Lecture 14
Integrating Servlets and JSP: The Model View Controller (MVC) Architecture

Lecture Agenda

1. Understanding the benefits of MVC
2. Using RequestDispatcher to implement MVC
3. Forwarding requests from servlets to JSP pages
4. Handling relative URLs
5. Choosing among different display options
6. Comparing data-sharing strategies

MVC Motivation

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 and JSP?

Introduction

- Typical picture: use JSP to make it easier to develop and maintain HTML
  - 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.
- Be aware ….
  - For complex processing, starting with JSP is awkward
  - Despite the ease of separating the real code into separate classes, beans and custom tags, the assumptions behind JSP is that a ***single*** page gives a ***single*** basic look.

Handling a Single Request

Possibilities

1. **Servlet Only**: Works well when …
   - Output is a binary type (ex. an image)
   - There is no output (ex. you doing forwarding or redirection).
   - Format/layout of page is highly variable (ex. portal)
2. **JSP only**: Works well when …
   - Output is mostly character data (ex. HTML)
   - Format/layout is mostly **fixed**
3. **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 Web development and the Business Logic.
   - You perform complicated data processing, but have a relatively **fixed** layout.

MVC Misconceptions

Basic Misconceptions

- An elaborate framework is necessary
  - Frameworks are often useful
    - JSF (JavaServer Faces)
    - Struts
    - Spring MVC
  - They are not required to implement MVC
    - Implementing MVC within the built-in RequestDispatcher works very well for most simple and even moderately complex applications.

Jab

- MVC totally changes your system design
  - You can use MVC for individual requests
  - Think of it as the **MVC approach**, <u>not</u> the **MVC architecture**
  - **MVC approach** is often referred to as **Model 2 approach**

MVC-Based Alternative to Servlet and JSP

JSF 2 – MVC Based Framework

- Servlet and JSP
  - Well-established standard
  - Used by google, ebay, Walmart, and thousands of other popular sites
  - Relatively low-level by today's standards
- JSF (JavaServer Faces) Version 2
  - Presently an official part of Java EE
  - Higher-level features: integrated Ajax support, field validation, page templating, rich third-party component libraries designed around MVC approach

Review: Java Beans

Review: Beans

- Java classes that must follow certain conventions
  1. Must have zero-argument (empty) constructor
     - You can satisfy this requirement either by explicitly defining such a constructor or omitting all constructors.
     - In this version of MVC, it is not required to have zero argument if you only instantiate from Java code.
  2. Should have **no** public instance variables (fields)
     - Use accessor methods instead of accessing members directly.
  3. Data members 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: Example

Method Name	Property Name	Example JSP Usage
getFirstName setFirstName	firstName	**<jsp:getProperty** … **property**=“firstName” /> **<jsp:setProperty** … **property**=“firstName” **value**=“…”/> **${**customer.firstName**}**
isExecutive setExecutive (boolean property)	Executive	**<jsp:getProperty** … **property**=“executive” /> **<jsp:setProperty** … **property**=“executive” **value**=“…”/> **${**customer.executive**}**
getExecutive setExecutive (boolean property)	Executive	**<jsp:getProperty** … **property**=“executive” /> **<jsp:setProperty** … **property**=“executive” **value**=“…”/> **${**customer.executive**}**
getZIP setZIP	ZIP	**<jsp:getProperty** … property=“ZIP”/> **<jsp:setProperty** … property=“ZIP” value=“…”/> **${**address.ZIP**}**

Example: StringBean

```
public class StringBean {
  private String message = "No message specified";

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

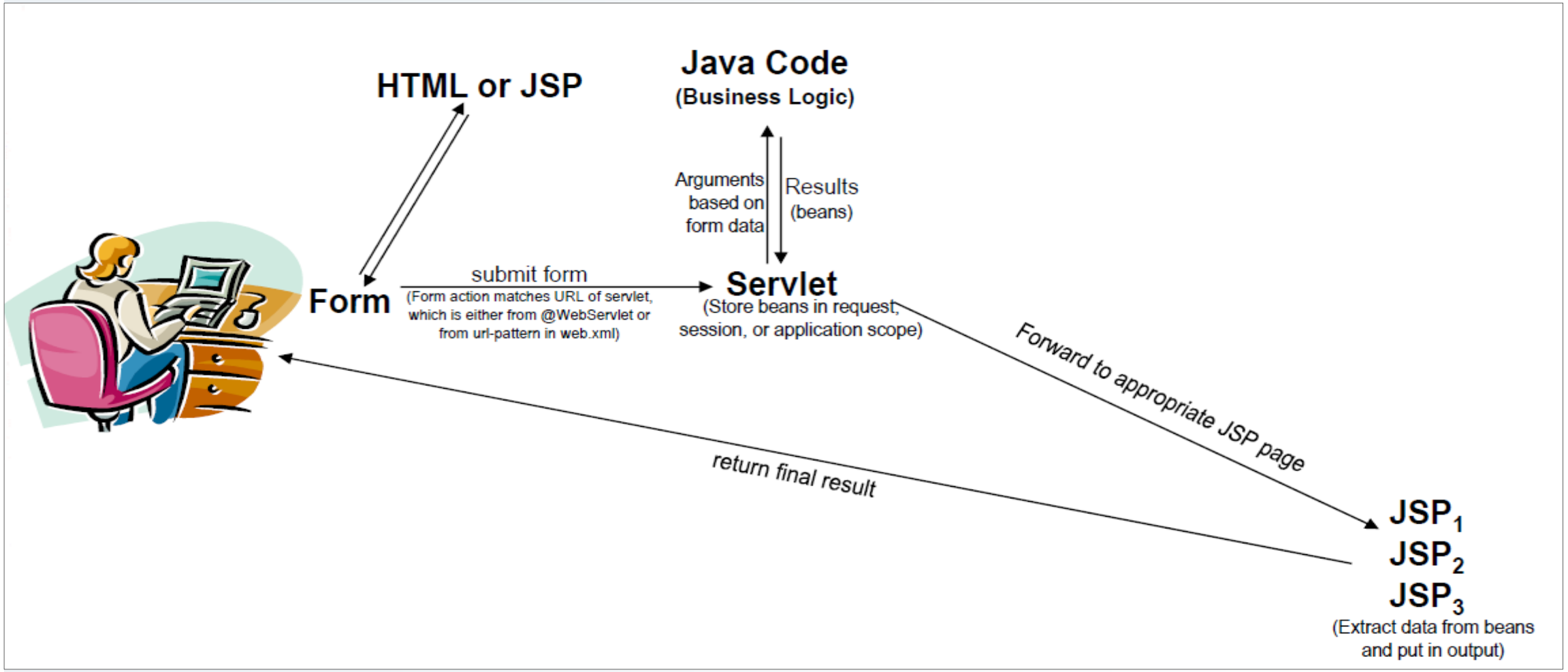
private member

accessor methods

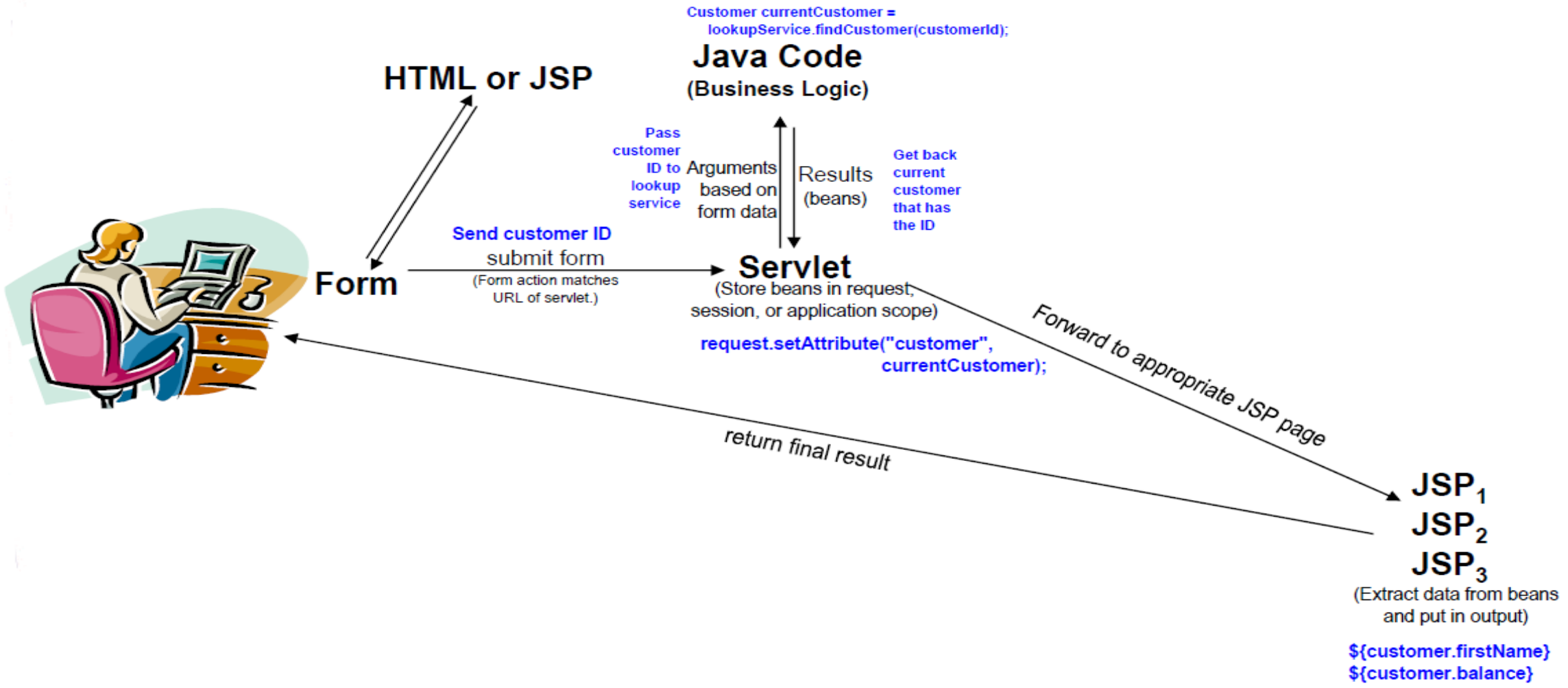
- Beans installed in normal Java directory
  - **Eclipse**: src/folderMatchingPackageName
  - **Deployment**: WEB-INF/classes/folderMatchingPackageName
- Beans **must** always be in packages!!!

Basic MVC Design

MVC Flow of Control



MVC Flow of Control (Annotated Example)



Parts in blue are examples for a banking application.

Implementing MVC with RequestDispatcher

1. Identify and define bean(s) to represent result data
   - ordinary Java classes with at least **one** get (accessor) method
2. Use servlet to handle requests
   - Servlet reads request parameters, checks for **missing and malformed data**, calls necessary 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 ServletContext.
   - The servlet calls **setAttribute** on the request, session, or servlet context objects to store a reference to the bean(s) 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 jsp page.
6. Extract the data from the beans
   - The JSP page uses **${nameFromServlet.property}** to output bean properties.
   - The JSP page does not create or modify a bean, it merely extracts and displays data that the servlet created.

Example: Request Forwarding

```
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);
```

Two step process

jsp:useBean in MVC vs. Standalone JSP Pages

- The JSP page should **not** create 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
  - Use jsp:getProperty **not** jsp:setProperty.

Scopes:
request, session and application (ServletContext)

Scopes

- Idea
  - A “scope” is a place that the bean is stored.
  - This controls where and for how long the bean is visible.
- Three choices
  1. 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.
  2. Session
     - Data stored in the session scope 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.
  3. Application (Servlet Context)
     - Data stored in the servlet context is visible to all users and all pages in the application. Rarely used.

Scopes: Request-Based Data Sharing

Request-Based Data Sharing

- Servlet Example Code:

LookupSevrice performs backend retrieval from some persistence

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

- **JSP 2.0**

Name chosen by the servlet.

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

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

- **JSP 1.2**

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

To retrieve the key value on the JSP side

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"/>
```

Scopes: Session-Based Data Sharing

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 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 with sendRedirect(…) (only one with RequestDispatcher.forward(…)).
- Advantages 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 the user can visit JSP page without going through servlet first, bean data might not be available (so JSP code needs added logic to detect this).

Scopes: ServletContext (Application)-Based Data Sharing

Session-Based Data Sharing

note we synchronize in application scope. Why?

- **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" />
```

Relative URLs in JSP Pages

- **Note:**
  - Forwarding with a request dispatcher is transparent to the client.
  - Original URL (the URL in the form action) is the **only** URL browser knows about.

Example: Bank Balance Lookup

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 balance to decide on the appropriate result page to forward to.
- **JSP pages to display results**
  - Negative balance: warning page
  - Regular balance: standard page
  - High balance: page with advertisements
  - 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);
```

findCustomer instantiates customer object and returns

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 JSP) the requirement for a zero-argument constructor is eliminated.

Since bean state is only set 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

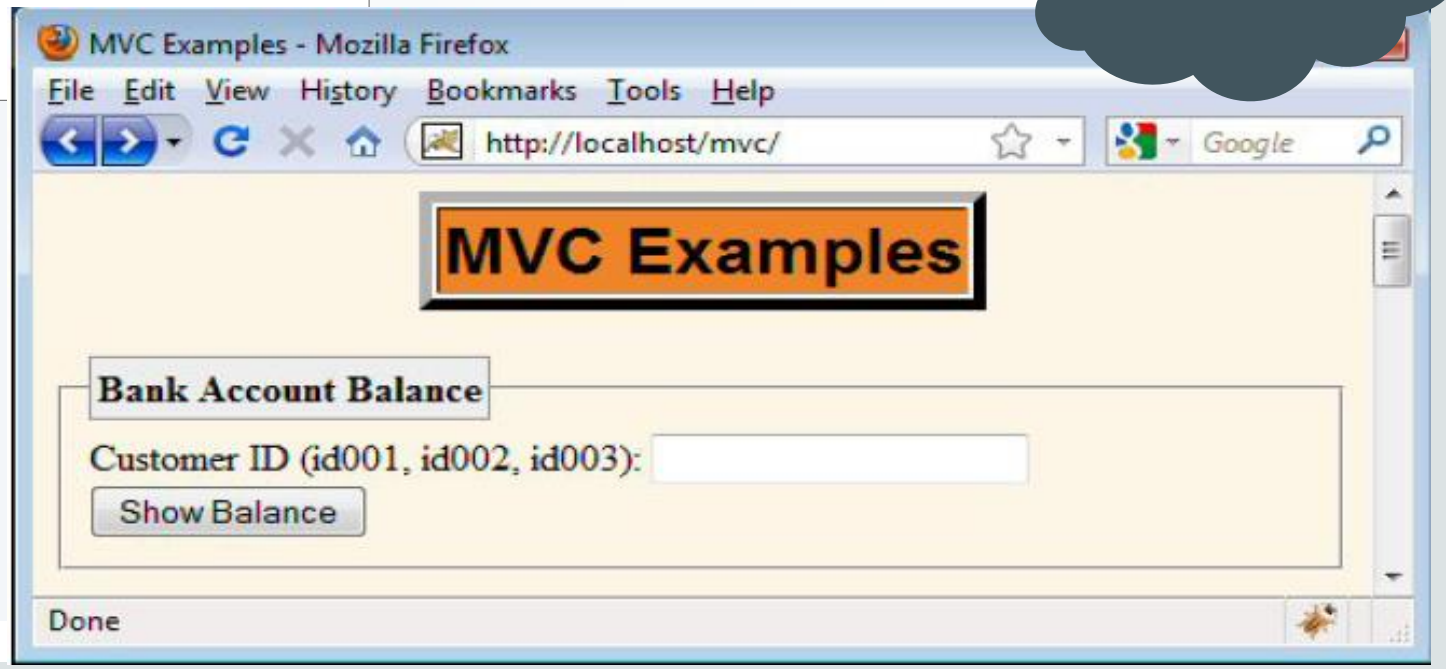
```
public class CustomerSimpleMap
        implements CustomerLookupService {
  private Map<String,Customer> customers;

  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>
```

source code for form

form screenshot



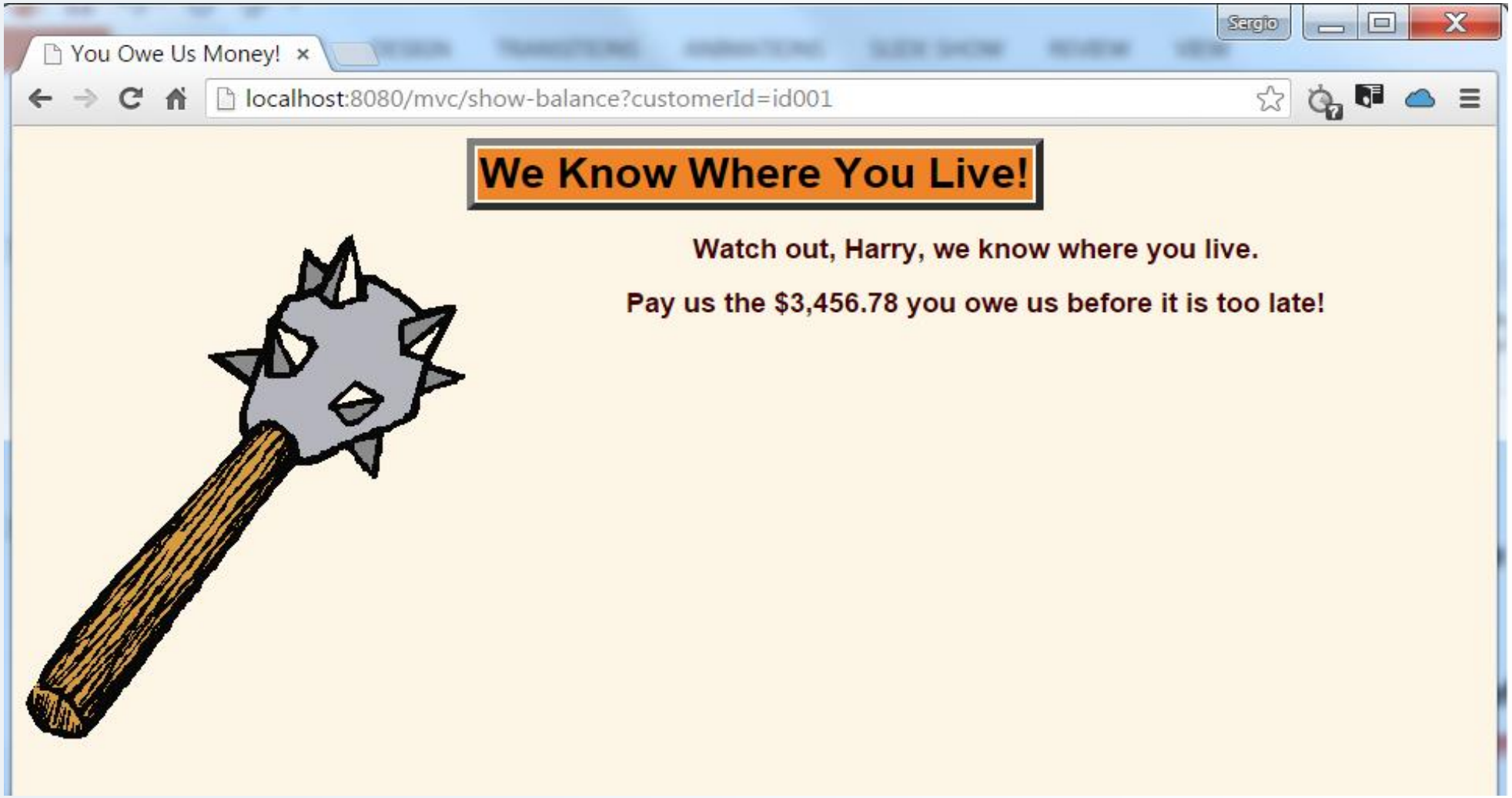
Bank Account Balances: Negative balance (JSP 2.0)

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

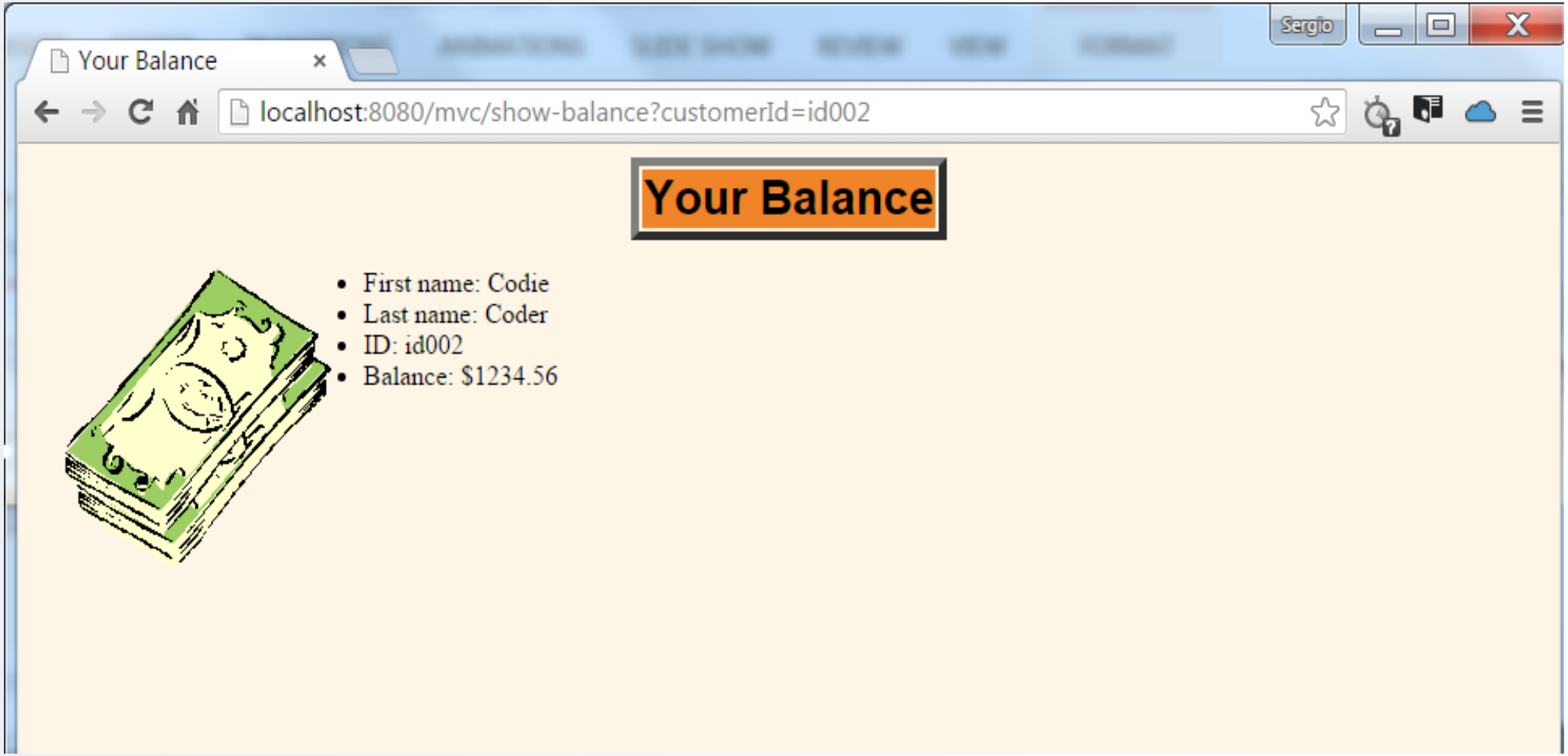
Bank Account Balances: Negative balance Result



Bank Account Balances: Normal balance (JSP 2.0)

```
…<body>
<table border="5" align="center">
<tr><th class="title">Your Balance</th></tr>
</table>
<p/>
<img src="./images/money.gif" align="left" hspace="20"/>

<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>
```

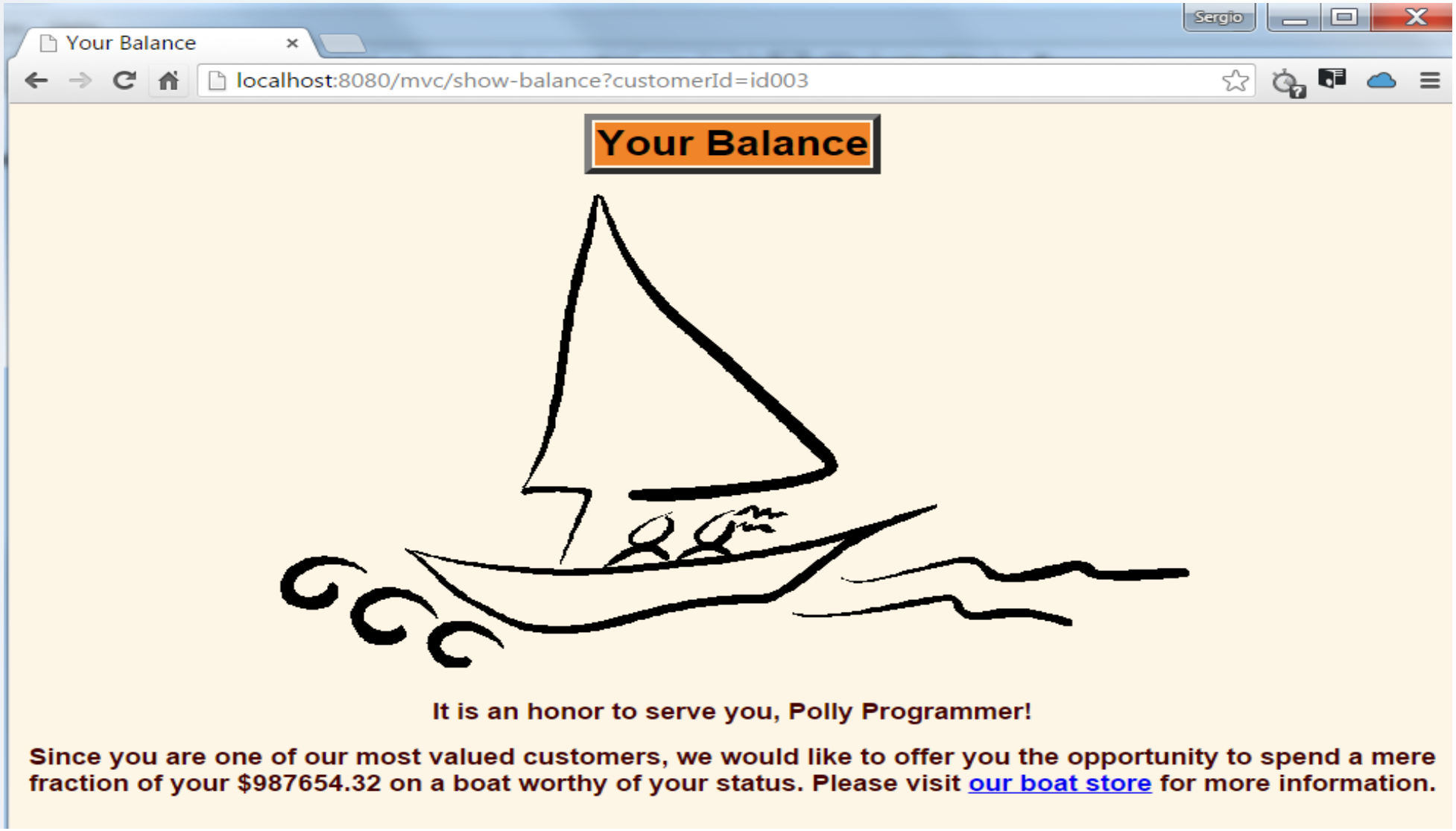
Bank Account Balances: Normal balance Result



Bank Account Balances: High balance (JSP 2.0)

```
<body>
<div align="center">
…
<img src="./images/sailing.gif"/><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>
```

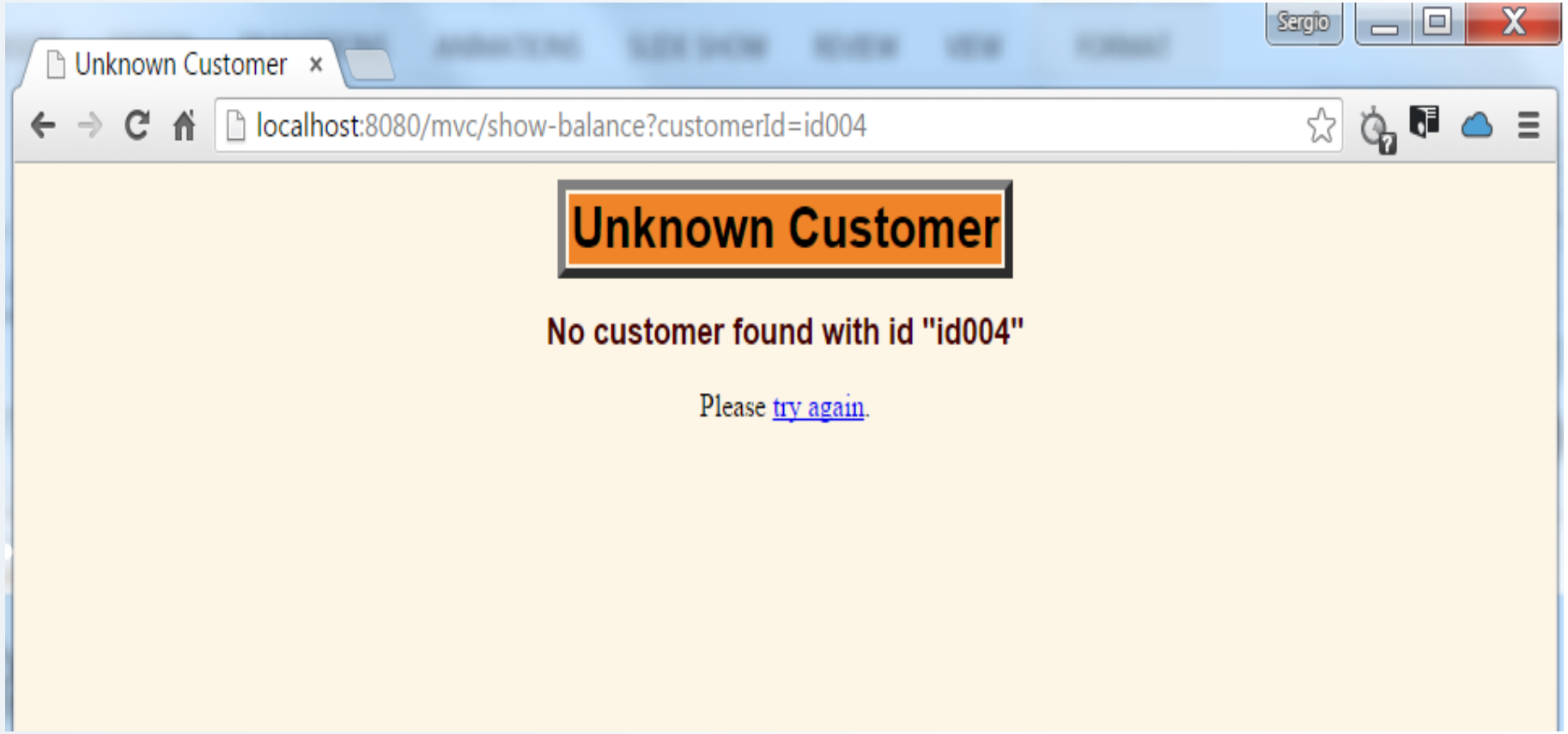
Bank Account Balances: High balance Result



Bank Account Balances: Unknown Customer (JSP 2.0)

```
<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: Unknown Customer Result



Comparing Data-Sharing Approaches

Review

Scope Review

- Request Scope
  - A 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.
  - Should make bean Serializable
- Application(ServletContext) scope
  - Once created, the same bean instance is used for **all** requests and all users.

Comparing Data-Sharing Approaches: Request Example

Comparing Data-Sharing Approaches

Request

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

Request-Based Sharing: Bean

Request

```
public class NumberBean {
  private final double num;

  public NumberBean(double number) {
    this.num = number;
  }

  public double getNumber() {
    return(num);
  }
}
```

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

Servlet Code

```
@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

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

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>
...
```

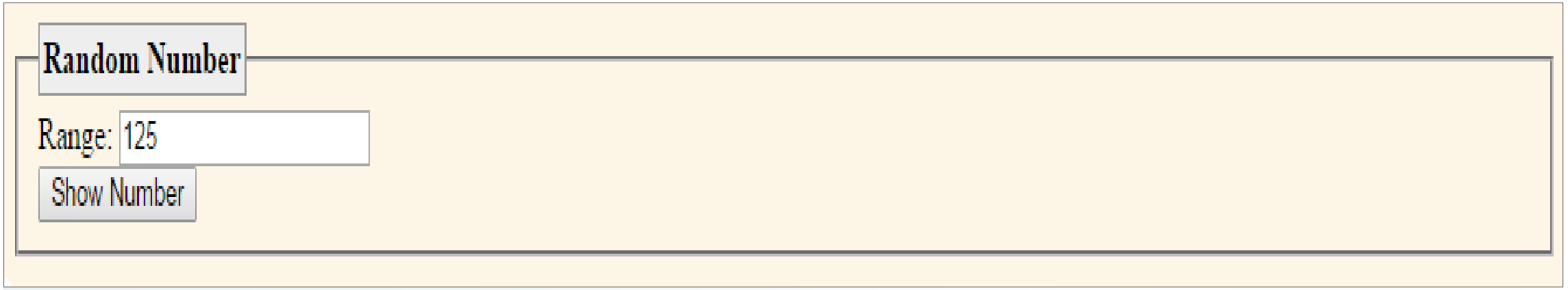
Request-Based Sharing: Results Page

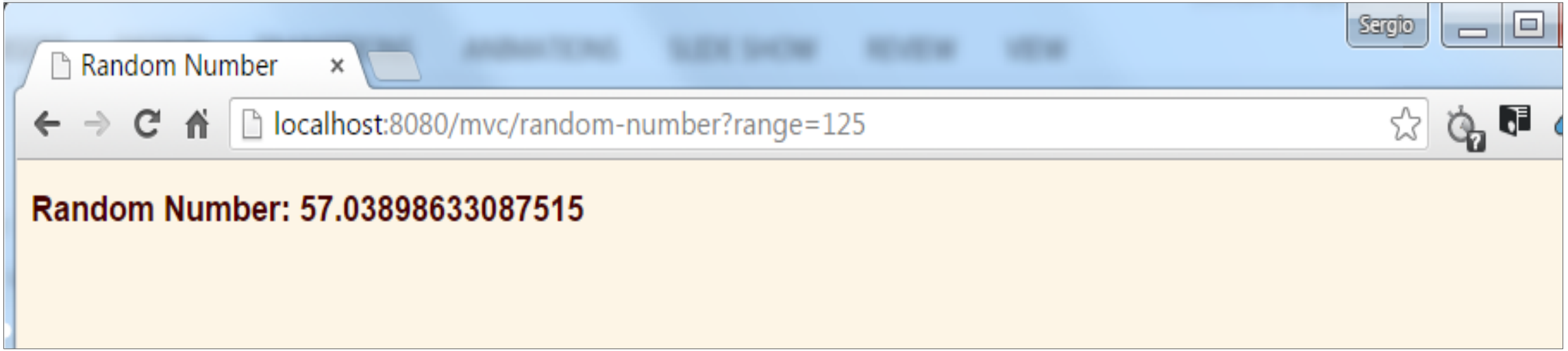
Results Page

```
<!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

Results





Comparing Data-Sharing Approaches: Session Example

Comparing Data-Sharing Approaches

Session

- Goal
  - Display users' first and last name.
  - If the user fails to provide a name, use the name they provided 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

Session

```
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("")));
  }
}
```

notice serializable

Session-Based Sharing: Servlet

Servlet Code

Synchronization block

```
@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);
    }
  }
}
```

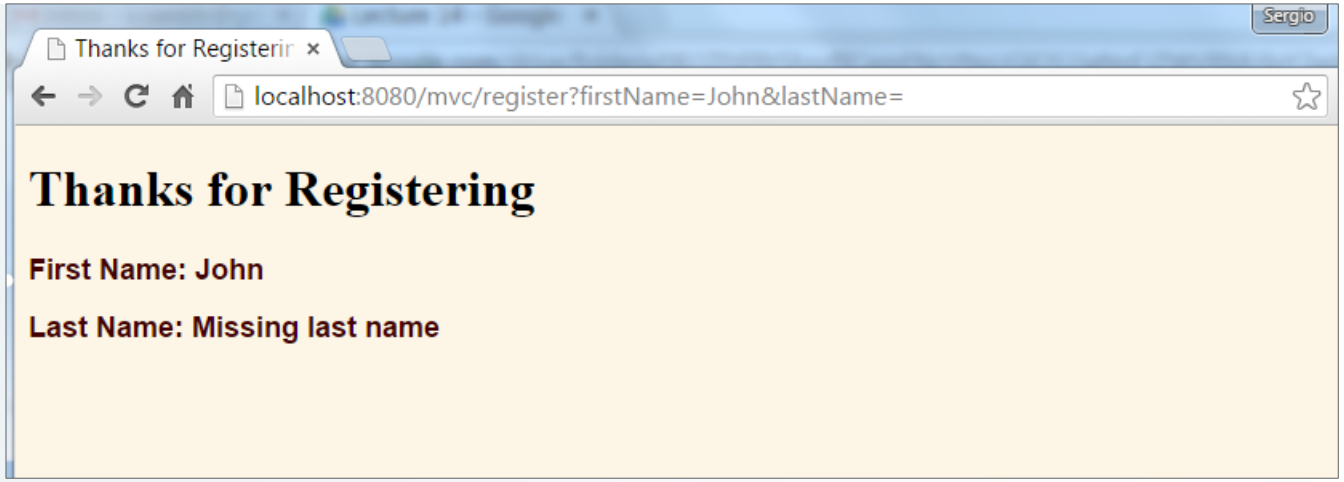
Session-Based Sharing: Results Page

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

Results Screenshot



Comparing Data-Sharing Approaches: ServletContext Example

Comparing Data-Sharing Approaches

ServletContext (Application)

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

ServletContext-Based Sharing: Bean

Session

```
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

Servlet Code

```
@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
}
```

Synchronization block

If data is missing or null, use prime from previous data stored in application scope

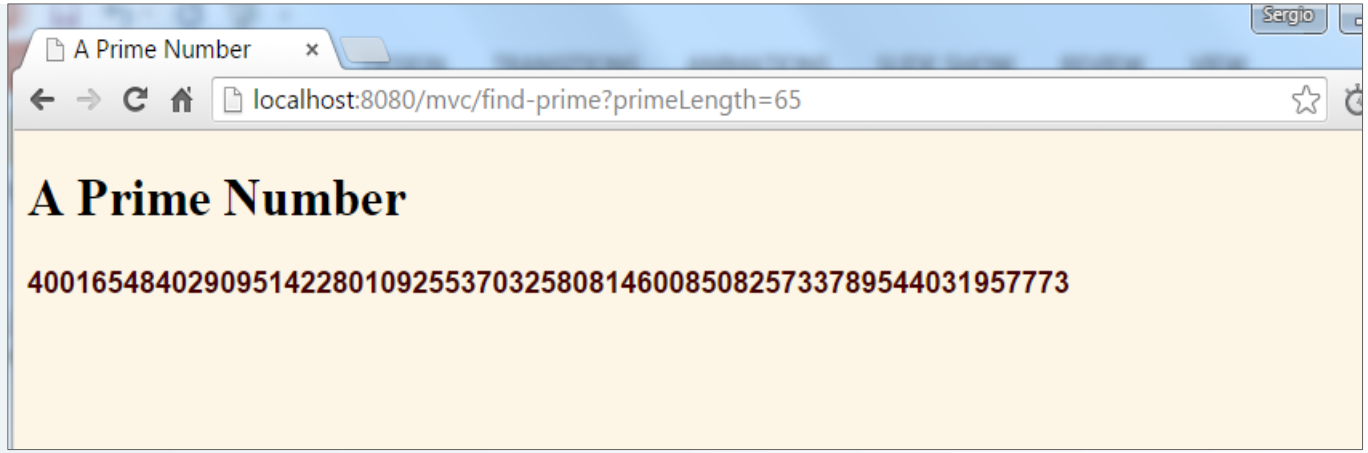
ServletContext-Based Sharing: Results Page

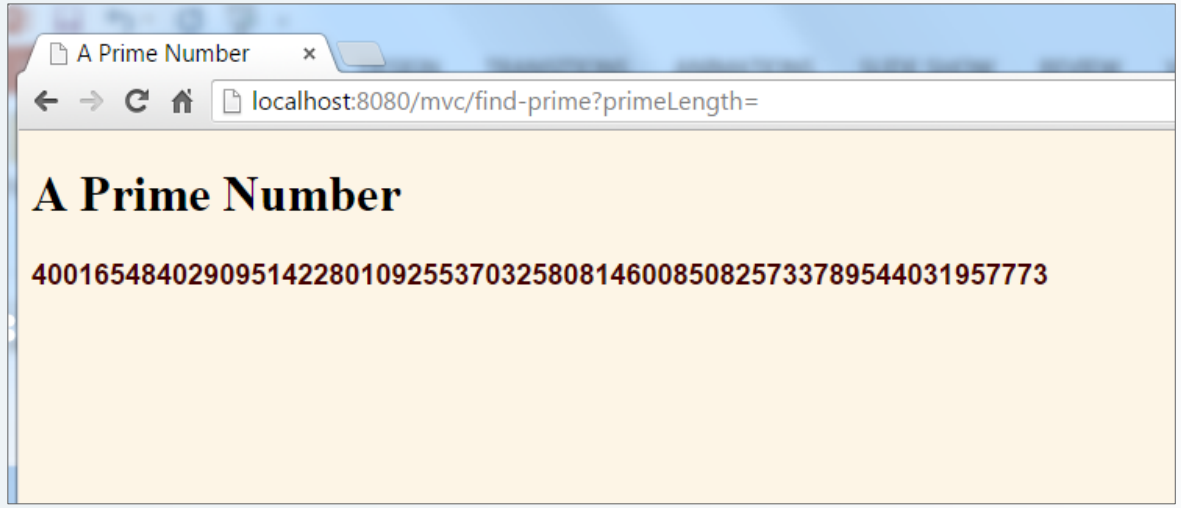
Results Page

```
<!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

Results Screenshot





Forwarding and Including

Forwarding

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 Practice
  - Business and control logic belongs in **servlets**
  - Keep **JSP** focused on presentation.

Including Pages vs Forwarding Pages

Include pages instead of **Forwarding** 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
  - Applications
    - Portal-Like applications
    - Setting content-type for the output

Using RequestDispatcher.include()

Portals Example

portal example

```
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);
```

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 the 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 (JSP 2.0 +): **${**beanName.propertyName**}**
    - Older Serves (JSP 1.2): jsp:useBean with appropriate scope (request, session, application) plus jsp:getProperty

Questions?