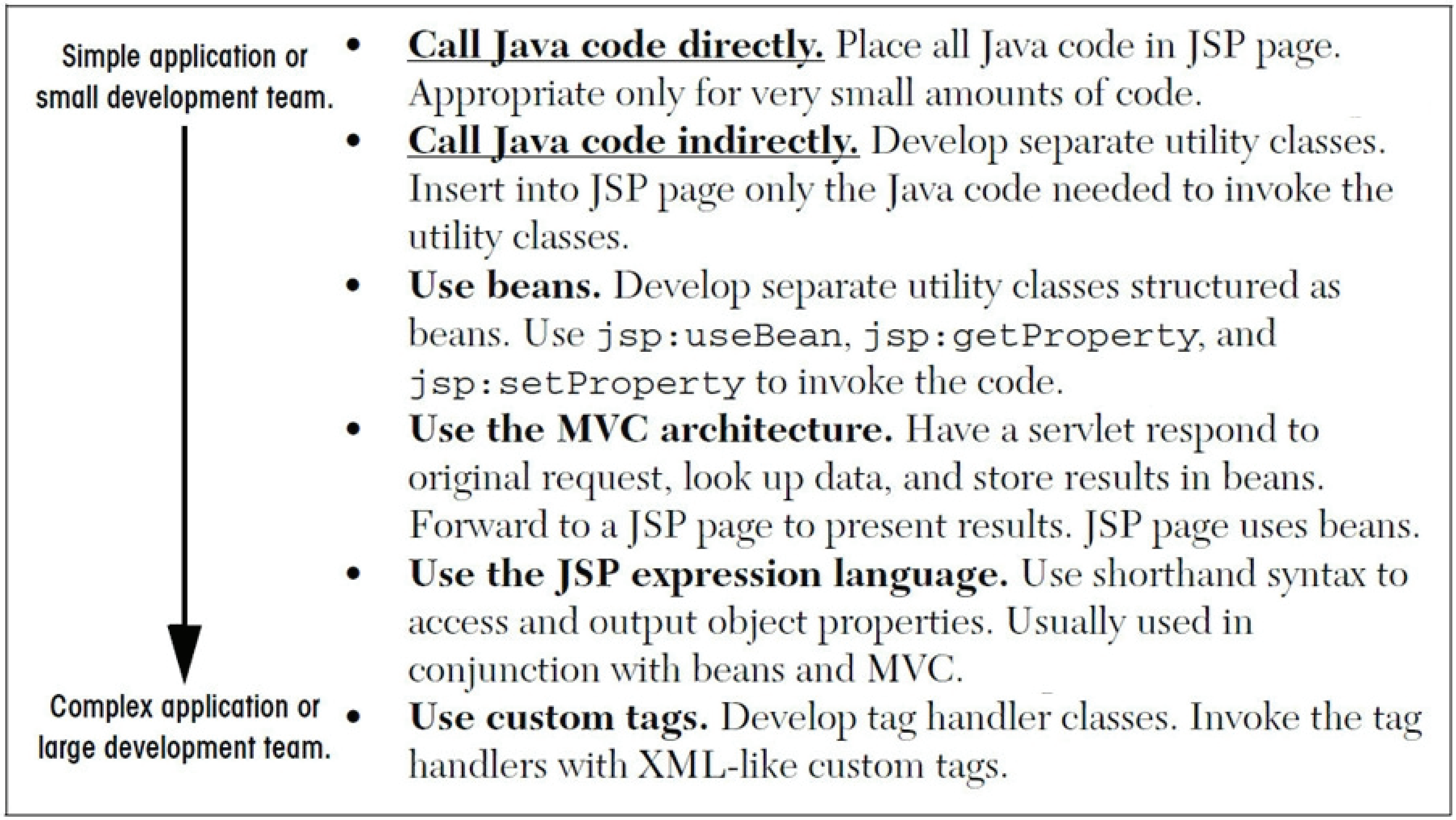
Using JavaBeans Components

Topics Covered:

- Understanding the benefits of beans
- Creating beans
- Installing bean classes on your server
- Accessing bean properties
- Explicitly setting bean properties
- Automatically setting bean properties from request parameters
- Sharing beans among multiple servlets and JSP pages

We'll discuss the third general strategy for inserting dynamic content in JSP pages by means of JavaBeans components.


Simple application or small development team.
Complex application or large development team.

- **Call Java code directly.** Place all Java code in JSP page. Appropriate only for very small amounts of code.
- **Call Java code indirectly.** Develop separate utility classes. Insert into JSP page only the Java code needed to invoke the utility classes.
- **Use beans.** Develop separate utility classes structured as beans. Use `jsp:useBean`, `jsp:getProperty`, and `jsp:setProperty` to invoke the code.
- **Use the MVC architecture.** Have a servlet respond to original request, look up data, and store results in beans. Forward to a JSP page to present results. JSP page uses beans.
- **Use the JSP expression language.** Use shorthand syntax to access and output object properties. Usually used in conjunction with beans and MVC.
- **Use custom tags.** Develop tag handler classes. Invoke the tag handlers with XML-like custom tags.

Limiting the Amount of Java Code in JSP Pages

- As we have discussed the benefit of using separate Java classes instead of embedding large amounts of code directly in JSP pages, separate classes are easier to
  - Write
  - Compile
  - Test
  - Debug
  - Reuse

what do beans provide that other classes do not?

- After all, beans are merely regular Java classes that follow some simple conventions defined by the JavaBeans specification;
  - beans extend no particular class
  - are in no particular package, and
  - use no particular interface.
- Although it is true that beans are merely Java classes that are written in a standard format, there are several advantages to their use.
- With beans in general, visual manipulation tools and other programs can automatically discover information about classes that follow this format and can create and manipulate the classes without the user having to explicitly write any code.

advantages of using JavaBeans components over scriptlets

- **No Java syntax.**
  - By using beans, page authors can manipulate Java objects using only XML-compatible syntax: no parentheses, semicolons, or curly braces. This promotes a stronger separation between the content and the presentation and is especially useful in large development teams that have separate Web and Java developers.
- **Simpler object sharing.**
  - When you use the JSP bean constructs, you can much more easily share objects among multiple pages or between requests than if you use the equivalent explicit Java code.
- **Convenient correspondence between request parameters and object properties.**
  - The JSP bean constructs greatly simplify the process of reading request parameters, converting from strings, and putting the results inside objects.

What Are Beans?

- Beans are simply Java classes that are written in a standard format to expose data through properties (attributes).

- Full coverage of JavaBeans is beyond the scope of this class, but for the purposes of use in JSP, all you need to know about beans are the three simple points outlined in the following list.

1. A bean class must have a zero-argument (default) constructor.

- You can satisfy this requirement either
  - by explicitly defining such a constructor or
  - by omitting all constructors

Note:

- "default constructor" refers to a nullary constructor that is automatically generated by the compiler if no constructors have been defined for the class.
- The default constructor is also empty, meaning that it does nothing.
- A user defined constructor that takes no parameters is called a default constructor too.

2. A bean class should have no public instance variables (fields).

- To be a bean that is accessible from JSP, a class should use accessor methods instead of allowing direct access to the instance variables.
  - You should already be familiar with this practice since it is an important design strategy in object-oriented programming.

3. Persistent values should be accessed via ***getXxx*** *and* ***setXxx****.*

- For example,
  - if your Car class stores the current number of passengers, you might have methods named getNumPassengers and setNumPassengers.
  - In such a case, the Car class is said to have a *property named* numPassengers.

- If the class has a ***getXxx*** method but no ***setXxx***, *the class is said to have a* read-only property named *xxx.*

- The one exception to this naming convention is with boolean properties: they are permitted to use a method called ***isXxx*** *to look* up their values.

- So, for example, your Car class might have methods called isLeased and setLeased, and would be said to have a boolean property named leased

Although you can use JSP scriptlets or expressions to access arbitrary methods of a class, standard JSP actions for accessing beans can only make use of methods that use the get*Xxx*, *setXxx or isXxx*, set*Xxx naming convention.*

Building a JavaBean

So what does a JavaBean look like?

- For this example, we'll define a JavaBean class called CarBean that we could use as a component in a car sales Web site.
- It'll be a component that will model a car and have one property—the make of the car.

Here's the code:

```
package packageName;

public class CarBean
{
   private String make = "Ford";

   public CarBean() {}

   public String getMake()
   {
     return make;
   }

   public void setMake(String make)
   {
     this.make = make;
   }
}
```

Using Beans – Basic Tasks

- jsp:useBean
- jsp:getProperty
- jsp:setProperty.

jsp:useBean

- In the simplest case, this element builds a new bean.
- It is normally used as follows:

```
<jsp:useBean id=“beanName” class=“package.ClassName” scope=“scopeType” />
```

- If you supply a scope attribute, the jsp:useBean element can either build a new bean or access a preexisting one.

jsp:getProperty

- This element reads and outputs the value of a bean property.
- Reading a property is a shorthand notation for calling a method of the form get*Xxx*
- *This element is used as follows:*

```
<jsp:getProperty name=“beanName” property=“propertyName” />
```

`jsp:setProperty`

- This element modifies a bean property (i.e., calls a method of the form set*Xxx).*
- *It is normally used as follows:*

```
<jsp:setProperty name="beanName“ property="propertyName“ value="propertyValue" />
```

Installing Bean Classes

- The bean class definition should be placed in the same directories where servlets can be installed, *not in the directory that contains the JSP file.*
- *Just remember to use* packages.

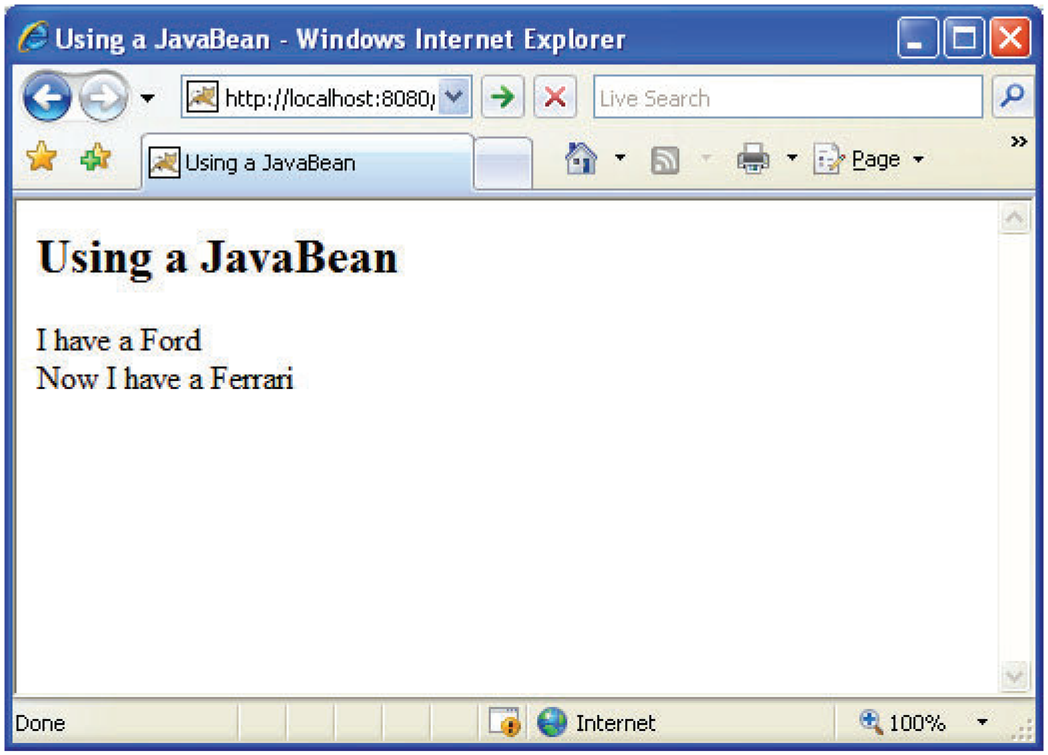
In-class exercise: creating a simple bean

```
package com.me.cars;

public class CarBean
{
  private String make = "Ford";

  public CarBean() {}

  public String getMake()
  {
    return make;
  }

  public void setMake(String make)
  {
    this.make = make;
  }
}
```

Now, create a JSP page to use the Bean

```
<jsp:useBean id="myCar" class="com.yusuf.cars.CarBean" />

<html>
    <head>
        <title>Using a JavaBean</title>
    </head>
    <body>

    <h2>Using a JavaBean</h2>

    I have a <jsp:getProperty name="myCar" property="make" /><br>

    <jsp:setProperty name="myCar" property="make" value="Ferrari" />

    Now I have a <jsp:getProperty name="myCar" property="make" />

    </body>
</html>
```

Here is the output



Using a JavaBean

```
I have a Ford
Now I have a Ferrari
```

Sharing Beans

- We have treated the objects that were created with *jsp:useBean*
- As though they were simply bound to local variables in the _jspService method (which is called by the service method of the servlet that is generated from the page).
- Although the beans are indeed bound to local variables, that is not the only behavior.
- They are also stored in one of four different locations, depending on the value of the optional scope attribute of jsp:useBean.

```
<jsp:useBean ... scope="page" />          (default)
<jsp:useBean ... scope="request" />
<jsp:useBean ... scope="session" />
<jsp:useBean ... scope="application" />
```

Using scope

- When you use scope, the system first looks for an existing bean of the specified name in the designated location.
- Only when the system fails to find a preexisting bean, it creates a new one.
- This behavior lets a servlet handle complex user requests by
  - setting up beans,
  - storing them in one of the standard shared locations (the request, the session, or the servlet context),
  - then forwarding the request to one of several possible JSP pages to present results appropriate to the request data.
- We'll discuss this approach (Model View Controller Architecture) next.

JavaBeans or Enterprise JavaBeans?

- **EJBs are an advanced topic and beyond the scope of this class.**
- JSP is one part of the Java 2 Enterprise Edition (J2EE) architecture, namely the *presentation tier*.
- Enterprise JavaBeans (EJBs) are another part of this architecture.
- However, as you create bigger and better JSP Web applications, you'll inevitably come across the term.
- We want to warn you not to confuse the JavaBeans you've learned about in this lecture with EJBs.
- Although they share a similar name, they have very different capabilities, designs, and uses.
- JavaBeans are general-purpose components that can be used in a variety of different applications, from very simple to very complex.
- EJBs, on the other hand, are components designed for use in complex business applications.

  They support features commonly used in these types of programs, such as automatically saving and retrieving information to a database, performing many tasks in a single transaction that can be safely aborted if parts of the transaction fail, or communicating other Java components across a network and so on
- Although you could accomplish any one of these EJB features with normal JavaBeans, EJBs make using these complex features easier.
- *However, EJBs are considerably more complicated to understand, use, and maintain than JavaBeans, and you'll have your hands full learning JSP and Servlets in this class, so we won't discuss EJBs.*

Summary

- JavaBeans are a simple but helpful addition to JSP.
- As used by JSP, a JavaBean is really nothing more than a fancy name for a way to code a Java class.
- By following certain design restrictions, it is easy to create a set of JSP actions that can manipulate and use any of those classes.
- JavaBeans are an example of a set of design restrictions, primarily get and set methods, and the JavaBean standard actions are available for use with JSP.