JSF 2.0: Introduction and Overview

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/jsf2/

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

For live training on JSF 1.x or 2.0, please see courses at http://courses.coreservlets.com/.

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 <u>your</u> organization.

- Courses developed and taught by Marty Hall
  - Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF 1.x & 2.0, Ajax, GWT, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, Google Closure) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Different views of JSF**
- **Pros and cons of JSF**
  - Vs. standard servlet and JSP technology
  - Vs. Apache Struts
  - Vs. other Ajax approaches
- **New features in JSF 2.0**
  - Vs. JSF 1.*x*

Overview

What is JSF?

- **A set of Web-based GUI controls and handlers?**
  - JSF provides many prebuilt HTML-oriented GUI controls, along with code to handle their events.
- **A device-independent GUI control framework?**
  - JSF can be used to generate graphics in formats other than HTML, using protocols other than HTTP.
- **An MVC-based Web app framework?**
  - Like Apache Struts, JSF can be viewed as an MVC framework for building HTML forms, validating their values, invoking business logic, and displaying results.
- **An Ajax library?**
  - JSF 2.0 provides very easy-to-use Ajax support. So, JSF 2.0 can be viewed as an alternative to jQuery or GWT.
- **But which is the proper way to view JSF?**
  - The answer depends on what you are going to use it for, but 1 & 3 are the most common wasy of looking at JSF.

JSF vs. Servlets/JSP

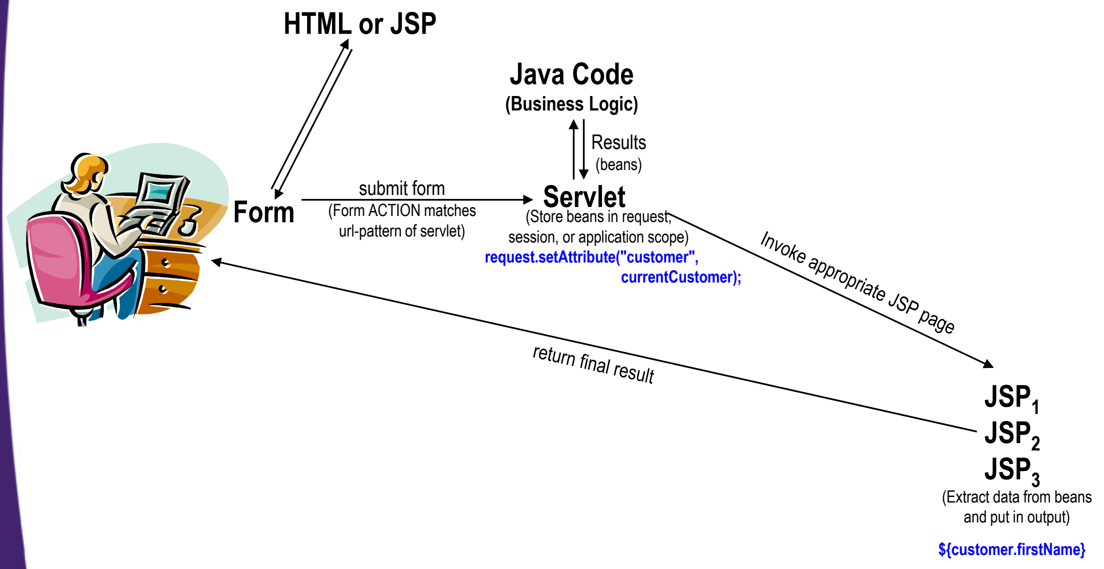
The MVC Architecture

- **Bad JSP design**
  - Many books or articles on Java-based Web frameworks start with this example:

```
<h1>Tiny bit of HTML</h1>
<% Java
     Java
     Java
     More Java %>
<h1>Tiny bit of HTML</h1>
```

  - Then, they redo the example using the framework and comment on how much better it is.
    - This proves nothing
- **MVC**
  - A fair comparison is to look at the use of the framework vs. the use of MVC with servlets and JSP

A Quick Review of MVC


HTML or JSP
Java Code
(Business Logic)
Results
(beans)
Form
submit form
(Form ACTION matches url-pattern of servlet)
Servlet
(Store beans in request, session, or application scope)
Invoke appropriate JSP page
return final result
JSP1
JSP2
JSP3
(Extract data from beans and put in output)

```
request.setAttribute("customer",
                     currentCustomer);
```

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

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

```
public class ShowBalance extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    BankCustomer currentCustomer =
      BankCustomerLookup.getCustomer
                     (request.getParameter("id"));
    request.setAttribute("customer", currentCustomer);
    String address;
    if (currentCustomer == null) {
      address =
        "/WEB-INF/bank-account/UnknownCustomer.jsp";
    } else if (currentCustomer.getBalance() < 0) {
      address =
        "/WEB-INF/bank-account/NegativeBalance.jsp";
    } ...
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
```

Bank Account Balances: Bean

```
public class BankCustomer {
  private final String id, firstName, lastName;
  private final double balance;

  public BankCustomer(String id,
                      String firstName,
                      String lastName,
                      double balance) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.balance = balance;
  }

  // Getters for four instance variables. No setters.

  public double getBalanceNoSign() {
    return(Math.abs(balance));
  }
}
```

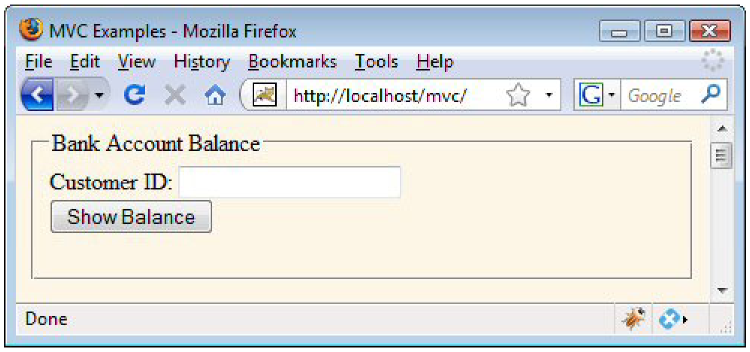
Bank Account Balances: Business Logic

```
public class BankCustomerLookup {
  private static Map<String,BankCustomer> customers;

  static {
    // Populate Map with some sample customers
  }

  …

  public static BankCustomer getCustomer(String id) {
    return(customers.get(id));
  }
}
```

Bank Account Balances: Input Form

```
…
<fieldset>
  <legend>Bank Account Balance</legend>
  <form action="./show-balance">
    Customer ID: <input type="text" name="id"><br>
    <input type="submit" value="Show Balance">
  </form>
</fieldset>
…
```

For the servlet, use the address http://*host*/*appName*/show-balance that is set via url-pattern in web.xml


MVC Examples - Mozilla Firefox

http://localhost/mvc/

Bank Account Balance

Customer ID:

Show Balance

Done

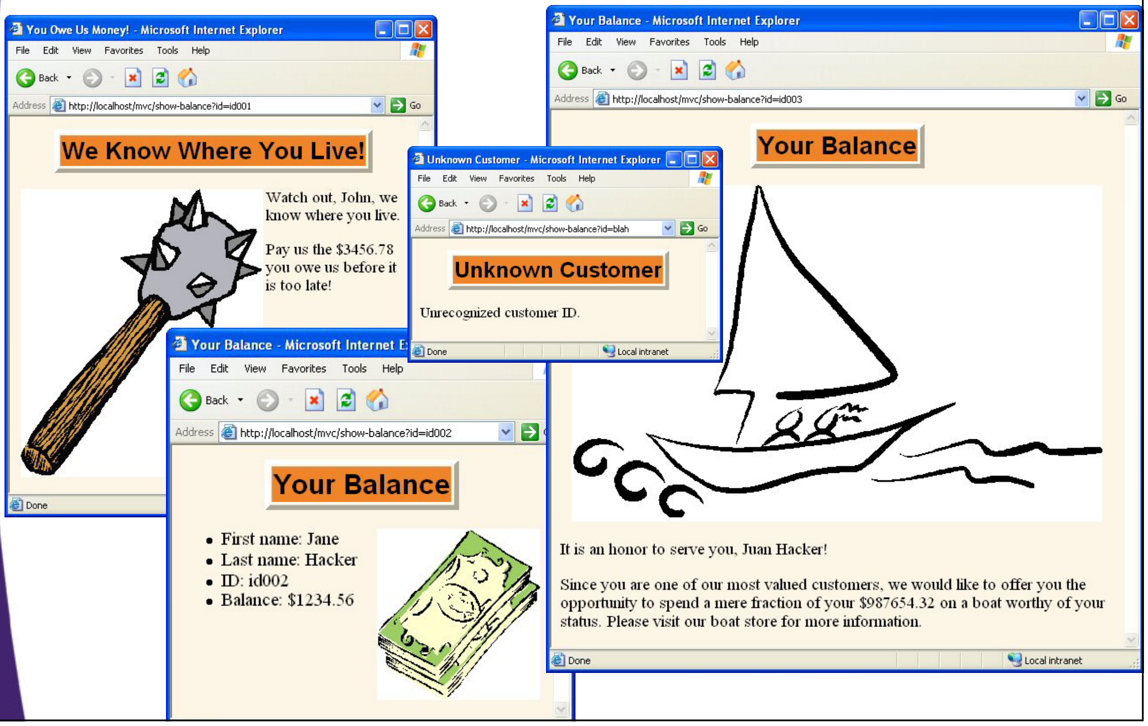
Bank Account Balances: JSP Code (Negative Balance)

```
…
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      We Know Where You Live!</TABLE>
<P>
<IMG SRC="/bank-support/Club.gif" ALIGN="LEFT">
Watch out, ${customer.firstName},
we know where you live.
<P>
Pay us the $${customer.balanceNoSign}
you owe us before it is too late!
</BODY></HTML>
```

Bank Account Balances: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" ...>
  <!-- Use the URL http://host/app/show-balance instead of
       http://host/app/servlet/coreservlets.ShowBalance -->
   <servlet>
    <servlet-name>ShowBalance</servlet-name>
    <servlet-class>coreservlets.ShowBalance</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ShowBalance</servlet-name>
    <url-pattern>/show-balance</url-pattern>
  </servlet-mapping>
...
</web-app>
```

Bank Account Balances: Results


You Owe Us Money! - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=id001
Unknown Customer - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=blah
Your Balance - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=id002
Your Balance - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=id003

We Know Where You Live!

Watch out, John, we know where you live.

Pay us the $3456.78 you owe us before it is too late!

Unknown Customer

Unrecognized customer ID.

Your Balance

- First name: Jane
- Last name: Hacker
- ID: id002
- Balance: $1234.56

Your Balance

It is an honor to serve you, Juan Hacker!

Since you are one of our most valued customers, we would like to offer you the opportunity to spend a mere fraction of your $987654.32 on a boat worthy of your status. Please visit our boat store for more information.

Advantages of JSF (vs. MVC Using RequestDispatcher)

- **Custom GUI controls**
  - JSF provides a set of APIs and associated custom tags to create HTML forms that have complex interfaces
    - There are many extra-rich third-party JSF libraries
- **Event handling**
  - JSF makes it easy to designate Java code that is invoked when forms are submitted. The code can respond to particular buttons, changes in particular values, certain user selections, and so on.
- **Managed beans**
  - In JSP, you can use property="*" with jsp:setProperty to automatically populate a bean based on request parameters. JSF extends this capability and adds in several utilities, all of which serve to greatly simplify request param processing.
- **Integrated Ajax support**
  - You can use jQuery, Dojo, or Ext-JS with servlets and JSP. However, JSF lets you use Ajax without explicit JavaScript programming and with very simple tags. Also, the Ajax calls know about the server-side business logic.

Advantages of JSF (vs. Standard MVC), Continued

- **Form field conversion and validation**
  - JSF has builtin capabilities for checking that form values are in the required format and for converting from strings to various other data types. If values are missing or in an improper format, the form can be automatically redisplayed with error messages and with the previously entered values maintained.
- **Centralized file-based configuration**
  - Rather then hard-coding information into Java programs, many JSF values are represented in XML or property files. This loose coupling means that many changes can be made without modifying or recompiling Java code, and that wholesale changes can be made by editing a single file. This approach also lets Java and Web developers focus on their specific tasks without needing to know about the overall system layout.
- **Consistent approach**
  - JSF encourages consistent use of MVC throughout your application.

Disadvantages of JSF (vs. MVC with RequestDispatcher)

- **Bigger learning curve**
  - To use MVC with the standard RequestDispatcher, you need to be comfortable with the standard JSP and servlet APIs. To use MVC with JSF, you have to be comfortable with the servlet API *and* a large and elaborate framework that is almost equal in size to the core system.
  - Similarly, if you have an existing app and want to add in some small amounts of Ajax functionality, it is moderately easy with jQuery (quite easy if you know JavaScript already). Switching your app to JSF 2.0 is a big investment.
- **Worse documentation**
  - Compared to the standard servlet and JSP APIs, JSF has fewer online resources, and many first-time users find the online JSF documentation confusing and poorly organized. True for both Mojarra and MyFaces.

Disadvantages of JSF (vs. Standard MVC), Continued

- **Less transparent**
  - With JSF applications, there is a lot more going on behind the scenes than with normal Java-based Web applications. As a result, JSF applications are:
    - Harder to understand
    - Harder to benchmark and optimize
- **Undeveloped tool support**
  - There are many IDEs with strong support for standard servlet and JSP technology. JSF 2.0 is pretty new, and IDE support is weak as of 7/2010.
- **Rigid approach**
  - The flip side of the benefit that JSF encourages a consistent approach to MVC is that JSF makes it difficult to use other approaches.

JSF vs. Struts

Struts is a Serious Alternative

- **Situation**
  – You decide that for your application, the benefits (power) of using a Web application framework [vs. servlets/JSP with MVC] outweigh the disadvantages (complexity)
- **Problem**
  – JSF is not the only game in town
- **Alternatives**
  – Struts is the most popular alternative to JSF
  – But there are many more
    - Spring MVC
    - Apache Wicket
    - Apache Tapestry
    - jMaki
    - …

Advantages of JSF (vs. Struts)

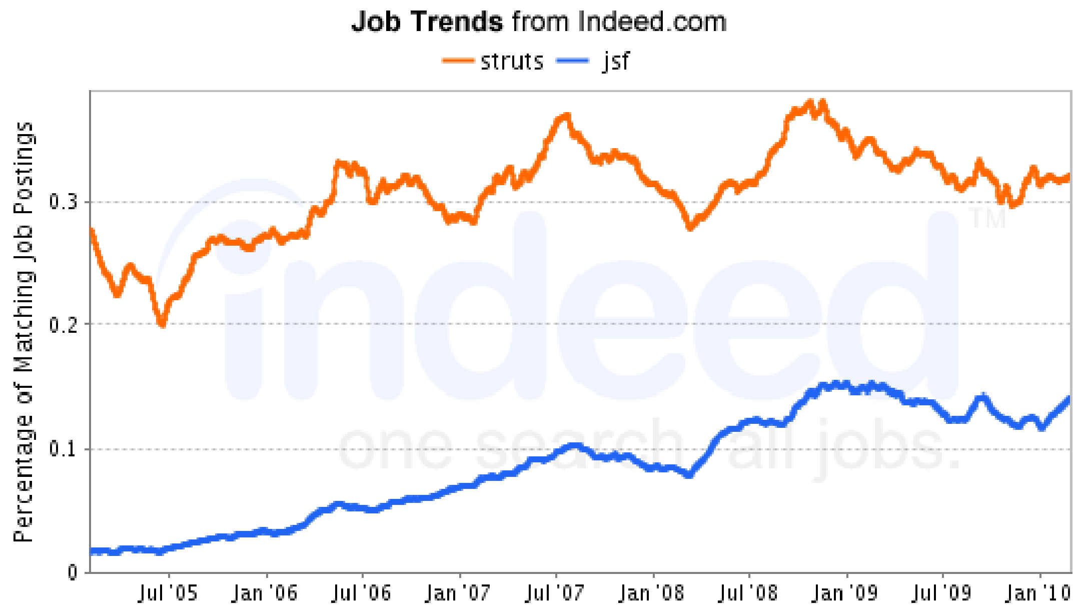
- **Custom components**
  - JSF makes it relatively easy to combine complex GUIs into a single manageable component; Struts does not
  - There are many existing third-party component libraries for JSF, virtually none for Struts
    - JBoss RichFaces, Oracle ADF, IceFaces, Apache Tomahawk, Woodstock, Web Galileo, Pretty Faces, …
- **Support for other display technologies**
  - JSF is not limited to HTML and HTTP; Struts is
- **Access to beans by name**
  - JSF lets you assign names to beans, then you refer to them by name in the forms. Struts has a complex process with several levels of indirection.
- **Official part of Java EE**
  - JSF is part of the Java EE spec, and all servers that support Java EE include JSF (JSF 2.0 is part of Java EE 6)

Advantages of JSF (vs. Struts), Continued

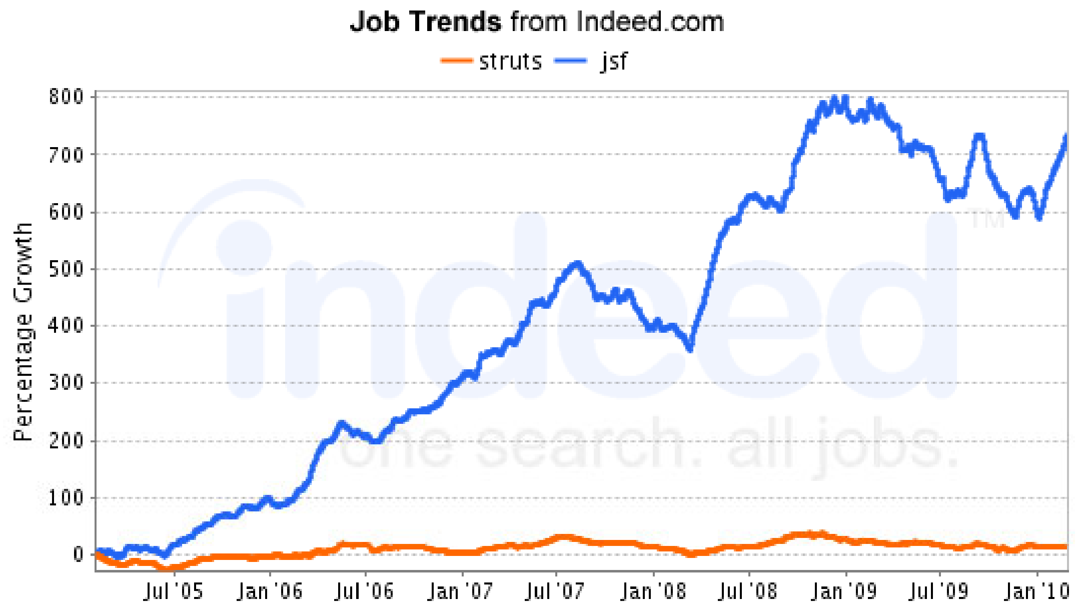
- **Expression language**
  - The JSF expression language is more concise and powerful than the Struts bean:write tag.
    - This is less advantageous vs. Struts 2.0
- **Simpler controller and bean definitions**
  - JSF does not require your controller and bean classes to extend any particular parent class (e.g., Action) or use any particular method (e.g., execute). Struts does.
- **Simpler config file and overall structure**
  - The faces-config.xml file is much easier to use than is the struts-config.xml file. In general, JSF is simpler.
- **More powerful potential tool support**
  - The orientation around GUI controls and their handlers opens possibility of simple to use, drag-and-drop IDEs

Disadvantages of JSF (vs. Struts)

- **Established base**



Counter-Arguments re Established Base



Disadvantages of JSF (vs. Struts), Continued

- **Support for other display technologies**
  - JSF is not limited to HTML and HTTP; Struts is
    - Hey! Didn't I say this was an *advantage* of JSF?
- **Confusion vs. file names**
  - The actual pages used in JSF end in *.xhtml*. But the URLs used end in *.faces* or *.jsf*. This causes many problems; in particular, in JSF:
    - You cannot browse directories and click on links
    - It is hard to protect raw XHTML pages from access
    - It is hard to refer to non-faces pages in faces-config.xml
- **Self-submit approach**
  - With Struts, the form (blah.*jsp*) and the handler (blah.*do*)have different URLs; with JSF they are the same (blah.jsf).

Disadvantages of JSF (vs. Struts), Continued

- **Much weaker automatic validation**
  - Struts comes with validators for email address, credit card numbers, regular expressions, and more. JSF only comes with validators for missing values, length of input, and numbers in a given range.
    - You can use third-party validation libraries with JSF (e.g., MyFaces/Tomahawk built on the Struts/Commons validation library), but still not as powerful as Struts
- **Lack of client-side validation**
  - Struts supports JavaScript-based form-field validation; JSF does not

JSF 2.0 vs. Other Ajax Approaches

Three Main Options

- **Traditional JavaScript library**
  - Add JavaScript code to your existing Web app to add richer GUIs and to support asynchronous, incremental updates to your page
    - E.g., jQuery, Dojo, Ext-JS, YUI, Prototype/Scriptaculous, Google Closure, Mootools
- **GWT**
  - Write everything in Java. Use RPC mechanism to talk to server. Client-side code gets compiled to JavaScript.
- **JSF 2.0 (or another framework with integrated Ajax support)**
  - Use simple tags to perform Ajax requests, without explicit JavaScript programming

Using a Traditional JavaScript Library

- **Scenario**
  - You already built a Web app (using Java, .NET, PHP, Ruby on Rails, or whatever)
  - You want to upgrade it by adding richer GUI elements and Ajax-enabled features
  - You do not want to start over and reimplement the existing functionality
- **Best approach**
  - Use JavaScript library with rich widgets and Ajax support
    - jQuery, Ext-JS, Dojo, YUI, Prototype/Scriptaculous, Google Closure, Mootools
  - Which library is best is another hard question
    - Different libraries have different strengths. I have an entire talk on this question.

Example: jQuery for Bank Customer Info

- **Functionality**
  - Enter a customer ID
  - Show the first name, last name, and bank account balance of customer with that ID
  - Show error message if ID does not exist
- **Goals**
  - Do not reload the page
  - Use existing server-side logic
- **Approach**
  - Use jQuery to pass ID to server via Ajax request and to insert result into page
  - Extend server to format customer as <ul> list

Bank Example: HTML

```
…
<fieldset>
  <legend>Find Customer Info</legend>
  <form id="customer-form">
    Customer ID:
    <input type="text" name="customerID"/>
    <input type="button" value="Show Customer"
           id="customer-button"/>
  </form>
  <div id="customer-region"/>
</fieldset>
…
```

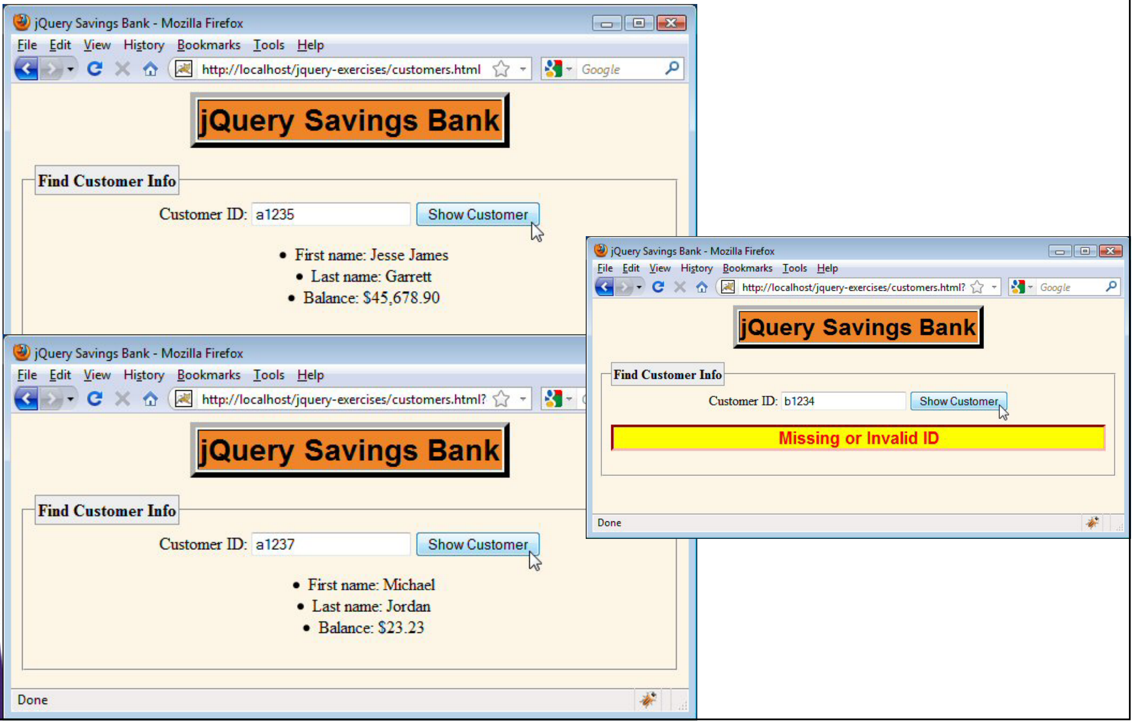
Bank Example: JavaScript

```
$(function() {
    $("#customer-button").click(showCustomer);
});

function showCustomer() {
  var queryData = $("#customer-form").serialize();
  $("#customer-region").load("find-customer-by-id",
                             queryData);
}
```

Bank Example: Java

```
public class FindCustomerByID extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String customerID = request.getParameter("customerID");
    Customer customer =
      CustomerUtils.getCustomer(customerID);
    String customerData;
    if (customer == null) {
      customerData = ResultUtils.makeErrorMessage("ID");
    } else {
      customerData = ResultUtils.makeBulletedList(customer);
    }
    PrintWriter out = response.getWriter();
    out.print(customerData);
  }
}
```

Bank Example: Results



jQuery Savings Bank

Find Customer Info

Customer ID: a1235

- First name: Jesse James
- Last name: Garrett
- Balance: $45,678.90

jQuery Savings Bank

Find Customer Info

Customer ID: a1237

- First name: Michael
- Last name: Jordan
- Balance: $23.23

jQuery Savings Bank

Find Customer Info

Customer ID: b1234

Missing or Invalid ID

jQuery Savings Bank - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/jquery-exercises/customers.html
Show Customer
Done

Using GWT

- **Scenario**
  - You are starting a new Web app
  - You want a large pure-Ajax app
    - Looks like a desktop application
    - Has complex client-server communication
  - For client-side code, you want strong typing, many data structures, and few runtime errors
    - You want Java, not JavaScript for the client!
- **Best approach**
  - Use the Google Web Toolkit
    - Client-side Java code gets compiled to JavaScript
    - Powerful RPC mechanism for talking to server

Example: GWT for Bank Customer Info

- **Functionality**
  - Enter a customer ID
  - Show the first name, last name, and bank account balance of customer with that ID
  - Show error message if ID does not exist
- **Goals**
  - Do not reload the page
  - Use straightforward server-side logic (no HTTP methods)
  - Return Customer object, not just String, to client
- **Approach**
  - Use GWT
  - Use Customer class on both client and server

Bank Example: Java (Customer Class)

```
public class Customer implements Serializable {
  private String id, firstName, lastName;
  private double balance;

  public Customer(String id,
                  String firstName,
                  String lastName,
                  double balance) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.balance = balance;
  }
  …
  // Getters and setters
}
```

Bank Example: Java (Core Client Code – Part 1)

```
private class ButtonHandler implements ClickHandler {
  public void onClick(ClickEvent event) {
    String id = idBox.getText();
    serviceProxy.findCustomer(id, new ButtonCallback());
  }
}
```

Bank Example: Java (Core Client Code – Part 2)

```
private class ButtonCallback implements AsyncCallback<Customer>
{
   public void onSuccess(Customer result) {
     if (result != null) {
       String message = result.getFirstName() + " " +
                        result.getLastName() +
                        " has balance of $" +
                        result.getBalance();
       customerInfo.setHTML(message);
     } else {
       customerInfo.setHTML("No such ID");
     }
   }

   public void onFailure(Throwable caught) {
     Window.alert("Unable to get data from server.");
   }
 }
```

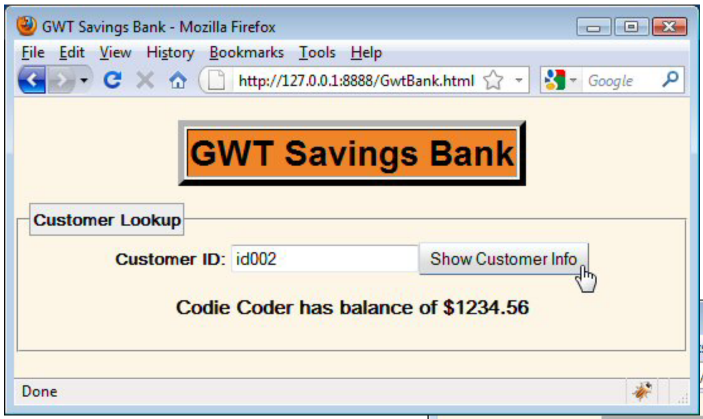
Bank Example: Java (Core Server Code)

```
public class DataServiceImpl extends RemoteServiceServlet
                             implements DataService {
  public Customer findCustomer(String id) {
    CustomerLookupService service =
      new CustomerSimpleMap();
    return(service.findCustomer(id));
  }
}
```

Bank Example: JavaScript

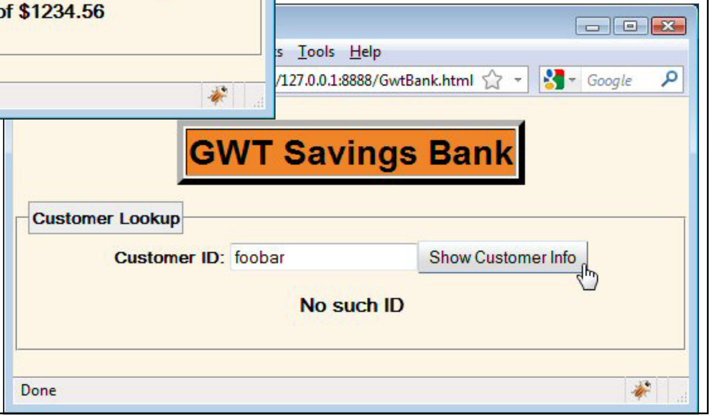
This page deliberately left blank

Bank Example: Results


GWT Savings Bank - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:8888/GwtBank.html
Google
GWT Savings Bank
Customer Lookup
Show Customer Info
Done

Customer ID: id002

Codie Coder has balance of $1234.56



Customer ID: foobar

No such ID

Using JSF 2.0

- **Scenario**
  - You are starting a new Web app
  - You want a hybrid application
    - Regular form submission and page navigation
    - Ajax-enabled content
    - Richer controls
  - You don't want to write a lot of JavaScript by hand
- **Best approach**
  - Use a Web app framework that includes rich controls and integrated Ajax support
    - Java developers: JSF 2.0, Struts 2.0, or Spring MVC 3.0
    - .NET developers: ASP.NET Ajax

Example: JSF 2.0 for Bank Customer Info

- **Functionality**
  - Enter a customer ID
  - Show the first name, last name, and bank account balance of customer with that ID
  - Show error message if ID does not exist
- **Goals**
  - Do not reload the page
  - Use existing server-side logic and bean names
  - Use familiar JSF tags, not explicit JavaScript
- **Approach**
  - Use JSF 2.0 <f:ajax> tag
  - Leave server code unchanged

Bank Example: JSF (Facelets)

```
…
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBeanAjax.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBeanAjax.password}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankingBeanAjax.showBalance}">
    <f:ajax execute="@form"
            render="ajaxMessage1"/>
  </h:commandButton>
  <br/>
  <h2><h:outputText value="#{bankingBeanAjax.message}"
                    id="ajaxMessage1"/></h2>
</h:form>
…
```

Bank Example: JavaScript

This page deliberately left blank

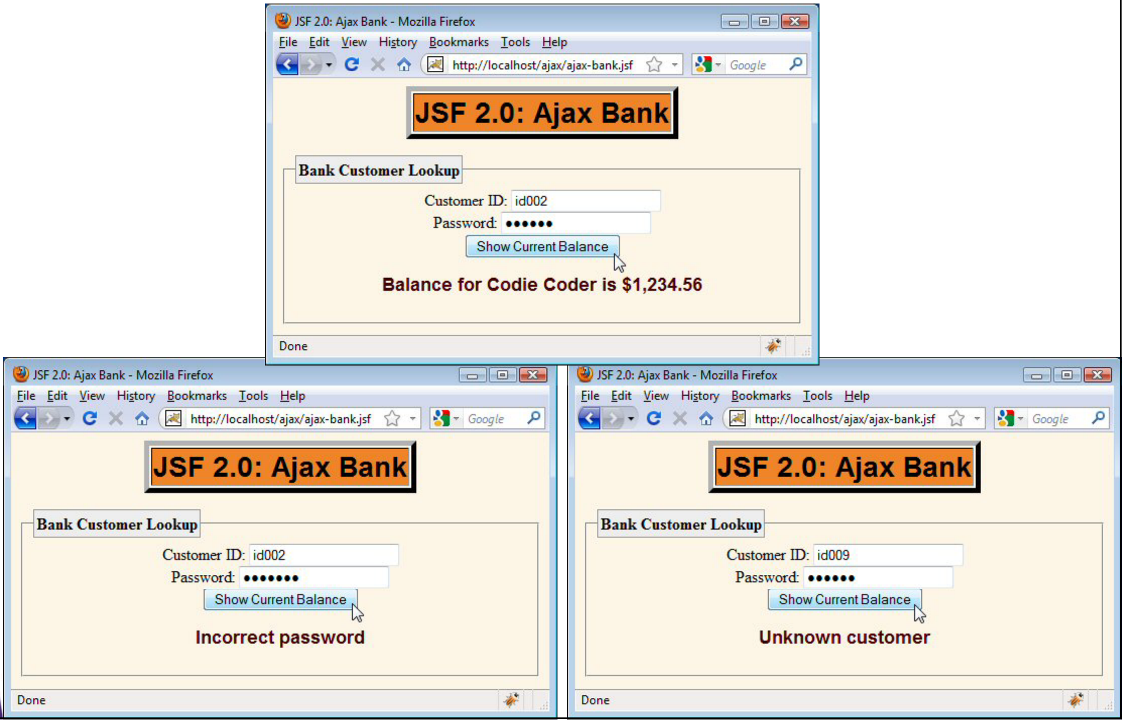
Bank Example: Java

```
@ManagedBean
public class BankingBeanAjax extends BankingBeanBase {
  private String message = "";

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

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

Bank Example: Java (Continued)

```
  public String showBalance() {
    if (!password.equals("secret")) {
      message = "Incorrect password";
    } else {
      CustomerLookupService service =
        new CustomerSimpleMap();
      customer = service.findCustomer(customerId);
      if (customer == null) {
        message = "Unknown customer";
      } else {
        message =
          String.format("Balance for %s %s is $%,.2f",
                        customer.getFirstName(),
                        customer.getLastName(),
                        customer.getBalance());
      }
    }
    return(null);
```

Bank Example: Results


JSF 2.0: Ajax Bank - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/ajax/ajax-bank.jsf
Google
Done

JSF 2.0: Ajax Bank

Bank Customer Lookup

Customer ID: id002
Password: ••••••
Show Current Balance

Balance for Codie Coder is $1,234.56

JSF 2.0: Ajax Bank

Bank Customer Lookup

Customer ID: id002
Password: •••••••
Show Current Balance

Incorrect password

JSF 2.0: Ajax Bank

Bank Customer Lookup

Customer ID: id009
Password: ••••••
Show Current Balance

Unknown customer

Bottom Line

- **Use traditional JavaScript library when:**
  – You have existing app already built
  – You want to incrementally add rich GUIs/Ajax to it
- **Use GWT when:**
  – You are starting a new Web app
  – You want it to look like a desktop app (no page nav)
  – You have complex client-server comms
- **Use JSF 2.0 when:**
  – You are starting a new Web app
  – It will be a combination of traditional form submissions and page navigation plus rich GUIs/Ajax content

JSF 2.0 vs. JSF 1.x

Overview of JSF 2.0

- **New features vs. JSF 1.x**
  - Smart defaults
  - Annotations to replace many faces-config.xml entries
  - Ajax support
  - Integrated support for facelets
  - Simpler custom components
  - More components and validators
  - Support for Groovy
  - Ability to bookmark results pages
  - Lots more
- **Downside**
  - Simple installation and testing instructions hard to find
    - Rectifying this is the main point of next section
    - Next section gives a whirlwind tour of main new JSF 2 features
    - Later sections give detailed tutorial on JSF 2.0

Main JSF 2.0 Implementations

- **Sun/Oracle Mojarra**
  - Main page: https://javaserverfaces.dev.java.net/
  - Runs in any server supporting servlets 2.5 or later
  - Also integrated into Glassfish 3
- **Apache MyFaces**
  - Main page: http://myfaces.apache.org/core20/
  - Runs in any server supporting servlets 2.5 or later
  - Also integrated into Apache Geronimo 3
- **Any Java EE 6 server**
  - JSF 2.0 is an official part of Java EE 6
    - JBoss 6, Glassfish 3, WebLogic 11

Wrap-Up

Summary

- **Alternatives: traditional Web apps**
  - Servlets/JSP (with MVC)
  - Struts 2.0
  - JSF 2.0
- **Alternatives: Ajax-based Web apps**
  - Add jQuery (etc) to existing app
  - Use the Google Web Toolkit and write everything in Java, following a Swing-like style
  - Use JSF 2.0 and add Ajax tags to page
- **JSF 2.0 vs. JSF 1.x**
  - Version 2 is significantly more powerful and easier to use than version 1

Questions?