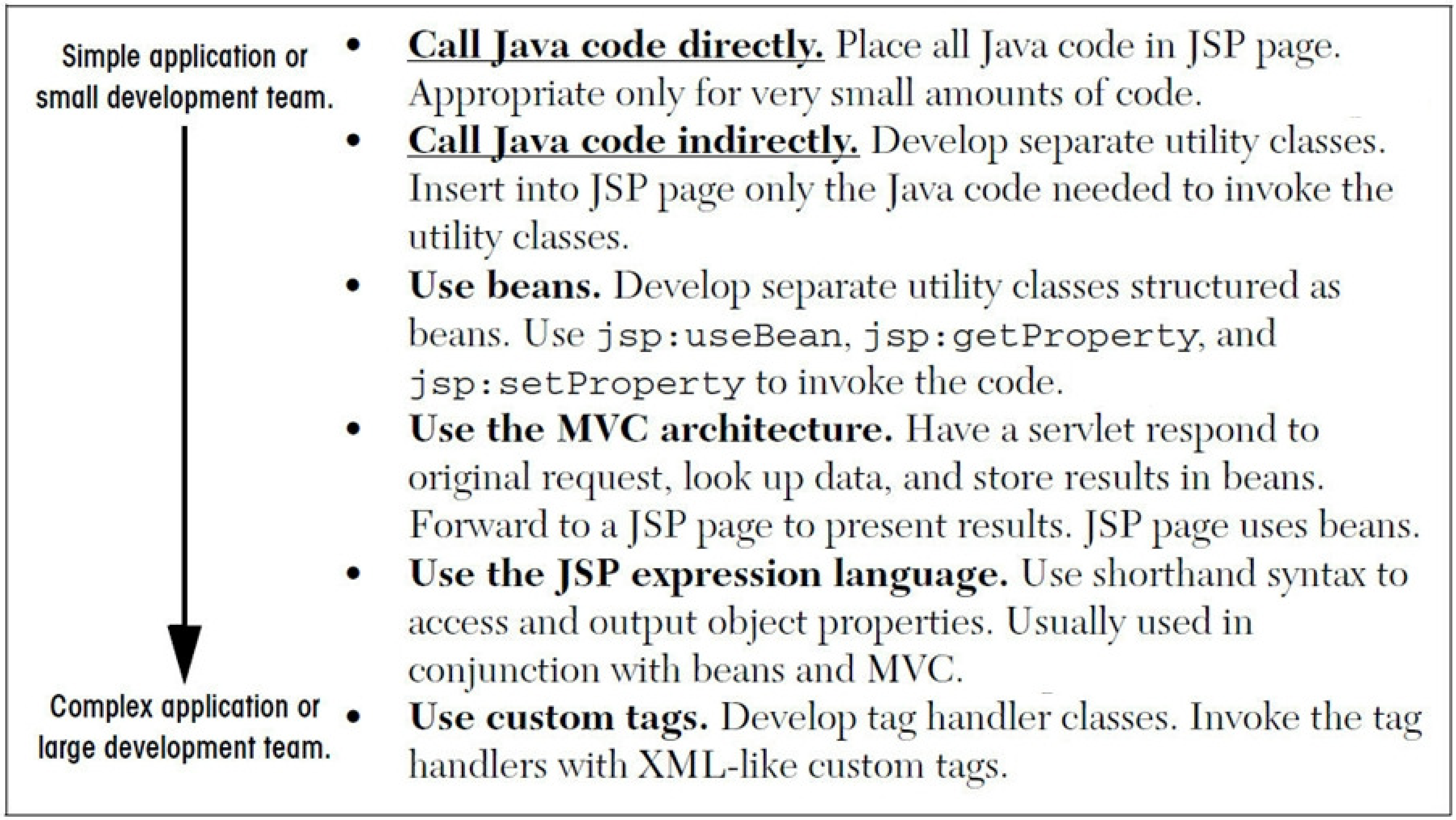
Using JavaBeans Components

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.

- **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.

Complex application or large development team.

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.

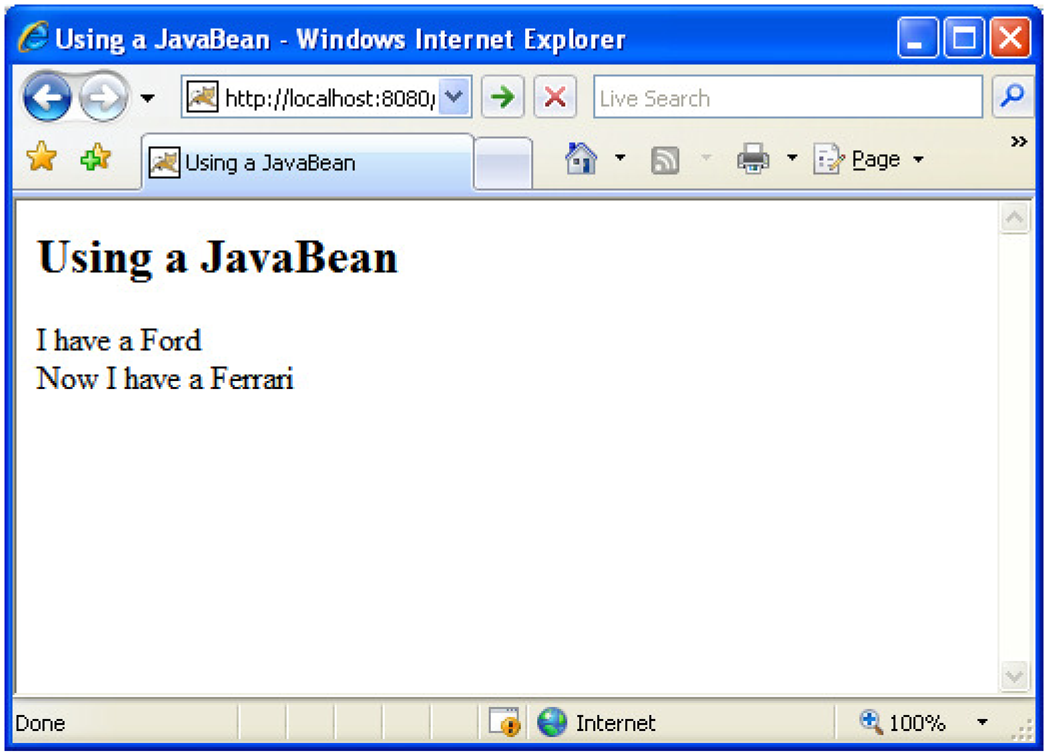
In-class exercise: creating a simple bean

```
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.

Architecture of Web Applications

- One approach to Web application development is to gather the requirements and quickly chalk out some JSP pages and JavaBeans.
- Though this approach would probably work to some degree,
  - the end result is unlikely to be maintainable,
  - it may well be hard to change a certain feature or to introduce new features without breaking other parts of the application.
  - It's also probably not *reusable*.
- If you need to introduce new functionality, you'd have to repeat a lot of work that you had already done but that's locked up in JSP pages and beans in such a way that you can't use it.
- By structuring your applications correctly, you can go much further toward achieving the goals of maintainability and reusability.

- **When considering how to construct an application, you must decide how to divide up responsibilities and how the pieces will interact with one another.**
- In a Web application, that means that you'll assign work to static HTML pages, JSPs, servlets, and other objects. The container and application server will pitch in and take care of some things for you.
- You want applications that are useful, easy to construct, and maintainable.
- Because requirements or uses seem to always change, you're also interested in software that is flexible or extensible.
- **The trick in building software that does all of this for you is to properly delegate the work to the right components.**
- For example, presenting static content to the client can be accomplished using a plain HTML page or a servlet or JSP. Writing a servlet to do this is more work than you need to do and makes the application complex.
- Also, you probably recognize that, in the workplace, employees are often assigned tasks making the best use of their skills.
- From a business perspective, it's useful to design software so that the work can be divided into parcels that are completed by employees whose training and talents are appropriate to the tasks.
- **One of the objectives of the JSP and servlet specifications is to make it possible to neatly divide the labor between graphics designers who work on the presentation and programmers who define the behavior of an application.**

Why Use a Design Pattern?

- **Reduce Development Time:**

  A good design pattern helps conceptually to break down a complex system into manageable tasks. This allows developers to individually code parts of the Web Application they are best suited for. It also allows individual components to be built and replaced without harming the existing code base.

- **Reduced Maintenance Time:**

  The majority of projects involve maintenance of an already existing system. Maintenance can be a nightmare should a system be hacked together when it was initially built. Good design plans ahead to simplify future maintenance concerns.

- **Collaboration:**

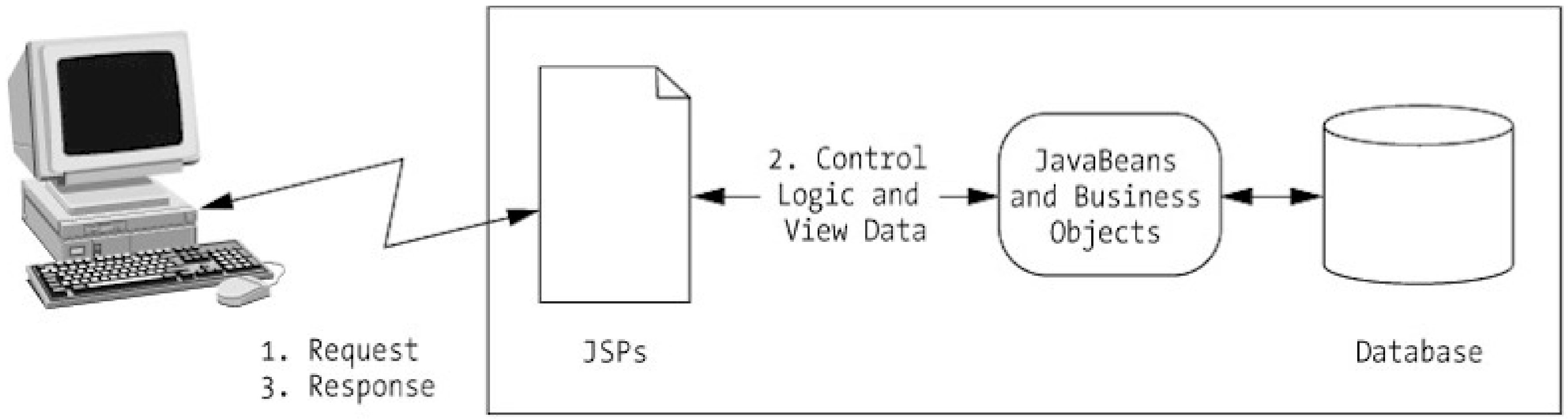
  Not all developers share the same expertise. Often a mixed group of developers is assigned to a task especially in the case of a larger project. A good design can successfully enforce separation of a project's functionality into areas that collaborating developers are most familiar, and ensure separate parts of a project seamlessly fit together

Model 1 Architecture

- Model 1 is used to refer to what is usually the intuitive approach to using JSP and Servlets; a Model 1 architecture is what a new JSP and Servlet developer are likely to build.
- The concept behind a Model 1 architecture is simple: code functionality wherever the functionality is needed.
- The approach is very popular because it's both simple and provides instant gratification.
  - Should security be needed, code it in.
  - Should a JSP need information from a database, code in the query.
- In the Model 1 architecture, JSPs accept client requests, decide which actions to take next, and present the results.
- JSPs work with JavaBeans or other services to affect business objects and generate the content.
- **Model 1 is acceptable for applications containing up to a few thousand lines of code, and especially for programmers, but the JSP pages still have to handle the HTTP requests, and this can cause headaches for the page designers.**

Model 1 Architecture

In Model 1 architecture, the application is page-centric. The client browser navigates through a series of JSP pages in which any JSP page can employ a JavaBean that performs business operations. However, the highlight of this architecture is that each JSP page processes its own input. Applications implementing this architecture normally have a series of JSP pages where the user is expected to proceed from the first page to the next. If needed, a servlet or an HTML page can be substituted for the JSP page in the series.


2. Control Logic and View Data
JavaBeans and Business Objects
1. Request
3. Response
JSPs
Database

Features of the JSP Model 1 Architecture

- You use HTML or JSP files to code the presentation.
- The JSP files can use JavaBeans or other Java objects to retrieve data if required.
- JSP files are also responsible for all the business and processing logic, such as receiving incoming requests, routing to the correct JSP page, instantiating the correct JSP pages, and so on.
- This means that the Model 1 architecture is a page-centric design: All the business and processing logic is either present in the JSP page itself or called directly from the JSP page.
- Data access is usually performed using custom tags or through JavaBean calls.
- Some quick-and-dirty projects use scriptlets in their JSP pages instead.
- Therefore, there's a tight coupling between the pages and the logic in Model 1 applications.
- The flow of the application goes from one JSP page to another using anchor links in JSP pages or the action attribute of HTML forms.
- This is the only kind of application you've seen so far.

Drawbacks of the JSP Model 1 Architecture

- The Model 1 architecture has one thing going for it:
  - simplicity.
- If your project is small, simple, and self-contained, it's the quickest way to get up and running. But the previous example, although by no means large or complicated, already illustrates a number of the disadvantages of Model 1 architecture:
- It becomes very hard to change the structure of such Web applications because the pages are *tightly coupled*.
- They have to be aware of each other. What if you decide that, after updating the quantities in a shopping cart, you want to redirect the user back to the catalog? This could require moving code from the shopping cart page to the catalog page.
- Large projects often involve teams of programmers working on different pages, and in the Model 1 scenario, each team would have to have a detailed understanding of the pages on which all of the other teams were working; otherwise, modifying pages could break the flow of the application.

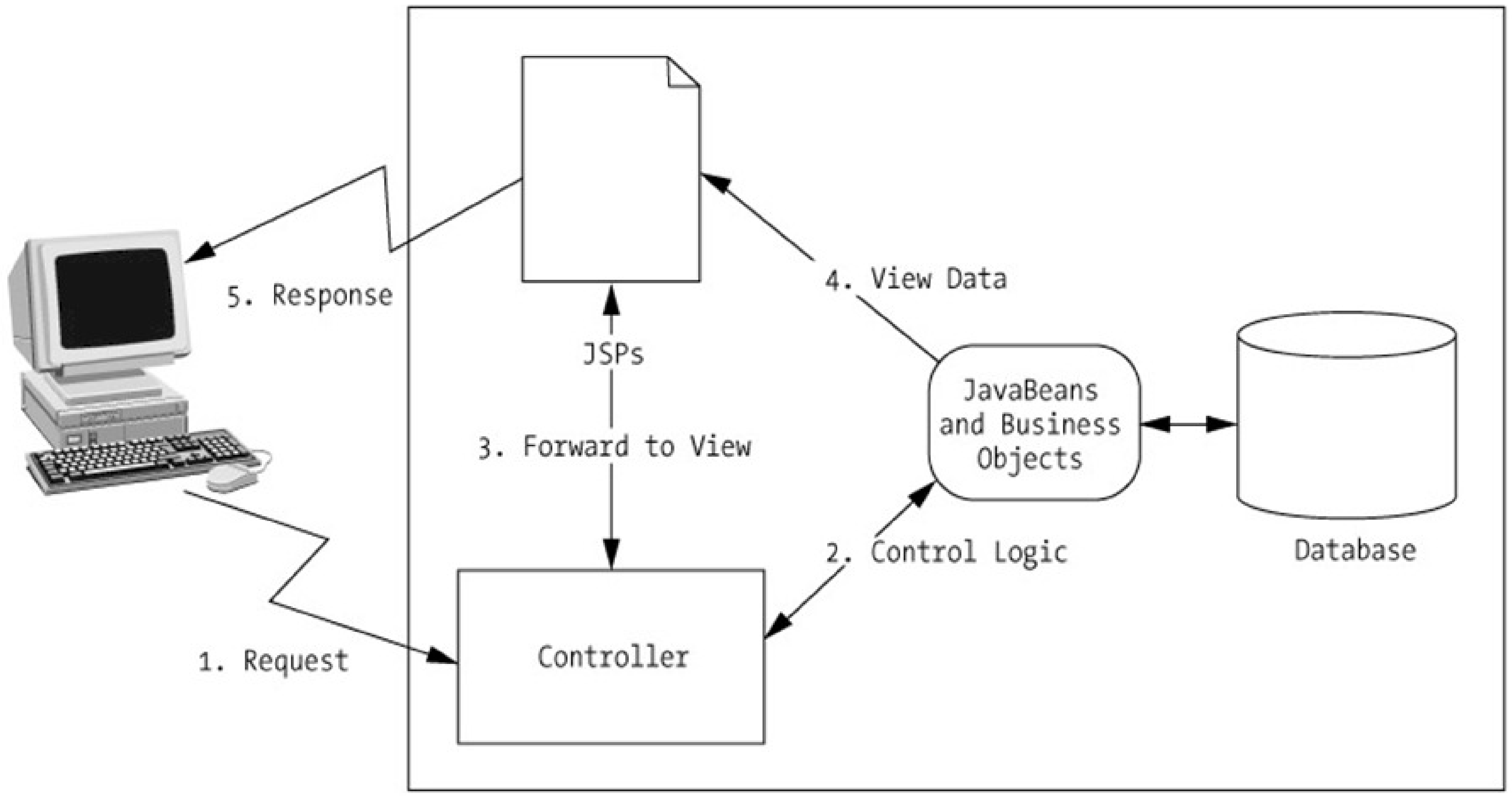
Drawbacks Cont'd

- Pages that are linked to from many other pages have to handle those other pages' logic, such as a cart update. In this way, they can accumulate a large amount of code that hasn't got an awful lot to do with the page itself. This reduces their *coherence*, making them harder to understand and to maintain.
- Presentation and application control logic are mixed up, making it hard for a Web designer to change the pages without messing up the Java code.
- And vice versa: it's very hard for a developer to change control logic that may be hidden in lots of HTML markup.
- The lack of separation between presentation and logic means that providing multiple presentations carries a very high cost.
- What if you decide to sell pizzas via Wireless Application Protocol (WAP) or personal digital assistant (PDA) devices?
  - These devices are radically different from Web browsers and require completely different presentation structure and formatting. Your only choice would be to produce a whole new set of pages and duplicate the application logic.
  - You'd have to implement every subsequent change to the logic in more than one place. Soon enough, changes will become very hard to manage, and the implementations will start to diverge.

JSP Model 2 Architecture

- A better solution, also suitable for larger applications, is to separate application logic and page presentation.
- This solution comes in the form of the JSP Model 2 architecture, also known as the model-view-controller (MVC)
- With this model,
  - a servlet
    - processes the request,
    - handles the application logic, and
    - instantiates the Java beans.
  - JSP
    - obtains data from the beans and
    - can format the response without having to know anything about what's going on behind the scenes.

Model 2


5. Response
JSPs
4. View Data
JavaBeans and Business Objects
3. Forward to View
Database
2. Control Logic
Controller
1. Request

Understanding the Need for MVC

- Servlets are great when your application requires a lot of programming to accomplish its task.
- As discussed in the previous lectures, servlets can
  - manipulate HTTP status codes and headers,
  - use cookies,
  - track sessions,
  - save information between requests,
  - compress pages,
  - access databases,
  - generate JPEG images on-the-fly, and
  - perform many other tasks flexibly and efficiently.
- But, generating HTML with servlets can be tedious and can yield a result that is hard to modify.
- That's where JSP comes in, JSP lets you separate much of the presentation from the dynamic content. That way, you can write the HTML in the normal manner, even using HTML-specific tools and putting your Web content developers to work on your JSP documents.
- JSP expressions, scriptlets, and declarations let you insert simple Java code into the servlet that results from the JSP page, and directives let you control the overall layout of the page.
- For more complex requirements, you can wrap Java code inside beans or even define your own JSP tags.

JSP provides a single overall presentation.

- *What if you* want to give totally different results depending on the data that you receive?
  - Scripting expressions, beans, and custom tags, although extremely powerful and flexible, don't overcome the limitation that the JSP page defines a relatively fixed, top-level page appearance.
- Similarly, what if you need complex reasoning just to determine the type of data that applies to the current situation?
  - JSP is poor at this type of business logic.

The solution is to use both servlets and JavaServerPages.

- *In this approach, known* as the Model View Controller (MVC) or Model 2 architecture, you let each technology concentrate on what it excels at.
  - The original request is handled by a servlet.
  - The servlet invokes the business-logic and data-access code and creates beans to represent the results (that's the *model).*
  - *Then, the servlet decides which JSP page is appropriate* to present those particular results and forwards the request there (the JSP page is the *view).*
  - *The servlet decides what business logic code applies and which JSP* page should present the results (the servlet is the *controller).*

Motivation behind the MVC

- The key motivation behind the MVC approach is the desire to separate the code that creates and manipulates the data from the code that presents the data.
- The basic tools needed to implement this presentation-layer separation are standard in the servlet API and are the topic of this lecture.
- However, in very complex applications, a more elaborate MVC framework is sometimes beneficial. Some popular of these frameworks are
  - Apache Struts
  - Spring MVC

Quick summary of the required steps in MVC

1. Define beans to represent the data.

Your first step is define beans to represent the results that will be presented to the user.

2. Use a servlet to handle requests.

In most cases, the servlet reads request parameters as described in previous lectures.

3. Populate the beans.

The servlet invokes business logic (applicationspecific code) or data-access code to obtain the results. The results are placed in the beans that were defined in step 1.

4. Store the bean in the request, session, or servlet context.

The servlet calls setAttribute on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.

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 page.

6. Extract the data from the beans.

The JSP page accesses beans with jsp:useBean and a scope matching the location of step 4.
The page then uses jsp:getProperty to output the bean properties.
The JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.

Defining Beans to Represent the Data

- Beans are Java objects that follow a few simple conventions.
- In this case, since a servlet or other Java routine (never a JSP page) will be creating the beans, the requirement for an empty (zero-argument) constructor is waived.
- So, your objects merely need to follow the normal recommended practices of keeping the instance variables private and using accessor methods that follow the get/set naming convention.
- Since the JSP page will only access the beans, not create or modify them, a common practice is to define *value objects: objects that represent results but have little or* no additional functionality.

Writing Servlets to Handle Requests

- Once the bean classes are defined, the next task is to write a servlet to read the request information.
- Since, with MVC, a servlet responds to the initial request, the normal approaches of previous lectures are used to read request parameters and request headers, respectively.
- Although the servlets use the normal techniques to read the request information and generate the data, they do not use the normal techniques to output the results.
- In fact, with the MVC approach the servlets do not create *any output; the output is* completely handled by the JSP pages.
- So, the servlets do not call response.set- ContentType, *response.getWriter*, or *out.println.*

Populating the Beans

- After you read the form parameters, you use them to determine the results of the request.
- These results are determined in a completely application-specific manner.
- You might call some business logic code, invoke an Enterprise JavaBeans component, or query a database.
- No matter how you come up with the data, you need to use that data to fill in the value object beans that you defined in the first step.

Storing the Results

- You have read the form information.
- You have created data specific to the request.
- You have placed that data in beans.
- Now you need to store those beans in a location that the JSP pages will be able to access
- A servlet can store data for JSP pages in three main places:
  - in the HttpServlet-Request,
  - in the HttpSession, and
  - in the ServletContext.
- These storage locations correspond to the three nondefault values of the scope attribute of jsp:useBean: that is,
  - request,
  - session,
  - application.

Forwarding Requests to JSP Pages using RequestDispatcher

- You forward requests with the forward method of RequestDispatcher.
- You obtain a RequestDispatcher by calling the getRequestDispatcher method of ServletRequest, supplying a relative address.
- You are permitted to specify addresses in the WEB-INF directory; clients are not allowed to directly access files in WEB-INF, but the server is allowed to transfer control there.
- Using locations in WEB-INF prevents clients from inadvertently accessing JSP pages directly, without first going through the servlets that create the JSP data.
- Once you have a RequestDispatcher, you use forward to transfer control to the associated address. You supply the HttpServletRequest and HttpServlet-Response as arguments.

forward vs. redirect

- Note that the forward method of RequestDispatcher is quite different from the sendRedirect method of HttpServletRequest.

- With forward, there is no extra response/request pair as with sendRedirect.
  - Thus, the URL displayed to the client does not change when you use forward.

- *When you use the forward method of RequestDispatcher, the client sees the URL of the original servlet, not the URL of the final JSP page.*

Summary of the behavior of ***forward***

- Control is transferred entirely on the server.
- No network traffic is involved.
- The user does not see the address of the destination JSP page and pages can be placed in WEB-INF to prevent the user from accessing them without going through the servlet that sets up the data.
- This is beneficial if the JSP page makes sense only in the context of servlet generated data.

Summary of the behavior of ***sendRedirect***

- Control is transferred by sending the client a 302 status code and a Location response header.
- Transfer requires an additional network round trip.
- The user sees the address of the destination page and can bookmark it and access it independently.
  - This is beneficial if the JSP is designed to use default values when data is missing.
- For example, this approach would be used when redisplaying an incomplete HTML form or summarizing the contents of a shopping cart.

Summary:

- We have covered the two models of Java web application design:
  - Model 1 and Model 2
- Model 1 architecture provides rapid development for small projects, and is suitable for small projects that will remain small or for building prototypes.
- Model 2 is the recommended architecture for any medium-sized to large projects.
- Model 2 is harder to build, but it provides more maintainability and extensibility.
- In very complex applications, a more elaborate MVC framework is sometimes beneficial, such as *Apache Struts, and Spring MVC*
- Although Struts, and Spring MVC are useful and widely used, you should not feel that you must use Struts or Spring MVC in order to apply the MVC approach.
- **For simple and moderately complex applications, implementing MVC from scratch with RequestDispatcher is straightforward and flexible.**