Managed Beans III – Advanced Capabilities

Originals of slides and source code for examples: http://www.coreservlets.com/JSF-Tutorial/jsf2/
Also see the PrimeFaces tutorial – http://www.coreservlets.com/JSF-Tutorial/primefaces/
and customized JSF2 and PrimeFaces training courses – http://courses.coreservlets.com/jsf-training.html

Topics in This Section

- Overview of bean scopes
- Session scope
- Session scope with redirects
- Getting the “raw” request and response objects
- Dependency injection

Overview of Bean Scopes

Bean Scopes

- **Idea**
  - Designates how long managed beans will stay “alive”, and which users and requests can access previous bean instances.
    - Request scope is the default
- **JSF 1.*x* scopes**
  - request, session, application
  - Specified in faces-config.xml
- **JSF 2.0 and 2.1 scopes**
  - Same as before plus view, none, custom
  - Specified either in faces-config.xml or by one of the new annotations (e.g., @SessionScoped)
- **JSF 2.2 scopes**
  - Same as before plus flow scope (see later tutorial section)
  - Specified either in faces-config.xml or by one of the new annotations (e.g., @FlowScoped)

Request Scope: Interpretation

- **Meaning**
  - A new instance of the bean is created for every HTTP request, regardless of whether it is the same user or the same page. This is the most commonly used scope in all of JSF (session scope is second-most common).
- **Annotation**
  - @RequestScoped
  - But, request scope is default, so most developers simply omit the annotation

Request Scope: Quick Example

- **Java**

```
@ManagedBean
public class BankForm { … }
```

- **Facelets**

```
<h:inputText value="#{bankForm.customerId}"/>
```

- **Behavior**
  - Bean is instantiated twice for each submission: once when form is displayed (and getCustomerId is called) and again when form is submitted (and textfield value is passed to setCustomerId).
  - If same bean name appears on a different page, different instances are used.

Application Scope: Interpretation

- **Meaning**
  - The bean is instantiated the first time any page with that bean name is accessed. From then on, the same bean instance is used, even if it is different user or different page. However, different Web apps are independent.
    - Never use application scope for beans that contain user data! Testing on your local machine with a single user might show correct results, but with multiple simultaneous users, you have race conditions with one user's data overwriting another's.
- **Annotation**
  - @ApplicationScoped
    - Optionally, use with @ManagedBean(eager=true)
    - This option causes object to be instantiated when app loaded

Application Scope: Quick Example

- **Java**

```
@ManagedBean
@ApplicationScoped
public class Messages { … }
```

- **Facelets**

```
#{messages.message1}
```

- **Behavior**
  - The first time this page (or any page with that bean name) is accessed, Messages is instantiated. From then on, the same bean instance is used for all users and on all pages that use that bean name.

Session Scope: Interpretation

- **Meaning**
  - The bean is instantiated the first time any page with that bean name is accessed by a particular user. From then on, the same bean instance is used if it is same bean name and same user, even if it is different page. However, different users get different instances. User determined by JSESSIONID cookie (usually) or jsessionid URL parameter (sometimes).
    - Second-most common scope, after request scope.
    - Often used for user preferences (fonts, colors, languages). Also used for accumulating data over time (shopping carts, questions on exams).
    - The bean should be Serializable so that session data can live across server restarts and so that on clustered server, session data can be shared between nodes.
- **Annotation**
  - @SessionScoped

Session Scope: Quick Example

- **Java**

```
@ManagedBean
@SessionScoped
public class UserLocale implements Serializable { … }
```

- **Facelets**

```
<f:view locale="#{userLocale.selectedLanguage}"/>
```

  - f:view and locales covered in section on event handling
- **Behavior**
  - Bean is instantiated first time a particular user accesses any page that refers to that bean name.
  - Same instance is used for that user on all pages that use that bean name.

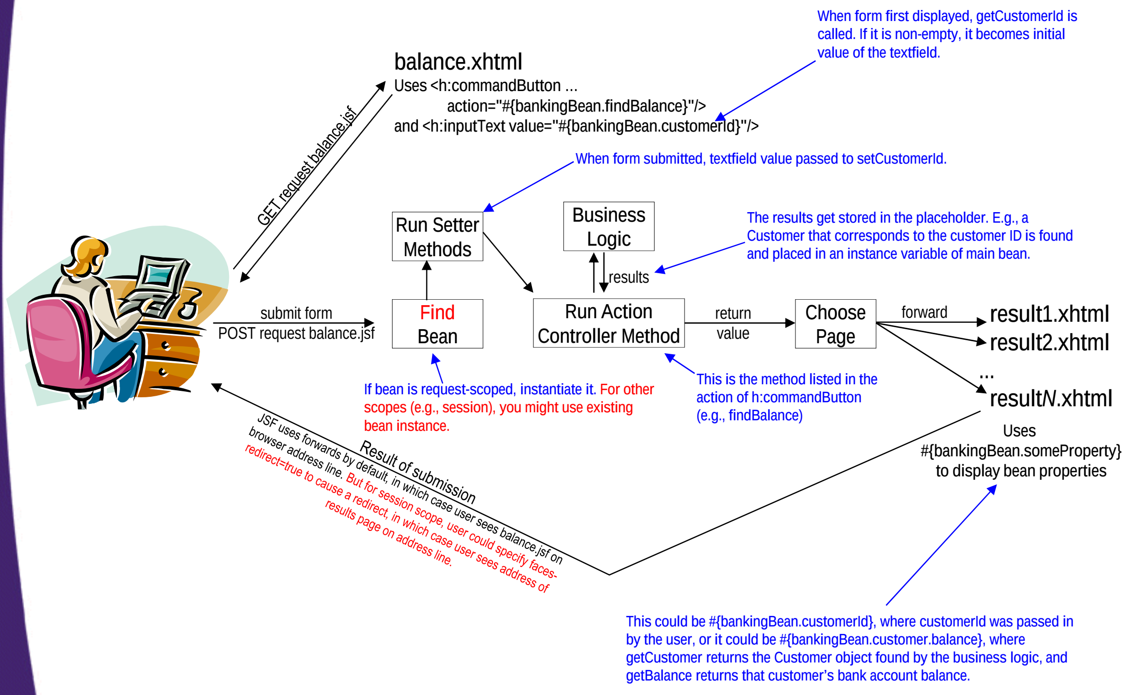
Annotations to Specify Bean Scope

- **@RequestScoped**
  - Default. Make a new instance for every HTTP request. Since beans are also used for initial values in input form, this means bean is generally instantiated twice (once when form is displayed, once when form is submitted).
- **@SessionScoped**
  - Put bean in session scope. If same user with same cookie (JSESSIONID) returns before session timeout, same bean instance is used. You should make bean Serializable.
- **@ApplicationScoped**
  - Put bean in application scope. Shared by all users. Bean either should have no mutable state or you must carefully synchronize access. Usually immutable.

Annotations to Specify Bean Scope (Continued)

- **@ViewScoped**
  - Same bean instance is used as long as same user is on same page (e.g., with master/detail pages or Ajax)
    - Bean should implement Serializable
- **@FlowScoped**
  - Same bean instance is used as long as it is same user on same *set* of pages
    - New scope in JSF 2.2. See separate tutorial section
- **@CustomScoped(value="#{someMap}")**
  - Bean is stored in Map; programmer can control lifecycle
- **@NoneScoped**
  - Bean is not placed in a scope. Useful for beans that are referenced by other beans that *are* in scopes

JSF Flow of Control (Updated but Still Simplified)



Uses <h:commandButton ...
action="#{bankingBean.findBalance}"/>
and <h:inputText value="#{bankingBean.customerId}"/>

When form first displayed, getCustomerId is called. If it is non-empty, it becomes initial value of the textfield.

When form submitted, textfield value passed to setCustomerId.

The results get stored in the placeholder. E.g., a Customer that corresponds to the customer ID is found and placed in an instance variable of main bean.

This is the method listed in the action of h:commandButton (e.g., findBalance)

If bean is request-scoped, instantiate it. For other scopes (e.g., session), you might use existing bean instance.

Uses #{bankingBean.someProperty} to display bean properties

JSF uses forwards by default, in which case user sees balance.jsf on browser address line. But for session scope, user could specify faces-redirect=true to cause a redirect, in which case user sees address of results page on address line.

This could be #{bankingBean.customerId}, where customerId was passed in by the user, or it could be #{bankingBean.customer.balance}, where getCustomer returns the Customer object found by the business logic, and getBalance returns that customer's bank account balance.

Session Scope

Session Scope: Main Points

- **Bean instance reused if**
  - Same user (even if different page in the app)
  - Same browser session
    - Usually based on cookies, but can be based on URL rewriting
- **Useful for**
  - Remembering user preferences
  - Prefilling values from previous entries
  - Accumulating lists of user data (ala shopping carts)
- **Normal Java session tracking rules apply**
  - Custom classes should be Serializable
    - Some servers save session data to disk on restart
    - Distributed Web apps need this to replicate sessions

Session Scope: Example

- **Idea**
  - Small variation of banking example
  - Remember previously entered id
    - If end user comes back to input page, the ID they entered last time is already filled in
- **What you need**
  - Add @SessionScoped to bean declaration
  - Make bean Serializable

bank-lookup2.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
…
<h:body>
…
<fieldset>
<legend>Bank Customer Lookup (with Session Scope)</legend>
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBean2.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBean2.password}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankingBean2.showBalance}"/>
</h:form>
</fieldset>
…
</h:body></html>
```

Same as bank-lookup.xhtml except for bean name and some minor changes to text.

BankingBean2.java

```
import java.io.*;
import javax.faces.bean.*;

@ManagedBean
@SessionScoped
public class BankingBean2 extends BankingBean
                           implements Serializable {
  @Override
  public String showBalance() {
    String origResult = super.showBalance();
    return(origResult + "2");
  }
}
```

If a page uses the name bankingBean2 and is accessed by the same user in the same browser session, the same bean instance will be used.

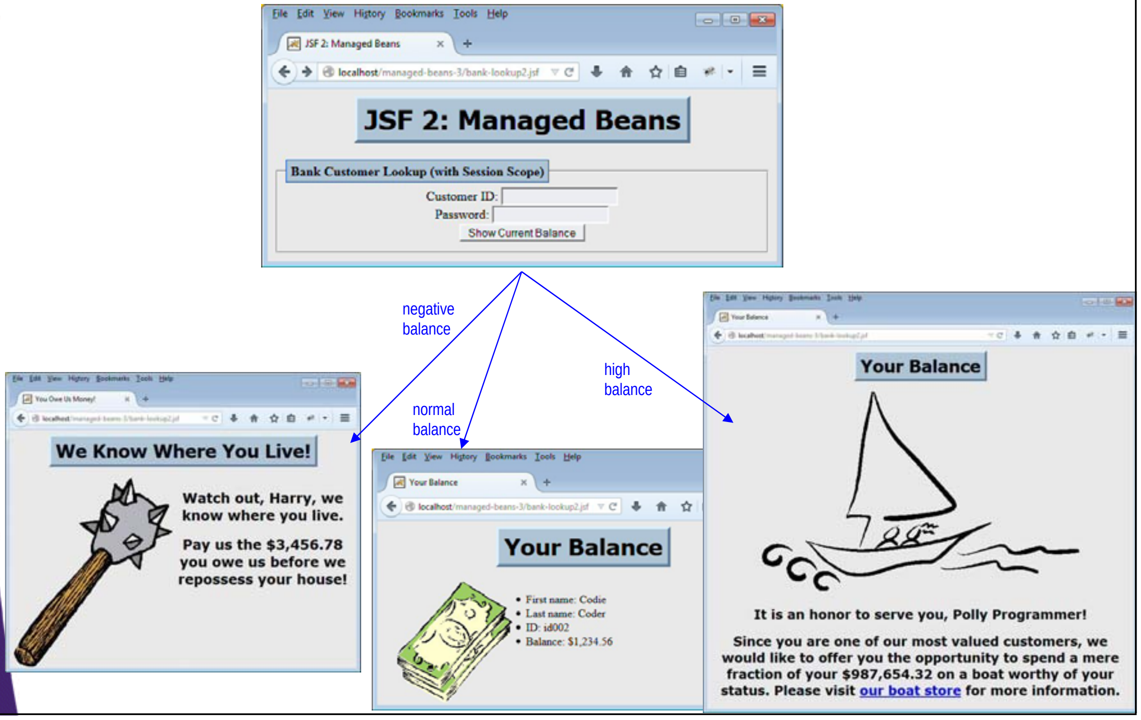
Results pages are negative-balance2.xhtml, normal-balance2.xhtml, etc. But, this example still uses forwards, so, as usual, name of results page will not exposed to end user. Next example uses redirects, where page is shown to end user.

normal-balance2.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<ul>
  <li>First name: #{bankingBean2.customer.firstName}</li>
  <li>Last name: #{bankingBean2.customer.lastName}</li>
  <li>ID: #{bankingBean2.customer.id}</li>
  <li>Balance: $#{bankingBean2.customer.balanceNoSign}</li>
</ul>
…
</h:body></html>
```

Same as normal-balance.xhtml except for bean name.
negative-balance2.xhtml and high-balance2.xhtml are similar.

Results (Initial Access)



JSF 2: Managed Beans

Bank Customer Lookup (with Session Scope)

Customer ID:
Password:

We Know Where You Live!

Watch out, Harry, we know where you live.

Pay us the $3,456.78 you owe us before we repossess your house!

Your Balance

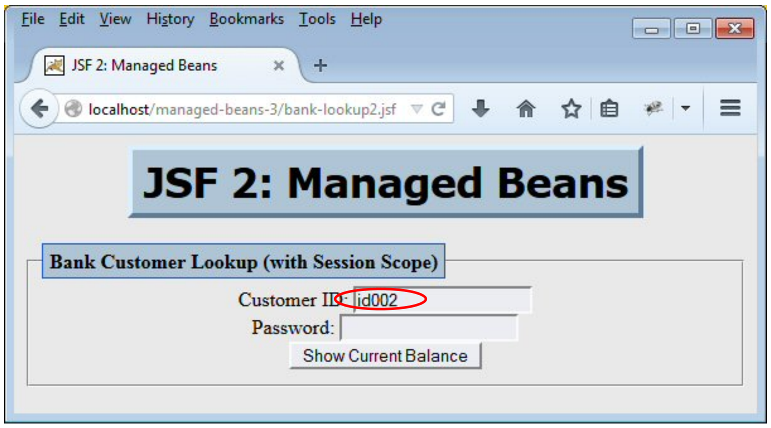
- First name: Codie
- Last name: Coder
- ID: id002
- Balance: $1,234.56

Your Balance

It is an honor to serve you, Polly Programmer!

Since you are one of our most valued customers, we would like to offer you the opportunity to spend a mere fraction of your $987,654.32 on a boat worthy of your status. Please visit our boat store for more information.

Results (User Returns in Same Browser Session)


File Edit View History Bookmarks Tools Help
JSF 2: Managed Beans
localhost/managed-beans-3/bank-lookup2.jsf
Show Current Balance

JSF 2: Managed Beans

Bank Customer Lookup (with Session Scope)

Customer ID: id002
Password:

Question: why wasn’t password “remembered” like the id was?

Session Scope with Redirects

Session Scope: Example

- **Idea**
  - Small variation of session-scoped banking example
  - Use redirects instead of forwards
    - So names of results pages are exposed to end users, who can bookmark them and navigate to them directly.
    - This is extra work, because you must consider situation where user follows bookmark in new session, when there is no stored data. However, point is that this is possible with session data, but not with request data.
- **What you need**
  - Add faces-redirect=true to end of return values, to tell JSF to redirect (instead of forward) to results pages
    - Allows users to access results pages directly
      - Later, when we use faces-config.xml for navigation rules, we can supply <redirect/> there instead of faces-redirect=true

bank-lookup3.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
…
<h:body>
…
<fieldset>
<legend>Bank Customer Lookup (Session Scope plus Redirects)</legend>
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBean3.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBean3.password}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankingBean3.showBalance}"/>
</h:form>
</fieldset>
…
</h:body></html>
```

Same as bank-lookup-2.xhtml except for bean name and some minor changes to text.

BankingBean3.java

```
import java.io.*;
import javax.faces.bean.*;

@ManagedBean
@SessionScoped
public class BankingBean3 extends BankingBean
                                 implements Serializable {
  @Override
  public String showBalance() {
    String origResult = super.showBalance();
    return(origResult + "3?faces-redirect=true");
  }
}
```

If a page uses the name bankingBean3 and is accessed by the same user in the same browser session, the same bean instance will be used.

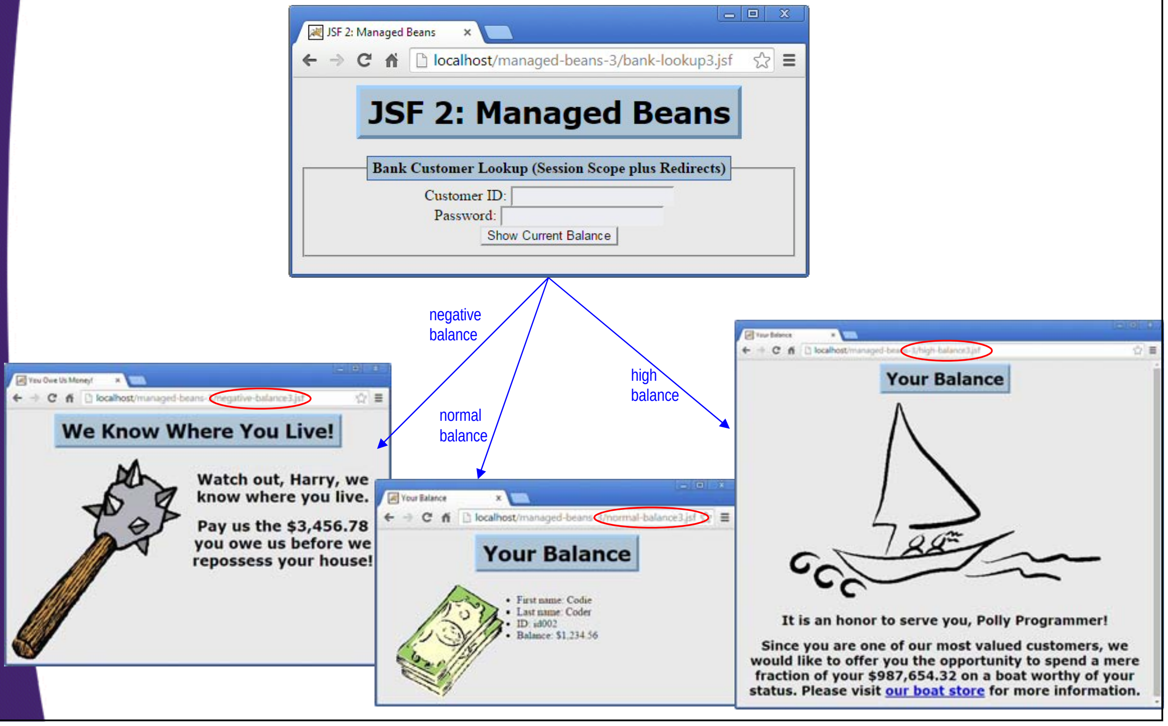
Results pages are negative-balance3.xhtml, normal-balance3.xhtml, etc. By also appending faces-redirect=true, JSF will redirect instead of forward to the results pages, thus exposing the URLs of the results pages and letting users navigate directly to them later.

normal-balance3.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<ul>
  <li>First name: #{bankingBean3.customer.firstName}</li>
  <li>Last name: #{bankingBean3.customer.lastName}</li>
  <li>ID: #{bankingBean3.customer.id}</li>
  <li>Balance: $#{bankingBean3.customer.balanceNoSign}</li>
</ul>
…
</h:body></html>
```

Same as normal-balance.xhtml except for bean name.
negative-balance3.xhtml and high-balance3.xhtml are similar.

Results (Initial Access)


JSF 2: Managed Beans
localhost/managed-beans-3/bank-lookup3.jsf
negative balance
normal balance
high balance
You Owe Us Money!
localhost/managed-beans-3/negative-balance3.jsf
Your Balance
localhost/managed-beans-3/normal-balance3.jsf
localhost/managed-beans-3/high-balance3.jsf

JSF 2: Managed Beans

Bank Customer Lookup (Session Scope plus Redirects)

Customer ID:

Password:

Show Current Balance

We Know Where You Live!

Watch out, Harry, we know where you live.

Pay us the $3,456.78 you owe us before we repossess your house!

Your Balance

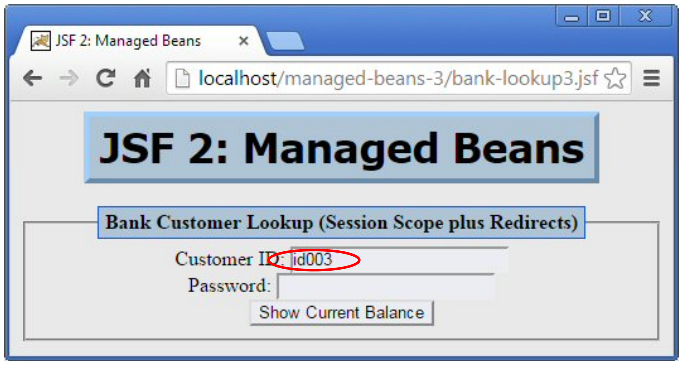
- First name: Codie
- Last name: Coder
- ID: id002
- Balance: $1,234.56

Your Balance

It is an honor to serve you, Polly Programmer!

Since you are one of our most valued customers, we would like to offer you the opportunity to spend a mere fraction of your $987,654.32 on a boat worthy of your status. Please visit our boat store for more information.

Results (User Returns in Same Browser Session)



JSF 2: Managed Beans

Bank Customer Lookup (Session Scope plus Redirects)

Customer ID: id003

Password:

Getting the Request and Response Objects

Issue

- **No automatic access to request & response**
  - JSF action controller methods do not have direct access
    - Unlike in Struts, where action controller method (execute) gets request and response automatically
- **Good news**
  - In most cases, only use for request and response objects is for explicit user data (request parameters), and JSF provides a much simpler way to get them.
  - Having your form beans be POJOs is very convenient
- **Bad news**
  - In the cases where you need the request and response objects, code is more awkward
  - JSF programmers forget how valuable the request and response objects are

Why You Need Request and Response Objects

- **Uses for request object**
  - Explicit session manipulation
    - E.g., changing inactive interval or invalidating session
  - Explicit cookie manipulation (e.g., long-lived cookies)
  - Reading request headers (e.g., User-Agent)
  - Looking up requesting host name
- **Uses for response object**
  - Setting status codes
  - Setting response headers
  - Setting long-lived cookies

Solution

- **Static methods**
  - If they are needed, use static method calls to get them

```
ExternalContext context =
   FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request =
   (HttpServletRequest)context.getRequest();
HttpServletResponse response =
   (HttpServletResponse)context.getResponse();
```

- **Note**
  - In some environments, you cast results of getRequest and getResponse to values other than HttpServletRequest and HttpServletResponse
    - E.g., in a portlet environment, you might cast result to PortletRequest and PortletResponse

Example

- **Idea**
  - Collect a search string and a search engine name, show the results of a search with that search engine.
- **Input form**
  - Use textfield for arbitrary search string. Use drop down menu to list only search engines that app supports.
- **Managed bean**
  - Construct a URL by concatenating a base URL (e.g., http://www.google.com/search?q=) with the URL-encoded search string
  - Do response.sendRedirect
    - Must use static methods to get the HttpServletResponse
  - Return normal strings for error pages

Input Form (search-engine-form.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>…</h:head>
<h:body>
…
<h:form>
  Search String:
  <h:inputText value="#{searchController.searchString}"/><br/>
  Search Engine:
  <h:selectOneMenu value="#{searchController.searchEngine}">
    <f:selectItems value="#{searchController.searchEngineNames}"/>
  </h:selectOneMenu><br/>
  <h:commandButton value="Search"
                   action="#{searchController.doSearch}"/>
</h:form>
</h:body></html>
```

Managed Bean (Part 1 – Properties for Input Elements)

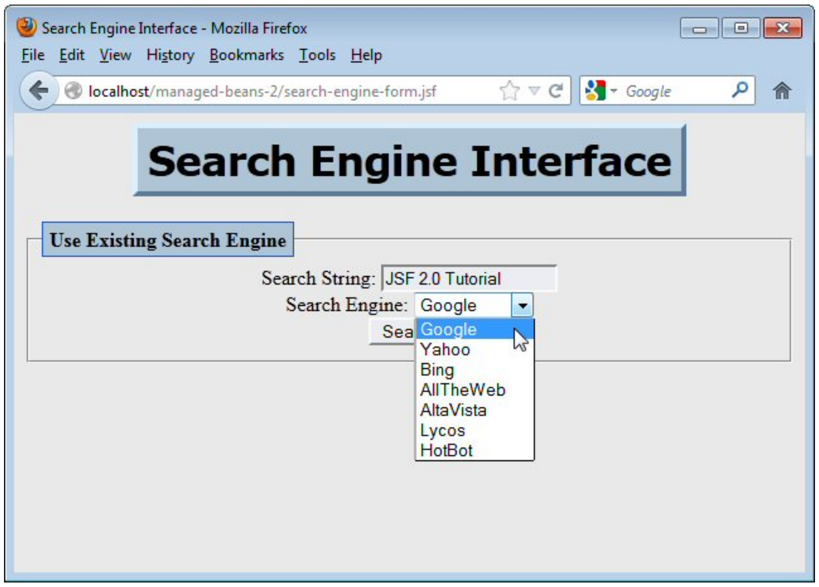
```
@ManagedBean
public class SearchController {
  private String searchString="", searchEngine;

  public String getSearchString() {
    return(searchString);
  }
  public void setSearchString(String searchString) {
    this.searchString = searchString.trim();
  }
  public String getSearchEngine() {
    return(searchEngine);
  }
  public void setSearchEngine(String searchEngine) {
    this.searchEngine = searchEngine;
  }
  public List<String> getSearchEngineNames() {
    return(SearchUtilities.searchEngineNames());
  }
```

Managed Bean (Part 2 – Action Controller)

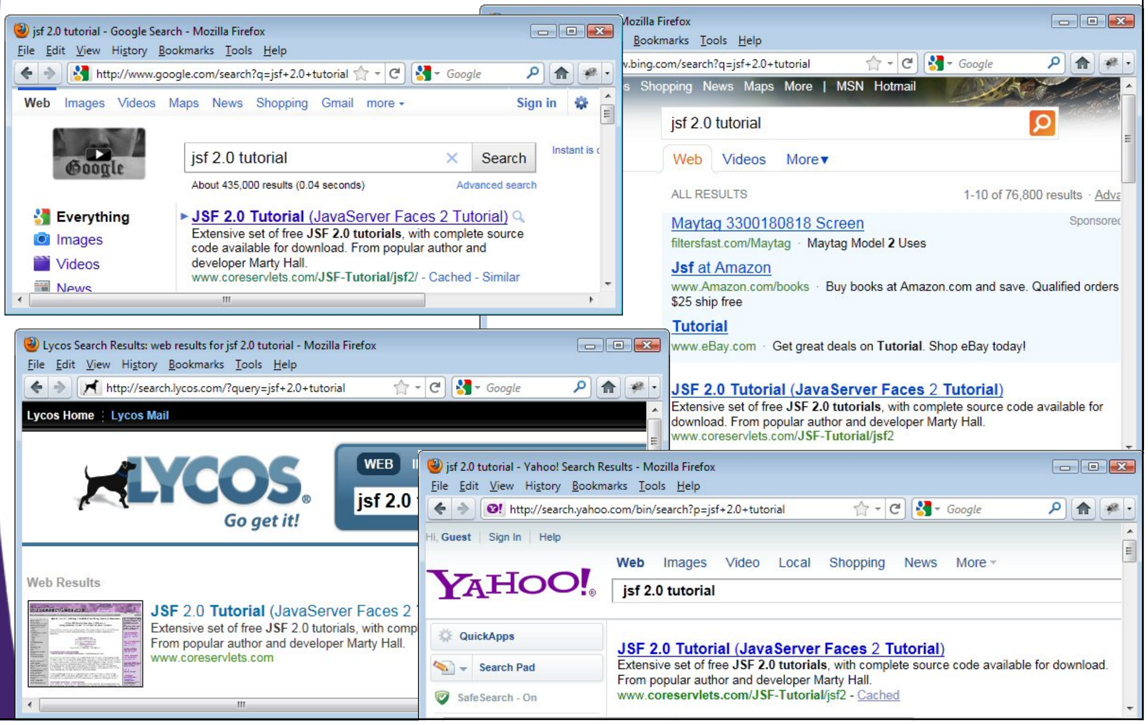
```
public String doSearch() throws IOException {
  if (searchString.isEmpty()) {
    return("no-search-string");
  }
  searchString = URLEncoder.encode(searchString, "utf-8");
  String searchURL =
    SearchUtilities.makeURL(searchEngine, searchString);
  if (searchURL != null) {
    ExternalContext context =
      FacesContext.getCurrentInstance().getExternalContext();
    HttpServletResponse response =
      (HttpServletResponse)context.getResponse();
    response.sendRedirect(searchURL);
    return(null);
  } else {
    return("unknown-search-engine");
  }
}
```

Results (Input Form)


Search Engine Interface - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/managed-beans-2/search-engine-form.jsf
Google
Search Engine Interface
Use Existing Search Engine
Sea
Google
Yahoo
Bing
AllTheWeb
AltaVista
Lycos
HotBot

Search String: JSF 2.0 Tutorial
Search Engine: Google

Results (Forwarding Results)


jsf 2.0 tutorial - Google Search - Mozilla Firefox
http://www.google.com/search?q=jsf+2.0+tutorial
jsf 2.0 tutorial
About 435,000 results (0.04 seconds)
JSF 2.0 Tutorial (JavaServer Faces 2 Tutorial)
Extensive set of free JSF 2.0 tutorials, with complete source code available for download. From popular author and developer Marty Hall.
www.coreservlets.com/JSF-Tutorial/jsf2/ - Cached - Similar
http://www.bing.com/search?q=jsf+2.0+tutorial
jsf 2.0 tutorial
1-10 of 76,800 results
JSF 2.0 Tutorial (JavaServer Faces 2 Tutorial)
Lycos Search Results: web results for jsf 2.0 tutorial - Mozilla Firefox
http://search.lycos.com/?query=jsf+2.0+tutorial
jsf 2.0 tutorial - Yahoo! Search Results - Mozilla Firefox
http://search.yahoo.com/bin/search?p=jsf+2.0+tutorial
jsf 2.0 tutorial
JSF 2.0 Tutorial (JavaServer Faces 2 Tutorial)
Extensive set of free JSF 2.0 tutorials, with complete source code available for download. From popular author and developer Marty Hall.
www.coreservlets.com/JSF-Tutorial/jsf2 - Cached

Advanced Topic: Using @ManagedProperty

Motivation: Analogous Example

- List<String> names = new ArrayList<>();
- Questions
  - What is benefit of using List instead of ArrayList above?
    - We answered this in the Managed Beans 1 lecture, using an interface for the lookup service.
  - What if you want to switch from ArrayList to LinkedList without changing *any* code in the main class?
    - Answered here: use dependency injection.

@ManagedProperty: Main Points

- **JSF supports simple dependency injection**
  - That is, you can assign values to a managed bean property (i.e., a value the main bean depends on) without hardcoding it into the class definition
    - Not as powerful as with Spring, but still useful
  - @ManagedProperty lets you do this with annotations
    - @ManagedProperty(value="#{someBean}")
      private SomeType someField;
  - <managed-property> lets you do it in faces-config.xml
    - This is same as in JSF 1.*x*
- **Setter method for the property is required**
  - E.g., in example above, you must have setSomeField
    - You can use “name” attribute of @ManagedProperty if setter method name does not match field name

@ManagedProperty: Secondary Points

- **You can instantiate beans at app load time**
  - @ManagedBean(eager=true)
    @ApplicationScoped
    public class SomePojo { … }
    - This is useful for the bean that you inject into the main bean. The injected bean is often something shared like a lookup service, whereas the main bean usually contains user-specific data
- **faces-config better than annotations (?)**
  - One of the points of dependency injection is to let you change the concrete class without changing Java code
    - Using annotations in the Java code partially violates that principle: you do not change main class, but do change at least one other class. This is debatable, but some people prefer annotations.
  - faces-config.xml lets you specify Map elements
    - The annotation does not
- **Spring is best of all (?)**
  - For situations that warrant the extra complexity, Spring has nice JSF integration, so you can directly use Spring configuration files (e.g., applicationContext.xml). See separate lecture.

@ManagedProperty: Example

- **Idea**
  - Refactoring of banking example
  - Customer lookup service will be injected into main bean via @ManagedProperty
  - Lookup service will be application scoped and created at application load time
- **What you need**
  - Main bean

```
@ManagedProperty(value="#{lookupServiceBeanName}")
 private CustomerLookupService service;
 public void setService(…) { … }
```

  - Lookup service bean (the bean being injected)
    - @ManagedBean(eager=true)
      @ApplicationScoped

bank-lookup4.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
…
<h:body>
…
<fieldset>
<legend>Bank Customer Lookup</legend>
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBean4.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBean4.password}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankingBean4.showBalance}"/>
</h:form>
</fieldset>
…
</h:body></html>
```

Same as bank-lookup.xhtml except for bean name and some minor changes to text.

BankingBean4.java

```
@ManagedBean
public class BankingBean4 extends BankingBean {
  @ManagedProperty(value="#{currentLookupService}")
  private CustomerLookupService service;

  public void setService(CustomerLookupService service) {
    this.service = service;
  }
 public String showBalance() {
    if (!password.equals("secret")) {
      return("wrong-password3");
    }
    customer = service.findCustomer(customerId);
    if (customer == null) {
      …
  }
}
```

There is now no explicit reference to the concrete class that provides the lookup service. So, it is easier to change among test implementations and real implementations without changing this code. However, when you use annotations only, the bean name is often tied closely to the bean class. The bean name need not be the class name, but you still specify the name in the Java code. So, although it is clear that dependency injection of some sort can make your code more flexible (you can change the lookup service without any changes to the main class), is not entirely clear that the JSF 2 annotation-based way of doing dependency injection is better than the JSF 1.*x* way of doing dependency injection using faces-config.xml.

Even though you must put the @ManagedProperty before the field (instance variable), the setter is what is called by JSF. If the setter name does not match the field name, use @ManagedProperty(*name*…, value…).

CustomerSimpleMap2.java

```
@ManagedBean(name="currentLookupService", eager=true)
@ApplicationScoped
public class CustomerSimpleMap2
        extends CustomerSimpleMap {
}
```

Since the lookup service is shared by all instances of the banking bean, it is declared application scoped. Since it might take some time to initialize the Map, it is instantiated at application load time via "eager". You are required to have @ApplicationScoped if you use eager=true.

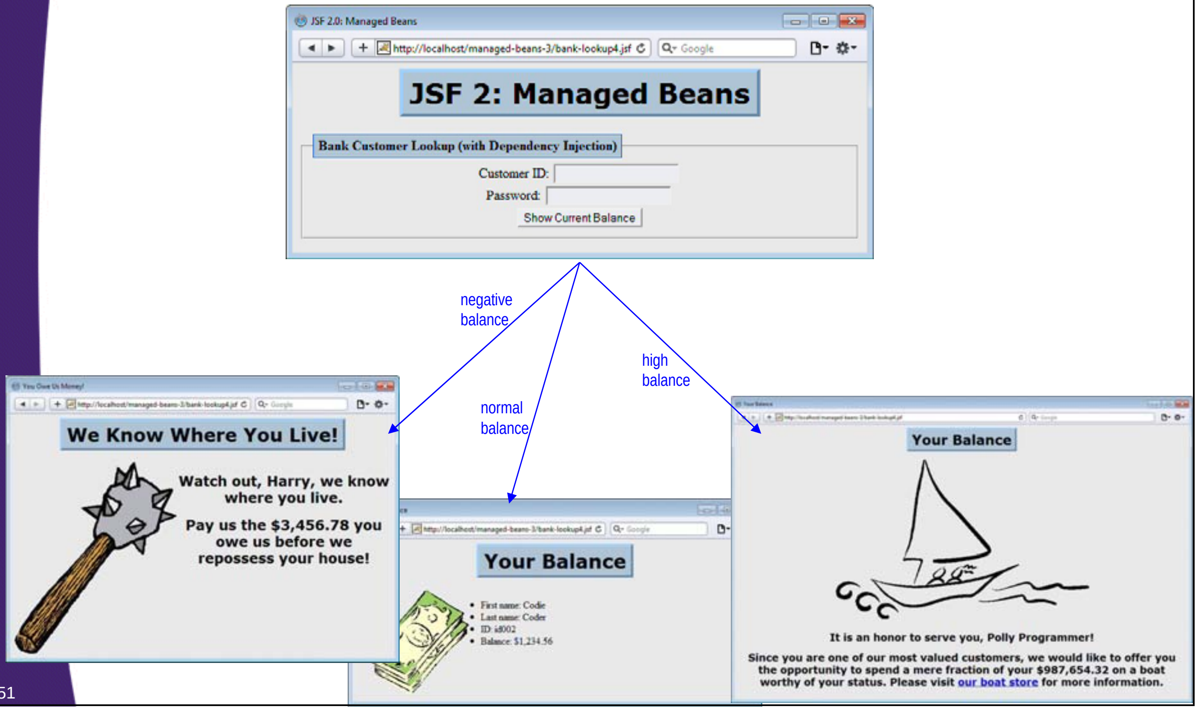
The only reason for the subclass (instead of just putting the annotations on CustomerSimpleMap) is so that I can use "eager" here but not in previous example that used CustomerSimpleMap. Also note that the lookup service is immutable (does not change), so we don't have to worry about synchronizing access to it.

normal-balance4.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<ul>
  <li>First name: #{bankingBean4.customer.firstName}</li>
  <li>Last name: #{bankingBean4.customer.lastName}</li>
  <li>ID: #{bankingBean4.customer.id}</li>
  <li>Balance: $#{bankingBean4.customer.balanceNoSign}</li>
</ul>
…
</h:body></html>
```

Same as normal-balance.xhtml except for bean name.
negative-balance4.xhtml and high-balance4.xhtml are similar.

Results (Same Behavior as First Banking Example)



JSF 2.0: Managed Beans

http://localhost/managed-beans-3/bank-lookup4.jsf

JSF 2: Managed Beans

Bank Customer Lookup (with Dependency Injection)

Customer ID:
Password:
Show Current Balance

We Know Where You Live!

Watch out, Harry, we know where you live.

Pay us the $3,456.78 you owe us before we repossess your house!

Your Balance

- First name: Codie
- Last name: Coder
- ID: id002
- Balance: $1,234.56

Your Balance

It is an honor to serve you, Polly Programmer!

Since you are one of our most valued customers, we would like to offer you the opportunity to spend a mere fraction of your $987,654.32 on a boat worthy of your status. Please visit our boat store for more information.

Wrap-Up

Summary

- **Session scope**
  - Session scope commonly used for user preferences and other user-specific data that must survive from request to request
    - @SessionScoped
    - Can sometimes use redirects instead of forwards, but you must worry about users following bookmarks in sessions with no data
- **The raw request and response objects**
  - Needed for general redirects, headers, long-lived cookies, etc.
    - Get these objects via static method calls on ExternalContext
- **Dependency injection**
  - Code can be more flexible if services (and other classes that commonly change) are passed in instead of hard coded
    - We used @ManagedProperty, but also consider declaring injection in faces-config.xml, or using the Spring framework

Questions?

More info:
http://www.coreservlets.com/JSF-Tutorial/jsf2/ – JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/primefaces/ – PrimeFaces tutorial
http://courses.coreservlets.com/jsf-training.html – Customized JSF and PrimeFaces training courses
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training