# Table of Contents – Fast Track to Java EE 5 with Servlets/JSP and JDBC

**Fast Track to Java EE 5 with Servlets/JSP and JDBC**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workshop Overview</td>
<td>2</td>
</tr>
<tr>
<td>Workshop Objectives</td>
<td>3</td>
</tr>
<tr>
<td>Workshop Agenda</td>
<td>4</td>
</tr>
<tr>
<td>Typographic Conventions</td>
<td>5</td>
</tr>
<tr>
<td>Labs</td>
<td>6</td>
</tr>
<tr>
<td>Release Level</td>
<td>7</td>
</tr>
</tbody>
</table>

**Session 1 - Java EE Introduction and Architecture**

| Session Objectives - You Should Be Able to:                           | 9    |

**Java EE Overview**

| Java EE's Place within Java                                           | 11   |
| Java EE - Distributed Multitier Applications                          | 12   |
| Important Java EE 5 APIs                                               | 13   |
| Commercial Java EE Products                                            | 14   |
| Common Commercial Java EE App Servers                                 | 15   |
| Enterprise Applications - the Three Pillars                            | 16   |
| Persistent Storage                                                     | 17   |
| Business Components                                                    | 18   |
| Presentation Components                                                | 19   |
| Presentation Logic - Servlets                                          | 20   |
| Presentation View - JavaServer Pages                                   | 21   |
| Java EE Containers                                                     | 22   |

**Java EE Architecture**

| Multitiered Architecture                                              | 24   |
| Web-Based Architecture                                                | 25   |
| Web-Based Architecture with EJB                                       | 26   |
| Swing Client Architecture with EJB                                    | 27   |
| Supporting Both Web Clients and Rich Clients                          | 28   |
| Playing Server Games                                                  | 29   |
| SOA – Service Oriented Architecture                                   | 30   |
| SOA Architecture                                                       | 31   |
| What's Next?                                                           | 32   |
| Session Review Questions                                               | 33   |
| Session Summary                                                        | 34   |

**Topic : Web Applications**

| Topic Overview                                                        | 36   |
| Objectives                                                            | 37   |
| Workshop Agenda                                                       | 38   |

**Session 2: Web Application Basics**

| Lesson Objectives                                                     | 39   |

**How the Web Works**

| Clients and Servers                                                   | 42   |
| HTML - The Language of the Web                                        | 43   |
| HTTP, Addressing, Requests, and Responses                             | 44   |
| Browsing a Simple Web Page                                            | 45   |
| Java EE Web Applications                                              | 46   |
Session 4: Additional Servlet Capabilities

Lesson Objectives

Working with HttpServletResponse

HTTP Responses - Status and Errors

HttpStatus/Response Status/Error Methods

HTTP Response Headers

Setting Response Headers

MIME Types

Specifying a MIME Type

Initialization

Initialization Overview

The ServletConfig Interface

The ServletContext Interface

Declaring and Mapping Servlets

Servlet Init Parameters

Retrieving Initialization Parameters

Web App Init Parameters

A Problem with SearchServlet

null and Empty String Parameters

Error Handling

Specifying Error Pages

Error Page Configuration

Error Handling

Lab 4.1 - Performing the Search

Review Questions

Lessons Learned

Session 5: JavaServer Pages (JSP)

Lesson Objectives

JavaServer Pages (JSP)

Shortcomings of Servlets

What is a JSP?

A Very Simple JSP - simple.jsp

JSPs Look Like HTML

JSP Expressions

JSP Comments

JSPs are Really Servlets

The Generated Servlet

Lab 5.1 - Creating a JSP

Model View Controller (MVC) Architecture

Servlet and JSP Architecture

Model View Controller (MVC)

Servlets as the Controller

JSP as the View

JavaBeans As The Glue and The Model

Servlets as Controllers

RequestDispatcher

Servlets as Controller

Forwarding to a Resource

Including Output of Another Resource

RequestDispatcher Paths
# Session Objectives

## HTTP Session Overview
- HTTP is a Stateless Protocol
- Online Shopping With a Stateless Protocol
- Storing Contextual Information
- Hidden Form Fields

## Cookies
- Cookies
- Cookie Details
- Accessing Cookies With the Servlet API
- Persistent Cookies and Session Cookies
- Cookie Usability Issues

## Sessions
- Servlet/JSP Sessions
- Using Sessions
- Using a Session Example
- How Session Tracking Works
- Sessions and Cookies - How They Work
- Persistent Cookies and Session Cookies
- Cookie Usability Issues

### Lab 7.1 - Sessions: Implement the Cart
- Review Questions
- Lessons Learned

## Session 8: More JSP Capabilities
- Lesson Objectives

## Error Pages
- Servlet Exceptions and Error Pages
- The Implicit `exception` Object
- JSP 2.0 Error Handling
- Using `errorData`
- Exception Handling in a JSP
- `web.xml` - Declarative Exception Handling
- Server Logging

### [Optional] Lab 8.1 - Error Pages
- Review Questions
- Lessons Learned

## JSPs as XML Files
- JSP Pages as XML Documents
- JSP Pages as XML Documents

## Java Scriptlets
- Scriptlets
- Scriptlet Syntax
- How do Scriptlets Work?
- Another Simple JSP - `simple.jsp`
- Generated Code Fragment For `simple.jsp`
- When to Use Scriptlets
- Declarations
- Declarations Example

### [Optional] Lab 8.2 - Scriptlets
- Review Questions
- Lessons Learned

## Session 9 – More JSTL and EL

---

20101108 Copyright © 2004-10, LearningPatterns Inc. All rights reserved
<p>| Session Objectives - You Should Be Able to: | 335 |
| More About JSTL | 336 |
| JSTL Tag Libraries at a glance | 337 |
| Custom Tag Architecture | 338 |
| More about JSTL URLs and Prefixes | 339 |
| Some Other Common Core Actions | 340 |
| Formatting Library | 341 |
| SQL Library | 342 |
| XML Library | 343 |
| XML Action Example | 344 |
| JSTL 1.1 Functions Library | 345 |
| JSTL Function Library | 346 |
| More About the JSP EL | 347 |
| EL Syntax | 348 |
| EL Identifiers | 349 |
| EL Operators | 350 |
| . and [] Operator Usage | 351 |
| JSP EL Implicit Objects (1 of 3) | 352 |
| JSP EL Implicit Objects (2 of 3) | 353 |
| JSP EL Implicit Objects (3 of 3) | 354 |
| pageContext Properties (1 of 2) | 355 |
| pageContext Properties (2 of 2) | 356 |
| JSP EL Literal Values | 357 |
| Conditional Operator | 358 |
| Using Common Tags | 359 |
| Putting JSTL and EL in JSP | 360 |
| Using &lt;c:if&gt; | 361 |
| Using &lt;c:if&gt; Test Expressions | 362 |
| &lt;c:choose&gt; | 363 |
| Lab 9.1 - More JSTL | 365 |
| More on &lt;c:forEach&gt; | 366 |
| &lt;c:set&gt; | 367 |
| &lt;c:set&gt; Examples | 368 |
| <a href="">fmt:formatDate</a> | 369 |
| <a href="">fmt:formatDate</a> Examples | 370 |
| <a href="">fmt:formatNumber</a> | 371 |
| <a href="">fmt:formatNumber</a> Example | 372 |
| Lab 9.2 - JSTL Format Tags | 373 |
| Review Questions | 374 |
| Lessons Learned | 375 |
| Session 10 – Security | 376 |
| Session Objectives - You Should Be Able to: | 377 |
| Java EE Security Overview | 378 |
| Security Requirements | 379 |
| Java EE security | 380 |
| Web Tier Security in Java EE | 381 |
| Transport Level Security with HTTPS/SSL | 382 |
| Declarative Security | 383 |
| Declarative, Role-Based Security | 384 |
| Example: Security Role Declaration | 385 |</p>
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying Security Constraints</td>
<td>395</td>
</tr>
<tr>
<td>Security Constraints - Deployment Descriptor</td>
<td>396</td>
</tr>
<tr>
<td>Security Constraints - Deployment Descriptor</td>
<td>397</td>
</tr>
<tr>
<td>Mapping Users to Roles</td>
<td>398</td>
</tr>
<tr>
<td>Security - The Big Picture</td>
<td>399</td>
</tr>
<tr>
<td>Web Authentication</td>
<td>400</td>
</tr>
<tr>
<td>Authentication</td>
<td>401</td>
</tr>
<tr>
<td>HTTP Basic Authentication</td>
<td>402</td>
</tr>
<tr>
<td>HTTP Basic Authentication on the Browser</td>
<td>403</td>
</tr>
<tr>
<td>Configuring Basic Authentication</td>
<td>404</td>
</tr>
<tr>
<td>Basic Authentication Pros and Cons</td>
<td>405</td>
</tr>
<tr>
<td>Form Based Authentication</td>
<td>406</td>
</tr>
<tr>
<td>Using Form Based Authentication</td>
<td>407</td>
</tr>
<tr>
<td>Form Based Authentication Pros and Cons</td>
<td>408</td>
</tr>
<tr>
<td>Digest Authentication</td>
<td>409</td>
</tr>
<tr>
<td>Digest Authentication Pros and Cons</td>
<td>410</td>
</tr>
<tr>
<td>HTTPS Client Authentication</td>
<td>411</td>
</tr>
<tr>
<td>Programmatic Security</td>
<td>412</td>
</tr>
<tr>
<td>Programmatic Security</td>
<td>413</td>
</tr>
<tr>
<td>Programmatic Security - HttpServletRequest</td>
<td>414</td>
</tr>
<tr>
<td>[Optional] Lab 10.1 - Security</td>
<td>415</td>
</tr>
<tr>
<td>Session 11 – Additional Topics</td>
<td>420</td>
</tr>
<tr>
<td>Session Objectives - You Should Be Able to:</td>
<td>421</td>
</tr>
<tr>
<td>Custom Tags Using Tag Files</td>
<td>422</td>
</tr>
<tr>
<td>Creating Custom Tags with Tag Files</td>
<td>423</td>
</tr>
<tr>
<td>Adding Attributes to Tags</td>
<td>424</td>
</tr>
<tr>
<td>The tag Directive</td>
<td>425</td>
</tr>
<tr>
<td>Other Tag File Capabilities</td>
<td>426</td>
</tr>
<tr>
<td>Servlet Filters</td>
<td>427</td>
</tr>
<tr>
<td>Servlet Filter Overview</td>
<td>428</td>
</tr>
<tr>
<td>Filter API</td>
<td>429</td>
</tr>
<tr>
<td>Filter Processing</td>
<td>430</td>
</tr>
<tr>
<td>Filter Processing</td>
<td>431</td>
</tr>
<tr>
<td>Filter Chain</td>
<td>432</td>
</tr>
<tr>
<td>A Simple Filter</td>
<td>433</td>
</tr>
<tr>
<td>A Simple Filter - Continued</td>
<td>434</td>
</tr>
<tr>
<td>Filter is Mapped in the web.xml File</td>
<td>435</td>
</tr>
<tr>
<td>How a Filter Can Modify a Request or Response</td>
<td>436</td>
</tr>
<tr>
<td>The Way a Response Normally Works</td>
<td>437</td>
</tr>
<tr>
<td>Manipulating the Response</td>
<td>438</td>
</tr>
<tr>
<td>Wrapping the Response</td>
<td>439</td>
</tr>
<tr>
<td>JavaServer Faces (JSF) Overview</td>
<td>440</td>
</tr>
<tr>
<td>Vanilla Servlet/JSP Shortcomings</td>
<td>441</td>
</tr>
<tr>
<td>JavaServer Faces</td>
<td>442</td>
</tr>
<tr>
<td>JSF Architecture</td>
<td>443</td>
</tr>
<tr>
<td>JSF Capabilities</td>
<td>444</td>
</tr>
<tr>
<td>JSF Components</td>
<td>445</td>
</tr>
<tr>
<td>The UI Runs on the Server</td>
<td>446</td>
</tr>
<tr>
<td>Event Handling</td>
<td>447</td>
</tr>
<tr>
<td>Process Flow</td>
<td>448</td>
</tr>
<tr>
<td>JSF Advantages</td>
<td>449</td>
</tr>
<tr>
<td>JSF Disadvantages</td>
<td>450</td>
</tr>
</tbody>
</table>
Recap
Recap of What We've Done
What Else is There

Topic: Java Database Connectivity (JDBC)
Topic Overview

Session 12 – JDBC Introduction and Architecture
Session Objectives - You Should Be Able to:

Relational Database and JDBC Overview
The Power of Information
Relational Database Overview
Relational Database Description
Database Table Relationships
Web-Based Architecture
Database Access Wish List
JDBC Overview
JDBC Characteristics

Lab 12.1 - Setup the JDBC Environment

JDBC Architecture and API
JDBC Architecture
The Fundamental JDBC API
The DriverManager Class
What is a JDBC Driver?
Drivers Must Be Loaded
Naming Databases with URLs
JDBC URL Examples

Database Connections
Database Connections, Connection Interface
Establishing a Database Connection
Database Connection - Example

Lab 12.2 - Connecting to a Database
The DatabaseMetaData Interface
Database Exceptions - SQLException
Wrapped Database Exceptions - Illustrated
Database Exceptions - SQLWarning
Closing JDBC Objects
try-catch May Not Be Good Enough
Using finally to Close JDBC Objects

Lab 12.3 - Metadata, Exceptions, Cleanup
Session Review Questions
Session Summary

Session 13 - Data Access with JDBC
Session Objectives - You Should Be Able to:

Data Access Objects and O-R Mappings
Data Access Objects - Rationale
Value Objects - Rationale
Object-Relational (O-R) Mappings
JavaTunes O-R Mapping
Value Objects and JavaBeans
Data Access Object - Illustrated

Processing Database Data
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Statements</td>
<td>522</td>
</tr>
<tr>
<td>The Statement Interface</td>
<td>523</td>
</tr>
<tr>
<td>Using Statement - Example</td>
<td>524</td>
</tr>
<tr>
<td>The ResultSet Interface</td>
<td>525</td>
</tr>
<tr>
<td>Extracting Data from a ResultSet</td>
<td>526</td>
</tr>
<tr>
<td>SQL - Java Type Mappings</td>
<td>527</td>
</tr>
<tr>
<td>Extracting Data from a ResultSet - Example</td>
<td>528</td>
</tr>
<tr>
<td>Dealing with Result Set NULL Values</td>
<td>529</td>
</tr>
<tr>
<td>Closing JDBC Objects - Revisited</td>
<td>530</td>
</tr>
<tr>
<td>Using finally to Close JDBC Objects</td>
<td>531</td>
</tr>
<tr>
<td><strong>Lab 13.1 - Data Access Object</strong></td>
<td>532</td>
</tr>
<tr>
<td>Prepared Statements</td>
<td>533</td>
</tr>
<tr>
<td>Benefits of Prepared Statements</td>
<td>540</td>
</tr>
<tr>
<td>The PreparedStatement Interface</td>
<td>541</td>
</tr>
<tr>
<td>Using PreparedStatement - Example</td>
<td>542</td>
</tr>
<tr>
<td>Calling Database Stored Procedures</td>
<td>543</td>
</tr>
<tr>
<td>The CallableStatement Interface</td>
<td>544</td>
</tr>
<tr>
<td>Using CallableStatement - Example</td>
<td>545</td>
</tr>
<tr>
<td><strong>Lab 13.2 - Prepared Statements</strong></td>
<td>546</td>
</tr>
<tr>
<td>Update Statements</td>
<td>554</td>
</tr>
<tr>
<td>Executing an Update - Example</td>
<td>555</td>
</tr>
<tr>
<td>Transactions - Overview</td>
<td>556</td>
</tr>
<tr>
<td>Controlling Transactions in JDBC</td>
<td>557</td>
</tr>
<tr>
<td>Example of a Transaction</td>
<td>558</td>
</tr>
<tr>
<td>Executing a Transaction - Example</td>
<td>559</td>
</tr>
<tr>
<td>Storing Dates</td>
<td>560</td>
</tr>
<tr>
<td><strong>Lab 13.3 - Update Statements</strong></td>
<td>561</td>
</tr>
<tr>
<td>JDBC Driver Types</td>
<td>567</td>
</tr>
<tr>
<td>JDBC Driver Types</td>
<td>568</td>
</tr>
<tr>
<td>JDBC Driver Types</td>
<td>569</td>
</tr>
<tr>
<td>Session Review Questions</td>
<td>570</td>
</tr>
<tr>
<td>Session Summary</td>
<td>571</td>
</tr>
<tr>
<td><strong>Session 14 – Optional Additional Topics</strong></td>
<td>572</td>
</tr>
<tr>
<td>Java Persistence API - Overview -</td>
<td>573</td>
</tr>
<tr>
<td>Java Persistence API (JPA)</td>
<td>574</td>
</tr>
<tr>
<td>JPA Goals</td>
<td>575</td>
</tr>
<tr>
<td>More about O-R Mapping</td>
<td>576</td>
</tr>
<tr>
<td>O-R Mapping Tools</td>
<td>577</td>
</tr>
<tr>
<td>Entity Classes</td>
<td>578</td>
</tr>
<tr>
<td>An Example Entity Class</td>
<td>579</td>
</tr>
<tr>
<td>Retrieving an Object From the DB</td>
<td>580</td>
</tr>
<tr>
<td><strong>Advanced JDBC Features - Overview -</strong></td>
<td>581</td>
</tr>
<tr>
<td>Batch Updates</td>
<td>582</td>
</tr>
<tr>
<td>Executing a Batch Update - Example</td>
<td>583</td>
</tr>
<tr>
<td>Prepared Statement Batch Update - Example</td>
<td>584</td>
</tr>
<tr>
<td>Scrollable, Updatable Result Sets</td>
<td>585</td>
</tr>
<tr>
<td>Result Set Concurrency Types</td>
<td>586</td>
</tr>
<tr>
<td>Creating a Scrollable Result Set - Example</td>
<td>587</td>
</tr>
<tr>
<td>Rowset Overview</td>
<td>588</td>
</tr>
<tr>
<td>The RowSet Interface</td>
<td>589</td>
</tr>
<tr>
<td>Using RowSet - Example</td>
<td>590</td>
</tr>
<tr>
<td>Rowset Implementations</td>
<td>591</td>
</tr>
<tr>
<td>BLOBs and CLOBs</td>
<td>592</td>
</tr>
</tbody>
</table>
Retrieving BLOB Data - Example

Topic: Database Integration and Additional Technologies

Topic Overview

Session 15 - Java EE Database Integration

Session Objectives - You Should Be Able to:

DataSource, ENC, and JNDI

Java EE and JDBC

Why DataSource?

How do Clients Get Access to a DataSource?

Resource References and the ENC

Defining a Resource Reference in web.xml

@Resource – Injecting/Using a DataSource

More About Resource Injection

Mapping the ENC Name to the Resource

JNDI – Java Naming and Directory Interface

Server Binds a Reference Into JNDI

Client Looks Up Reference In JNDI

JNDI Overview

JNDI Name Tree

JNDI Tree Structure

The ENC and JNDI

Context Interface and Resource Lookup

ENC and JNDI - Putting It All Together

Connection Pooling

Connection Pooling

Non-Connection Pooling Example

Connection Pooling Example

Pooled Connections - Illustrated

Getting a Pooled Connection - Illustrated

Closing a Pooled Connection - Illustrated

Statement Pooling

Statement Pooling - Illustrated

Getting a Pooled Statement - Illustrated

Closing a Pooled Statement - Illustrated

Be Neat

Lab 15.1 - Database Integration

Session Review Questions

Session Summary

Session 16 - Additional Topics

Session Objectives

EJB Overview

What is EJB

EJB Goals

Types of Enterprise JavaBeans

EJB and Java EE

EJB in Java EE Architecture

EJB Transaction Support

EJB 3.0 Overview

EJB 3.0 Goals

Session Bean Usage

Session Bean Usage

MDB Usage
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benefits of using EJB</td>
<td>652</td>
</tr>
<tr>
<td>Issues with EJB</td>
<td>653</td>
</tr>
<tr>
<td>Defining a Session Bean</td>
<td>654</td>
</tr>
<tr>
<td>Stateless Session Bean Definition</td>
<td>655</td>
</tr>
<tr>
<td>Using an EJB</td>
<td>656</td>
</tr>
<tr>
<td><strong>Web Services Overview</strong></td>
<td></td>
</tr>
<tr>
<td>Service Oriented Architecture (SOA)</td>
<td>657</td>
</tr>
<tr>
<td>The Evolution of Computing</td>
<td>658</td>
</tr>
<tr>
<td>SOA Concepts</td>
<td>659</td>
</tr>
<tr>
<td>Some Advantages of SOA</td>
<td>660</td>
</tr>
<tr>
<td>Some Issues with SOA</td>
<td>661</td>
</tr>
<tr>
<td>Web Services Defined</td>
<td>662</td>
</tr>
<tr>
<td>Value Proposition</td>
<td>663</td>
</tr>
<tr>
<td>Web Services Defined</td>
<td>664</td>
</tr>
<tr>
<td>WS-* Web Services Specifications</td>
<td>665</td>
</tr>
<tr>
<td>The Web Services Stack</td>
<td>666</td>
</tr>
<tr>
<td>Web Services Stacks</td>
<td>667</td>
</tr>
<tr>
<td>Architectural Details</td>
<td>668</td>
</tr>
<tr>
<td>SOAP Concepts</td>
<td>669</td>
</tr>
<tr>
<td>WSDL Concepts</td>
<td>670</td>
</tr>
<tr>
<td>Java Web Services (JWS) Standards</td>
<td>671</td>
</tr>
<tr>
<td>Java Web Services Standards</td>
<td>672</td>
</tr>
<tr>
<td>Defining a Web Service</td>
<td>673</td>
</tr>
<tr>
<td>Writing a JAX-WS Client</td>
<td>674</td>
</tr>
<tr>
<td><strong>Design Considerations</strong></td>
<td></td>
</tr>
<tr>
<td>Web App Design Considerations</td>
<td>675</td>
</tr>
<tr>
<td>User Action - JavaTunes Catalog Search</td>
<td>676</td>
</tr>
<tr>
<td>The Simple Approach - Illustrated</td>
<td>677</td>
</tr>
<tr>
<td>The Simple Approach - Analysis</td>
<td>678</td>
</tr>
<tr>
<td>Using a JSP for the View - Illustrated</td>
<td>679</td>
</tr>
<tr>
<td>Using a Data Access Object - Illustrated</td>
<td>680</td>
</tr>
<tr>
<td>A More Modular Approach - Analysis</td>
<td>681</td>
</tr>
<tr>
<td>Using EJB - Illustrated</td>
<td>682</td>
</tr>
<tr>
<td>Using EJB - Analysis</td>
<td>683</td>
</tr>
<tr>
<td>Using Web Services - Illustrated</td>
<td>684</td>
</tr>
<tr>
<td>Using Web Services - Analysis</td>
<td>685</td>
</tr>
<tr>
<td>General Design – Divide and Conquer</td>
<td>686</td>
</tr>
<tr>
<td>Minimize Network Overhead</td>
<td>687</td>
</tr>
<tr>
<td>Scalability - Clustering</td>
<td>688</td>
</tr>
<tr>
<td>Scalability – Separating Static Content</td>
<td>689</td>
</tr>
<tr>
<td>Scalability – Java Virtual Machine</td>
<td>690</td>
</tr>
<tr>
<td>Scalability – CPUs and 64 bit Machines</td>
<td>691</td>
</tr>
<tr>
<td>The Java BluePrints</td>
<td>692</td>
</tr>
<tr>
<td><strong>Recap</strong></td>
<td></td>
</tr>
<tr>
<td>Recap of what we've done</td>
<td>693</td>
</tr>
<tr>
<td>What Else is There</td>
<td>694</td>
</tr>
<tr>
<td>Online Resources</td>
<td>695</td>
</tr>
<tr>
<td>◆</td>
<td>696</td>
</tr>
</tbody>
</table>
Fast Track to Java EE

The Java Developer Education Series
Workshop Overview

- This is a 5-day hands-on course that covers, in depth, two core technologies of Java Enterprise Edition (Java EE)
  - Servlets and JavaServer Pages / JSP
  - JDBC (Java Database Connectivity)
  - Assembling these components into a Java EE application

- It includes an overview of other core Java EE technologies
  - XML and Web Services basics
  - Enterprise JavaBeans (EJB) overview

- The workshop is 50% discussion, 50% hands-on lab exercises
  - The labs follow a common fictional case study: JavaTunes, an online music store
  - You'll build a complete Web system, from database to browser

Notes:
# Workshop Objectives

- Understand the scope and important technologies of Java EE  
  – With primary focus on Servlets/JSP, and JDBC and an overview of the other technologies

- Build the components of a Java EE application, assemble them into a working system, and deploy that application to an application server

- Understand the architectural issues in designing a Java EE application

- Understand the role of the application server and the services it provides to Java EE application components

---

**Notes:**
Workshop Agenda

◆ Topic 1 – Java EE Introduction and Architecture

◆ Topic 2 – Servlets and JavaServer Pages

◆ Topic 3 – Java Database Connectivity

◆ Topic 4 - Component Integration and additional Java EE Technologies

Notes:
Typographic Conventions

- Code that is inline in the text will appear in a fixed-width code font, such as this:
  ```java
  JavaTeacher teacher = new JavaTeacher()
  ```
  - Any class names, such as `JavaTeacher`, method names, or other code fragments will also appear in the same font
  - If we want to emphasize a particular piece of code, we'll also bold it (and in the slide, change it's color) such as `HttpServlet`
  - Filenames will be in italics, such as `SearchServlet.java`
  - We sometimes denote more info in the notes with a `star` *
  - Lastly, longer code examples will appear in a separate code box as shown below

```java
package com.javatunes.teach;
public class JavaInstructor implements Teacher {
    public void teach() {
        System.out.println("Servlets/JSP are way cool");
    }
}
```

Notes:
The workshop consists of approximately 50% discussion, 50% hands-on lab exercises, including a series of brief labs

- Many of the labs follow a common fictional case study - JavaTunes, an online music store
- The labs are contained directly in the course book, and have detailed instructions on what needs to be done

The course includes setup zip files that contain skeleton code for the labs

- Students just need to add code for the particular capabilities that they are working with
- There is also a solution zip file that contains completed lab code

Lab slides contain an icon like the one in the upper right corner of this slide

- The end of each lab is clearly marked with a stop like this one to the right

Notes:
Release Level

◆ This manual contains instructions for creating and running the labs using the following platforms:

  – Java 5 or Java 6
  – Tomcat 6.0
  – Eclipse Java EE Edition
  – Derby database (Open Source DB available from Apache)

◆ All labs have been tested on Windows XP

◆ Complete lab instructions for this platform are included in the student manual

Notes:

◆ The instructions for the labs should work with almost any Eclipse Java EE edition
  – However, the basic Eclipse version does not have the server deploy support, and these instructions will not be suitable for basic Eclipse

◆ Sun Microsystems recently changed the version numbering of the Java Software Development Kit.
  – After version 1.4.2, the next version was called Java 5, and the next one Java 6
  – However, note that the default home directory path for a 5.0 installation contains references to the fact that it is really version 1.5 renamed, i.e., C:\Program Files\Java\jdk1.5.0.
Session 1 - Java EE Introduction and Architecture

Java EE Overview
Java EE Architecture

Notes:
Session Objectives - You Should Be Able to:

- Define Java EE, explain its role in the Java landscape, and list some of its important component APIs

- List the fundamental components of an enterprise application and offer one or more Java EE technologies for implementing each one

- Illustrate several Java EE application architectures and discuss some of the characteristics of each

Notes:
Java EE Overview

Java EE Overview

Java EE Architecture
Java EE's Place within Java

- **Java Standard Edition (Java SE)**
  - Platform for desktop computers and high-end small devices
  - Contains some APIs that are essential to enterprise applications (e.g., JDBC and JNDI)

- **Java Micro Edition (Java ME)**
  - Platform for small devices such as PDAs and cell phones
  - Subset of J2SE, provides a small-footprint Java environment

- **Java Enterprise Edition (Java EE)**
  - Platform for multitier enterprise applications
  - Depends upon Java SE
  - Adds enterprise-specific APIs such as Servlets/JSP and EJB
  - Current version - **Java EE 5**, Upcoming version **Java EE 6**
  - See note about Java EE 5 and Platform Names

**Notes:**
- The naming of the Java platform has changed - In Previous releases, the names included the "2" from Java 2 platform
  - So – the old name for Java EE was J2EE or Java 2 Platform Enterprise Edition - the previous release was called J2EE 1.4 - the current release is called Java EE 5
  - The old name for Java SE was J2SE, or Java 2 Platform Standard Edition

- In this course, we are covering both J2EE 1.4 and Java EE 5
  - We will mostly use the name Java EE

- Although Sun is the official owner of all Java specifications, it does not write them alone. It works with vendors and industry experts via the *Java Community Process* (JCP) in order to develop them collaboratively.
Java EE - Distributed Multitier Applications

- Java EE specification provides technologies to more easily develop multitier, enterprise services
  - Defines the APIs and how they interact
  - Java EE 5 is very available – some applications still running on J2EE 1.4 – though potentially using newer technologies (e.g. EJB 3), Java EE 6 is available though not widely adopted
  - For all things Java EE: http://java.sun.com/javaee/

- Other parts of the Java EE platform Include
  - **Glassfish Application Server** - Sun-based open source Java EE server - replaces the Reference Implementation (RI)
  - **Java EE Compatibility Test Suite** - Set of standard tests to determine if a particular product conforms to the specification
  - **Java EE Blueprints** - Documents from Sun that describe design guidelines for Java EE applications

Notes:
- Enterprises need to extend their reach
  - New customers, new services, Web access
- Need to combine existing EIS (Enterprise Information Systems) with new business functions
  - Highly Available, Secure, Reliable, Scalable
- These services generally built as multitier applications
- Glassfish is a full production level application server based on Sun's Java Application Server
  - Support available from Sun
  - Information available at: https://glassfish.dev.java.net/ and http://java.sun.com/javace/community/glassfish/
- Information about Java EE compatibility and Java EE-certified products can be found at http://java.sun.com/javace/overview/compatibility.jsp
- The Java Blueprints include a sample application – Java Pet Store
## Important Java EE 5 APIs

- **Servlet 2.5**  Java Servlet
- **JSP 2.1**  JavaServer Pages
- **JSTL**  JSP Standard Tag Library
- **JSF 1.2**  JavaServer Faces
- **EJB 3.0**  Enterprise JavaBeans
- **JPA**  Java Persistence API (JEE 5)
- **JMS**  Java Message Service
- **JAX-WS 2.0**  Java API for XML-based Web Services
- **JSR 181**  Web Service MetaData for Java (JEE 5)
- **SAAJ**  SOAP with Attachments API for Java
- **JAXB 2.0**  Java Architecture for XML Binding
- **J2EE Connector Architecture 1.5**

**Notes:**
- We list the spec number of both J2EE 1.4 and Java EE 5 technologies in the list above because we are still in a transition phase between the two.

- Recall that Java EE relies upon Java SE. Some of the important APIs from Java SE that are employed in Java EE include:
  - **RMI**  Remote Method Invocation
  - **RMI-IIOP**  RMI over Internet Inter-ORB Protocol
  - **JDBC**  Java Database Connectivity
  - **JNDI**  Java Naming and Directory Interface
  - **JAXP**  Java API for XML Processing
Commercial Java EE Products

◆ Since Java EE is an open standard, vendors compete on the quality of their implementations
  – Price, performance, ease of use, support, reliability, scalability
  – Additional functionality, e.g., clustering
  – Vendor reputation and stability

◆ The products on the market support different versions of the Java EE specification (Java EE 5, upcoming Java EE 6)

◆ Many vendors offer a suite of Java EE products
  – Base application server platform
  – Products with functionality layered on top of the base server
  – Scaled-down, low-cost alternative to the base server
  – Tools to aid in development and deployment

Notes:
◆ Information about Java EE-certified products, including what version of Java EE is supported, can be found at http://java.sun.com/javaee/overview/compatibility.jsp
◆ 

Session 1: Java EE Introduction
### Common Commercial Java EE App Servers

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>WebSphere Application Server</td>
</tr>
<tr>
<td>BEA (acquired by Oracle)</td>
<td>WebLogic Server</td>
</tr>
<tr>
<td>JBoss</td>
<td>JBoss Application Server (open source)</td>
</tr>
<tr>
<td>Oracle</td>
<td>Oracle Application Server</td>
</tr>
<tr>
<td>Sun (acquired by Oracle)</td>
<td>Sun Application Server for Java</td>
</tr>
<tr>
<td></td>
<td>Glassfish Open Source App Server</td>
</tr>
<tr>
<td>Apache</td>
<td>Geronimo</td>
</tr>
<tr>
<td>SAP</td>
<td>NetWeaver</td>
</tr>
</tbody>
</table>

*This list is not exhaustive!*

**Notes:**

- For some projects, an open source server such as JBoss might be a reasonable choice. For example, the Apache Web Server is open source and is one of the most popular Web servers. (In fact, some vendors in the list above use the Apache HTTP server in their Java EE products, adding functionality and support.)
Enterprise Applications - the Three Pillars

- **Persistent storage** maintains data
  - Typically a relational database, but could be a legacy system

- **Business logic** implements the rules of the business

- **Presentation logic** implements interactions with users

- Java EE has relevant technologies:
  - **JDBC** connects Java applications to databases
  - **Enterprise JavaBeans** and/or **POJOs** (Plain Old Java Objects) can be used for business logic
  - **Servlets and JavaServer Pages** can be used for Web presentation

Notes:
- There could be different types of legacy systems
  - Perhaps a mainframe application
  - The generic term is Enterprise Information System (EIS)

- It is not required that you use the Java EE technologies above to implement these fundamental components of an enterprise application, but they are the standard way to do so.
  - This is because Java EE specifies their behavioral semantics and thus the application server plays a major role in supporting the other requirements of the system, e.g., scalability, security, transactions, etc.
  - There are other presentation technologies, and some system designers choose not to use EJB at all, implementing business logic in classes of their own design.

- Swing is also available for presentation
  - It is not as widely used as Servlets/JSP
  - Web access is by far the most common choice for systems being built today
Persistent Storage

- **JDBC** provides connectivity to relational databases
  - Connects the Java parts of your application to SQL databases
  - You need a JDBC **driver** for each type of database in use
    - Drivers are specific to a database product and typically obtained from the database vendors
  - Part of Java SE, in the `java.sql` and `javax.sql` packages
  - See: [http://java.sun.com/jdbc](http://java.sun.com/jdbc)

- **Java Connector Architecture (JCA)** defines an architecture for connectivity to Enterprise Information Systems (EIS)
  - EIS vendors provide **resource adapters** specific to a type of EIS, analogous to a driver in JDBC
  - Resource adapters are plugged into an application server, enabling communication with the EIS
  - See: [http://java.sun.com/j2ee/connector](http://java.sun.com/j2ee/connector)

**Notes:**
- For the JCA, **system contracts** define the interactions between application server and resource adapter
  - The resource adapter implements the EIS side of these contracts
Business Components

- Business logic is the heart of an enterprise application
  - It's where business rules are implemented in code
  - In Java EE, business rules can be implemented with Enterprise JavaBeans or POJOs

- Enterprise JavaBeans (EJB) handle common enterprise system needs (e.g. security, scalability, transactions)
  - Frees developers from security and transaction coding and allows them to focus on the actual business logic
  - Java Persistence API (JPA) provides generation of persistent storage code, freeing the developer from JDBC coding, as well

- POJOs (Plain Old Java Objects) can also be used
  - Many organizations have opted not to use EJB because of its complexity and performance issues
  - POJOs are simple to program, but you don't get added value of EJB capabilities - with EJB 3.0, this distinction is smaller

Notes:
- Examples of business rules:
  - Shipments over 100 units get a 10% discount
  - Notify our sales team about daily customers
  - Get an authorization code for foreign shipments

- There is no requirement to use EJBs for business logic – you could certainly use regular Java components instead.
  - For small- to mid-sized systems, the overhead of EJB may actually impede performance.
  - You need experienced EJB developers who understand its complexities - EJB components are not trivial to write, and EJB projects often require longer development cycles.

- For certain kinds of systems, however, it is sensible to use EJB.
  - Typically, these are large-scale, distributed systems that require the benefits of resource pooling and make extensive use of the security and transaction services provided by the application server.

- EJB 3.0 simplifies the EJB programming model
  - It also makes EJB components more POJO-like
  - There is still a substantial leap in complexity and a performance cost over POJO
Presentation Logic

- Presentation logic determines how the application visually appears to its users
  - It also determines how users interact with the application

- Examples of user interactions:
  - Click on a hyperlink to find out current account balance
  - Fill out an HTML form and submit it to the application

- Java EE provides several options for presentation
  - Servlets
  - JavaServer Pages
  - JavaServer Faces
  - Standalone JFC/Swing clients
Presentation Logic - Servlets

- Servlets are Java components that respond to users making application requests via a Web browser
  - A servlet typically processes a request by reading input data from a user's request and interacting with a database
  - It can then generate a dynamic HTML response to the user

- Servlets are great for processing user requests, but not so great for presenting responses
  - Putting HTML response code in a Java servlet makes it harder to create and maintain, because there is no clean separation
  - Your HTML person should be editing it, not your Java person

- Servlets can forward requests to JSPs for display of data
  - The HTML resides in a JSP

Notes:

- Servlets are a replacement for CGI (Common Gateway Interface).
  - CGI programs are coded in C, Perl, and other languages.

- Servlets are more efficient than CGI programs because the invocation cost is lower.
  - Every request to a CGI program spawns a new process.
  - Every request to a servlet spawns a new thread inside an existing process.
JavaServer Pages are mostly HTML with just a little Java
- There is a JSP syntax and a set of JSP tags that get processed on the server side, before the page is sent back to the browser

They provide a way to separate the Java from the HTML
- Servlets don't include HTML response output
- Instead, they store any relevant data in memory and forward client requests to JSPs for presentation of that data
- Can use custom tag libraries to make the JSP even simpler

Servlets/JSP is an example of Model-View-Controller
- Data=model, JSP=view, servlet=controller

You can also use a Swing (Java) client for the presentation
- Web front ends are much more common, but these are still useful when the user interface is very complex, or otherwise not suitable for the Web

A typical servlet/JSP interaction works like this:
1. A user clicks on a link or submits a form, generating a request to the application.
2. A servlet acts as the controller by processing the request, possibly accessing a database and storing some data (the model) in memory for use by an ensuing JSP.
3. The servlet forwards the request to a JSP.
4. The JSP accesses the model and generates a view of it for the client, most likely via an HTML page.

JSPs are only one presentation solution. Servlets are pretty well accepted as an excellent solution for processing user requests, but there are several options for the presentation of data.

Servlets and JSPs are used when the client application is a Web browser
- No application code resides or executes on the client machine

For more elaborate user interfaces, you can use a J2SE client
- JFC/Swing is J2SE's graphical user interface toolkit
  - Swing clients are sometimes called rich clients

Implies that some application code resides and executes on the client machine
- Harder to maintain and update
Java EE Containers

- **Containers** are the interface between components (e.g. Web components) and low-level system functionality supporting them (e.g. threading, security, networking, etc)
  - In general, enterprise applications make heavy use of system capabilities like threading
  - Containers insulate the application from having to deal with these underlying services

- Java EE components are generally assembled and deployed to a container before being executed
  - For example, a Web app might be assembled in a WAR file, deployed to a Web container, and then be invoked by a client
  - We'll cover the details of all this in the course
Multitiered Architecture

- Presentation logic, business logic, and persistent storage are separated into different tiers

- These tiers may or may not be on physically separate machines
  - Though the database server is typically on its own machine

- Many enterprise applications will thus involve multiple physical servers and possibly multiple databases
Web-Based Architecture

- This is a popular Java EE architecture for Web applications
- It is relatively simple, yet suitable for many types of applications

Notes:
- The presentation code and business code reside in the same tier.
  - Database access can be implemented directly in the servlets/JSPs, but is often encapsulated into a data access layer.
  - Many types of systems can be built with this relatively simple architecture.
Web-Based Architecture with EJB

- This extends the previous architecture by using EJBs for business logic and data access
- This is another fairly standard Java EE architecture used for larger systems

**Notes:**

- You'll see this picture, or a variation of it, in almost every overview document about Java EE.
  - The difference between this and the previous solution is that the business code has been implemented in EJBs and exists in a different tier than the presentation code.
  - Larger, more scalable systems can be built with this architecture.

- Although EJBs are distributable and have remote method invocation (RMI) support built in, it is not required that they be invoked over a network via RMI.
  - The Web components and EJBs don't have to be on separate machines.
  - *Local* EJBs are resident in the same JVM as their client components and can be invoked via regular method calls. This reduces the overhead associated with RMI and allows for parameters and return values to be passed by reference instead of serialized and sent over a network.
## Swing Client Architecture with EJB

- This architecture utilizes client processing power and allows for rich client GUIs
  - Note: Persistent storage is not shown in remaining diagrams and will use one of the previous architectures shown

### Diagram:

```
+----------------+        +----------------+
|     client     |        | application server |
|                |        |                    |
| Swing GUI      |        | EJBs               |
|                |   RMI  |                    |
```

### Notes:
- Rich clients can interact directly with the business tier.
Supporting Both Web Clients and Rich Clients

- Web clients communicate via HTTP
- Rich clients can communicate via HTTP or RMI
  - This includes applets running in a browser

Notes:
- This architecture may be attractive because you can support both Web browser clients and Swing clients, and do so in several different ways.

- Web browser clients interact with the EJB business tier indirectly, via the servlet/JSP presentation tier.

- Swing clients can be installed on a J2SE client machine or written as a downloadable applet that executes in the JVM of a Web browser.
  - Swing clients can generate HTTP requests to the servlet/JSP presentation tier or can interact directly with the EJB business tier. You might want to do this to reuse an existing servlet/JSP interface or to use HTTP to get through a firewall.
### Playing Server Games

- Logical tiers can be located on different physical machines

![Diagram showing logical tiers of a Java EE application](image)

**Notes:**

- Once you have split your application into logical tiers, it is possible to move them onto different physical machines.
  - This flexibility is one of the strengths of this architecture.
  - The need for this kind of architecture might also influence your use of EJB.

- We will also cover some additional technologies
  - Web Services for HTTP/XML based service provision
  - JMS for asynchronous messaging

- There is no single correct way to architect a system.
  - It all depends on requirements and available resources.
SOA – Service Oriented Architecture

- SOA is an important basis for enterprise architectures

- Based on providing resources on a network
  - As independent services independent of their implementation
  - Results in loosely coupled architecture

- Usually based on Web Services Technologies
  - Such as SOAP
  - Provide standard APIs and implementation independence

- Services are still implemented using standard Java EE technologies
  - Such as servlets or EJB
**SOA Architecture**

- Another popular Java EE architecture
- Provides loosely coupled access via standard protocols

![Diagram showing SOA Architecture]

**Notes:**

- The presentation code and business code reside in the same tier.
  - Database access can be implemented directly in the servlets/JSPs, but is often encapsulated into a data access layer.
  - Many types of systems can be built with this relatively simple architecture.
What's Next?

- We will cover two of the key Java EE technologies
  - Servlets and JavaServer Pages
  - JDBC

- We will give an overview of some of the other technologies
  - EJB
  - Web Services
  - JMS
Session Review Questions

1. What is Java EE?
2. Name some Java EE technologies/APIs.
3. Does Java EE depend on J2SE? Why or why not?
4. Why is the Java EE Reference Implementation important?
5. What are the "three pillars" of an enterprise application? Which Java EE technologies can be used to implement each?
6. What is the role of the application server? Who provides it?

Notes:
Session Summary

- We defined Java EE and its affiliated components and APIs
- We discussed the "three pillars" of enterprise applications and Java EE technologies that can be used to implement them
- We illustrated several Java EE application architectures and discussed their characteristics

Notes:
Topic: Web Applications
We now cover the use of **Servlets** and **JSP** (JavaServer Pages) with **JSTL** (JSP Standard Tag Library) to build dynamic Web applications.

The topic starts with the basic architecture of the Web & how servlets work within that:

- It moves on to cover JSP (JavaServer Pages)
- Custom tags and the JSTL are introduced for creating complex actions
- Key design issues are introduced and illustrated
Objectives

At completion of this topic you should be able to

- Understand the design and development of Web applications using Servlets, JSPs and the JSTL (JSP Standard Tag Library)
- Develop Servlets to process HTML forms
- Understand and create JavaServer Pages (JSPs)
- Understand and use custom tag libraries
- Understand and use the JSTL
- Link Servlets and JSPs, and share data between them
- Store and process session information
- Deal with concurrency issues
- Understand and use good design techniques including MVC (Model-View-Controller) and the Model 2 architecture
Workshop Agenda

- Session 2: Web Application Basics
- Session 3: Servlet Basics
- Session 4: Additional Servlets Capabilities
- Session 5: JSP (JavaServer Pages)
- Session 6: Using Custom Tags
- Session 7: Session Management
- Session 8: More JSP Capabilities
- Session 9: More JSTL and EL
- Session 10: Security
- Session 11: Additional Capabilities
Session 2: Web Application Basics

How the Web Works
Java EE Web Applications
Servlet Basics - Generating HTML Dynamically
Lesson Objectives

- Understand the Basics of HTML and HTTP
  - Understand Clients, Servers and the Internet, HTML tags, and HTTP requests

- Get an overview of Java EE
  - Understand Java EE Web Applications structure
  - Create a simple Java EE Web Application

- Gain an understanding of what servlets are
  - Learn the basics of the Servlet API
  - Create a Simple Servlet
  - Learn how to map a servlet in web.xml
How the Web Works

Java EE Web Applications
A Simple Servlet
Clients and Servers

- The World Wide Web is composed of clients and servers connected by the Internet
- Clients - Programs that **request** information on the web
  - e.g. browsers like Mozilla Firefox or Microsoft Internet Explorer
- Servers - Programs that **respond** with information to clients
  - Web servers or application servers – e.g. Apache Tomcat, WebSphere Application Server, GlassFish, etc.

If you want to see more graphs of the internet, check out http://www.caida.org
- Their skitter tool probes the internet for topology and performance
- It can visualize network from the data it collects

- The most common client is a web browser, used by a person
  - Other clients such as Swing applications can also exist
  - Computer programs can be clients also

- Servers may do many other things than simply serving up web pages
  - It depends on the server, and upon the system that is running on it
HTML - The Language of the Web

- Web servers supply pages to browsers as Hypertext Markup Language (HTML)
  - HTML is a plain text format with added tags
  - Browsers use HTML tags to interpret and format content
- HTML uses tags to mark different parts of a document
  - i.e. - To denote the document structure, such as the page title
  - `<TITLE>Search page</TITLE>`
- HTML also uses tags for formatting (see notes)
  - `<B>This will appear in bold in a browser</B>`

Notes:
- Basic HTML Tags
  - `<HTML>` - Starts an HTML page
  - `<HEAD>` - The heading part of a page
  - `<TITLE>` - The title of a page
  - `<BODY>` - The body (main content) of a page
  - `<B>` - Bold text
  - `<BR>` - “Break”, or start a new line
  - `<P>` - Start a new paragraph
  - `<HR>` - Insert a horizontal line
  - `<A>` - Insert a hypertext link
    `<A href="elsewhere.htm"> Click me!</A>`
- Go to http://www.w3.org/MarkUp/ for more info on HTML
- All HTML pages should have at least the following the `<HTML>` and `<BODY>` elements
  - Most will have `<HEAD>` and `<TITLE>` elements
  - Browsers are very forgiving. Pages with errors often work
HTTP, Addressing, Requests, and Responses

- Servers and browsers communicate via the Hypertext Transfer Protocol (HTTP)

- A client specifies a server with an IP address
  - This can be a name, like LearningPatterns.com
  - Names translate to numeric addresses, like 206.98.161.38
  - HTTP servers run on specific ports
  - The default is port 80, Tomcat runs on port 8080

- The browser sends an HTTP request to a server for a page
  - With a URL: http://www.mycompany.com/index.html
  - The server returns the page in an HTTP response
  - Both requests and responses include extra bits of information, called headers
    - The request is mostly composed of headers
    - The response includes headers and the HTML itself

Notes:

- A basic understanding of HTTP us useful in knowing how the web, and servlets, work
  - You’ll be interacting with it later through the servlet API
- You can get more information on HTTP at: http://www.w3.org/Protocols/
- If the server is using a different port, the client should ask for it specifically - e.g.
  - http://www.learningpatterns.com:8080
- URL = Uniform Resource Locator
- Finding a server is like looking up a phone number and making a phone call
  - The symbolic name, is like the name of the person you are calling.
  - Using the symbolic name, you look up the IP address (using something called DNS)
  - The IP address is like a phone number. It specifies a specific computer on an IP network
  - The port is like an extension
  - There may be a number of applications on the same computer listening for connections
  - The port is what directs incoming connections to a specific program (in this case an HTTP server listening on port (extension) 80
Browsing a Simple Web Page

GET /javatunes/index.html

Hello

... </HTML>

www.learningpatterns.com

Notes:
Java EE Web Applications

How the Web Works
Java EE Web Applications
A Simple Servlet

Notes:
Java EE - Java Platform Enterprise Edition

- Java EE is Java’s standard architecture for building scalable, distributed, reliable, Enterprise applications
  - We will use a Java EE architecture to study and implement all of these capabilities
  - Formerly (and still widely) called Java EE
  - Java EE includes technologies such as Servlets/JavaServer Pages (JSP), JDBC, EJB, JMS, JavaMail, etc.

- We will focus solely on the Web tier during this topic
  - And the base Servlet/JSP technology
  - We will delve into the other technologies such as JDBC for database access, and take a brief look at EJB for creating transactional, scalable components afterward

Notes:
- Web applications are part of the Java EE standard
  - This gives your applications consistency and portability across implementations
Simple Web-Based Architecture

- This is a popular Java EE architecture for Web applications
- It is relatively simple, yet suitable for many types of applications

Notes:
- The presentation code and business code reside in the same tier.
  - Database access can be implemented directly in the servlets/JSPs, but is often encapsulated into a data access layer.
  - Many types of systems can be built with this relatively simple architecture.
Java EE Web Applications

- **Web application** - A standard structure defined in Java EE for organizing the resources in a Java EE system
  - It is a collection of servlets, JavaServer Pages, and other files
  - This is the standard Java EE way to package up server programs for the Web

- Every web app has a standard directory structure and includes an XML configuration file - **web.xml**
  - **web.xml** (commonly called the *deployment descriptor* or DD) contains configuration information (XML) for the Web app
  - You can declare servlets, servlet mappings URLs and more

- Often, the Web app is packaged into a Web Archive (WAR)
  - Based on jar, and is the standard packaging for a Web app
  - Can be deployed directly in servers supporting Java Web apps
  - Often servers support "exploded" deployments *

**Notes:**

- A jar file is basically a zip file with added information in it
  - It is compatible with the zip format, and jar files can be manipulated using most zip tools
  - There is also a utility program that comes with Java to create and work with jar files

- A WAR file is just a jar containing a Web app
  - Some servers also support the deployment of the Web app as a directory containing the needed files
  - This directory would have the standard Web app structure, and contain the same contents as in the WAR file
  - This is convenient if you have to change something (e.g. an HTML or JSP page) in the Web app
Web Application Structure

- `<Web application base directory>`
  - `[static content files: HTML, forms, images, etc.]`
  - `[dynamic content: JSPs]`
  - `[other directories for content]`

- WEB-INF
  - `web.xml`
  - `classes`
    - `[.class files: servlets and others]`
  - `lib`
    - `[JAR files]`

- The root of the directory hierarchy serves as the document root for files that are part of the application
  - The WEB-INF node is not part of the public document tree
  - Nothing in WEB-INF can be served directly to a client

Notes:

- The WEB-INF directory is just a normal directory of this name
  - It’s an unusual name, but don’t be confused by that. It’s just a directory

- You can put things in WEB-INF that you don't want visible directly to clients
  - For example application configuration files
  - You can still forward/include them via the RequestDispatcher that we will cover soon
  - Some developers put all the jsp files that are normally forwarded to by servlets
  - This prevents clients from browsing directly to them
Lab 2.1 - Create a Web Application
Lab 2.1 – Create a Web Application

◆ **Overview:** In this lab, we will setup and become familiar with the lab environment, and create a simple Web application
  – The Web app will just have a simple HTML file in it
  – We will build, deploy and run it from within Eclipse
  – This will take a little time to set up, but future labs will take less setup

◆ **Objectives:**
  – Become familiar with the lab structure and Eclipse
  – Create a Web application and deploy it using Eclipse

◆ **Builds on previous labs:** None

◆ **Approximate Time:** 30-40 minutes
Information Content and Task Content

- Within a lab, information only content is presented in the normal way – the same as in the student manual pages
  - Like these bullets at the top of the page
- Tasks that the student needs to perform are in a box with a slightly different look to help you identify them - like the one below

Tasks to Perform

- Look at these instructions, and notice the different look of the box as compared to that above
  - Make a note of how it looks, as future labs will use this format
- **Make sure that you have Eclipse installed**
  - Likely installed in a directory like C:\eclipse *
  - We'll refer to this directory as <eclipse>
  - If it is not installed, or you can't find it, tell your instructor
- **OK – Now get out your setup files; we're ready to start working**

Notes:

- If Eclipse is not installed, then you'll have to download and install it
- If you need to download Eclipse, go to http://www.eclipse.org/downloads
  - Under the Eclipse Packages section, click on the link for the Eclipse IDE for Java EE Developers
  - Save the zip file to your computer
- To install Eclipse, unzip the zip file (easiest location to unzip it to is C:\, but another location is fine as long as you can get to it to run the eclipse.exe executable)
Extract the Lab Setup Zip File

- To set up the labs, you'll need the course setup zip file *
  - It has a name like: JEE-Tomcat_LabSetup_20101108.zip
- Our base working directory for this course will be

  \texttt{C:\StudentWork\JEE}
  
  - This directory will be created when we extract the Setup zip
  - It includes a directory structure and files (e.g., Java files and other) that will be needed in the labs
  - All instructions assume that this zip file is extracted to C:\. If you choose a different directory, please adjust accordingly

Tasks to Perform

- Unzip the lab setup file to \texttt{C:\}
  
  - This will create the directory structure, described in the next slide, containing files that you will need for doing the labs

Notes:

- The setup zip will either be given to you on a CD or supplied by your instructor
  
  - The nnnn stands for the version number of the course

- The setup zip may also contain additional folders with extra content
  
  - \textbf{InstallFiles}: Extra software install files (e.g. simple editor)
  - \textbf{Resources}: Documentation, specifications, etc.
  
  - Which extra files are supplied will vary based on space limitations
Lab Directory Structure

- **StudentWork\JEE** contains
  - **Derby**: Database files
  - **Resources**: Extra files (e.g. docs)
  - **LabSetup**: Files needed for lab work
  - **workspace**: Lab working directories

- **StudentWork\JEE\workspace** will contain the following folders:
  - **common**: shared files
  - **javatunes**: Lab directory (a Web app)
    - **javatunes/src/**: Java source
    - **javatunes/build/classes/**: compiled code
    - **javatunes/WebContent/**: jsp, HTML
    - **javatunes/WebContent/lib**: Jar files
    - **javatunes/WebContentWEB-INF/**: web.xml

Notes:

- All our labs will be Web applications in this topic
  - For other topics, we'll also have plain Java applications
  - In that case, the lab directory will have a different structure suitable for a plain Java app
The Eclipse Platform

- Eclipse ([www.eclipse.org](http://www.eclipse.org)) is an open source platform for building integrated development environments (IDEs) -
  - Used mainly for Java development - can be extended via plugins and used in other areas (e.g. C# programming)
- Originally developed by IBM
  - Released into open source
  - IBM's RAD environment is built on top of Eclipse
- Eclipse products have two fundamental layers
  - The **Workspace** – files, packages, projects, resource connections, configuration properties
  - The **Workbench** – editors, views, and perspectives
- We will set up the workspace and workbench, then describe it in more detail at the end of the lab

Notes:
- The Workbench sits on top of the Workspace and provides visual artifacts that allow you to access and manipulate various aspects of the underlying Workspace resources.
Eclipse and Web Projects

- We'll use the **Eclipse Java EE** edition in this class
  - Has support for Java Web applications

- **Eclipse organizes Java Web apps using Dynamic Web Projects**
  - **Dynamic** Web projects contain Java EE resources such as servlets, JSP pages, plus static resources (HTML)
  - You establish project properties for the Web Project at creation time and can modify them later
  - It provides a custom editor for the `web.xml` file

- The **Eclipse Web project organization** is different from how the final Java EE Web application organization will be
  - It is designed to make it easy for you to work with the resources
  - When deployed, a standard WAR is build

**Notes:**
- **Eclipse also provides Static Web projects** that can be used when you aren't generating dynamic content and don't want any of the overhead associated with dynamic Web content

- When you create the project, you can set many properties for it, including:
  - Build path, project references, default server for deployment, and the application’s context root
  - Uses the build path value to resolve references in the compiling code

- The `web.xml` file holds many additional settings
  - Servlets to be run in project
  - Initial (welcome) pages and error pages
  - Environment values and JNDI references to resources made in servlets/JSP’s
Web Project Organization

- Organized in the following folders
  - **src**: Contains all Java source files
  - **WebContent**: Contains all Web resources
  - **WebContent\WEB-INF**: Same as Java EE WEB-INF
- Usually use **Web Perspective** or **Java EE Perspective**
- All visible elements are not necessarily deployed with the project
  - e.g. the src folder is not deployed - only compiled classes
  - Eclipse creates a standard WAR file when it deploys
- Before Web components are developed you must create and configure a new Web Project
  - You can specify the build path for the project to include external jar’s or class files
- When Web projects are created, Eclipse automatically creates the associated Deployment Descriptor (DD)
  - **web.xml** - DD for Web project in **WEB-INF** folder

Notes:
- The src folder contains all Java source
  - Servlets, supporting classes, JavaBeans
  - When you compile these classes, the compiled output is put in the WEB-INF\classes directory
- The WebContent folder contains all Web resources
  - HTML files, JSP files, image files, etc.
  - Only content in this folder (or a sub-folder) can be accessed in the Web application
- HTML and JSP pages must be correctly placed relative to the context root
  - Media files, JAR files, loose class files and other resource libraries must also be correctly placed
- Both the Web and Java EE views are useful for managing Web projects
  - Web perspective contains HTML/JSP oriented views not in J2EE perspective
Getting Started With Eclipse

Tasks to Perform
- Make sure you have **Eclipse installed** - likely in C:\eclipse
- **Launch Eclipse**: Go to c:\eclipse and run **eclipse.exe**
  - A dialog box should appear prompting for a workspace location
  - Set the workspace location to **C:\StudentWork\JEE\workspace**
  - If a different default workspace location is set, change it

**Lab 2.1: Create a Web Application**

**Notes:**
- You can also put a shortcut on your desktop to start Eclipse
Fast Track to Java EE

Task 1: Workbench and Java EE Perspective

**Tasks to Perform**
- Close the Welcome screen (click the X on its tab)
- You'll likely be in a Java EE perspective - depending on which Eclipse version you use
  - That's fine, this perspective is good for what we do
- If you need to open the Java EE perspective, (Shouldn't need to do this now) you can do so by clicking the Perspective icon at the top right of the Workbench, and select Other | Java EE (as shown below)

![Perspective Icon](image)

**Notes:**
- The workbench icon is in different places in Eclipse 3.4 (below on left) and 3.5 (below on right)
Notes:
Creating a Server

- We will use the Tomcat to run our Web applications - first we need to create a server in Eclipse *

**Tasks to Perform**

1. Go to the Servers view, right click, and select New | Server
2. In the next dialog, select Apache | Tomcat V6.0 * and click Next
3. In the next dialog, browse to your Tomcat install directory, and click Finish *

**Notes:**

- If you use a Tomcat version other than 6.0, then select the appropriate version in the dialog where you choose the server

- Eclipse for Java EE has support for deploying Web applications to a configured server
  - It also has support to start and stop the servers from within Eclipse

- If you click Next instead of Finish in Step 3, you'll come to a dialog that lets you configure the project to run on the server
  - We are going to do this slightly differently
Fast Track to Java EE

Lab 2.1: Create a Web Application

Create a Web Project

Tasks to Perform

- Create a Dynamic Web project
  - File | New | Dynamic Web Project *
  - Call it javatunes *
  - Use the Tomcat runtime *
  - Click Next (click Next twice if using Eclipse 3.5)
- In the next dialog, make sure the Context Root is javatunes
  - Click Finish

Notes:

- There are multiple ways to create a new project:
  - Click on the “New Wizard” button in left side of the toolbar
  - Right click in the Package Explorer View, select New → ...
  - etc.

- Note - When you call the project javatunes, it's stored in workspace\javatunes

- The Tomcat runtime should automatically be used, unless you have other servers installed
  - If that's the case, make sure the Tomcat runtime is selected
  - This will also set the Web Module to Servlets 2.5

- Eclipse 3.4 and 3.5 have slightly different dialogs for creating a Dynamic Web project - but they are very similar
  - You shouldn't have any problems with either
The Project Explorer View

- Open the Project Explorer View

- Java EE oriented display
  - Not file oriented
  - Organized into groups based on type of project
  - Resources in a project are displayed in a view specific way
  - For example the web.xml deployment descriptor
Open the Navigator View (Window | Show View | Navigator)

Look at the Navigator view to see the Web project you just created

File system – like view
  - Organizes Java source, Web content
  - Knows about deployment descriptors

Note that Eclipse has create the deployment descriptor, web.xml, for you

Double click on web.xml to open it for viewing and editing
The web.xml file

- Review the **web.xml** file by clicking the editor **source tab**
  - `<web-app>` is the root element, and contains the schema declaration from the servlet/Java EE specification
  - `<display-name>` is a descriptive name used by tools
  - `<welcome-file-list>` is a list of default files to be displayed if a file is not specified on the request URL

```xml
  <display-name>javatunes</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Lab 2.1: Create a Web Application

Notes:
Create an HTML file

**Tasks to Perform**

- In the **Project Explorer** View, right click on the **WebContent** folder
  - Select **New | HTML File**
  - In the dialog that comes up, call the page **index.html**, and click **Next**
  - Select **New HTML File (4.01 transitional)** in the Template pane
  - Click **Finish**

- Eclipse will create the web page in the **WebContent** folder, and open it in an editor for you
Edit the HTML File

Tasks to Perform

◆ Review the HTML file in the editor, and add some simple text or HTML into the body element

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
</body>
</html>
```

Add content

Notes:

◆ You can add the following HTML into the page if you want

```html
<body>
    <h1>Welcome to JavaTunes</h1>
    [add whatever else you want]
</body>
```
**Deploying an Application**

**Tasks to Perform**
- To deploy to the server, right click on the javatunes project, and select **Run As | Run on Server**
- In the next dialog box select the existing **Tomcat v6.0** server
- Also select **Set server as project default**, as we will always use this server
- Click **Finish**
- Note that running a Web app on the server will automatically start it (look at the Servers view)

**Notes:**
Viewing the Web Application

- Eclipse will automatically open a Web browser for you onto the Web application
  - Note that the built in browser (IE) is sometimes misleading because it caches Web pages, and it's hard to clear the cache
  - Also - sometimes the browser window comes up before the server has loaded the Web app - try a reload if a resource can't be accessed
  - If you ever feel your having browser issues, open an external browser viewing the same URL (note that you don't have to specify index.html since it's one of the files specified as a welcome file)
  - That's it – your Web app is up and running

Welcome to JavaTunes

Notes:
- Tomcat listens on port 8080 by default
Server Startup

- You can open the **Console** view to see output from the server startup
  - This is useful to look for exception stack traces in later labs
  - Note that server startup may take some time, especially the first time you start the server
  - You can also look at the server status in the **Servers** view

```
INFO: Starting service Catalina
Jul 8, 2008 2:15:49 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.10
INFO: Starting Coyote HTTP/1.1 on http-8080
Jul 8, 2008 2:15:50 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jul 8, 2008 2:15:50 PM org.apache.jk.server.JkMain start
INFO: JK running ID=0 time=0/78 config=null
Jul 8, 2008 2:15:50 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1641 ms
```

Notes:
A Simple Servlet

How the Web Works
Java EE Web Applications
A Simple Servlet
Servlets and Dynamic Content

- Servlets are Java objects that run on the server, in response to client requests
  - Servlets are instances of a class whose methods are invoked in response to client requests
  - They can be used to generate dynamic content for the response

- Some content must be generated dynamically
  - For example, the results of a search
  - Servlets can be used to generate this dynamic content

- Previous approaches to dynamic content had many limitations
  - Common Gateway Interface (CGI)
  - Server extensions like NSAPI and ISAPI

Notes:
- Some Web pages don’t ever change (static content)
  - Static Web content is simply files sitting on a server somewhere
- Dynamic content is generated on the fly from information
  - For example, if you do web banking, you would want to see your most current account information
- Early in the evolution of the web, there was no dynamic content
  - People were using it for very simple things
- Eventually, the Web evolved to where people were doing things that required dynamic content
- There were two approaches used
  - CGI - A separate process which when invoked generated the dynamic content
  - Proprietary server extensions which allowed you to generate dynamic content
  - The proprietary extensions were basically server side plug-ins
What are Servlets?

- Servlets run within a container that exists as part of a server
  - **Container** is a name for a set of services that must be available to a servlet for it to execute properly
  - It provides network services, handles MIME-based requests/responses, manages servlet lifecycle
- The servlet container is usually a Web server or server plugin
  - Container must support HTTP(1.0/1.1), and usually HTTPS
  - Servlet support is widespread
- Most servlet containers are also full-fledged Web servers
  - For example, Tomcat can run as an HTTP server
  - It can also be a plugin for Apache or other servers
- There are many other servers supporting servlets:
  - JBoss, BEA WebLogic, IBM WebSphere, Sun Application Server, etc.

Notes:

- Servlets are the Java EE way to respond dynamically to client requests
  - They work alongside JSPs which we talk about later
  - Servlets are really instances of a class whose methods are invoked by the servlet container

- The servlet container can be provided entirely by the server itself or by a server plug-in
  - Tomcat and other application servers can act as a plug-in servlet container for well known servers like Apache
  - You can architect your system differently depending on your needs
How a Servlet Runs

Browser sends request to server

Server passes request to servlet container

Servlet container passes request to servlet class

Servlet class

Servlet sends response to browser

Instance of Servlet class

Notes:

- A servlet is actually an instance of a Java class
  - It implements a specific interface
  - The methods of the interface are called by the container in a manner that is clearly defined in the servlet spec
  - You write the class to do whatever work you need done to generate your dynamic content
  - For example, you might do a database lookup and then generate HTML from the results of the lookup
# Advantages of Servlets

- **Fast performance**
  - A servlet call is just a method invocation in a thread
  - Every CGI call is a separate process

- **Rich APIs are available**
  - Functionality like sessions that wasn’t available in CGI
  - Easy communication between servlets in same JVM

- **Programmer productivity**
  - Written in Java and uses a standard API

- **Safety**
  - Protects against things like buffer overflows (won’t crash server)
  - Allows sandbox-style restrictions on servlets

## Notes:
- CGI was basically a big hack to enable dynamic content
  - When it was created, the systems using it were still relatively small
  - It was never meant to run today’s large scale web sites
  - It is slow, even with a later version called Fast CGI
  - It is not very portable, and is vulnerable to hacks/attacks
  - Servlets are much better

- **Proprietary extensions**
  - Are just that - proprietary
  - No portability at all
  - They were often hard to work with also
  - They were often written in C, and could crash your server

- **Servlets are much faster than CGI**
  - They are also standard
  - It is easier to find developers who can use servlets
  - You are very portable if you want to change platforms
Packages and Classes

- Servlets are a standard extension library

- `javax.servlet` is the core servlet package

- `javax.servlet.http` provides a set of HTTP-specific servlet functionality

- `javax.servlet.Servlet` is the core interface, implemented in the abstract class `GenericServlet`

- You will most often extend `javax.servlet.http.HttpServlet`, an abstract subclass of `GenericServlet`

- We will go over these types in detail in the next section

Notes:

- It is one of the key parts of Java EE (Java Enterprise Edition)
  - It is a required part of Java EE
  - Vendors that implement Java EE platforms must support servlets

- A standard extension library is a standardized Java API that is not part of the core Java implementation
  - Vendors can support it where it makes sense (for example in a web server), and ignore it where they don’t need it
  - It will be the same on all implementations
Creating a Servlet - The Simplest Way

- Subclass the `HttpServlet` class and define a `doGet()` method
  - This overrides the `doGet()` method defined in the `HttpServlet` class
  - For now, we'll show a very simple servlet without thinking about any other details

- We will actually create a servlet, then install it and invoke it
  - We'll see how to do in the upcoming lab
  - Don't worry about all the details that you see - we'll look at them all in detail later

- When this servlet is invoked, the container will execute the `doGet()` method
  - The resulting output will go back to the browser

Notes:
A Simple HTTP Servlet

```java
package com.javatunes.web;
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class SimpleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            // Here is where output is generated
            // This gets sent back to the client
            out.println("Hello World");
            out.close();
    }
}
```

Notes:

- There is a lot here we don’t understand yet

- The important point, is that we can write Java code here (out.println("Hello World")) and the container takes care of all the mechanics of getting this output from the servlet, back to the client (the browser)

- In a more complex application, you could first do more complicated things, like a database lookup, and then use it to send output to the client

- Note: In general, we will NOT generate HTML output this way (print statements in a servlet)
  - This is only illustration
  - We will see how to do this more elegantly when we learn JSP
Declaring and Mapping Servlets

- A servlet must be declared in web.xml so the container recognizes it
  - This includes the servlet class and its URL mappings, as shown below
  - Again - don't worry about the details, we'll cover them later

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
  <display-name>JavaTunes</display-name>
  <servlet>
    <servlet-name>SimpleServlet</servlet-name>
    <servlet-class>com.javatunes.web.SimpleServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SimpleServlet</servlet-name>
    <url-pattern>/simple</url-pattern>
  </servlet-mapping>
</web-app>
```

Notes:
- The mapping can use wildcards
  - For example / to map to all URLs
  - Or *.jsp to map to all files ending in JSP

- The schema elements shown are for the Servlet 2.5 specification
  - Depending on your app server, you might have a schema for a later or earlier specification, such as Servlet 2.4
  - The differences in terms of what you learn in this course are mainly in the schema element in <web-app>
  - Everything else that we cover in this course is the same in Servlet 2.4 and 2.5
Lab 2.2 - Create a Simple Servlet
Lab 2.2 - Create a Simple Servlet

◆ **Overview:** In this lab, you'll create a very simple servlet that sends HTML back to the client
  – It will be a subclass of `HttpServlet` that contains a `doGet()` method to handle a request
  – You will also be introduced to the JavaTunes online store which the labs are focused on

◆ **Objectives:**
  – Become familiar with servlets, `HttpServlet` and `web.xml`
  – Be introduced to the JavaTunes lab structure

◆ **Builds on previous labs:** 2.1

◆ **Approximate Time:** 25-35 minutes
The JavaTunes Online Store

- The Web application we are creating is a small piece of the JavaTunes online music store
  - It displays a search form
    - The search form sends a request to a servlet that does a search and forwards to a results page that displays the results
    - The results page allows you to add items to a shopping cart, which are then displayed on a shopping cart display page
    - We are creating the servlet that processes the search request
    - The complete flow for JavaTunes appears following this slide, but we'll go into more detail on it later

Notes:
Complete Web Application Flow

User submits the search form

User clicks on an “Add to cart” link

Item is added to shopping cart

Search results page displayed in browser

Cart contents page displayed in browser

SearchServlet

CartServlet

searchResults.jsp

cartDisplay.jsp

Notes:
Creating the Servlet

- In this lab, we will create a class called **SearchServlet**, a subclass of `javax.servlet.http.HttpServlet`
  - It should be in the `com.javatunes.web` package

- You will define a `doGet()` method like that in the manual examples that sends back simple HTML
  - You can send back different HTML in the `println` if you want
  - For example: "<html><body>Searching</body></html>"

- Eclipse does most of the work of creating the servlet
  - See the next slides

**Notes:**

- We will be putting all our code in packages.
  - This adds a little complication, but it is important to understand.
  - It also reflects what you'll see in the real world.
Declaring and Mapping Servlets

- All servlets need to be declared in *web.xml*
  - Including their implementing class and their associated URL mappings
  - You associate a name with the servlet, which is used in other parts of the *web.xml* file to refer to the servlet
  - This will be **done by Eclipse** in this lab

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" ... >
  <display-name>javatunes</display-name>
  <servlet>
    <description>Servlet to execute Search</description>
    <display-name>SearchServlet</display-name>
    <servlet-name>SearchServlet</servlet-name>
    <servlet-class>com.javatunes.web.SearchServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SearchServlet</servlet-name>
    <url-pattern>/search</url-pattern>
  </servlet-mapping>
</web-app>
```

**Notes:**
- The mapping can use wildcards
  - For example /* to map to all URLs
  - Or *.jsp to map to all files ending in JSP
Creating a Servlet

**Tasks to Perform**

- Right click on the `javatunes` project
  - Select **New | Servlet**
- In the first dialog box (there are several):
  - In Java package enter `com.javatunes.web`
  - For Class name enter `SearchServlet`
  - Click **Next**

**Notes:**

- Note that the project needs the servlet jar on its classpath to compile
  - Because we associated this project with the Tomcat runtime, Eclipse automatically puts the Tomcat servlet jar on the project's classpath
  - If we had not associated the project with the Tomcat runtime (for example, if we had created the project before we created the server), we would need to manually add the servlet jar to the classpath
  - For Tomcat 6.x, the jar is: `<Tomcat-Install>\lib\servlet-api.jar`
- To add the jar to the project manually (you should NOT normally need to do this) you would:
  - Right click on the javatunes project in Project Explorer and select **Build Path | Configure Build Path**
  - Select the **Libraries Tab**, and click **Add External Jars**
  - Browse to the appropriate directory (above) and add in the `servlet-api.jar` file
  - Click **OK**
Creating a Servlet

**Tasks to Perform**

- In the next dialog
  - In Servlet Name enter **SearchServlet**
  - Under description, enter "**Servlet to execute search**"
- **Select** the default `/SearchServlet` URL mapping and edit it to be `/search`
- Click **Finish**

*Notes:*
**Edit the Generated Servlet**

**Tasks to Perform**

- Eclipse generates a servlet for you, and should open it for editing

---

```java
package com.javatunes.web;

import java.io.IOException;

/*
 * Servlet implementation class SearchServlet
 */

public class SearchServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /*
     * Default constructor.
     */
    public SearchServlet() {
        // TODO Auto-generated constructor stub
    }

    /*
     * See HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        super.doGet(request, response);
    }

    /*
     * See HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) {
        super.doPost(request, response);
    }
}
```

**Notes:**

- If it is not open for editing, open it by double clicking on it in the Project Explorer view

- You may see a warning in the Problems view about servlet-api.jar not being exported, and that possibly causing problems
  - This isn't a problem, because the Web app will be running in a servlet container (Tomcat in this case) that has its own copy of servlet-api.jar
  - You can right click on this problem, and select Quick Fix, then select "Exclude the associated raw classpath entry … "
  - This will remove the warning
Finish the doGet() method

Tasks to Perform

- Finish the `doGet()` method so that it sends output to the browser
  - **Delete** the automatically generated `super.doGet()` line
  - See the manual examples that get a `PrintWriter` from the response object, and use that to send back back simple HTML
- You can send back different HTML in the `println` if you want
  - For example: "<html><body>Searching</body></html>"
- In Project Explorer, right click on `SearchServlet.java`, and select **Run As | Run on Server** (If prompted to restart the server, click OK)
  - This will open a browser pointing to `http://localhost:8080/javatunes/search`
  - The container will go to the `javatunes` web app
  - In the Web app, it will see that `/search` is mapped to a servlet
  - It will invoke the `doGet()` method on the servlet, and send the results back to the browser
  - Your browser will display the HTML (see notes)

Notes:

```
web.xml  index.html  SearchServlet.java  http://localhost:8080/javatunes/search

http://localhost:8080/javatunes/search

Search results: (not implemented yet)
```
The **web.xml** file contains information about the servlets and other resources in a Web application

- **Eclipse created a web.xml for us**
- Open it in source view and look at the elements listed below
- You don't need to do anything with these elements as Eclipse filled them in with the information you gave when creating the servlet

For a servlet, we can declare:

- `<servlet-name>` - An **internal name** used within the XML file
- `<display-name>` - A **descriptive name**, used in tools
- `<servlet-class>` - The actual **class name** of the servlet
- `<url-pattern>` - A **URL pattern**, which tells the server which URL requests map to the servlet

**Lab 2.2: Create a Simple Servlet**

Notes:

- There are other entries in the web.xml file that we will get to later
The screen shot shows a Servlet 2.4 version web.xml

- The 2.5 version would be exactly the same, but would have the schema elements for 2.5, as shown earlier
Review Questions

◆ What is a servlet?
◆ Why do we bother with servlets?
◆ How do you write a servlet?
Lesson Summary

- Servlets are Java classes that can be used by servers to respond to Web client requests
  - Providing a standard efficient mechanism to build Web apps
- Servlets are very easy to create
  - For example, you can just extend HttpServlet and override its doGet() or doPost() method
- HTML is a text based markup language that uses tags to format content that is displayed in a browser
- HMTL is used by marking up the information you want to display with appropriate HTML tags
- HTTP is the protocol that governs how information is exchanged between a Web client and a Web server
  - Information is requested by a client, and supplied to the client in a response from the server
Session 15 - Java EE Database Integration

DataSource, ENC, and JNDI
Connection Pooling
<table>
<thead>
<tr>
<th>Session Objectives - You Should Be Able to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>◆ Explain the advantages of using <em>DataSource</em> over <em>DriverManager</em></td>
</tr>
<tr>
<td>◆ Understand resource references and the environment naming context (ENC)</td>
</tr>
<tr>
<td>◆ Configure a <em>DataSource</em> reference in a Web application, and use <em>@Resource</em> to inject it into a servlet</td>
</tr>
<tr>
<td>◆ Be familiar with connection and statement pooling, and understand why it is used</td>
</tr>
</tbody>
</table>

Notes:
# Java EE and JDBC

- The Java EE platform introduces another layer between your code and the database

- Application code can employ the Java EE container in obtaining a connection factory
  - These database client objects are commonly servlets or EJBs

- This factory is an instance of `javax.sql.DataSource`
  - Obtained by directory lookup
  - Clients get connections via `getConnection()` invocations on the data source object

- The Java EE container manages the connection to the database
## Why DataSource?

- **Reduce dependencies**
  - Client code only needs to know the name of the data source
  - The Java EE container stores the specifics for the database
    - Driver class name
    - Connect URL
  - You can change these without having to modify your code

- **Authentication**
  - The JEE container can authenticate to the database
  - Usually with the username and password of a pseudo-user
  - Or application code can supply a username and password

- **Connection pooling**
How do Clients Get Access to a DataSource?

- Generally, you need a reference to invoke on an object
  - How does a client get this reference?
  - In this case, the client happens to be a servlet in the same server
Resource References and the ENC

- The short answer is that applications use **named resource references** to gain access to a resource
  - Applications define the resource references, usually in a deployment descriptor

- The **environment name context**, or **ENC**, is a special Java EE namespace for defining these resource reference names
  - It is a naming context local to the Web application
  - Resources can be defined in the web.xml, and their names are located in the ENC
  - Web components use these ENC lookup names in obtaining references to the resource, e.g., a data source or an EJB
  - Data sources, by convention, are usually bound to names in the ENC beginning with `jdbc` e.g. `/jdbc/JavaTunesDataSource`
Defining a Resource Reference in web.xml

```xml
<web-app version='2.4' ...>
  ...
  <resource-ref>
    <!-- lookup name used by Web components -->
    <res-ref-name>jdbc/JavaTunesDataSource</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
  ...
</web-app>
```

- This makes a *DataSource* object available to components (e.g. servlets) in the Web app
  - Its ENC name is *jdbc/JavaTunesDataSource*
  - The application server performs database authentication
  - The data source is shareable (able to be pooled)

Notes:
- The ENC also has a special JNDI prefix - *java:comp/env*
  - That means that you can use JNDI to directly look up resources in the ENC by prefixing their name with *java:comp/env*
  - So the full JNDI path name for the resource defined in the slide is *java:comp/env/jdbc/JavaTunesDataSource*
  - We'll see later how you can use the JNDI API directly to look up these resources
@Resource – Injecting/Using a DataSource

- The `javax.annotation.Resource` annotation is used to inject a reference to a resource
  - Usually applied to a field or (property setter) method
  - The ENC name of the reference is specified with the `name` element
- The resource is injected when application is initialized
  - The example below shows injection and use of a `DataSource` with the name `jdbc/JavaTunesDataSource`

```java
package com.javatunes.web;
public class SearchServlet extends HttpServlet {

    @Resource(name="jdbc/JavaTunesDataSource")
    DataSource ds;

    protected void doGet( /* … */ ) { // Most detail omitted
        Connection conn = ds.getConnection();
        // …
    }
}
```

Notes:
- If you don't specify the name element in `@Resource`, a default name is assumed
  - This is the fully qualified class name, followed by the field or property name
  - In our example, this would be `com.javatunes.web.SearchServlet/ds`
  - If you use the default name, then you don't need a `<resource-ref>` entry in the web.xml
  - However, in that case, each component that uses the resource will have it's own name for the resource – which is not really practical when multiple components use the same resource
  - In general, it's better practice to specify a name in web.xml, and in the `@Resource` annotation
- `@Resource` can also be specified on a JavaBean-style property setter method
  ```java
  @Resource(name="jdbc/JavaTunesDataSource")
  public void setDs(DataSource dsIn) {
      ds = dsIn;
  }
  ```
- It can also be specified at the class level – in which case no injection is done
  - It just makes a name available in the ENC, which can then be looked up using JNDI
More About Resource Injection

- With resource injection (Java EE 5+) the container handles the resource initialization when it's required
  - Only available to managed classes (e.g. servlets, EJBs, etc)
  - Makes acquisition of resources simple

- @Resource provides a number of elements, including
  - name: The resource reference name
    - The default is the fully-qualified-className/field-name, e.g. `com.javatunes.web.SearchServlet/ds`
  - type: The type of the resource (default is type of field)
  - authenticationType: CONTAINER or APPLICATION
  - mappedName: Non-portable name to which resource should be mapped (typically a global JNDI name – covered later)
    - Typically used to bypass the ENC, and go directly to the global name
    - Not considered a best practice
Mapping the ENC Name to the Resource

- The *web.xml* resource name must be mapped to a real name
  - It is basically just a local, logical name for the resource
  - Typically mapped in a deployment descriptor or admin console
- The resource and its real (JNDI) name must also be defined
  - Usually by either config files or a GUI-based admin tool
  - We’ll cover JNDI next

```xml
<jboss-web> <!-- JBoss vendor-specific DD for webapps -->
...
<resource-ref>
  <!-- web.xml name used by Web components for lookup -->
  <res-ref-name>jdbc/JavaTunesDataSource</res-ref-name>
  <!-- real JNDI name as defined in the app server -->
  <jndi-name>java:/DefaultDS</jndi-name>
</resource-ref>
...
</jboss-web>
```

Notes:
- In the example, we assume that an actual datasource exists with the JNDI name java:/DefaultDS
- Because the ENC name is just a logical name, it needs to be mapped to an actual resource
  - The reasoning behind this is that a deployer/administrator may deploy the app into different environments that have different resource names
  - Using this kind of mapping, you can do that without recompiling your applications simply by changing the deployment descriptor
- Vendor-specific DDs are not required to follow any specification or naming convention.
  - However, vendors strive to make it as easy as possible to work with them.
  - They often use the same (or similar) element names as in the application DDs, e.g., `resource-ref` and `res-ref-name` are used to map a resource reference to a JNDI name, as shown above.
  - They also tend to use DD file names modeled after the application DD file names, e.g., `sun-web.xml`, `weblogic-web.xml`, etc., and these DDs are colocated with the application DDs, i.e., they reside in the same directory.
- The acronym DD is often used for deployment descriptor.
<table>
<thead>
<tr>
<th>JNDI – Java Naming and Directory Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>◆ The longer answer is, use a <strong>distributed naming service</strong></td>
</tr>
<tr>
<td>– The service runs in a well known location accessible to both the <strong>DataSource</strong> container and the clients</td>
</tr>
<tr>
<td>◆ Think of a naming service as a telephone book</td>
</tr>
<tr>
<td>– A telephone number can be listed under your name</td>
</tr>
<tr>
<td>– Users can look up your number in the phone book via your name</td>
</tr>
<tr>
<td>◆ <strong>JNDI</strong> is the standard Java interface to naming services</td>
</tr>
<tr>
<td>– The ENC is just a special namespace in JNDI</td>
</tr>
<tr>
<td>◆ When you deploy a <strong>DataSource</strong> into a server, the <strong>server binds the DataSource into JNDI under its name</strong></td>
</tr>
<tr>
<td>– Clients can <strong>look up the DataSource in JNDI under that name</strong></td>
</tr>
<tr>
<td>– With resource injection, JNDI is often not used directly, but it is still good to know the basics</td>
</tr>
</tbody>
</table>

**Notes:**

◆ The **DataSource** name is generally configured when you deploy it
Server Binds a Reference Into JNDI

- As part of deployment, the server binds a reference into JNDI using a name the deployer configures
  - This reference is some sort of proxy to a container class

Notes:
Client Looks Up Reference In JNDI

- The client can use this JNDI name to access the datasource
  - It gets a reference, which it can use to invoke on the DataSource

Notes:
- Generally, the JNDI server is running within the application server itself
  - This doesn't have to be the case though
**JNDI Overview**

- **JNDI** = Java Naming and Directory Interface

- Standard API for naming and directory functionality
  - `javax.naming` package

- Binds names to objects
  - Defines how names are bound to objects
  - Defines how objects are found by name
  - These name-object bindings occur in a **context**

- Data sources are obtained by **JNDI lookup**

---

**Notes:**

- A simplified view of a JNDI directory is as a simple collection of name-value pairs.
  - Names are *Strings*.
  - Values are *Objects*.
**JNDI Name Tree**

- A JNDI name tree has a structure similar to a file system in which
  - The root directory is like the **initial context**
  - Directories are like **contexts**
  - Files are like **objects** bound to one or more contexts

- So, in JNDI terms, the path name “jdbc/JTunesDS” translates as follows
  - The **initial context** is the implicit starting point
  - **jdbc** is a **context** under the initial context
  - **JTunesDS** is an **object** bound into the ejb context

---

**Notes:**

- Remember, we are just using a file system as an analogy to understand the basic structure of JNDI. In the way that we are going to be using JNDI, the objects in JNDI do not correspond to anything in the underlying file system.
JNDI Tree Structure

jdbc context bound into Initial Context

jdbc

JTunesDS

Context

Object

JTunesDS Object bound into dbc context

object bound into two contexts

Initial Context

Notes:
The ENC and JNDI

- The environment name context, or ENC, is a special Java EE namespace for application components
  - Each component (e.g. a Web app) has its own ENC
  - It has a special JNDI prefix - java:comp/env, so the full JNDI name of App 1's object below is `java:comp/env/jdbc/MyDS`
  - ENC names are logical names, generally mapped to global JNDI names

```
ENC – App 1
  
  java:comp/env
  jdbc
  MyDS

ENC – App 2
  
  java:comp/env
  jdbc
  JTunesDS
```

Notes:
- The ENC has a special JNDI prefix - java:comp/env
  - That means that you can use JNDI to directly look up resources in the ENC by prefixing their name with java:comp/env
  - The full JNDI path name for the App 1 resource defined in the slide is `java:comp/env/jdbc/MyDS`
  - The full JNDI path name for the App 2 resource defined in the slide is `java:comp/env/jdbc/JTunesDataSource`
- Each application's ENC is totally separate from every other component's ENC
  - Even though they all have the same java:comp/env prefix
  - The container is responsible for determining the correct part of the JNDI tree when the java:comp/env prefix is used – for example, by something like keeping track of the thread(s) associated with a component and mapping those to specific parts of the JNDI tree that represent that component's ENC
  - We'll see later how you can use the JNDI API directly to look up these resources
Context Interface and Resource Lookup

- `javax.naming.Context` is the core interface in JNDI
  - Has methods to lookup an object by JNDI name (and much more)
  - With a DataSource ENC name of "jdbc/JavaTunesDataSource", the following code fragment looks it up in JNDI
  - Resource injection is more commonly used, but you may see this

```java
// get root naming context
Context ctx = new InitialContext();

// look up object by its ENC name in web.xml
Object lookup = ctx.lookup("java:comp/env/jdbc/JavaTunesDataSource");

// cast the returned Object to DataSource (see notes)
DataSource datasrc = (DataSource)PortableRemoteObject.narrow(lookup, DataSource.class);

// no username and password - container authenticates
Connection conn = datasrc.getConnection();
```

Notes:
- With resource injection, JNDI is often not needed to acquire resource
  - However, there is still a lot of existing code that uses JNDI lookup, so its good to know
- `javax.naming.Context` is the core interface in JNDI; it contains methods to
  - Lookup an object by its JNDI name (most commonly used method)
  - Bind/Unbind/Rename an object to a JNDI name
  - Create and destroy subcontexts
- For `lookup()`, names passed as arguments are relative to the context.
  - An empty name "" is used to name the context itself.
  - The return type is `java.lang.Object`; the caller must cast the object to the correct type.
- The narrow method of `PortableRemoteObject` returns an object that can be safely cast to the class specified in the second parameter. The use of narrow, which is needed to retain compatibility with CORBA, is recommended in EJB1.1 and required in EJB 2.0+
- The EJB container is responsible for publishing (i.e., binding) an instance of the home interface to the JNDI name space. The JNDI name is specified in the vendor-specific deployment descriptor.
As we saw before, resource injection is much simpler
– Still requires mapping of ENC name to global JNDI name in the DD

Notes:
◆ Web components don't look things up in JNDI directly.
  – They do ENC lookups, which get mapped to real JNDI lookups by vendor-specific deployment descriptors. This isolates the Web application from any administrative server changes that might take place regarding the real JNDI name.
  – For example, administrators might have specific JNDI naming conventions that include other things in the lookup name. In our example, the javatunes prefix in the data source's JNDI name marks it as belonging to the JavaTunes application. This particular administrator chooses to prefix all JavaTunes JNDI names with javatunes. If this were to change in the future, we would modify sun-web.xml to indicate the new JNDI name. The Web application itself remains unchanged.
Connection Pooling

DataSource, ENC, and JNDI

Connection Pooling
Connection Pooling

- Connection pooling is a technique for improving the connection efficiency of an application
  - Establishing a connection is an expensive operation

- Instead of creating a new connection every time you need to access the database, the Java EE container keeps a pool of open connections available
  - When you need a connection, you get one from the pool
  - When you are done with the connection, you return it to the pool

- Using a pooled connection is completely transparent to the client
  - Client calls `getConnection` and `close`, as usual
  - But this is a logical connection from the pool, not a physical one

Notes:
- Connection pooling implementations provide mechanisms to control how the pool operates.
  - **Minimum** and **maximum** number of connections in the pool.
  - Some allow you to set an **increment**, such that the number of connections increases by that value when more connections are needed.

- For example, you might set the minimum number to 5, the maximum number to 25, and the increment to 5.
  - When the pool is created, 5 connections are established.
  - If all the connections are currently in use and a 6th one is requested, 5 more will be established, bringing the total to 10.
  - The number of connections in the pool might also shrink back down to 5, if the demand for connections goes down.
  - Connection pools strive to efficiently satisfy the demand for connections, while not establishing more connections than are necessary.
Non-Connection Pooling Example

◆ Consider an application that performs a catalog search for users, based on a keyword

◆ Without connection pooling, the application would do something like this:
  – Establish a connection
  – Create a statement
  – Execute the query and display the results
  – Close the connection

◆ Establishing a connection is expensive
  – We need a way to reduce this overhead

Notes:

◆ A Web application is not a "connected" application. Contrast this to a client/server application, in which clients connect to a database and stay connected for the lifetime of the session.
  – Client/server applications have a finite number of clients – dedicated connections can be appropriate. And the connection overhead is only incurred once.
  – Opening up an application to the Web means you can have a very large number of end clients (browsers) accessing the system. And since browser clients do not hold persistent connections to the app server, connection overhead is incurred repeatedly, hence the need for connection pooling.
Connection Pooling Example

- Set up a pool of open connections when the application first initializes, or when starting the application server that does the pooling

- Then the application would do this:
  - Get a connection from the pool (already open, fast operation)
  - Create a statement
  - Execute the query and display the results
  - Return the connection to the pool

- This considerably improves the performance of the application
Pooled Connections - Illustrated

- When an application server starts up, it can initialize the pool of open connections
Getting a Pooled Connection - Illustrated

- When the client wants a connection, the app server just takes one from the pool
  - Much faster than creating a new connection
Closing a Pooled Connection - Illustrated

- When the client closes the connection, it doesn't really close
  - It just gets returned to the pool

Notes:
- So when do the physical connections in the pool get closed?
  - When the connection pool manager shuts down the pool.
Connection Pooling Implementations

- All application servers support connection pooling
- Your application code doesn't need to do anything differently
  - Database client objects get a data source via JNDI lookup and use connections from it
  - Behind the scenes, the app server is using pooling

Notes:
- An application server is not required to get connection pooling.
  - Though most Web applications would use the facilities in the server.
- Some other connection pooling implementations include:
  - Db Connection Broker
    - [http://www.javaexchange.com](http://www.javaexchange.com)
  - Hans Bergsten article
  - Thomas Davis article
Statement Pooling

◆ Connection pools most greatly benefit 3-tier environments
  – Especially Web applications, where database clients, e.g., servlets, do not hold connections for very long
  – Get connection, do DB work, close connection ("get in, get out")

◆ If closing a connection closes its statements:
  – Do we lose the efficiency benefits of prepared statements?
  – Get connection, prepare statement, execute (once), close
  – This works against the prepared statement lifecycle

◆ Prepared statements can also be pooled
  – Completely transparent to the client
  – Client calls prepareStatement and close, as usual

Notes:
Statement Pooling - Illustrated

- Each pooled connection has a pool of prepared statements
  - The first time any client prepares a statement with any connection, it is "precompiled" and added to that connection's statement pool

Notes:
Getting a Pooled Statement - Illustrated

- When a client issues a `prepareStatement` call:
  - If that statement has already been prepared with that connection, it just gets taken from that connection's pool.
  - If that statement has not already been prepared with that connection, it is precompiled and added to that connection's pool.

Notes:
- Over time, the application "warms up" and all of the prepared statements have been prepared with each one of the pooled connections.
**Closing a Pooled Statement - Illustrated**

- When the client closes the statement, it doesn't really close
  - It just gets returned to that connection's statement pool

**Notes:**
- So when do the prepared statements in the statement pools get closed?
  - When the connection pool manager shuts down the connection pool.
Be Neat

- Using connection pools requires tidy programming
  - Connections are now a shared resource

- You must release connections back into the pool when you are finished with them
  - Make sure you call the `close` method on the `Connection`

- You should be doing this anyway
  - It is always good practice to release resources when you are done using them

- The same applies to pooled statements
  - Follow the regular lifecycle: prepare, execute, `close`

Notes:
Lab 15.1 - Database Integration
Lab 15.1 - Database Integration

- **Overview:** In this lab, we will integrate the Web application built earlier in the course with the database
  - We'll do this by defining a *DataSource*, and using it to get database connections - which will then be passed to *ItemDAO*

- **Objectives:**
  - Integrate servlets with a database using a DataSource
  - Define and use a *DataSource* object in Tomcat

- **Builds on previous labs:** None
  - This lab will be done in the Lab15.1 directory - a completely new directory with all the files needed for the lab

- **Approximate Time:** 25-35 minutes
Overview

- SearchServlet and CartServlet previously used the SearchUtility class to fake a database lookup
- A SearchUtilityDB class with the same API that actually searches in a database has been partially written for you
  - It has findByxxx methods that use the DAO from the JDBC labs
  - You will finish these methods to get a connection from a datasource passed in as an argument to the find methods
- The servlets supplied in this lab, use SearchUtilityDB to do their searches, so they hit the real DB via the DAO
- To complete the lab you'll need to:
  - Configure a resource reference in web.xml
  - Inject a DataSource into the servlets using the reference name
  - Finish SearchUtilityDB to get a connection from a datasource
  - Provide the database driver for the server

Notes:

- In SearchUtilityDB, you will look for the // TODO comments, and finish the statements to get a connection from the DataSource that was passed in

- An alternative solution is to place the code directly in the servlets.
  - Although the code required to set up and use our DAO is not that substantial, placing it in a helper class like SearchUtilityDB makes the servlet even cleaner.
Java EE Perspective and Web Project

Tasks to Perform

◆ Switch back to a Java EE perspective by selecting it in the Perspective list at the top right of the Workbench

◆ Create a new **Dynamic Web Project**
  – File | New | Dynamic Web Project *
  – Call it **Lab15.1** and click Next
  – In the next dialog, change the Context Root to **javatunes (all lower case)**, and click **Finish**

◆ Remove javatunes from the server, as follows:
  – Right click on the server in the Servers view, and select **Add and Remove Projects …**
  – Remove the javatunes project
  – Click **Finish**

Notes:

◆ We supply a Lab15.1 directory with most of the required code supplied
  – You will only need to finish the datasource specific portions

◆ When you create the Lab15.1 project, Eclipse will detect the existing directory and use it to build the project in
  – It will examine the project contents, and do its best to identify source and construct the project appropriately
Complete the DD

**Tasks to Perform**

- Open the Web app DD – `web.xml` - for editing
- Finish the `resource-ref` entry to make our `DataSource` object available to the JavaTunes Web components
  - The `resource-ref` element is the last entry in `web.xml`
  - The name should be `jdbc/JavaTunesDataSource`
  - The type should be `javax.sql.DataSource`

```
<web-app>...
  <resource-ref>
    <!-- ENC lookup name used by Web components -->
    <res-ref-name>TODD</res-ref-name>
    <res-type>TODD</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

**Notes:**

- The ENC name of `jdbc/JavaTunesDataSource` should exactly match the name element of the `@Resource` annotation
- If you were to look this up using JNDI (which is also possible), you would need to add the special ENC prefix – e.g.
  
  ```
  ctx.lookup("java:comp/env/jdbc/JavaTunesDataSource")
  ```
Review the DataSource Configuration

**Tasks to Perform**

- Open the Tomcat specific context configuration file - which is where you configure DataSources in Tomcat
  - It is in `WebContent\META-INF\context.xml`
  - **There is nothing for you to do here** - just review the datasource configuration - shown below
  - It configures a resource by the name of `jdbc/JavaTunesDataSource`
  - The resource is a `javax.sql.DataSource`, and the connection and pooling information is specified in the various attributes
  - See the notes for more information

```xml
<?xml version='1.0' encoding='utf-8'?>
<Context>
  <Resource name="jdbc/JavaTunesDataSource" type="javax.sql.DataSource"
            auth="Container" username="guest" password="password"
            maxActive="5" maxIdle="2" maxWait="10000"
            driverClassName="org.apache.derby.jdbc.ClientDriver"
            url="jdbc:derby://localhost:1527/JavaTunesDB"/>
</Context>
```

**Notes:**

- Configuring a resource in the `context.xml` file makes it available in JNDI
  - The attributes in the `<Resource>` element above configure the driver class, connection url, and pooling attributes in addition to the name and type of the resource

- For Tomcat, you **need to use the same name** for the resource as in the resource-ref name in your `web.xml` file
  - For other, more capable, application servers, these names will usually be different, and you would map from one to another in some other configuration element

- The resource relies on Tomcat's built in support for datasources and JNDI
  - By default for a datasource, Tomcat uses the Commons DBCP (Database Connection Pool) package which is part of the Apache Commons projects, and is included in the Tomcat libraries
  - More information on DBCP's features, capability, and configuration is available at [http://commons.apache.org/dbcp/index.html](http://commons.apache.org/dbcp/index.html)
Lab 15.1: Database Integration

SearchUtilityDB and Driver Jars

Tasks to Perform

- Open SearchServlet and CartServlet for editing
  - Look for the DataSource field declaration in each, and add an @Resource annotation to inject the DataSource
  - The data source's ENC name is jdbc/JavaTunesDataSource

- Open com.javatunes.util.SearchUtilityDB for editing
  - Look for the TODO comments, and replace the null value with an expression that gets a connection from the passed in DataSource

- Copy JEE\Derby\lib\derbyclient.jar to the Tomcat lib directory
  - e.g. C:\apache-tomcat-6.0.10\lib
  - The server needs the database driver to create the datasource
  - Restart the server (Right click in Servers view, select Restart)

- Right click on the Lab15.1 project, & select Run | Run on Server
  - Try doing a search - it should work as before, except now you're actually going to the database!

Notes:

- Remember that you only need to complete the portions marked by the TODO comments
  - We have taken care of things such as catching exceptions, closing JDBC resources – the DAO and the Connection object – and returning the methods' return values.
Session Review Questions

1. Why is `DataSource` better than `DriverManager` in obtaining a database connection?
2. What is the relationship between a data source and JNDI?
3. How is a data source made available to an application?
4. How do Web components get a reference to a data source?
5. What is the ENC and what is its role in the above question?
6. Describe connection pooling and statement pooling.
7. How is an application written to take advantage of pooling?
### Session Summary

- We reviewed `DataSource` and JNDI, and discussed their relationship
- We showed how to make a data source available to an application, and how Web components get a reference to one
- You integrated your Web application with a database, tying all of these ideas together
- We presented use of connection pooling and statement pooling

**Notes:**