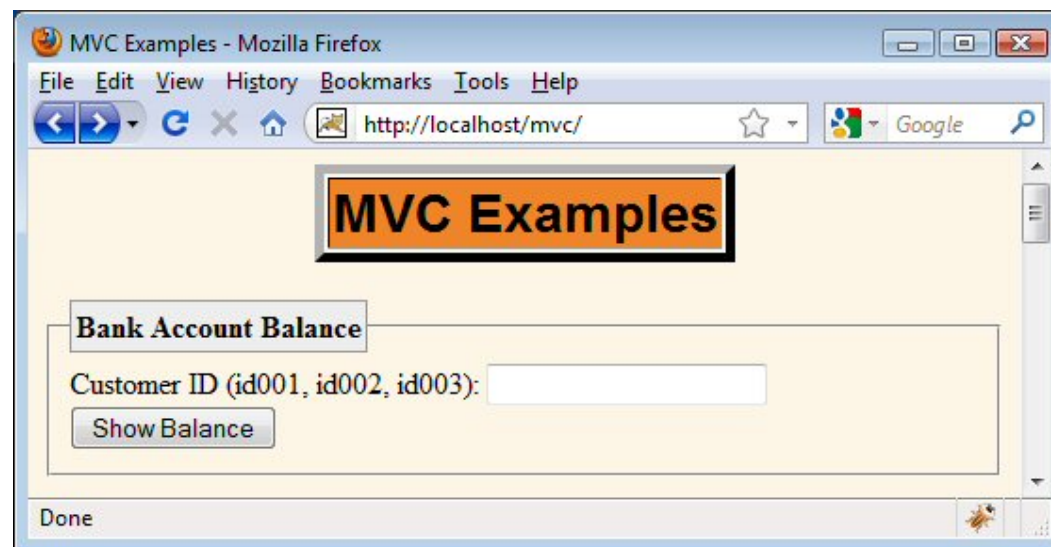
```
    <input type="submit" value="Show Balance"/>
```

```
  </form>
```

```
</fieldset>
```

...

The address `http://host/appName/show-balance` comes from the `@WebServlet` annotation in this case, but could also be set in older servers using the `url-pattern` entry in `web.xml`.

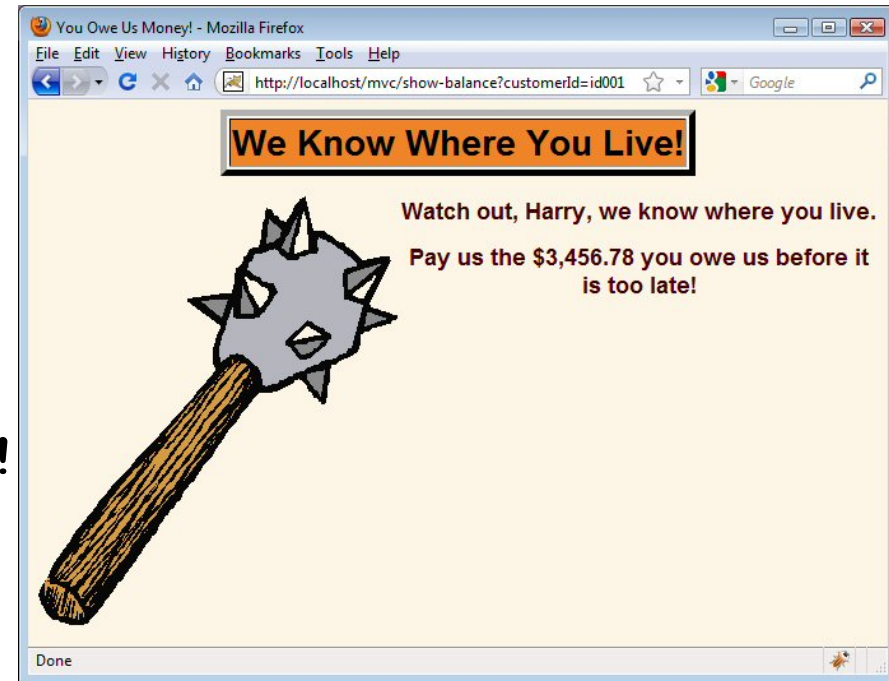


# Bank Account Balances: Negative Balance (JSP 2.0)

...

```
<body>
<div align="center">
<table border="5">
  <tr><th class="title">
    We Know Where You Live!
  </th></tr>
</table>
<p/>

<h2>Watch out, ${customer.firstName},
we know where you live. </h2>
<h2>Pay us the $$${customer.balanceNoSign}
you owe us before it is too late!</h2>
</div></body></html>
```



# Bank Account Balances: Negative Balance (JSP 1.2)

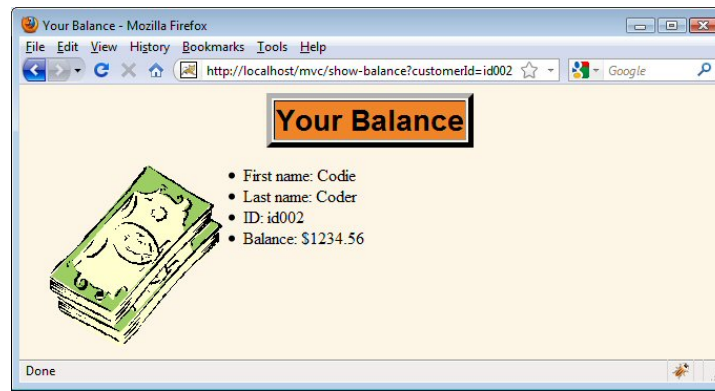
```
...<body>
<div align="center">
<table border="5">
  <tr><th class="title">We Know Where You Live!</th></tr>
</table>
<p/>

<jsp:useBean id="customer"
              type="coreservlets.Customer"
              scope="request" />
<h2>Watch out,
<jsp:getProperty name="customer"
                  property="firstName" />,
we know where you live. </h2>
<h2>Pay us the
$<jsp:getProperty name="customer"
                  property="balanceNoSign" />
you owe us before it is too late!</h2>
</div></body></html>
```

# Bank Account Balances: Normal Balance

```
...<body>
<table border="5" align="center">
  <tr><th class="title">Your Balance</th></tr>
</table>
<p/>

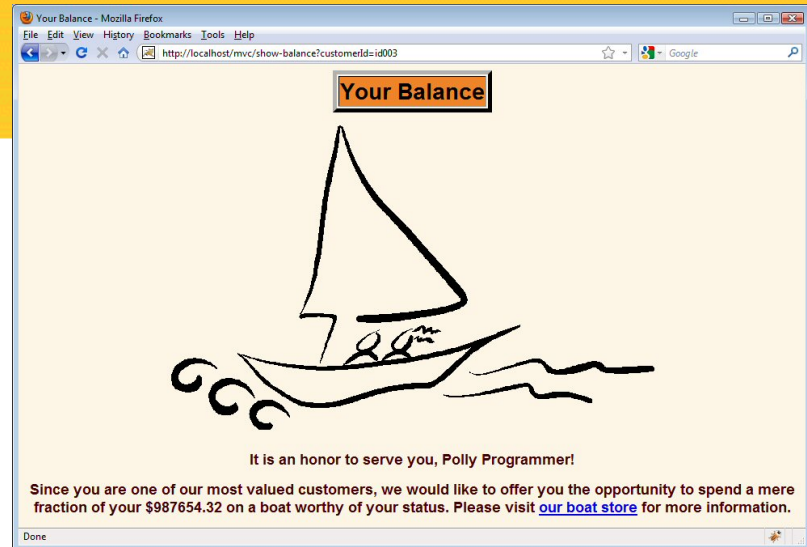
<ul>
  <li>First name: ${customer.firstName}</li>
  <li>Last name: ${customer.lastName}</li>
  <li>ID: ${customer.id}</li>
  <li>Balance: $$${customer.balance}</li>
</ul>
</body></html>
```



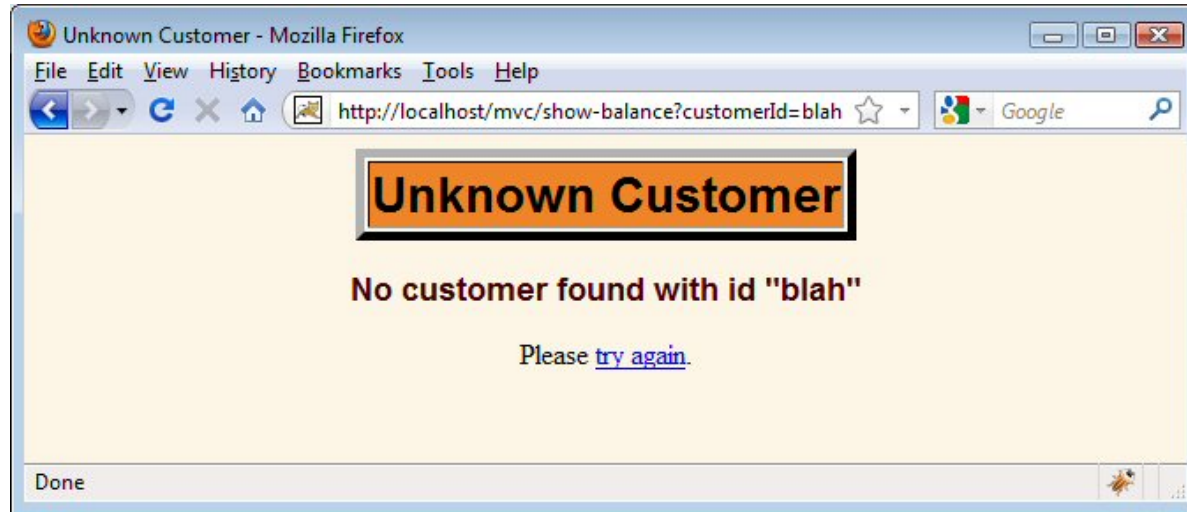
# Bank Account Balances: High Balance

```
...  
<body>  
<div align="center">  
...  
<br clear="all"/>  
<h2>It is an honor to serve you,  
${customer.firstName} ${customer.lastName}!  
</h2>  
<h2>
```

```
Since you are one of our most valued customers, we would like  
to offer you the opportunity to spend a mere fraction of your  
${customer.balance} on a boat worthy of your status.  
Please visit <a href="http://overpricedyachts.com">  
our boat store</a> for more information.  
</h2>  
</div></body></html>
```



# Bank Account Balances: Unknown Customer

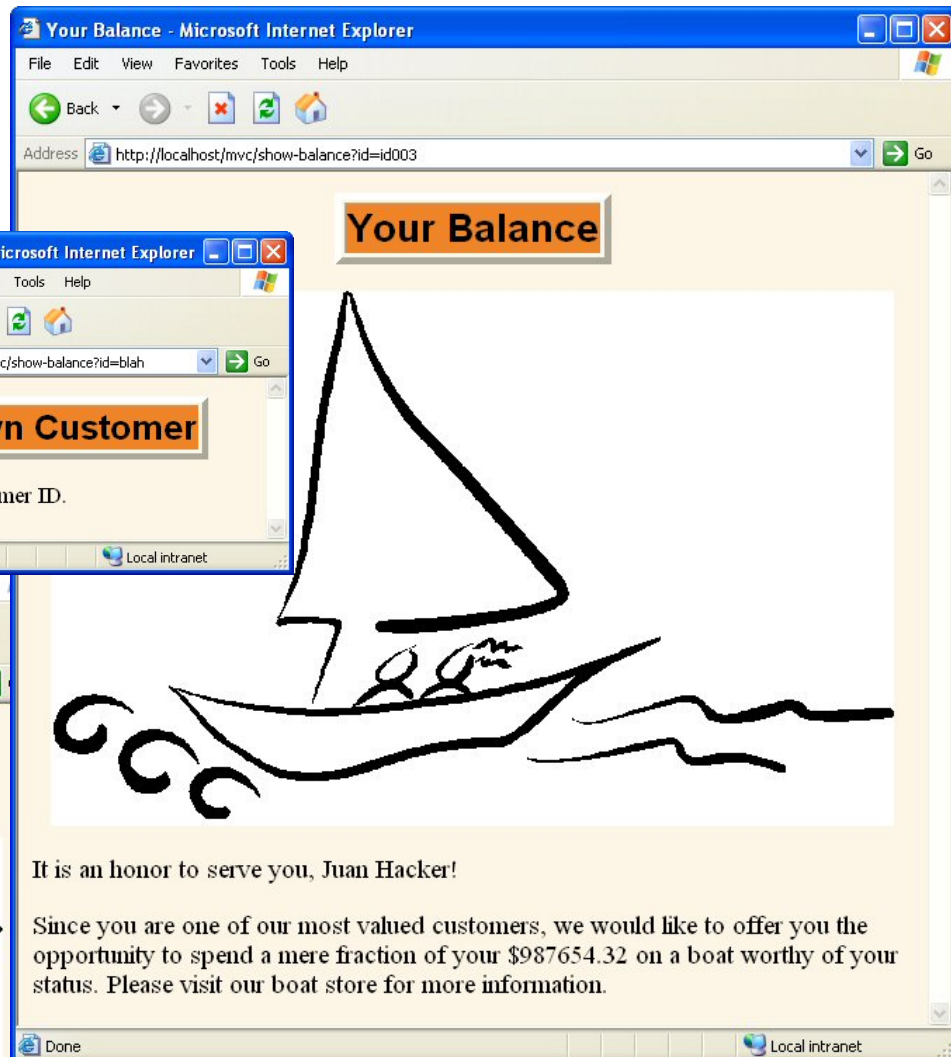
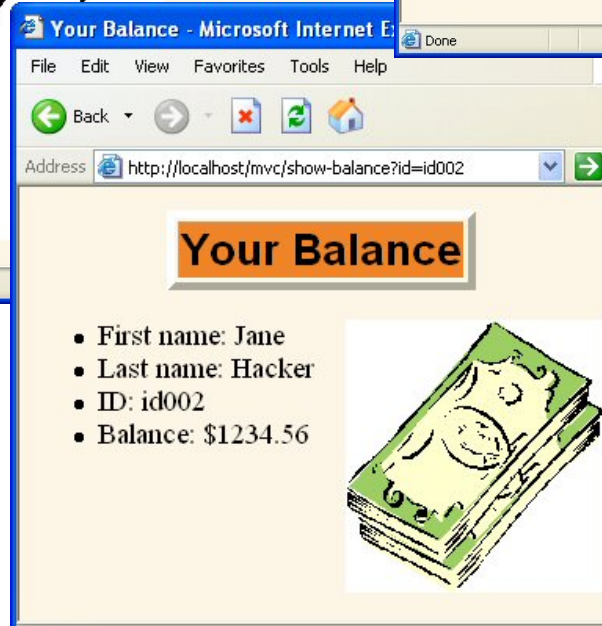
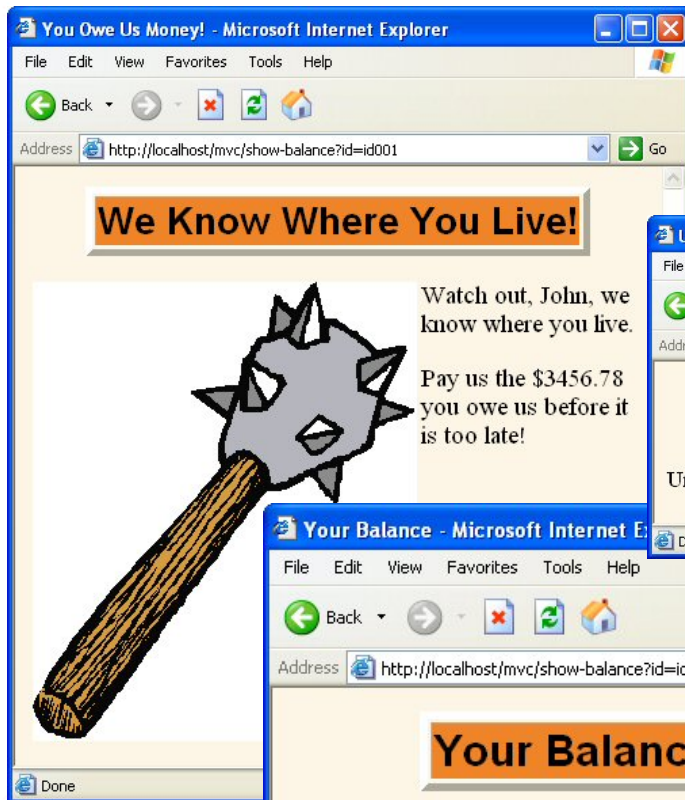


...

```
<body>
<div align="center">
<table border="5">
  <tr><th class="title">Unknown Customer</th></tr>
</table>
<p/>
<h2>No customer found with id "${badId}"</h2>
<p>Please <a href="index.html">try again</a>.</p>
</div></body></html>
```



# Bank Account Balances: Results







# Comparing Data Sharing Approaches

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Summary

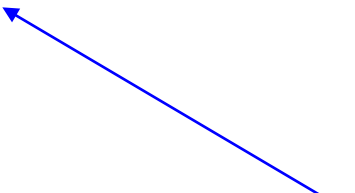
- **Request scope**
  - A new bean instance is made on every HTTP request.
  - The most common scope.
- **Session scope**
  - A bean instance could be reused if the request is from the same user in the same browser session. Useful for tracking user-specific data.
    - See session tracking lecture for details
    - Remember to make bean Serializable
- **Application (ServletContext) scope**
  - Once created, the same bean instance is used for all requests and all users.
    - Must synchronize. Very rare.

# Comparing Data-Sharing Approaches: Request

- **Goal**
  - Display a random number to the user
- **Type of sharing**
  - Each request should result in a new number, so request-based sharing is appropriate.

# Request-Based Sharing: Bean

```
public class NumberBean {  
    private final double num;  
  
    public NumberBean(double number) {  
        this.num = number;  
    }  
  
    public double getNumber() {  
        return(num) ;  
    }  
}
```



The property name in JSP will be "number". The property name is derived from the method name, not from the instance variable name. Also note the lack of a corresponding setter.

# Request-Based Sharing: Servlet

```
@WebServlet("/random-number")
public class RandomNumberServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        NumberBean bean =
            RanUtils.randomNum(request.getParameter("range"));
        request.setAttribute("randomNum", bean);
        String address = "/WEB-INF/results/random-num.jsp";
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}
```

# Request-Based Sharing: Business Logic

```
public class RanUtils {  
    public static NumberBean randomNum(String rangeString) {  
        double range;  
        try {  
            range = Double.parseDouble(rangeString);  
        } catch (Exception e) {  
            range = 10.0;  
        }  
        return (new NumberBean(Math.random() * range));  
    }  
  
    private RanUtils() {} // Uninstantiable class  
}
```

# Request-Based Sharing: Input Form

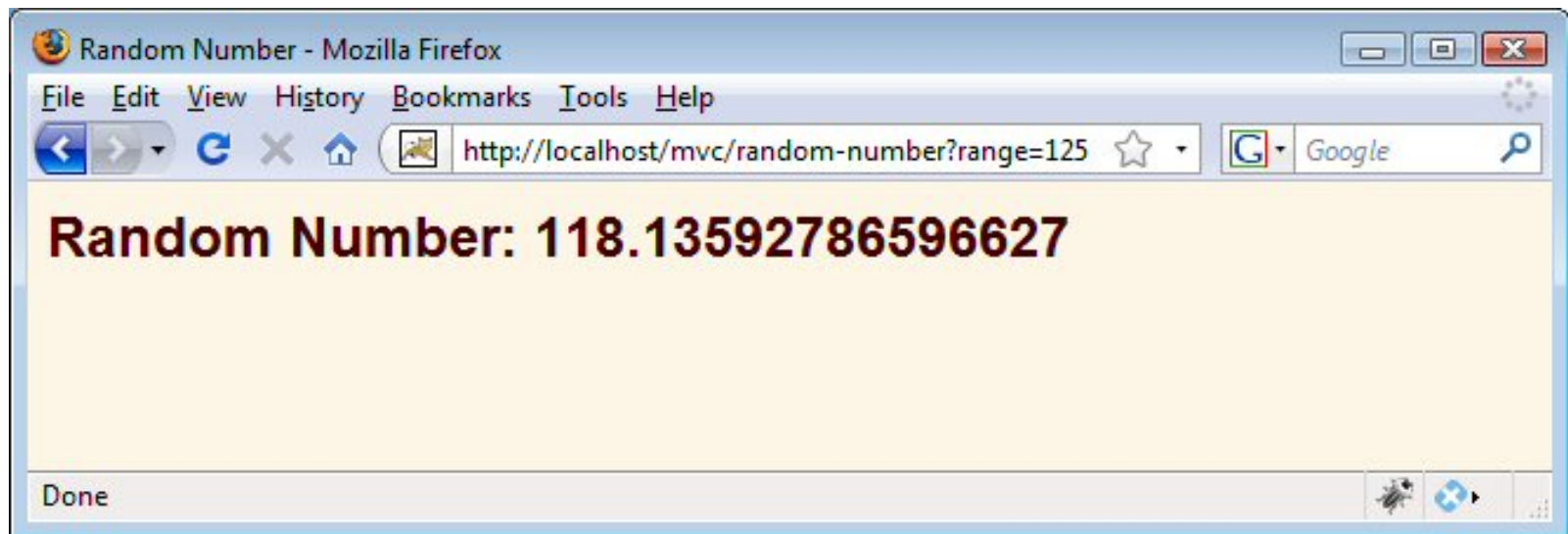
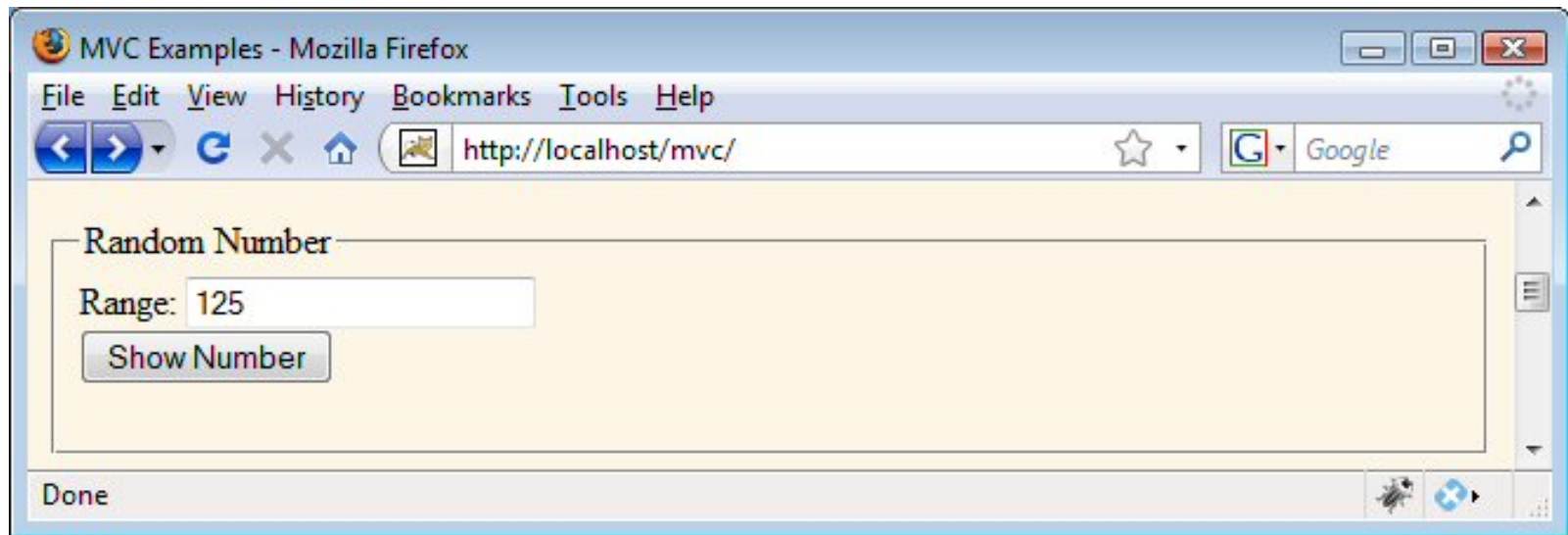
```
...  
<fieldset>  
  <legend>Random Number</legend>  
  <form action="random-number">  
    Range:  <input type="text" name="range"><br/>  
    <input type="submit" value="Show Number">  
  </form>  
</fieldset>  
...
```



# Request-Based Sharing: Results Page

```
<!DOCTYPE html>
<html>
<head>
<title>Random Number</title>
<link rel="stylesheet"
      href=" ./css/styles.css"
      type="text/css">
</head>
<body>
<h2>Random Number: ${randomNum.number}</h2>
</body></html>
```

# Request-Based Sharing: Results



# Comparing Data-Sharing Approaches: Session

- **Goal**
  - Display users' first and last names.
  - If the users fail to tell us their name, we want to use whatever name they gave us previously.
  - If the users do not explicitly specify a name and no previous name is found, a warning should be displayed.
- **Type of sharing**
  - Data is stored for each client, so session-based sharing is appropriate.

# Session-Based Sharing: Bean

```
public class NameBean implements Serializable {
    private String firstName = "Missing first name";
    private String lastName = "Missing last name";

    public String getFirstName() {
        return(firstName);
    }

    public void setFirstName(String firstName) {
        if (!isMissing(firstName)) {
            this.firstName = firstName;
        }
    }

    ... // getLastName, setLastName

    private boolean isMissing(String value) {
        return((value == null) || (value.trim().equals("")));
    }
}
```

# Session-Based Sharing: Servlet

```
@WebServlet("/register")
public class RegistrationServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();
        synchronized(session) {
            NameBean nameBean =
                (NameBean)session.getAttribute("name");
            if (nameBean == null) {
                nameBean = new NameBean();
                session.setAttribute("name", nameBean);
            }
            nameBean.setFirstName(request.getParameter("firstName"));
            nameBean.setLastName(request.getParameter("lastName"));
            String address = "/WEB-INF/mvc-sharing/ShowName.jsp";
            RequestDispatcher dispatcher =
                request.getRequestDispatcher(address);
            dispatcher.forward(request, response);
        }
    }
}
```

# Session-Based Sharing: Results Page

```
<!DOCTYPE html>
<html>
<head><title>Thanks for Registering</title>
<link rel="stylesheet"
      href=" ./css/styles.css"
      type="text/css" />
</head>
<body>
<h1>Thanks for Registering</h1>
<h2>First Name: ${name.firstName}</h2>
<h2>Last Name:  ${name.lastName}</h2>
</body></html>
```

# Session-Based Sharing: Results





# Comparing Data-Sharing Approaches: ServletContext

- **Goal**
  - Display a prime number of a specified length.
  - If the user fails to tell us the desired length, we want to use whatever prime number we most recently computed for *any* user.
- **Type of sharing**
  - Data is shared among multiple clients, so application-based sharing is appropriate.

# ServletContext-Based Sharing: Bean

```
package coreservlets;
import java.math.BigInteger;

public class PrimeBean {
    private BigInteger prime;

    public PrimeBean(String lengthString) {
        int length = 150;
        try {
            length = Integer.parseInt(lengthString);
        } catch (NumberFormatException nfe) {}
        this.prime = Primes.nextPrime(Primes.random(length));
    }

    public BigInteger getPrime() {
        return(prime);
    }
    ...
}
```

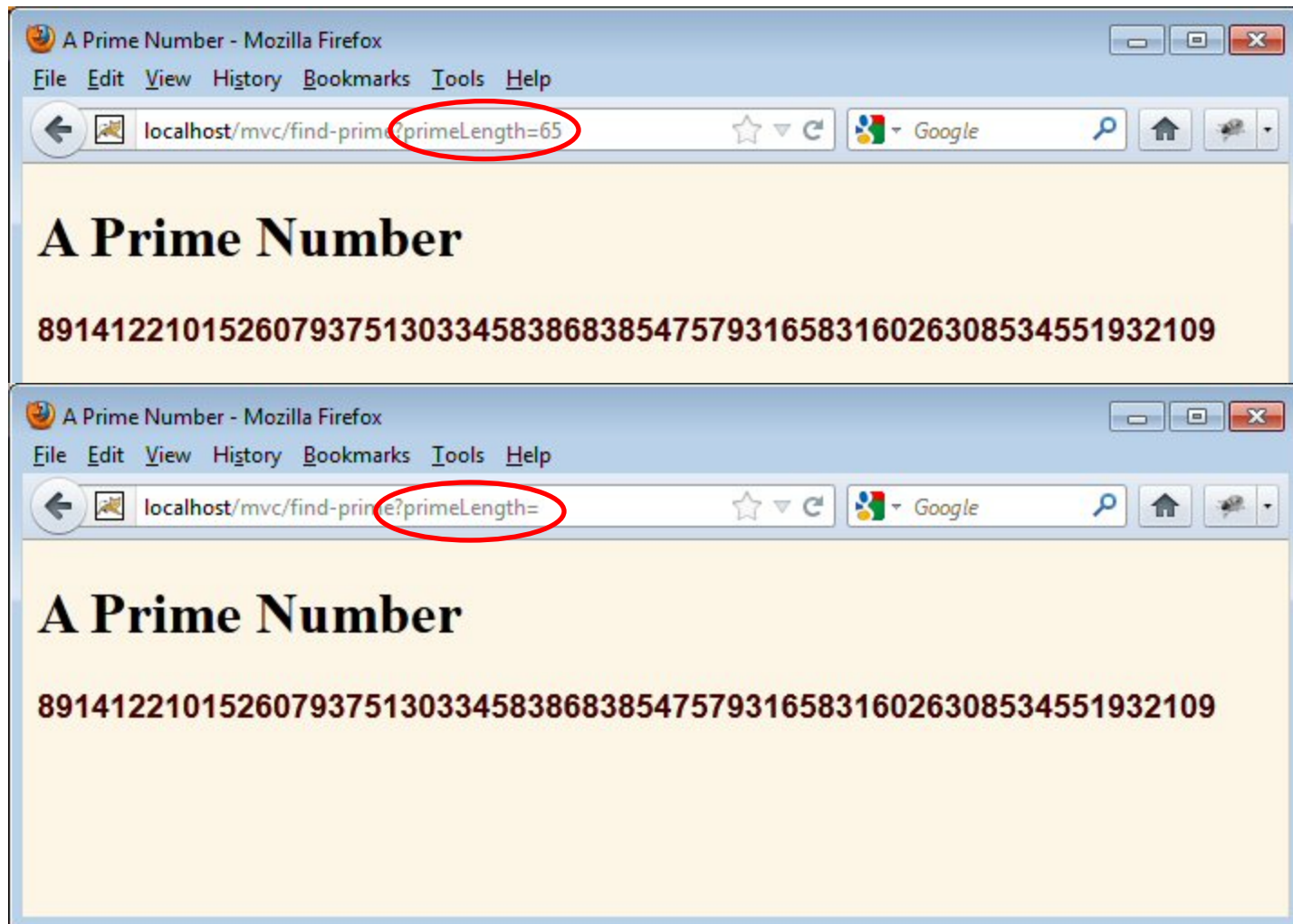
# ServletContext-Based Sharing: Servlet

```
@WebServlet("/find-prime")
public class PrimeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        String length = request.getParameter("primeLength");
        ServletContext context = getServletContext();
        synchronized(this) {
            if ((context.getAttribute("primeBean") == null) ||
                (!isMissing(length))) {
                PrimeBean primeBean = new PrimeBean(length);
                context.setAttribute("primeBean", primeBean);
            }
            String address = "/WEB-INF/mvc-sharing/ShowPrime.jsp";
            RequestDispatcher dispatcher =
                request.getRequestDispatcher(address);
            dispatcher.forward(request, response);
        }
    }
    ... // Definition of isMissing: null or empty string
}
```

# ServletContext-Based Sharing: Results Page

```
<!DOCTYPE html>
<html>
<head><title>A Prime Number</title>
<link rel="stylesheet"
      href="/css/styles.css"
      type="text/css"/>
</head>
<body>
<h1>A Prime Number</h1>
<h2>${primeBean.prime}</h2>
</body></html>
```

# ServletContext-Based Sharing: Results





# Forwarding and Including

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Forwarding from JSP Pages

```
<% String destination;  
    if (Math.random() > 0.5) {  
        destination = "/examples/page1.jsp";  
    } else {  
        destination = "/examples/page2.jsp";  
    }  
%>  
<jsp:forward page="<%= destination %>" />
```

- **Legal, but bad idea**
  - Business and control logic belongs in servlets
  - Keep JSP focused on presentation



# Including Pages Instead of Forwarding to Them

- **With the `forward` method**
  - New page generates all of the output
  - Original page or other pages *cannot* generate any output
- **With the `include` method**
  - Output can be generated by multiple pages
  - Original page *can* generate output before and after the included page
  - Original servlet does not see the output of the included page (for this, see later topic on servlet/JSP filters)
  - Applications
    - Portal-like applications (see first example)
    - Setting content type for output (see second example)

# Using RequestDispatcher.include: portals

```
response.setContentType("text/html");
String firstTable, secondTable, thirdTable;
if (someCondition) {
    firstTable = "/WEB-INF/results/sports-scores.jsp";
    secondTable = "/WEB-INF/results/stock-prices.jsp";
    thirdTable = "/WEB-INF/results/weather.jsp";
} else if (...) { ... }
RequestDispatcher dispatcher =
    request.getRequestDispatcher("/WEB-INF/results/header.jsp");
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher(firstTable);
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher(secondTable);
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher(thirdTable);
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher("/WEB-INF/results/footer.jsp");
dispatcher.include(request, response);
```

# Using `RequestDispatcher.include`: Setting Content-Type of Output

```
// From Ajax example
public void doGet(...) ... {
    ...
    if ("xml".equals(format)) {
        response.setContentType("text/xml");
        outputPage = "/WEB-INF/results/cities-xml.jsp";
    } else if ("json".equals(format)) {
        response.setContentType("application/json");
        outputPage = "/WEB-INF/results/cities-json.jsp";
    } else {
        response.setContentType("text/plain");
        outputPage = "/WEB-INF/results/cities-string.jsp";
    }
    RequestDispatcher dispatcher =
        request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
}
```

# cities-xml.jsp

```
<?xml version="1.0" encoding="UTF-8"?>
<cities>
  <city>
    <name>${cities[0].name}</name>
    <time>${cities[0].shortTime}</time>
    <population>${cities[0].population}</population>
  </city>
  ...
</cities>
```

- **Notes**

- Because I use .jsp (not .jspx) and classic JSP syntax, the default content-type is text/html
- I could use `<%@ page contentType="text/xml" %>` here, but it is more convenient to do it in calling servlet and keep this page simple and focused on presentation



# Wrap-Up

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Summary

- **Use MVC (Model 2) approach when:**
  - One submission will result in more than one basic look
  - Several pages have substantial common processing
  - Your application is moderately complex
- **Approach**
  - A servlet answers the original request
  - Servlet calls business logic and stores results in beans
    - Beans stored in `HttpServletRequest`, `HttpSession`, or `ServletContext`
  - Servlet invokes JSP page via `RequestDispatcher.forward`
  - JSP page reads data from beans
    - Most modern servers: `${beanName.propertyName}`
    - JSP 1.2 servers: `jsp:useBean` with appropriate scope (request, session, or application) plus `jsp:getProperty`





# Questions?

JSF 2, PrimeFaces, Java 7, Ajax, jQuery, Hadoop, RESTful Web Services, Android, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.