

# Java Server Faces (JSF)

# Table of Contents

# Chapter 2

# Managed Beans

# Chapter 2 Outline

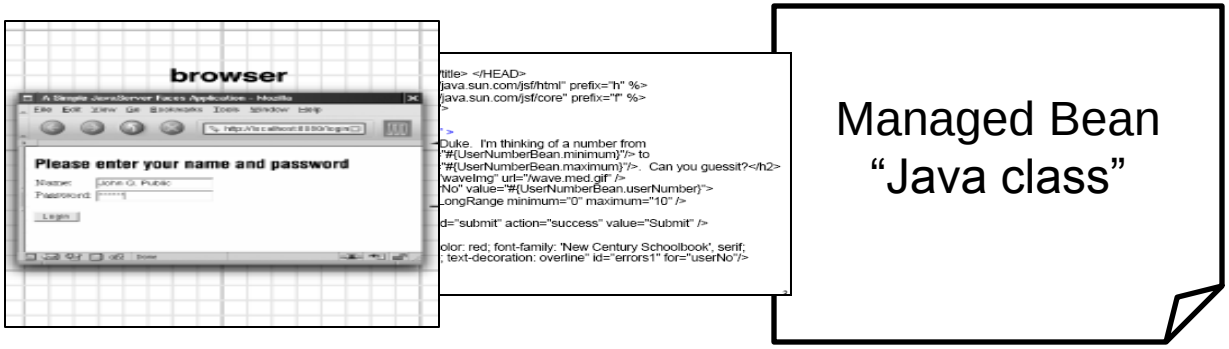
- What is a Bean?
- What is a Managed Bean?
- What is a Backing Bean?
- Bean Scopes
- Injecting Managed Beans
- Bean Life Cycle Annotations
- Configuring Managed Beans with XML
- JSF Expression Language

# What is a Bean?

- **What is a bean ?**
  - a reusable software component .
    - has a public constructor without parameters.
    - Properties (name and type).
    - Rules (getters, setters, services….).
- **Why we are using beans in JSF?**
  - The separation of presentation and business logic is a central theme of web application design
- **Bean Scopes (JSF 1.x):**
  - Request Scope
  - Session Scope
  - Application Scope

# What is a Managed Bean?

- **What is a managed bean ?**
  - JSF automatically "manages" the bean
  - is a JavaBeans object that a JavaServer Faces web application instantiates and stores in any scope (Controls its lifecycle).
  - JSF creates and discards beans as needed.
  - Managed Beans store the state of web pages:
    - Reads bean properties when displaying a web page
    - Sets bean properties when a form is posted



# What is a Managed Bean ? Cont'd

– Example:

- **<h:inputText value="#{user.userName}" >**

```
@ManagedBean(name="user")
@SessionScoped
public class UserBean {
...
}
```

Javax.faces.bean

- Or
- **<h:inputText value="#{userBean.userName}" >**

```
@SessionScoped
public class UserBean {
...
}
```

# What is a Backing Bean?

- is a special type of managed-bean consisting of properties that are UIComponents
- The **JSF page** is a template from which the server generates the HTML response to be displayed in the browser (the client).
- The **page bean** contains the logic that the server executes both when it generates the HTML response and after a user submits the page.
- contains some or all component objects of a web form
- What are Backing Beans ?
  - Beans contain variables like:
    - » A component's value: (Value expression)

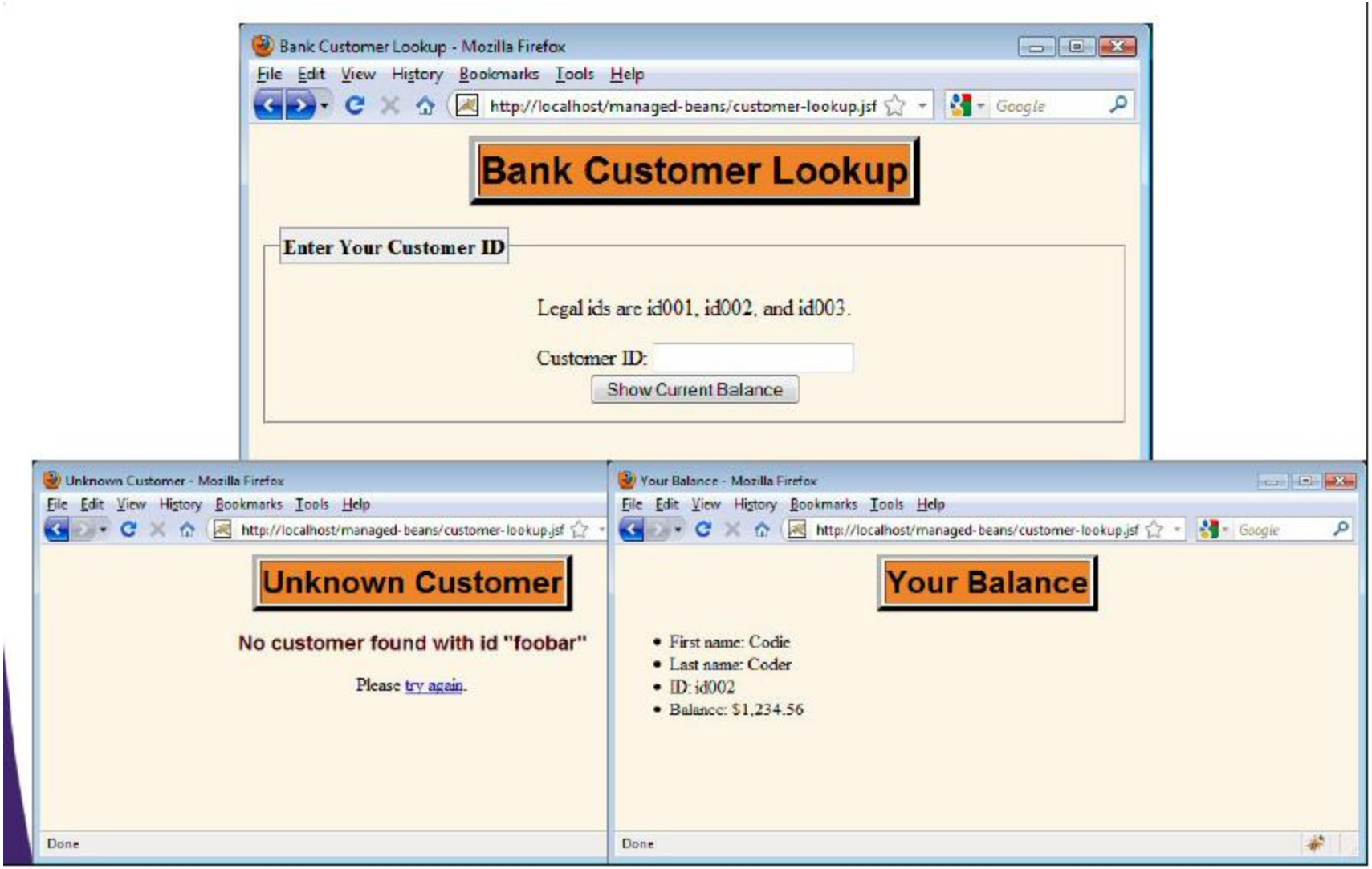
      ```
      <h:inputText value="#{bean1.UIValue}"...>
      ```

    - » A component instance:

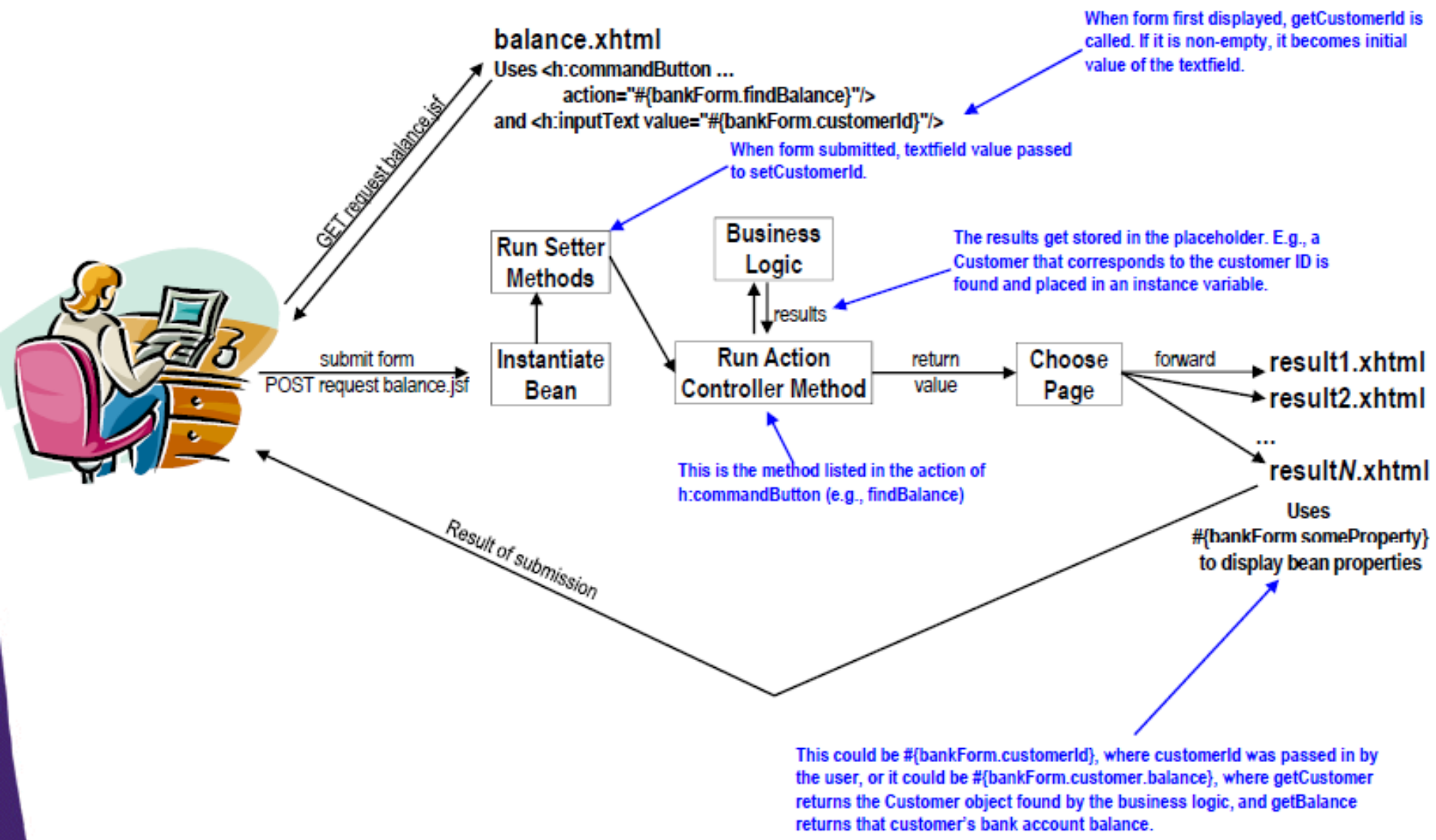
      ```
      <h:inputText binding="#{bean1.UIInstance}"...>
      ```

  - Beans contain Methods like:
    - » Getter and setter methods of properties
    - » Validation methods
    - » Event handler methods
    - » Navigation handling methods

# Example



# Example Cont'd

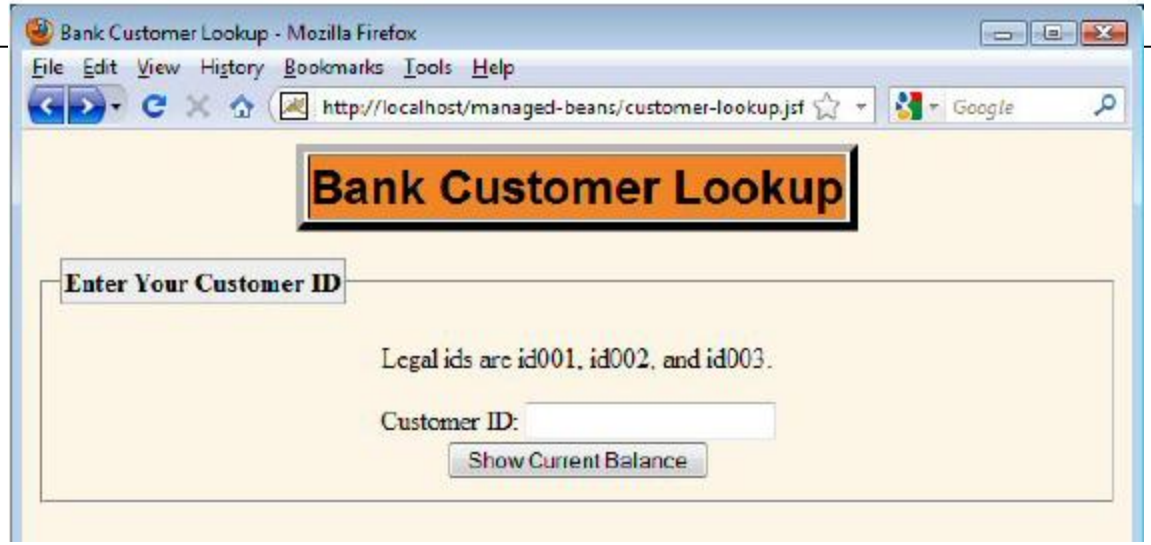


# Example Cont'd

balance.xhtml

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

<h:commandButton value="ShowCurrentBalance"
action="#{bankForm.findBalance}"/>
</h:form>
```



# Example Cont'd

## bankForm

```
public class BankForm {
      private String customerId;
      public String getCustomerId() {
      return(customerId);}
      public void setCustomerId(String customerId) {
      this.customerId = customerId;}

      private Customer customer;  // placeholder
      public Customer getCustomer() {
      return(customer); }

      public String findBalance() {
      customer = ……..;
            if (customer == null)
                  return("unknown-customer");
            else      return("show-customer");}
}
```

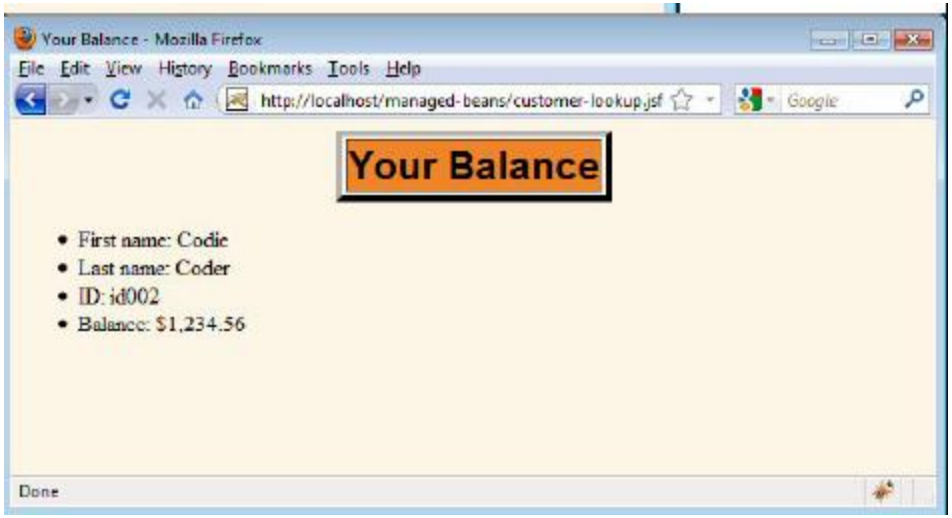
# Example Cont'd

show-balance.xhtml

```
<li>First name: #{bankForm.customer.firstName}</li>

<li>Last name: #{bankForm.customer.lastName}</li>

<li>ID: #{bankForm.customer.id}</li>
```



# Bean Scopes

- JSF container provides separate scopes.
- Each one manages a table of name/value bindings.
- Holds beans and other objects.

- **Annotations:** javax.faces.bean
  - @SessionScoped
  - @RequestScoped
  - @ApplicationScoped
  - @ViewScoped
  - @CustomScoped

# Bean Scopes (cont'd)

- **Application scope:**
  - lasts until the server stops the application.
  - Values that you store in an application bean are available to every session and every request that uses the same application map. ( *@ManagedBean(eager=true)* ) .. constructed before the first page of the application is displayed..
  - Bean either should have no state or you must carefully synchronize access.
- **Session scope:**
  - begins when a user first accesses a page in the web application and ends when the user's session times out due to inactivity, or when the web application invalidates the session.
- **Request scope:**
  - begins when the user submits the page and ends when the response is fully rendered, whatever page that is. (*Data for error and status messages* )

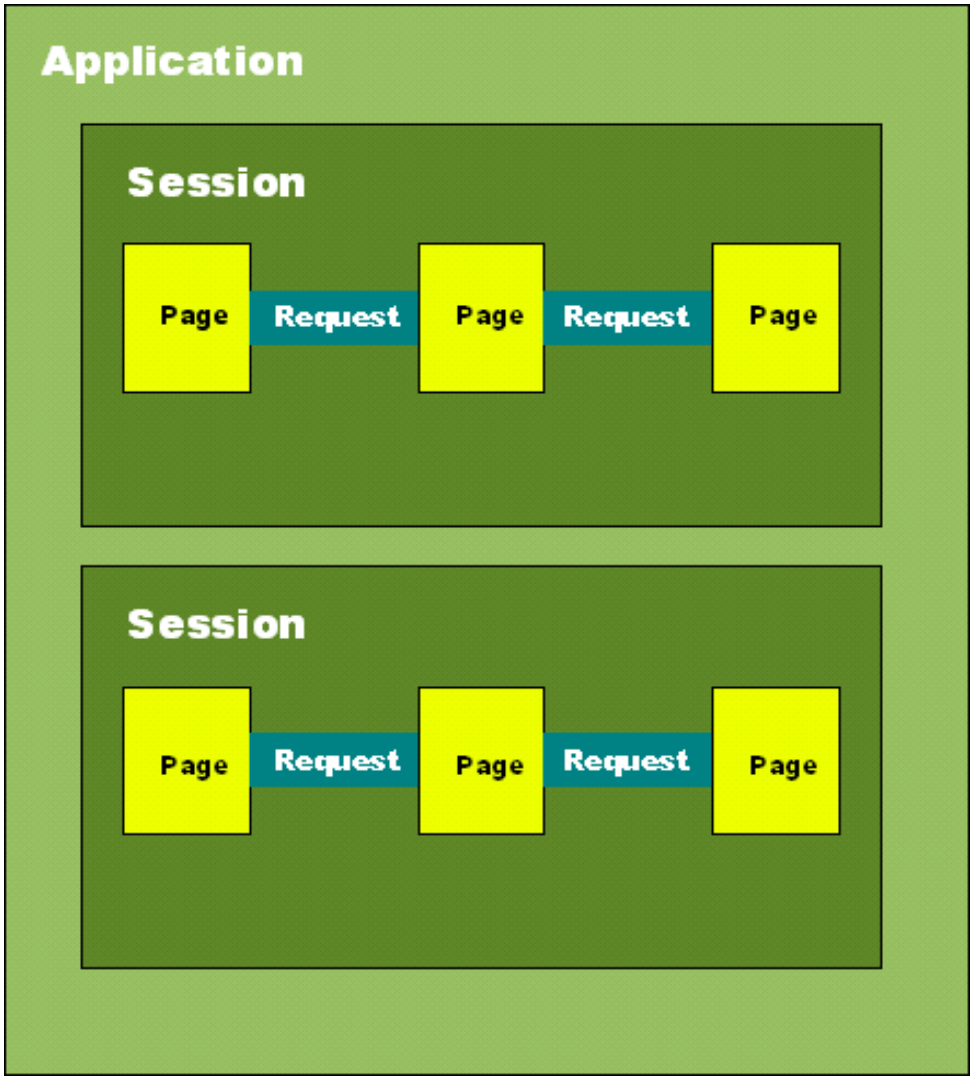
# Bean Scopes (cont'd)

- **View Scope:**
  - was added in JSF 2.0.
  - persists while the same JSF page is redisplayed. (*Page Scope*)
  - is particularly useful for Ajax applications

- **Custom scope:**
  - was added in JSF 2.0.
  - custom scopes—maps whose lifetimes you manage.
  - application is responsible for removing objects from the map

*@CustomScoped("#{expr}")*

# Bean Scopes (cont'd)



# Injecting Managed Beans

- @**ManagedProperty** annotation:

```
@ManagedBean (name= “edit”)
@SessionScoped

public class EditBean implements Serializable {

@ManagedProperty(value="#{user}")
private UserBean       currentUser;  // has another Bean with name user

public void setCurrentUser(UserBean newValue) { currentUser = newValue; } // Must
. . .
}
```

# Injecting Managed Beans (Cont'd)

- @**ManagedProperty** annotation:
  - The Scopes for these bean **must** be compatible..
  - The scope of the property **must** be no less than that of the containing bean.

| Container / property | Request | View | Session | Application |
|---|---|---|---|---|
| **Request** | Valid | | | |
| **View** | Valid | Valid | | |
| **Session** | Valid | Valid | Valid | |
| **Application** | Valid | Valid | Valid | Valid |

# Bean Life Cycle Annotations

- **@PostConstruct** and **@PreDestroy** annotations:

```
public class MyBean {
    @PostConstruct
    public void initialize() {
    // initialization code
    }
    @PreDestroy
    public void shutdown() {
    // shutdown code
    }
}
```

# Configuring Managed Beans with XML

- Before JSF 2.0:
  - all beans had to be configured with XML.
- JSF 2.0:
  - Annotations (at development Time)
  - Or XML configuration (at deployment Time):
    - WEB-INF/faces-config.xml
    - faces-config.xml or ending with .faces-config.xml inside the META-INF directory of a JAR file

# Configuring Managed Beans with XML (Cont'd)

- **Defining Beans:**

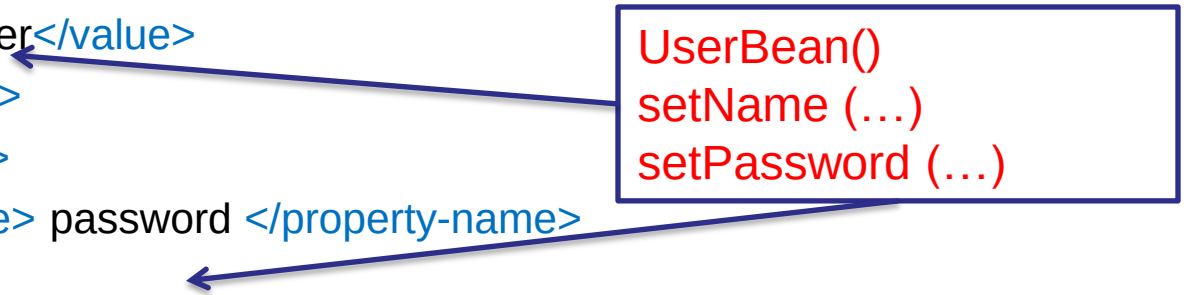
```
<faces-config>
    <managed-bean>
    <managed-bean-name> user </managed-bean-name>
    <managed-bean-class> com.corejsf.UserBean </managed-bean-class>
    <managed-bean-scope> session </managed-bean-scope>
    </managed-bean>
</faces-config>
```

- The scope can be: request, view, session, application, none, or a value expression that yields a custom scope map.

# Configuring Managed Beans with XML (Cont'd)

- **Setting Property Values:**

```
<managed-bean>
    <managed-bean-name> user </managed-bean-name>
    <managed-bean-class> com.corejsf.UserBean </managed-bean-class>
    <managed-bean-scope> session </managed-bean-scope>
    <managed-property>
        <property-name> name </property-name>
        <value> testUser</value>
    </managed-property>
    <managed-property>
        <property-name> password </property-name>
        <null-value/>
    </managed-property>
</managed-bean>
```

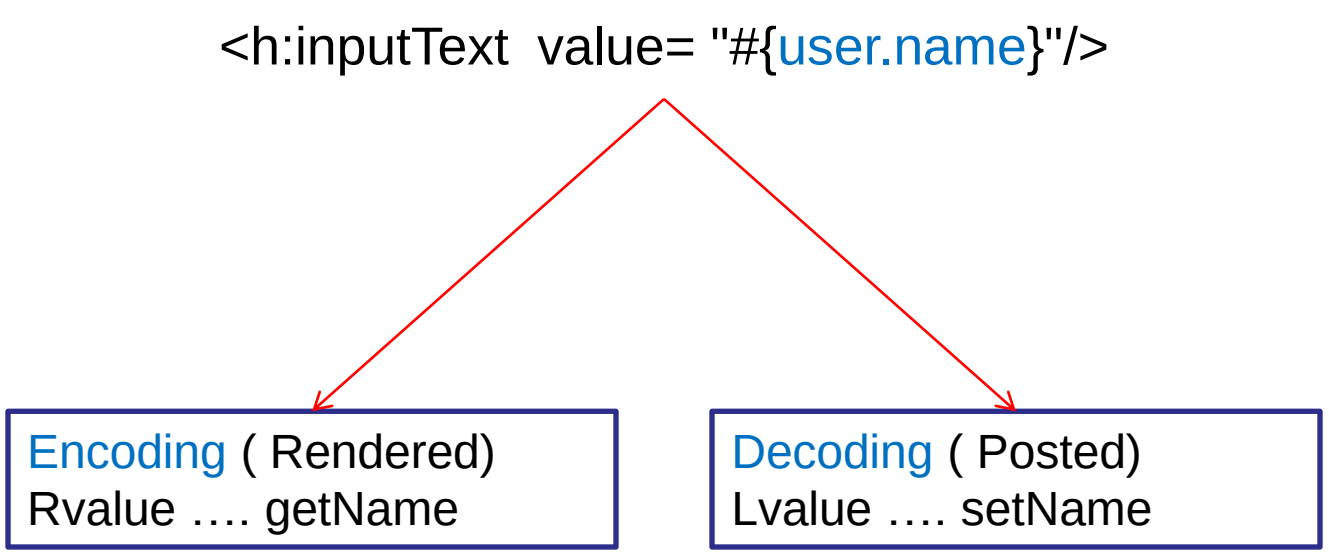


# Configuring Managed Beans with XML (Cont'd)

- **Setting Property Values (Cont'd):**

```
<managed-bean>
    <managed-bean-name> editBean </managed-bean-name>
    <managed-bean-class> com.corejsf.EditBean </managed-bean-class>
    <managed-bean-scope> session </managed-bean-scope>
    <managed-property>
        <property-name> user</property-name>
        <value> #{user}</value>
    </managed-property>
</managed-bean>
```

# JSF Expression Language



# JSF Expression Language (Cont'd)

- **Using Brackets:**

user.password  user["password"]  user['password']

msgs["error.password"]

value="#{user['password']}"  value='#{user["password"]}'.

# JSF Expression Language (Cont'd)

- **Calling Methods and Functions:**

  #{stockQuote.price("ORCL")}

- if the stockQuote bean has a method *double price(String)*
- Overloaded methods are not supported.
- JSTL functions library: xmlns:fn=http://java.sun.com/jsp/jstl/functions
- <……value="#{fn:toUpperCase(myBean.greeting)}"…..>

| Functions | Description |
|---|---|
| fn:replace(str, from, to) | Returns the result of replacing all occurrences of from in str with to. |
| fn:toLowerCase(str) | Returns the lowercase of str. |
| fn:toUpperCase(str) | Returns the uppercase of str. |
| fn:trim(str) | Returns str with leading and trailing whitespace removed. |
| fn:escapeXml(str) | Returns str with characters < > & escaped as XML entities. |

# JSF Expression Language (Cont'd)

- **Implicit Objects:**
  - header
  - headerValues
  - param
  - paramValues
  - cookie
  - initParam
  - requestScope
  - sessionScope
  - applicationScope
  - facesContext
  - View

header['User-Agent']

- viewScope
- resource
- component
- CC

# JSF Expression Language (Cont'd)

- **Composite Expressions:**
  - Arithmetic operators: + , - , * , / , %.
  - Relational operators: < , <= , > , >= , == , !=
    and their alphabetic variants lt , le , gt , ge , eq , ne
  - logical operators: && , || , !
    and their alphabetic variants AND , OR , NOT
  - the ***empty*** operator:
  - the ternary: ?: selection operator

# JSF Expression Language (Cont'd)

- **Examples:**
  - <h:inputText rendered="#{!bean.hide}" ... />
  - <h:commandButton value="#{msgs.clickHere}, #{user.name}!"/>
  - <h:commandButton action="#{user.checkPassword}"/>
  - <h:commandButton value="Previous" action="#{formBean.move(-1)}"/>
  - <h:commandButton value="Next" action="#{formBean.move(1)}"/>

```
public class FormBean {
    ...
    public String move(int amount) { ... }
}
```

# Lab Exercise

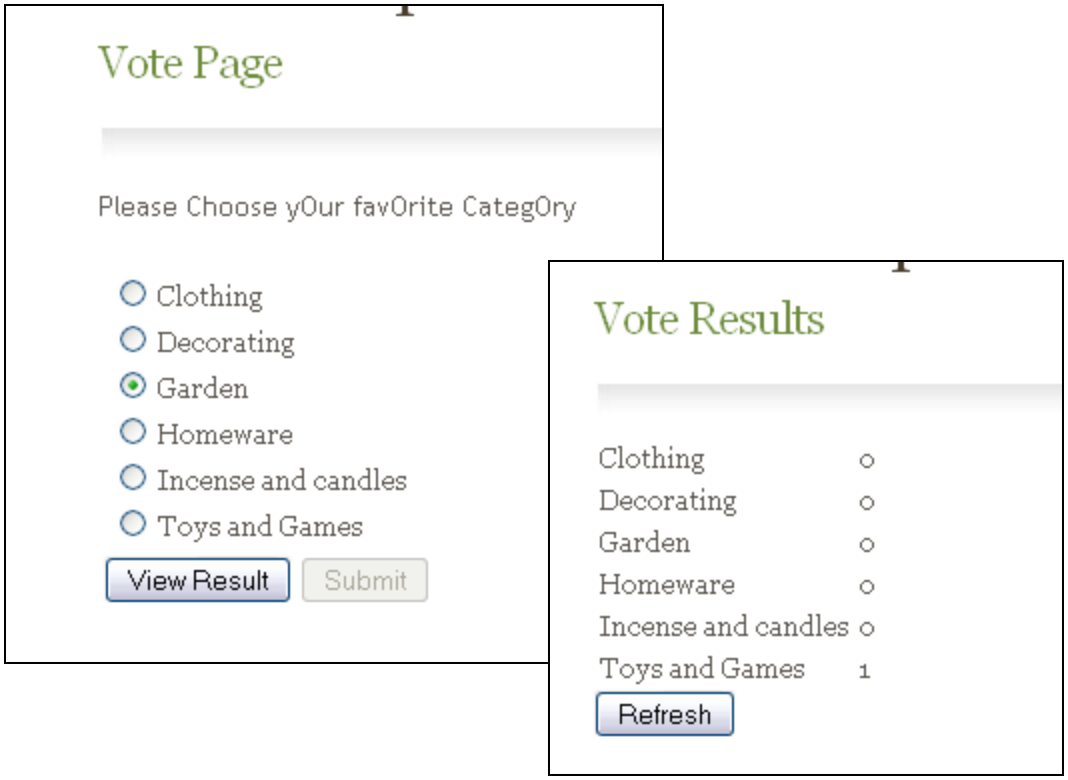
# Assignments

- Getting familiar with different  Java Beans Scopes

- Make a web application that uses:
  - **Options.** An array property that contains the list of ballot choices. Because the list remains the same for all sessions, this property goes in application scope.
  - **votes.** A hash map property that accumulates all the session votes. Because it must persist across sessions, it goes in application scope.
  - **hasVoted.** A boolean property that tracks whether the user has voted. Because the application needs to persist the value across several requests in a single session, the application stores the value in session scope.

# Assignments

- Getting familiar with different  Java Beans Scopes



# Assignments

- Getting familiar with different Java Beans Scopes


```
<h:selectOneRadio value="#{voteBean.choosen}" layout="pageDirection">

                <f:selectItems value="#{listBean.categoryList}" var="c"
                itemLabel="#{c.categoryLabel}" itemValue="#{c.categoryValue}"  />

</h:selectOneRadio>
```

```
<h:commandButton value="Submit" action="#{voteBean.voteButtonBack}"
disabled="#{voteBean.voted}">
```

List Bean:
•Category[] categoryList;
•Constructor.
•public void incrementCategory(String name)

Class Category:
•String categoryLabel;
•String categoryValue;
•int categoryVotes;

Vote Bean:

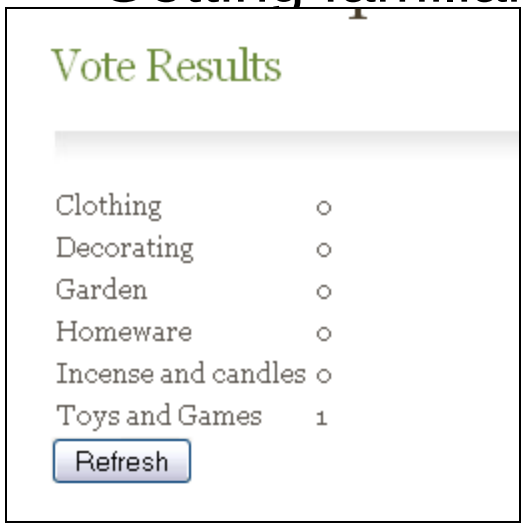
```
•@ManagedProperty(value ="#{listBean}")
ListArrayBean listBean;

•boolean voted;
•String choosen;

•public String voteButtonBack() {
        if (getChoosen() != null) {
            listBean.incrementCategory(getChoosen());
            setVoted(true);
            return "Results";
        } else {
            return null;}
    }
```

# Assignments

- Getting familiar with different  Java Beans Scopes




```
<table >
  <c:forEach items="#{listBean.categoryList}" var="v" >
    <tr><td>
            #{v.categoryLabel}
        </td>
        <td>
            #{v.categoryVotes}
        </td>
     </tr>
   </c:forEach>
</table>
```

```
<meta content="10;url=Results.xhtml" http-equiv="refresh"/>
```