Lightweight J2EE Framework

Struts, spring, hibernate

Software System Design

Zhu Hongjun

Introduction

- Teaching hours
  - Theory：40 class hours
  - Practice：20 class hours
- Grading rules
  - Theory（51%）+ Exercise（49%）=Total（100%）
  - Exercise=7 weeks * 7’ = 49

Contents

- Theory
  - Contents：The Basics of MVC、Struts 2.X Key Features、Actions & Results & Interceptors & Validators in Struts 2.x、Struts Tags、Message Handling & i18n、O/R Mapping & CRUD in Hibernate 4.x、HQL & QBC & QBE、Spring IoC & DI.
  - Evaluation：written test
- Practice
  - Contents：discussion on usages of MVC frameworks in your engineering practice project
- Exercise
  - Contents：programming on the key features of a lightweight controller

References

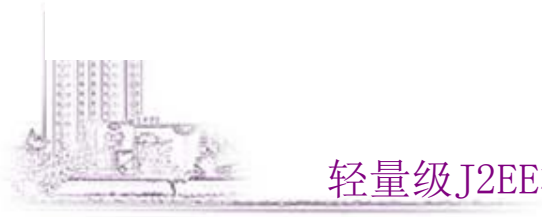
- References
  - Slides and Internet Materials
  - Internet
    - http://docs.spring.io/
    - http://docs.jboss.org/hibernate
    - http://struts.apache.org/
    - http://www.google.com
    - http://www.baidu.com

Software & Tools

- JDK
  - Downloading from: www.oracle.com
- Eclipse
  - Downloading from: www.eclipse.org
- Tomcat
  - Downloading from: http://www.apache.org
- Struts
  - Downloading from: http://struts.apache.org/
- Hibernate
  - Downloading from: http://www.hibernate.org/
- Spring
  - Downloading from: http://projects.spring.io/spring-framework/

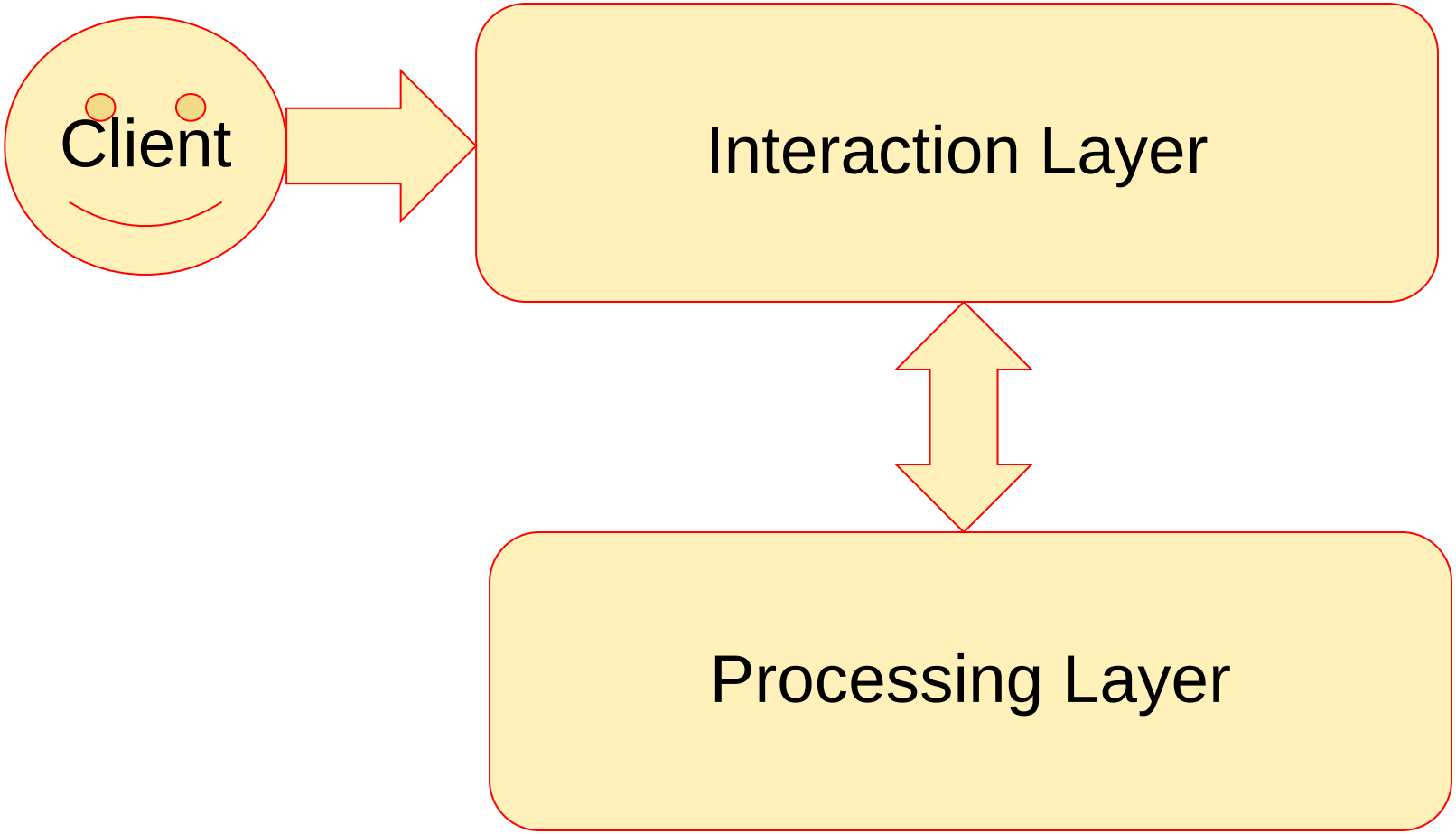
Session 1: Basics

- Layered (or tiered) Application Design
- Evolution of Web Application Design Architecture
- Basic MVC Design
- Web Application Frameworks



Layered (or tiered) application design

- Structure application in two layers
  - Interaction layer
    - Interface to client
    - Receive requests and perform required translations and transformations
    - Delegate request to processing layer for processing
    - Respond to clients
  - Processing layer
    - Process request by performing business logic
    - Access database & Integrate with EIS



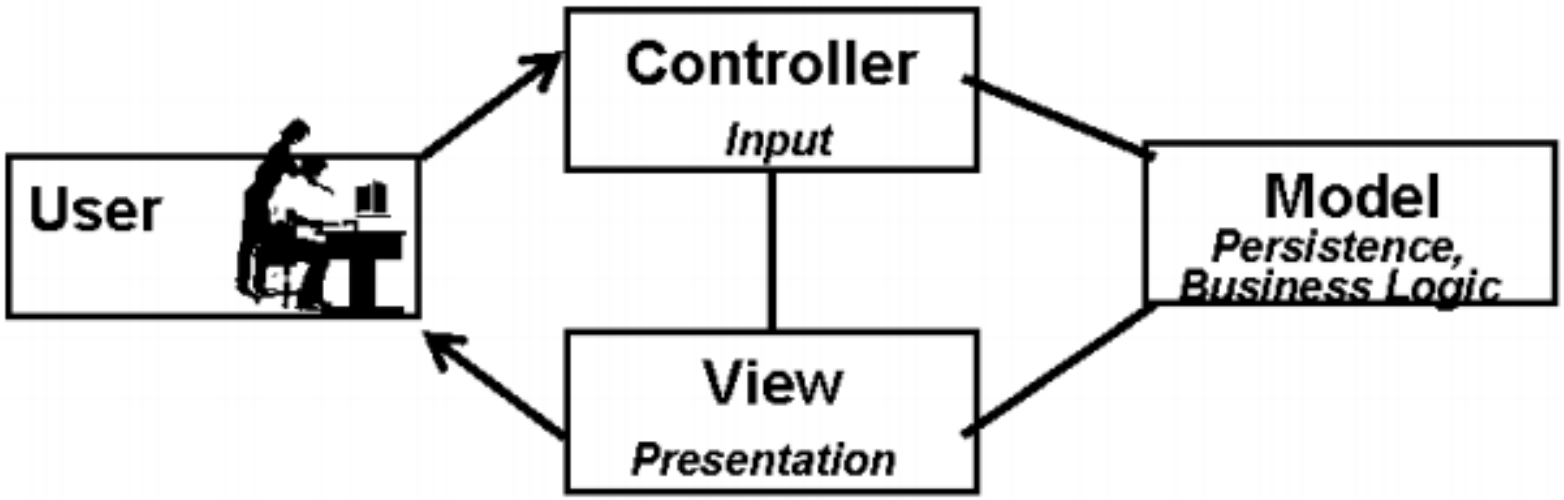
application in two layers

Layered (or tiered) application design

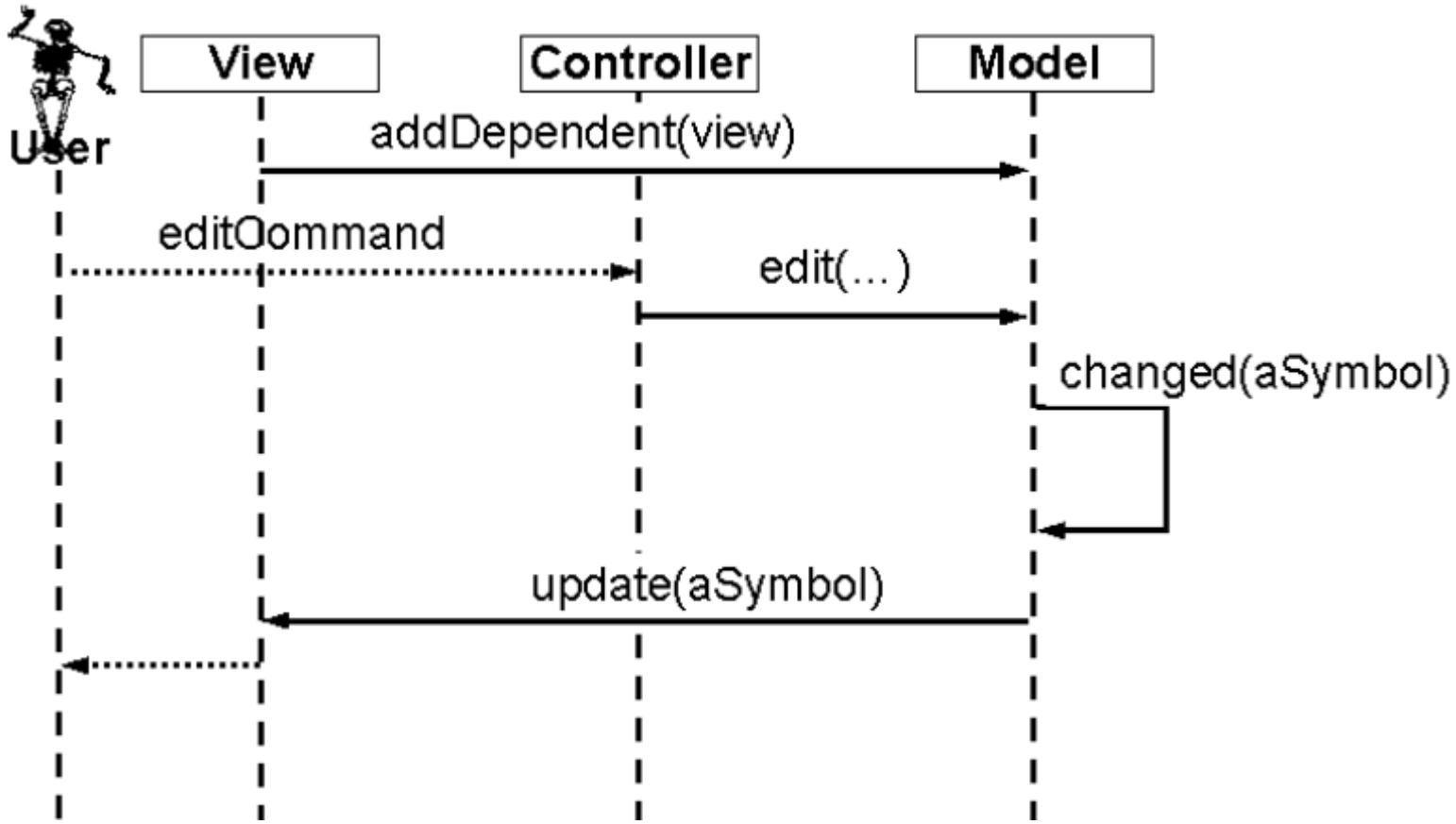
- Structure application in two layers (cont.)
  - Why Layered Application Design?
    - Clearly divide responsibilities
      - De-couple business logic from presentation
      - Change in business logic layer does not affect the presentation layer
    - Provide a common “place” for pre-processing and post-processing of requests and responses
      - logging, translations, transformations, etc

Layered (or tiered) application design

- MVC Pattern
  - Model (Business process layer)
    - Models the data and behavior behind the business process
    - Responsible for actually doing
      - Performing DB queries
      - Calculating the business process
      - Processing orders
    - Encapsulate of data and behavior which are independent of presentation



Smalltalk-80 MVC



Smalltalk-80 MVC Sequence Diagram

Layered (or tiered) application design

- MVC Pattern (cont.)
  - View (Presentation Layer)
    - Display information according to client types
    - Display result of business logic (Model)
      - Not concerned with how the information was obtained, or from where (since that is the responsibility of Model)
    - As interface of application for user
      - Receive request
      - Respond to user's request

Layered (or tiered) application design

- MVC Pattern (cont.)
  - Controller (Controller Layer)
    - Serves as the logical connection between the user's interaction and the business services on the back
    - Responsible for making decisions among multiple presentations
    - A request enters the application through the control layer, it will decide how the request should be handled and what information should be returned

Layered (or tiered) application design

- Attention to Layered Application Design
  - It is often advantageous to treat each layer as an independent portion of your application
  - Do not confuse logical separation of responsibilities with actual separation of components
  - Some or of the layers can be combined into single components to reduce application complexity

Layered (or tiered) application design

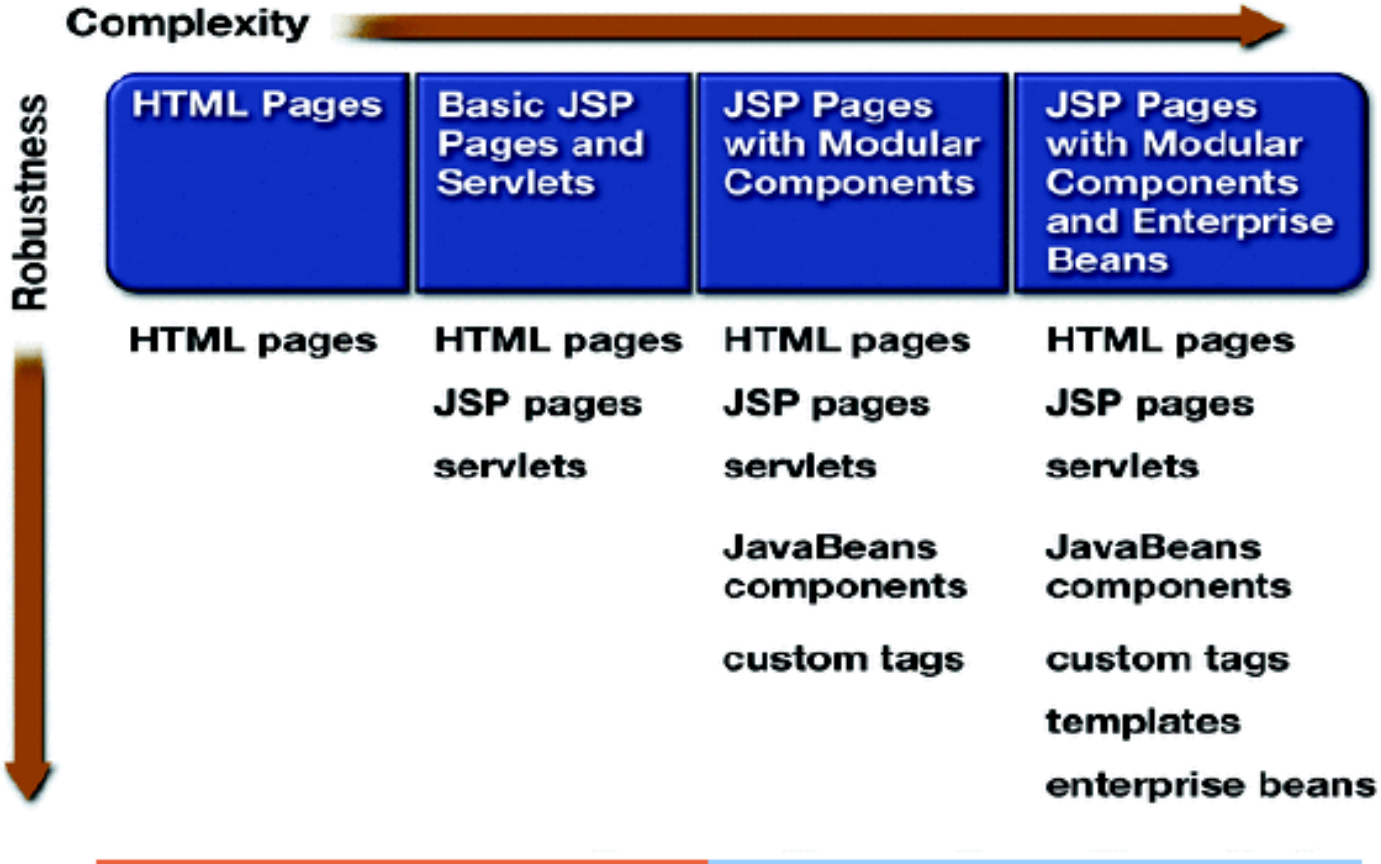
- MVC Misconceptions
  - An elaborate framework is necessary
    - Frameworks are sometimes useful
      - Struts
      - JavaServer Faces (JSF)
    - They are *not* required!
      - Implementing MVC with the builtin RequestDispatcher works very well for most simple and moderately complex applications
  - MVC totally changes your overall system design
    - You can use MVC for individual requests
    - Think of it as the MVC *approach*, not the MVC *architecture*
  - Also called the *Model 2* approach

Evolution of Web Application Design Architecture

- Evolution of MVC Architecture
  - No MVC
  - Page-centric MVC (Model 1)
  - Servlet-centric MVC (Model 2)
  - Web Application FrameWork
    - Struts, Spring, Hibernate
  - Standard-based Web application framework
    - JSF, JavaServer Faces (JSR-127)

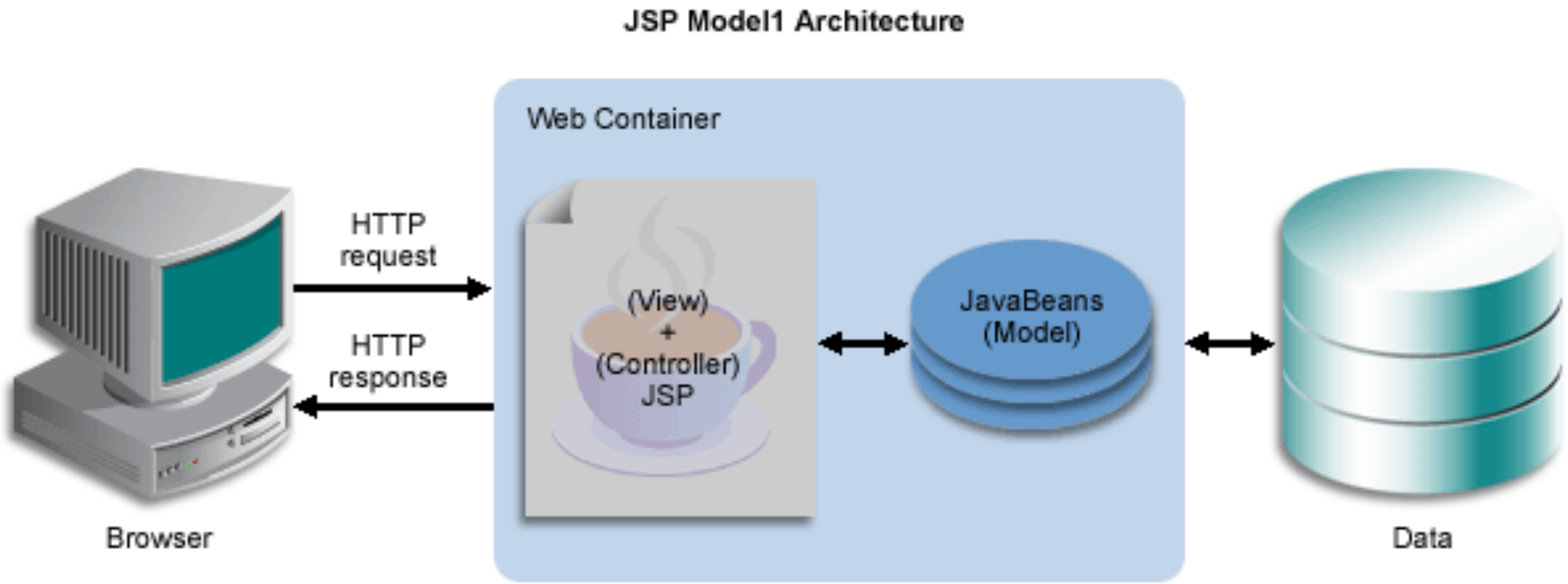
Evolution of Web Application Design Architecture

- No MVC & Model 1



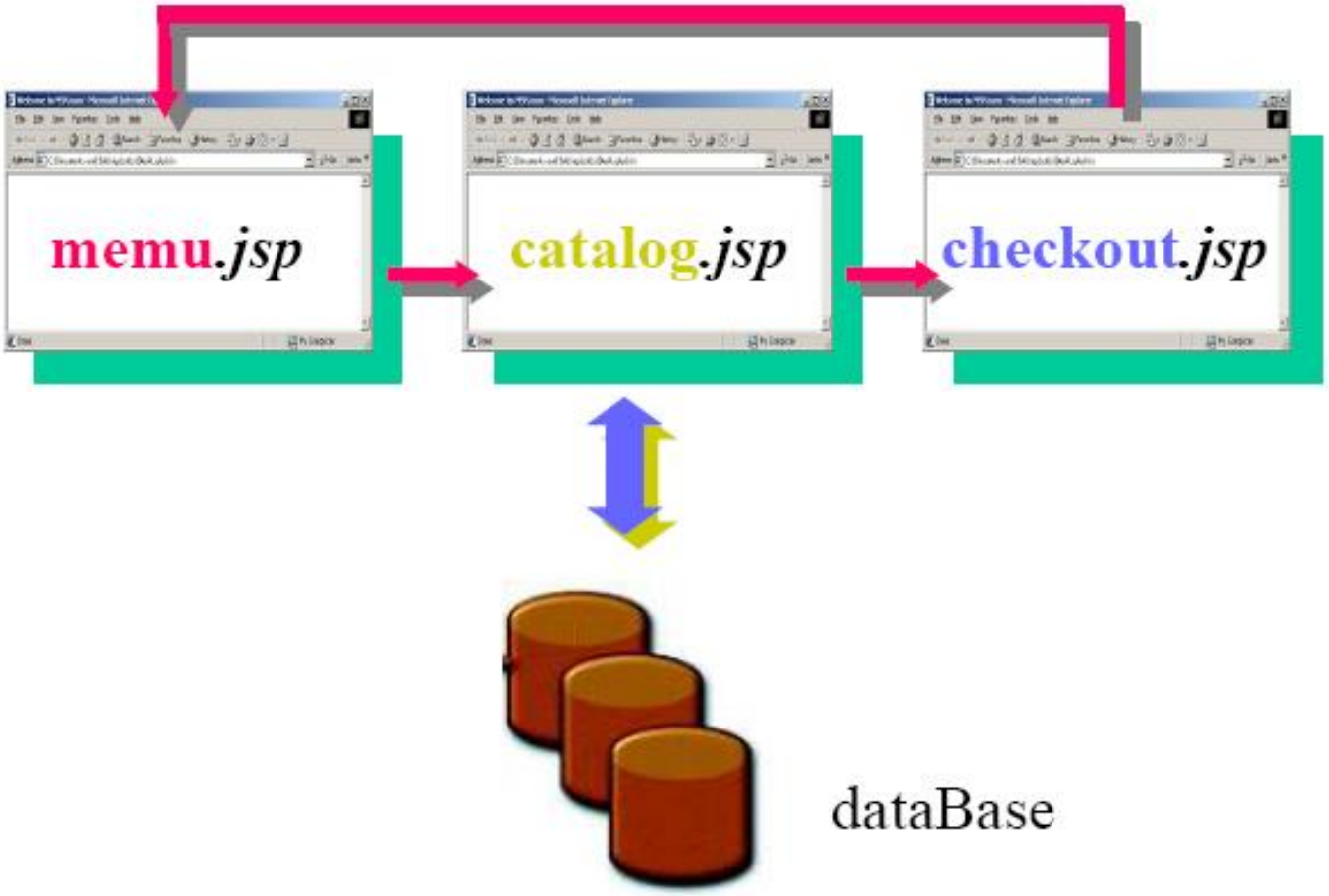
Evolution of Web Application Design Architecture

- Model 1 (Page-centric)



Evolution of Web Application Design Architecture

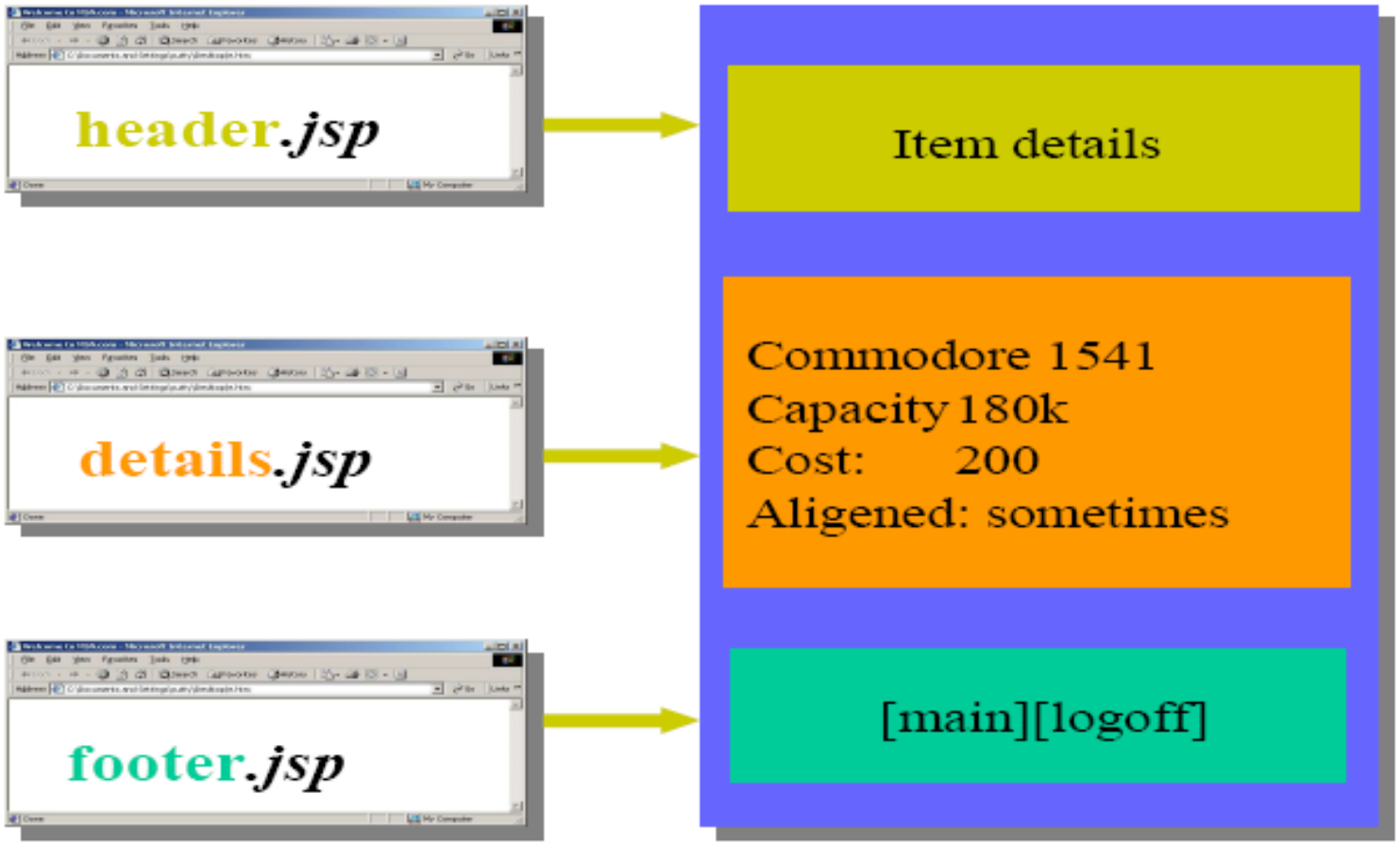
- Model 1 (cont.)
  - Composed of a series of interrelated JSP pages
    - JSP pages handle all aspects of the application – presentation, control, and business process
  - Business process logic and control decisions are hard coded inside JSP pages
    - in the form of JavaBeans, scriptlets, expression
  - eg. Next page selection is determined by
    - A user clicking on a hyper link, e.g. <A HERF="find.jsp>
    - Through the action of submitting a form, e.g. <FORM ACTION="search.jsp">



Model 1 Catalog Page Demo

Evolution of Web Application Design Architecture

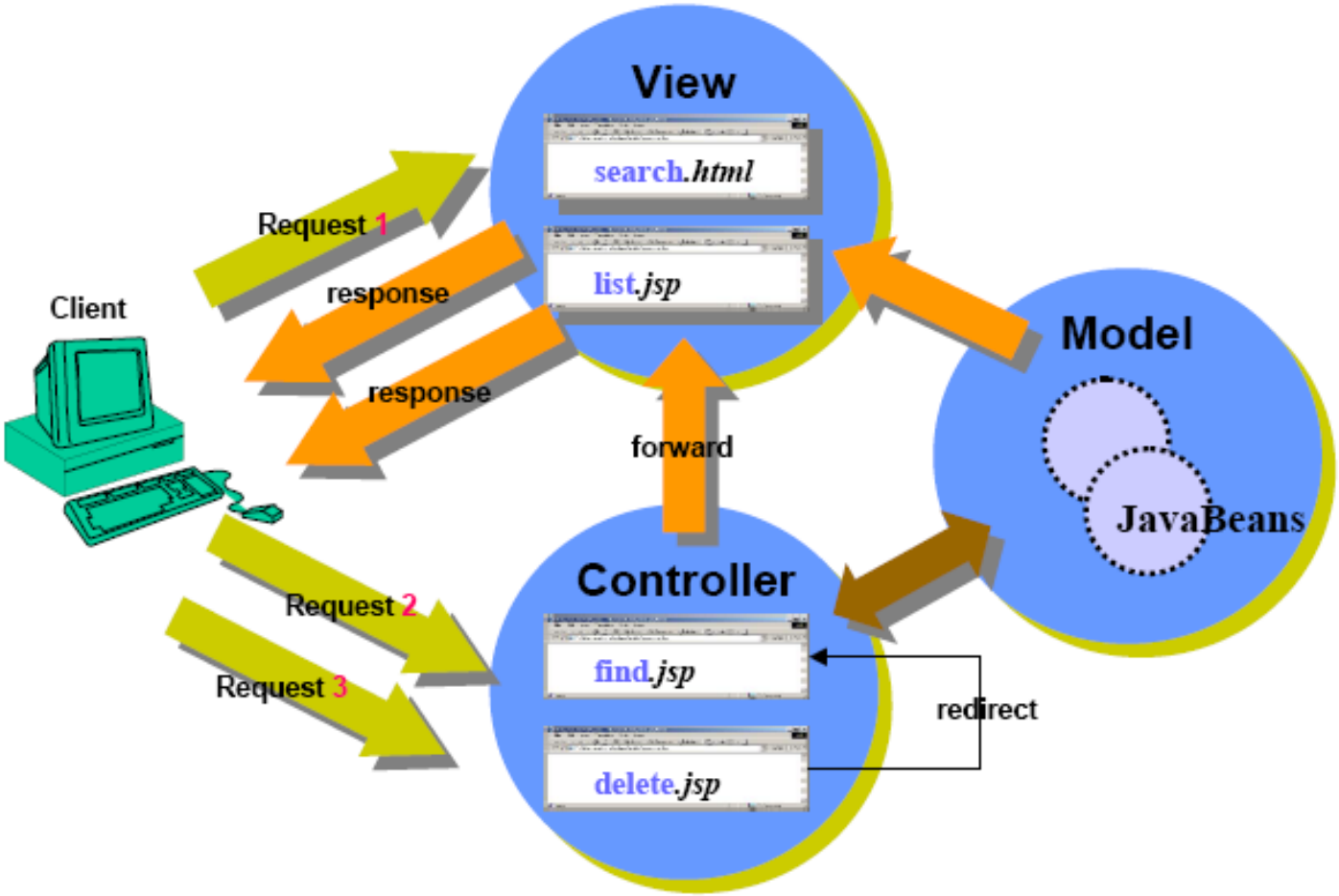
- Model 1 (cont.)
  - eg. One page might display a menu of options, another might provide a form for selecting items from the catalog,and another would be to complete shopping process
    - This doesn't mean we lose separation of presentation and content
    - Still use the dynamic nature of JSP and its support for JavaBeans component to factor out business logic from presentation
    - The pages are tightly coupled:
      - Need to sync up request parameters
      - Be aware of each other's URLs



Model 1 Component Page Demo

Evolution of Web Application Design Architecture

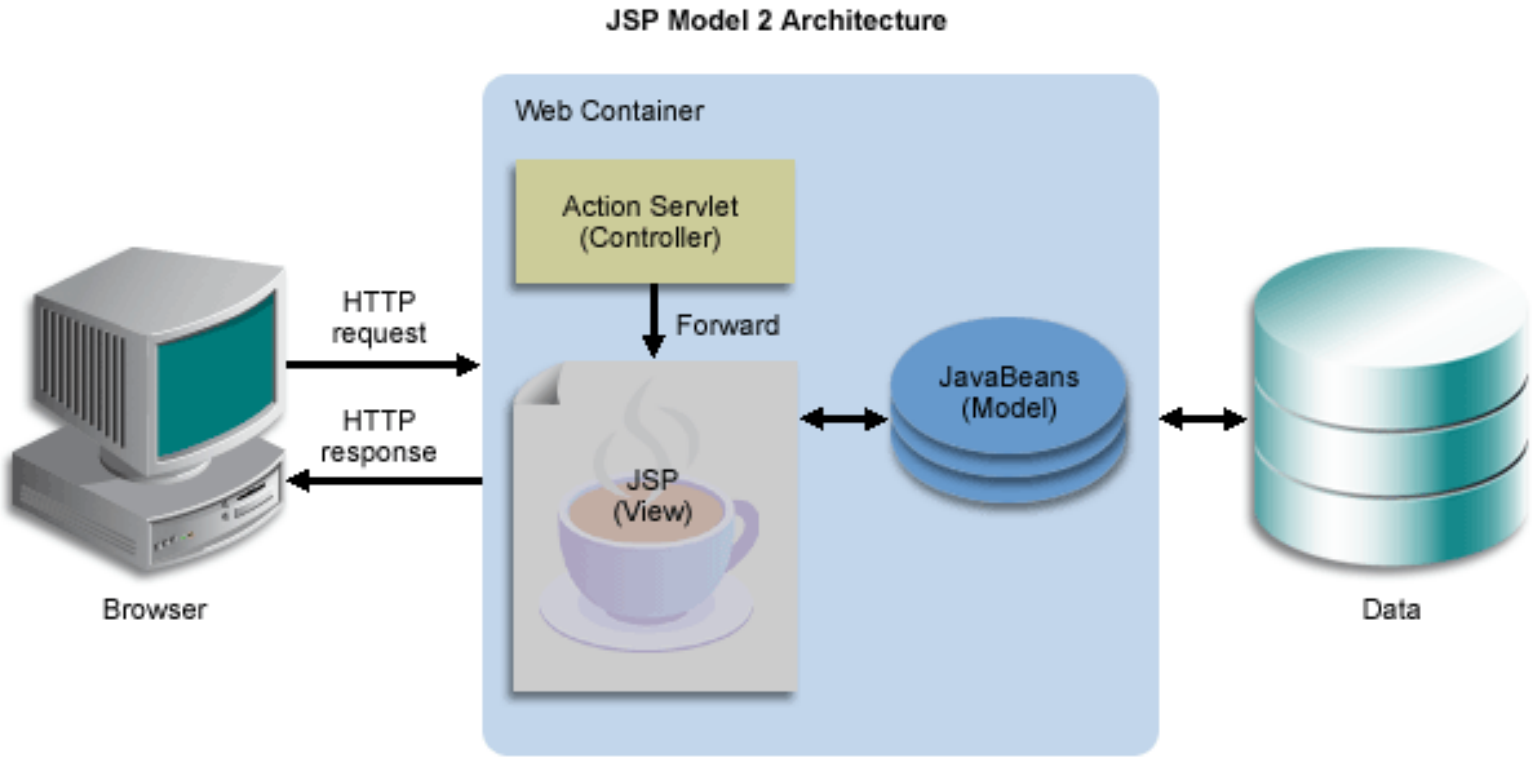
- Model 1 (cont.)
  - eg. Create headers, footers and navigation bars in JSP pages
    - Provides better flexibility and reusability.
    - Easy to maintain.
  - <%@ include file = "header.jsp" %>
    - Use it when the file (included) changes rarely.
    - Faster than jsp:include.
  - <jsp:include page="header.jsp" flush="true“>
    - Use it for content that changes often
    - if which page to include can not be decided until the main page is requested



Model 1 Scenario

Evolution of Web Application Design Architecture

- Model 2 (Servlet-centric)



Evolution of Web Application Design Architecture

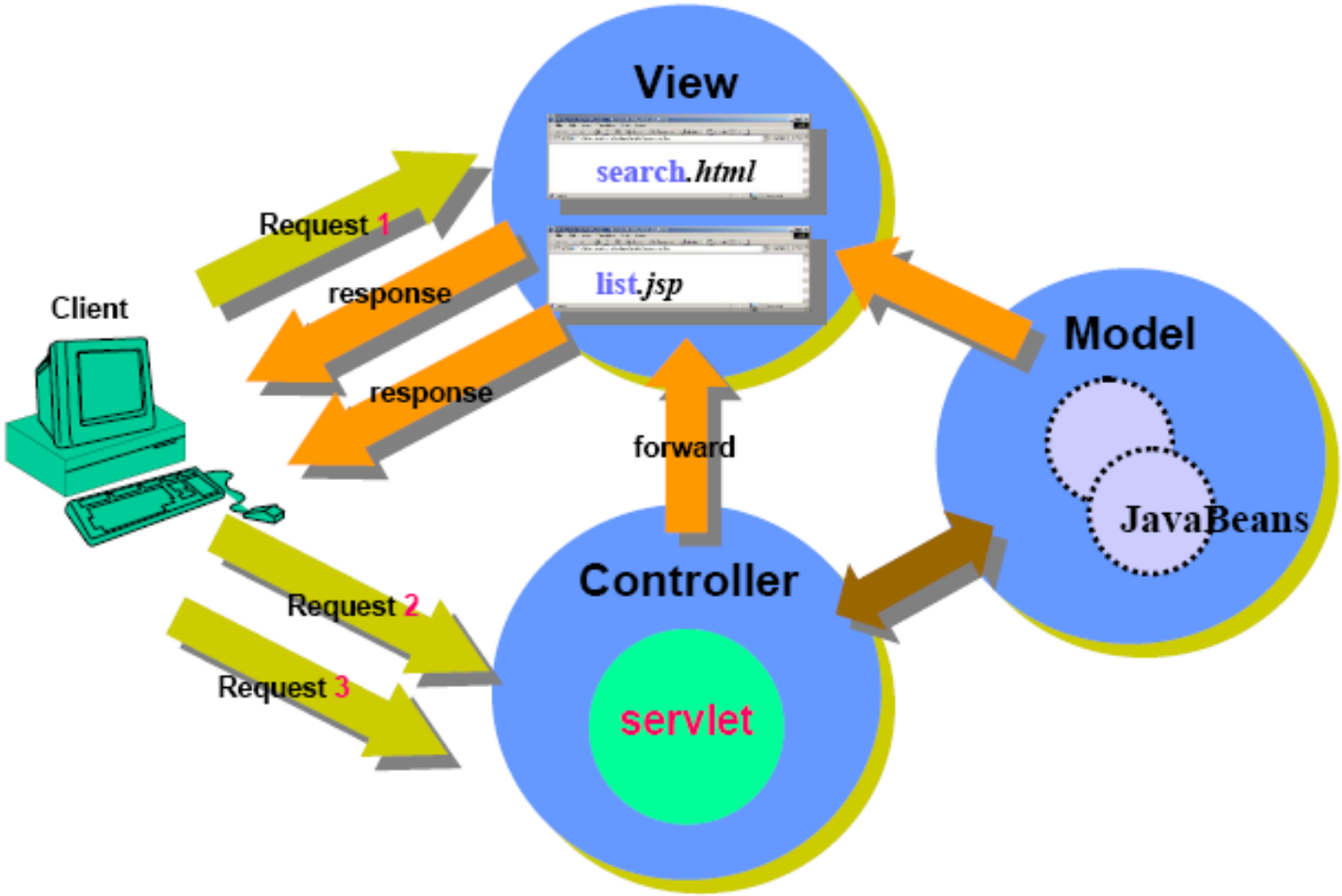
- Model 2 (cont.)
  - What if you want to present different JSP pages depending on the data you receive
    - JSP technology alone even with JavaBeans and custom tags (Model 1) cannot handle it well
  - Solution
    - Use Servlet and JSP together (Model 2)
    - Servlet handles initial request, partially process the data, set up beans, then forward the results to one of a number of different JSP pages

Evolution of Web Application Design Architecture

- Model 2 (cont.)
  - Architecture
    - JSP pages are used only for presentation
      - Control and application logic handled by a servlet (or set of servlets)
    - Servlet serves as a gatekeeper
      - Provides common services, such as authentication, authorization, login, error handling, and etc
    - Servlet serves as a central controller
      - Act as a state machine or an event dispatcher to decide upon the appropriate logic to handle the request
      - Performs redirecting

Evolution of Web Application Design Architecture

- Model 2 (cont.)
  - How many Servlets in Model 2
    - It depends on the granularity of your application
    - One master Servlet
    - One servlet per use case or business function
    - Combination of the two
      - master servlet handles common function (i.e. common login) for all business functions
      - master servlet then delegates to child servlets for further gatekeeping tasks



Model 2 Scenario

Evolution of Web Application Design Architecture

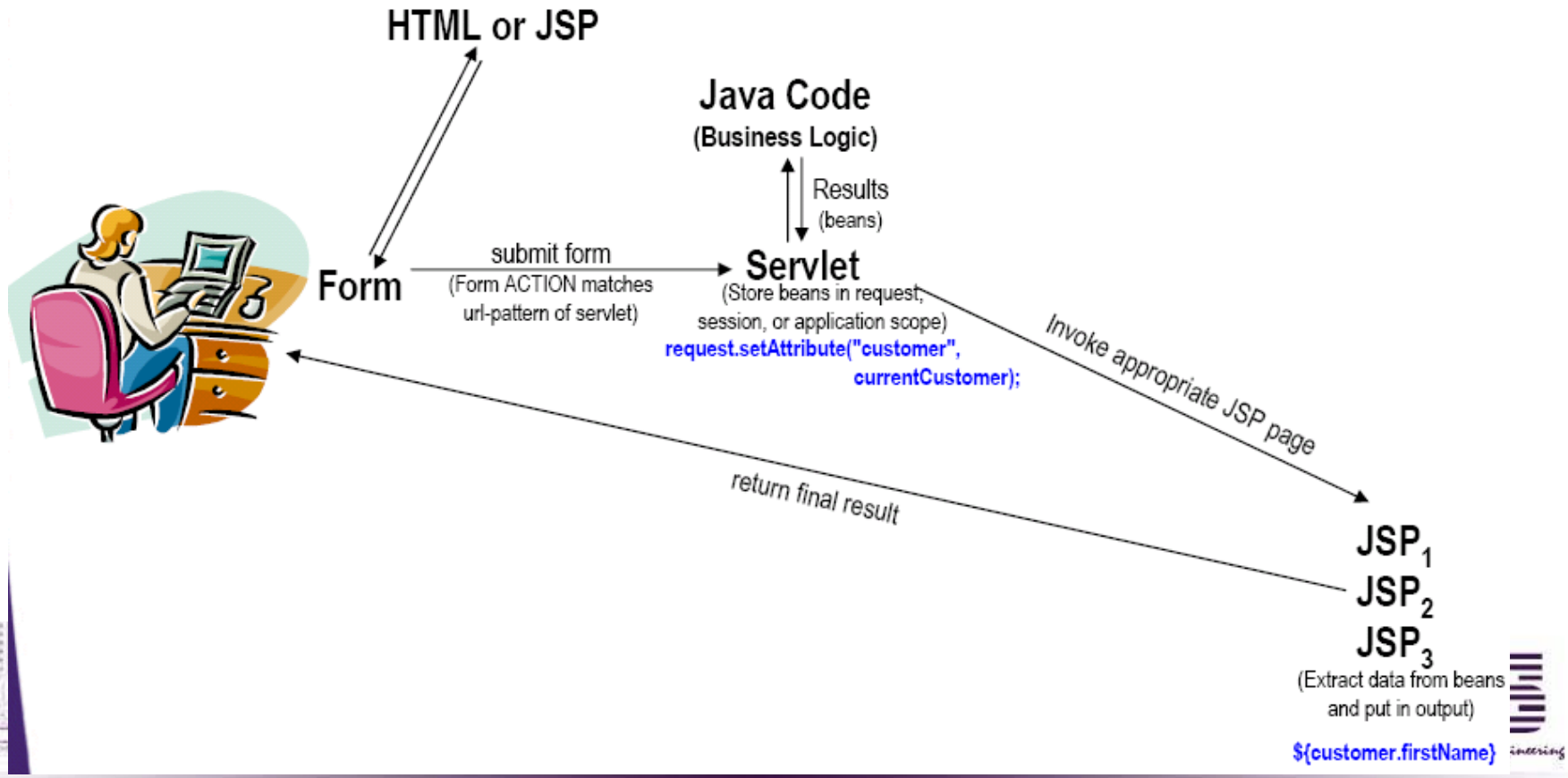
- Beans
  - **Java classes that follow certain conventions**
    - Must have a zero-argument (empty) constructor
      - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
    - Should have no public instance variables (fields)
      - I hope you already follow this practice and use accessor methods instead of allowing direct access to fields
    - Persistent values should be accessed through methods called get*Xxx* and set*Xxx*
      - If class has method **getTitle** that returns a String, class is said to have a String *property* named **title**
      - Boolean properties can use is*Xxx* instead of get*Xxx*

```
package coreservlets;

public class StringBean {
  private String message = "No message specified";

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

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

Bean demo

Basic MVC Design

- MVC Flow of Control



Basic MVC Design

- Implementing MVC with RequestDispatcher
  - Define beans to represent the data
  - Use a servlet to handle requests
  - Populate the beans
  - Store the bean in the request, session, or servlet context
  - Forward the request to a JSP page
  - Extract the data from the beans

Insert title here

http://localhost:8080/JeeProject/login?name=water&pass=123

Success!

Request

```
<servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>water.servlet.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>
```

Request

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    String name = request.getParameter("name");
    String pass = request.getParameter("pass");

    if (name.equals("water") && pass.equals("123")) {
        RequestDispatcher rd = request
                .getRequestDispatcher("pages//success.jsp");
        rd.forward(request, response);
    } else {
        RequestDispatcher rd = request
                .getRequestDispatcher("pages//fail.html");
        rd.forward(request, response);
    }
}
```

Web Application Frameworks

- Based on MVC Model 2 architecture
- Web-tier applications share common set of functionality
  - Dispatching HTTP requests
  - Invoking model methods
  - Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

Web Application Frameworks

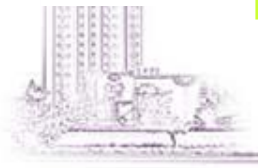
- Why Web Application Framework?
  - De-coupling of presentation tier and business logic into separate components
  - Provides a central point of control
  - Provides rich set of features
  - Facilitates unit-testing and maintenance
  - Availability of compatible tools
  - Provides stability
  - Enjoys community-supports
  - Simplifies internationalization

Web Application Frameworks

- Why Web Application Framework?
  - Simplifies input validation
  - Frameworks have evolved with Java Server technology
  - JSP/Servlets are still hard to use
  - Frameworks define re-usable components to make this job easier.
  - A good framework defines how components work to create a usable application

Web Application Frameworks

- Common web app frameworks
  - Apache Struts、JBoss Hibernate、Spring
  - JavaServer Faces (JSR-127)
    - A server side user interface component framework for Java TM technology-based web applications
  - Echo
    - http://echo.nextapp.com/site/
  - Tapestry
    - http://tapestry.apache.org/



Conclusions

- Layered (or tiered) Application Design
- Evolution of Web Application Design Architecture
- Basic MVC Design
- Web Application Frameworks