Session-2

J2EE
Java Server Side Programming

Contents

- What is J2EE?
- Evolution of Enterprise Application Development Frameworks
- J2EE Components and Container Architecture
- J2EE APIs and Technologies
- How to get started

What is J2EE?

- Open and standard based platform for
  - developing,
  - deploying and
  - managing
    - n-tier,
    - Web-enabled,
    - server-centric,
    - component-based enterprise applications
  - which are typically transactional, reliable and secure.

Issues of Enterprise Computing

Challenges

Robustness
Scalability and performance
Object-oriented design
Avoid complexity
Maintainable and extensible
Just on time
Easy to test
Promote reuse
Support for multiple client types
portability

Key Technologies

J2SE™
J2EE™
JMS
Servlet
JSP
Connector
XML
Data Binding
XSLT

Products

App Servers
Web Servers
Components
Databases
Object to DB Tools

Legacy Systems

Databases
TP Monitors
EIS Systems

❖ **J2EE is specifically designed to address all these issues.**

About Enterprise Applications

Things that make up an enterprise application :

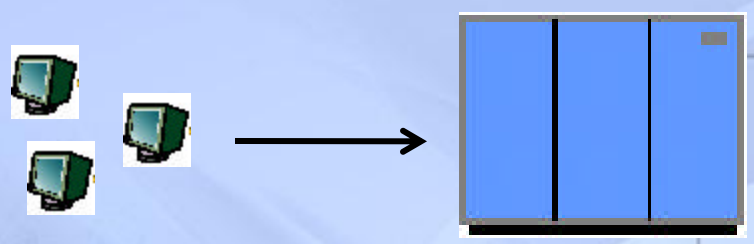
- Presentation logic
- Business logic
- Data access logic (and data model)
- System services

- The evolution of enterprise application framework reflects
  - How flexibly you want to make changes
  - Where the system services are coming from

Evolution of Enterprise Application Frameworks

- Single tier
- Two tier
- Three tier
  - RPC based
  - Remote object based
- Three tier (HTML browser and Web server)

Single Tier (Mainframe-based)



- Dumb terminals are directly connected to mainframe
- Centralized model (as opposed distributed model)
- **Presentation**, **business logic**, and **data access** are intertwined in one monolithic mainframe application

Single Tier (Cont.)

- Pros:
  - No client side management is required
  - Data consistency is easy to achieve
- Cons:
  - Functionality (presentation, data model, business logic) intertwined, difficult for updates and maintenance and code reuse

Two Tier

- Fat clients talking to back end database
  - SQL queries sent, raw data returned
- Presentation, Business logic and Data Model processing logic in client application

Two Tier (Cont.)

- Pros:
  - DB product independence (compared to single-tier model)
- Cons:
  - Presentation, data model, business logic are intertwined (at client side), difficult for updates and maintenance
  - Data Model is “tightly coupled” to every client: If DB Schema changes, all clients break
  - Updates have to be deployed to all clients making System maintenance nightmare
  - DB connection for every client, thus difficult to scale
  - Raw data transferred to client for processing causes high network traffic

Three Tier (RPC based)

- *Thinner client*: business & data model separated from presentation
  - Business logic and data access logic reside in middle tier server while client handles presentation
- Middle tier server is now required to handle system services
  - Concurrency control, threading, transaction, security, persistence, multiplexing, performance, etc.

Three Tier (Cont.)

- Pro:
  - Business logic can change more flexibly than 2-tier model
    - Most business logic reside in the middle-tier server
- Cons:
  - Complexity is introduced in the middle-tier server
  - Client and middle-tier server is more tightly coupled (than the three-tier object based model)
  - Code is not really reusable (compared to object model based)

Three Tier (Remote Object based)

- Business logic and data model captured in objects
  - Business logic and data model are now described in "abstraction" (interface language)
- Object models used: CORBA, RMI, DCOM
  - Interface language in CORBA is IDL
  - Interface language in RMI is Java interface

Three Tier (Remote Object based)

- Pro:
  - More loosely coupled than RPC model
  - Code could be more reusable
- Cons:
  - Complexity in the middle-tier still need to be addressed

Three-Tier (Web Server)

- Browser handles presentation logic
- Browser talks Web server via HTTP protocol
- Business logic and data model are handled by “dynamic contents generation” technologies (CGI, Servlet/JSP, ASP)

Three-Tier (Web Server)

- Pro:
  - Ubiquitous client types
  - Zero client management
  - Support various client devices
    - J2ME-enabled cell-phones
- Cons:
  - Complexity in the middle-tier still need to be addressed

Trends

- Moving from single-tier or two-tier to multitier architecture
- Moving from monolithic model to object-based application model
- Moving from application-based client to HTML-based client

Outstanding Issues & Solution

- Complexity at the middle tier server still remains
- Duplicate system services still need to be provided for the majority of enterprise applications
  - Concurrency control, Transactions
  - Load-balancing, Security
  - Resource management, Connection pooling
- How to solve this problem?
  - **Commonly shared container** that handles the above system services
  - Proprietary versus Open-standard based

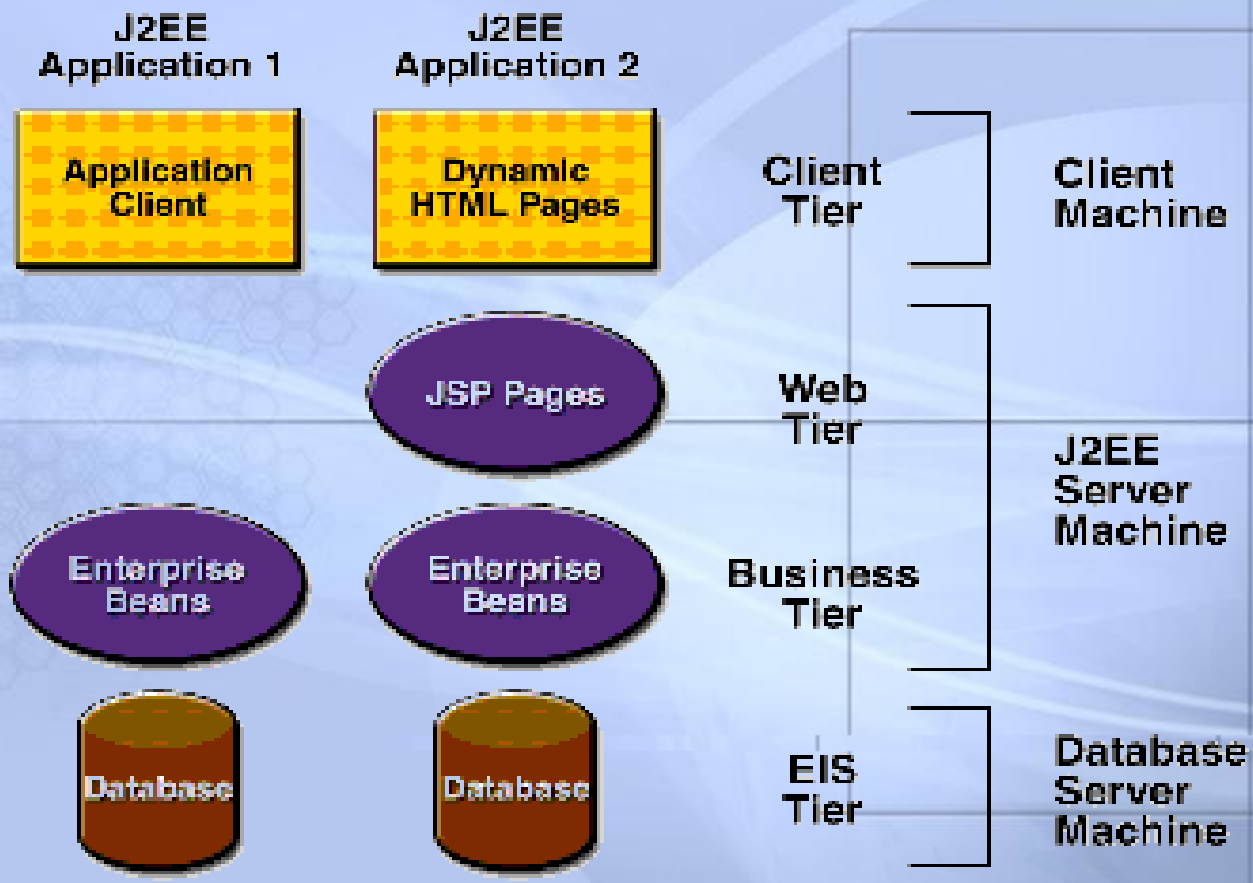
Open and Standard Solution

- Use "**component and container**" model in which container provides system services in a well-defined and as industry standard

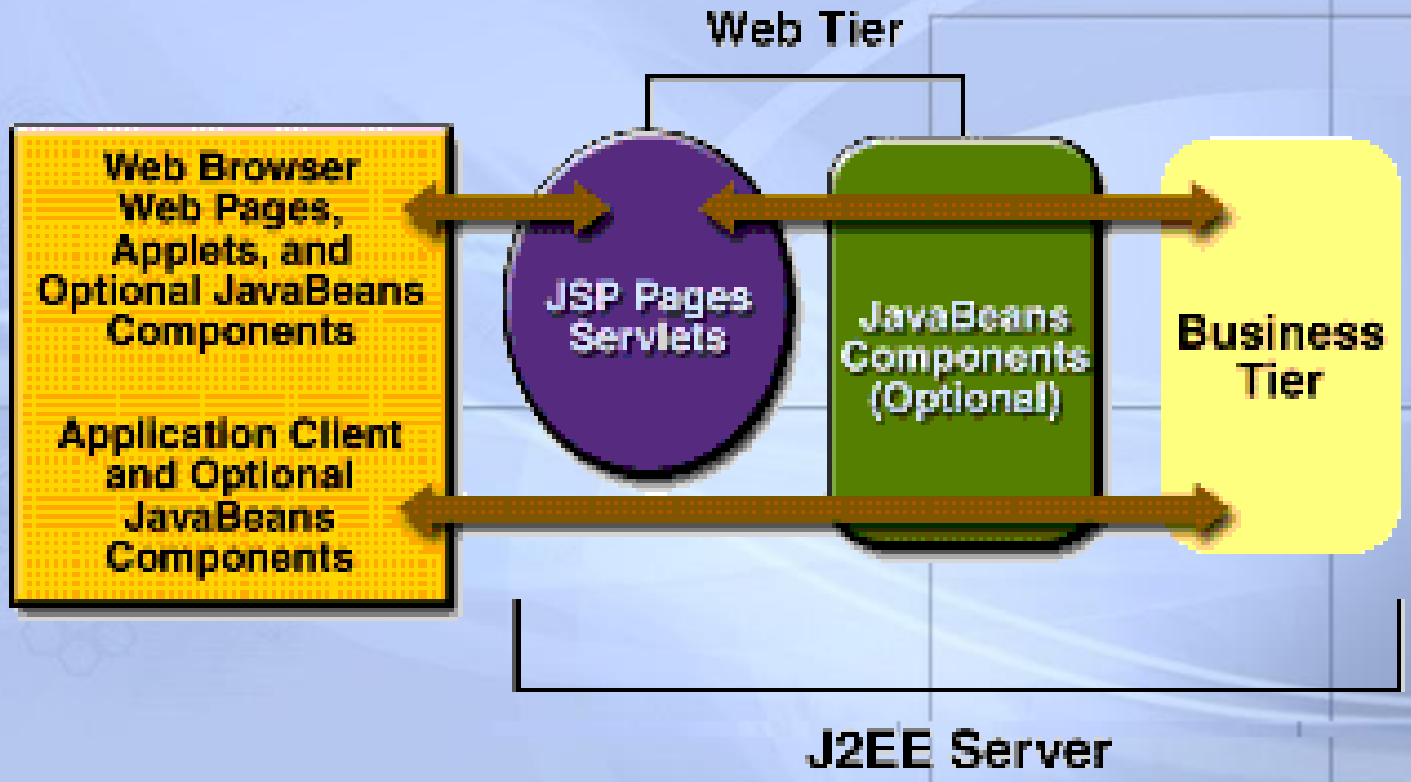
- J2EE is that standard , provides portability of code because it is based on Java technology and standard-based Java programming APIs

J2EE Architecture

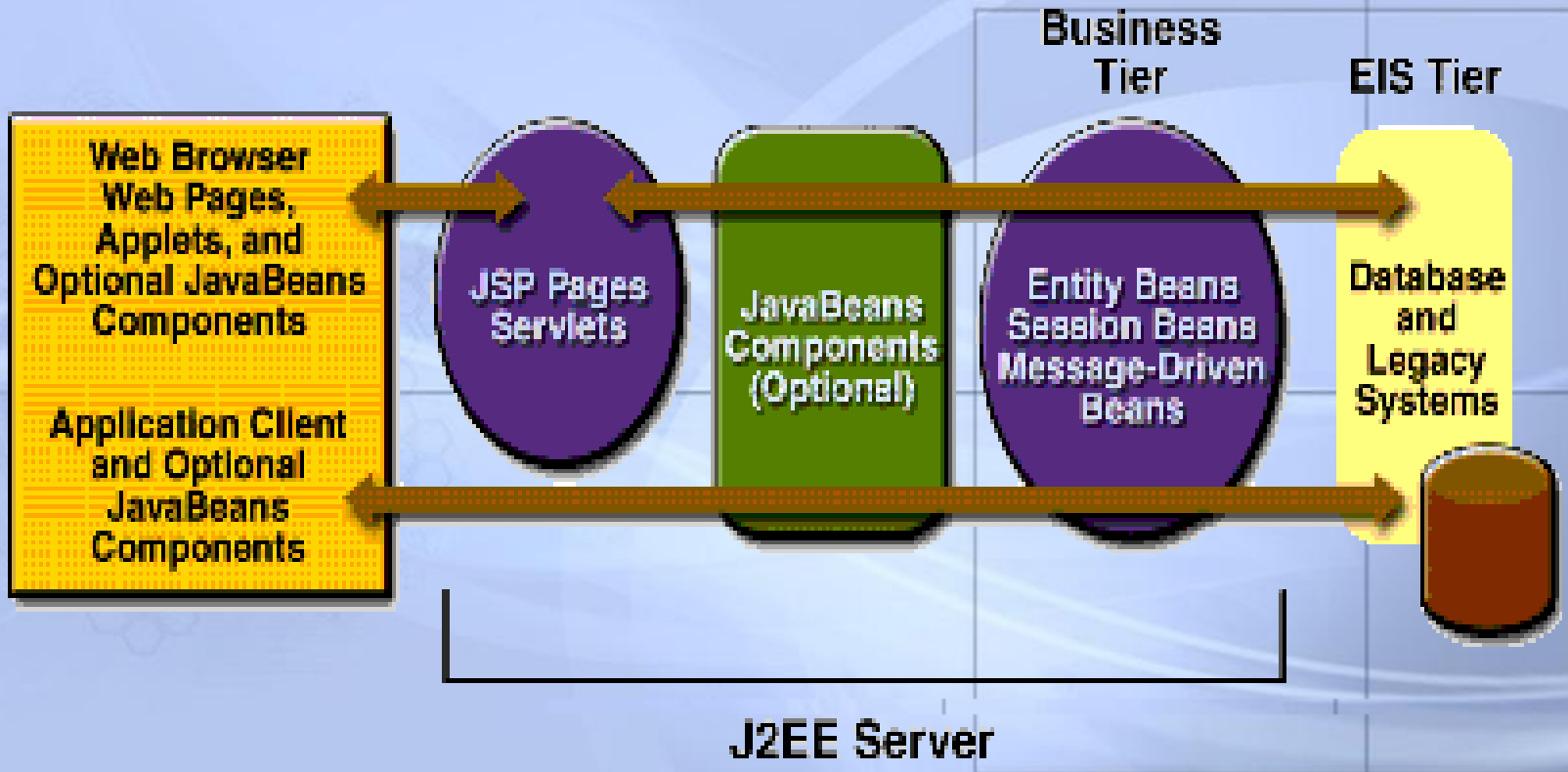
Multilayered Applications



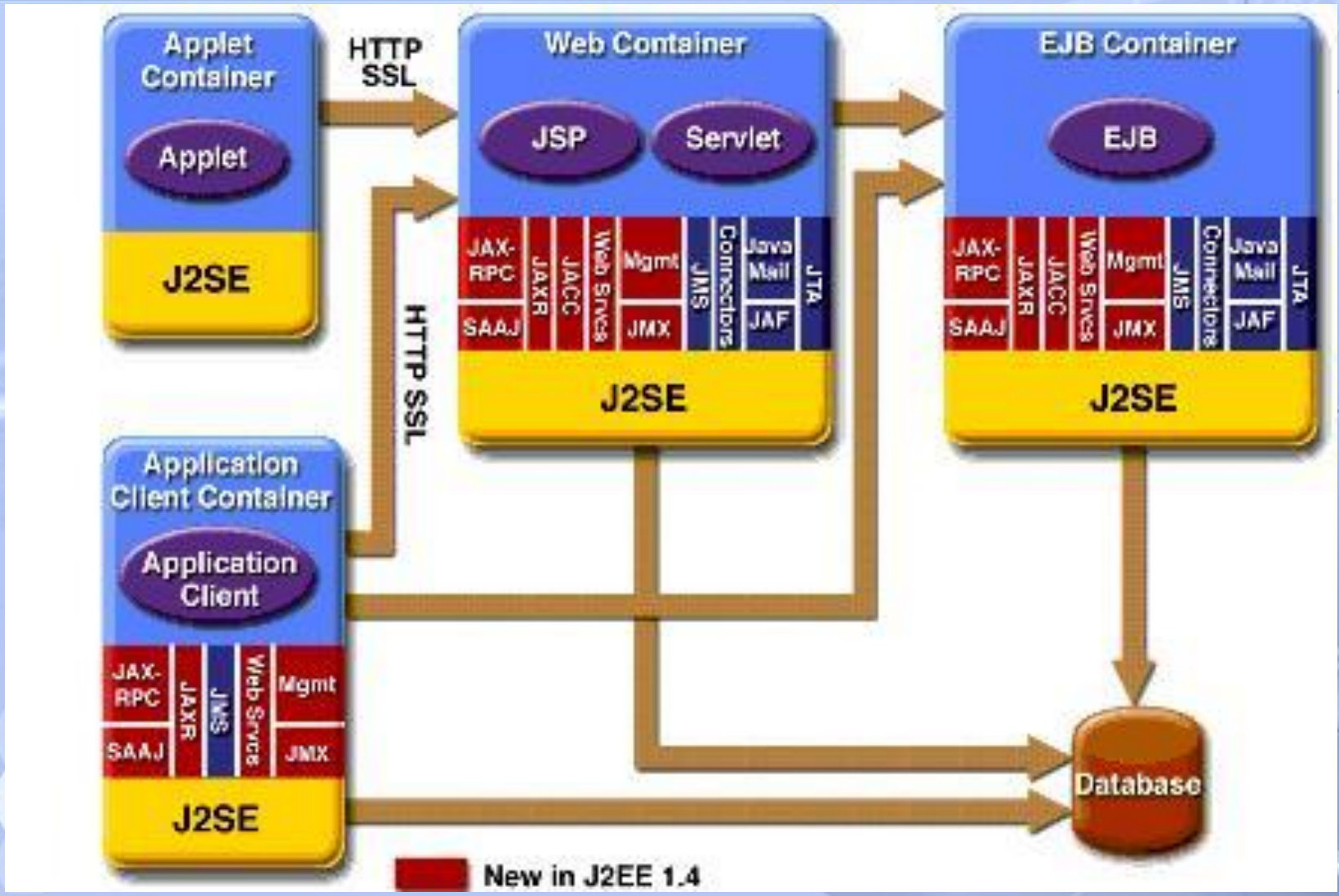
Web Tier only



Adding business and EIS tiers



J2EE Architecture



Introducing J2EE Components

J2EE Components

- J2EE specifies both the service APIs for building the application and the infrastructure for managing the application.

  – **J2EE Runtime**
    - A runtime infrastructure for managing and hosting applications. All runtime applications are located in this server.

  – **J2EE APIs and Technologies**
    - A set of Java APIs describes the programming model for J2EE application.

J2EE APIs and Technologies

Basic Requirement

- J2SE SDK & JVM
  - To run J2EE SDK and provides core APIs for writing J2EE components, core development tools.

- J2EE (SDK) APIs

J2EE APIs and Technologies

J2EE (SDK) APIs

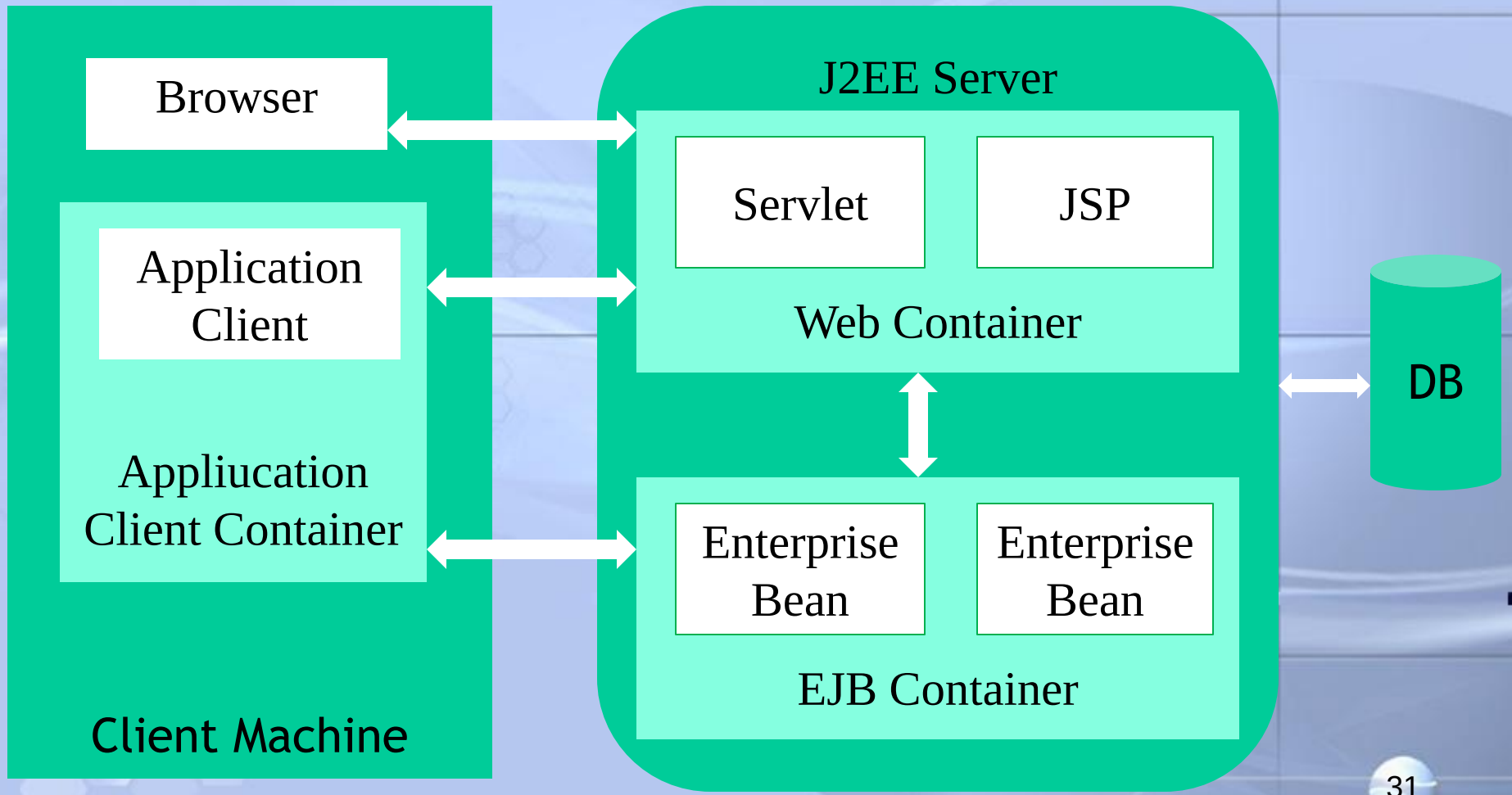
- Enterprise Java Beans (**EJB**) 2.1
- Java Servlet Technology 2.3
- Java Server Pages (**JSP**) 2.1
- JavaServer Faces (**JSF**)
- Java Message Service (**JMS**) 1.1
- Java Transaction API (**JTA**) 1.0
- Java Mail API 1.3
- Java Beans Activation Framework 1.0
- Java API for XML **(JAXP**) 1.3
- Java API for XML-based RPC (**JAX-RPC**) 1.5
- SOAP with Attachment API for Java (**SAAJ**) 1.2
- Java API for XML Registers (**JAXR**) 1.0
- J2EE Connector API
- **JDBC** 2.0
- Java Naming and Diorectory Interface (**JNDI**)
- Java Authentication and Authorization Service (**JAAS**) 1.3

Introducing J2EE Containers

J2EE Containers

- EJB Container
- Web Container
- Application Client Container
- Applet Container

Containers Types



Introducing J2EE Technologies

Types of J2EE Technologies

- Component Technologies
- Services Technologies
- Communication Technologies

Introducing J2EE Component Technologies

1. Web Component

- Concerning respond to HTTP requests
- Web components are in the form of either **Servlet or JSP**
- Web components run in a Web container
  - Tomcat and Resin are popular web containers
  - All J2EE compliant app servers (Sun Java System App Server) provide web containers
- Web container provides system services to Web components
  - Request dispatching, security, and life cycle management

Servlet

- Java™ objects which extend the functionality of a HTTP server
- Dynamic contents generation
- Better alternative to CGI, NSAPI, ISAPI, etc.
  - Efficient
  - Platform and server independent
  - Session management
  - Java-based

JSP (Java Server Page)

- Enables separation of business logic from presentation
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as Java Beans or custom tags
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology

2. Enterprise Java Bean (EJB) Component

- J2EE's business logic resides in EJB

- Easy development and deployment of Java technology-based application that are:
  - transactional,
  - distributed,
  - multi-tier,
  - portable,
  - scalable,
  - secure,
  - etc…

3. XML

- Defines how we manage, transport, process, and view data (as well as metadata)
- cross-platform, extensible, text-based standard for representing data.
- J2EE provides a number of APIs for developers working with XML:
  - Java API for XML Procerssing (JAXP)
  - Java API for XML Binding (JAXB)
  - Java API for XML Registers (JAXR)
  - Java API for XML Messaging (JAXM)
  - Java API for XML-based Remote Procedure Calls (JAX-RPC)

Introducing J2EE Service Technologies

Service Technologies

- Java Transaction API and Service
- JDBC
- JNDI
- Java Mail
- JMS
- JAAS

JMS (Java Message Service)

- Messaging systems (MOM- Message Oriented Middleware) provide
  - De-coupled communication
  - Asynchronous communication
  - Plays a role of centralized post office
- Benefits of Messaging systems
  - Flexible, Reliable, Scalable communication systems
- Point-to-Point, Publish and Subscribe
- JMS defines standard Java APIs to messaging systems

JAAS (Java Authentication & Authorization Service)

Authentication

- Pluggable authentication framework
  - Userid/password
  - Smartcard
  - Kerberos
  - Biometric

- Application portability regardless of authentication schemes underneath
  - JAAS provides authentication scheme independent API
  - Authentication schemes are specified Login configuration file, which will be read by JAAS

JAAS (Java Authentication & Authorization Service)

Authorization

- Without JAAS, Java platform security are based on
  - Where the code originated
  - Who signed the code
- The JAAS API augments this with
  - Who's running the code
- User-based authorization is now possible

Introducing J2EE Communication Technologies

Communication Technologies

- Internet Protocol
  - HTTP
  - TCP/IP
  - SSL
  - OMG Protocols
- Remote Object Protocol
  - RMI and RMI-IIOP
  - Messaging
  - Data Formats
  - Java IDL

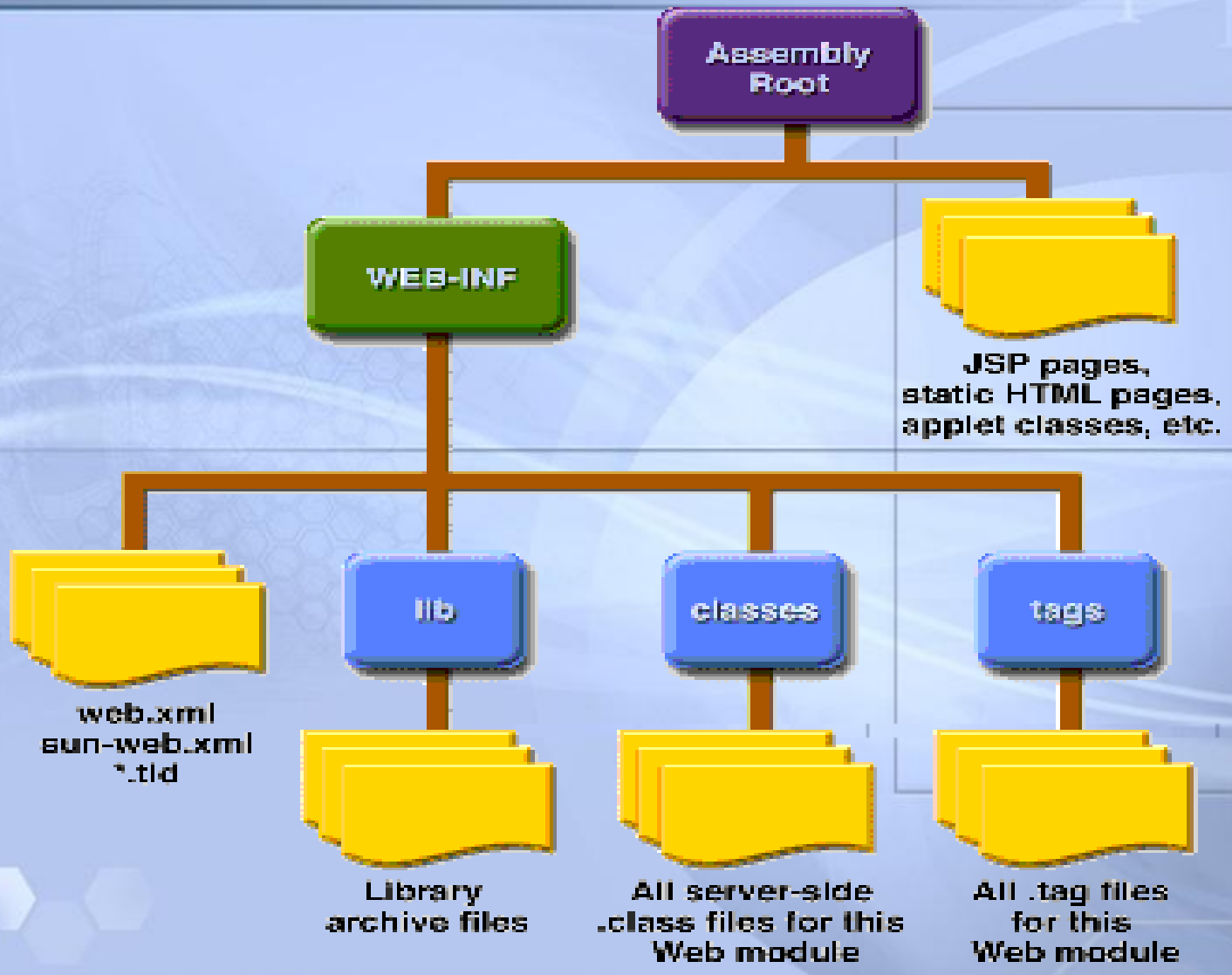
Building Web Applications

Web Application & Components

- Web Application is a deployable package
  - Web components (Servlets and JSP's)
  - Static resource files such as images
  - Helper classes
  - Libraries
  - Deployment descriptor (web.xml file)

- Web Application can be represented as
  - A hierarchy of directories and files (unpacked form) or
  - *.WAR file reflecting the same hierarchy (packed form)

Web Application Layout



Web Application Development and Deployment Steps

Web Application Development and Deployment Steps

1. Write (and compile) the Web component code (Servlet or JSP) and helper classes referenced by the web component code

2. Create any static resources (for example, images or HTML pages)

3. Create deployment descriptor (web.xml)

4. Build the Web application (*.war file or deployment-ready directory)

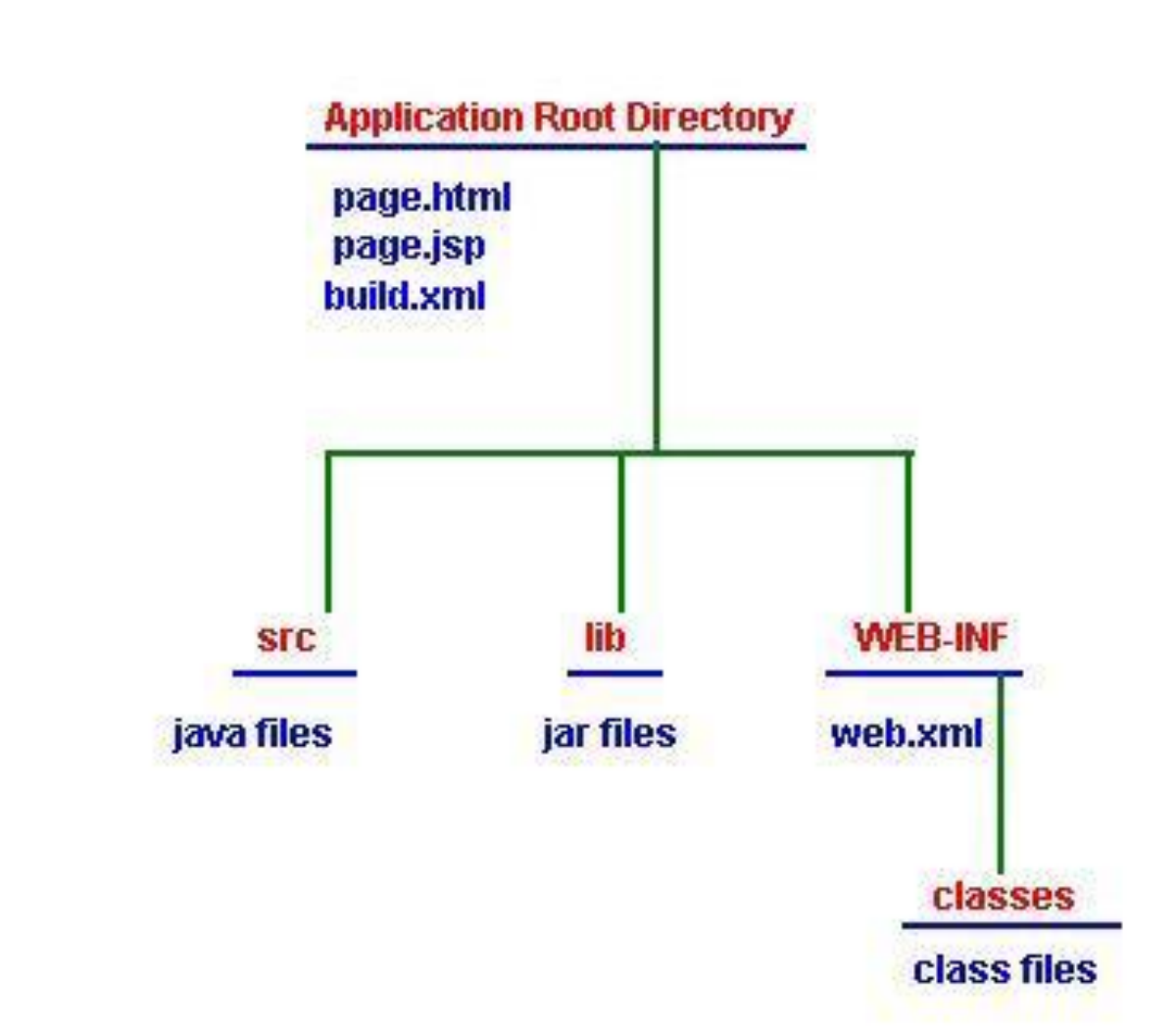
5. Deploy the web application into a Web container
   - ❖ Web clients are now ready to access them via URN

1. Write and compile the Web component code

- Create development tree structure
- Write either servlet code or JSP pages along with related helper code
- Create web.xml for building (and other application development life-cycle management) process

❖ IDE (i.e. Eclipse, NetBeans) handles all these chores

Development Tree Structure



"Hello" Example Tree Structure

Hello

- src/servlets
  - GreetingServlet.java
  - ResponseServlet.java
- Web
  - WEB-INF
    - web.xml
  - pic1.gif

– web.xml

2. Create any static resources

- HTML pages
  - Custom pages
  - Login pages
  - Error pages

- Image files that are used by HTML pages or JSP pages
  - Example: pic.gif

3. Create deployment descriptor (web.xml)

- Deployment descriptor contains deployment time runtime instructions to the Web container
  - URL that the client uses to access the web component
- Every web application has to have it

4. Build the Web application

- either *.WAR file or unpacked form of *.WAR file

- Build process is made of
  - create build directory (if it is not present) and its subdirectories
  - compile Java code into build/WEB-INF/classes directory
    - Java classes reside under ./WEB-INF/classes directory
  - copy web.xml file into build/WEB-INF directory
  - copy image files into build directory

5. Deploy Web application

- Deploy the application over deployment platform such as Sun Java System App Server or Tomcat

- 3 ways to deploy to Sun Java System App server
  - As admin deploy --port 4848 --host localhost – passwordfile "c:\j2ee\examples\common\adminpassword.txt“ -- user admin Hello.war
  - App server admin console
  - NetBeans,Eclipse

- From a browser, go to URN of the Web application

Configuring Web Application via web.xml

Web Applications Deployment Descriptor (web.xml)

- helps in managing the deployment configuration of web applications.
- stored in the /WEB-INF directory of the application.
- purposes:
  - Initialization of parameters for Servlets and web applications
  - Servlet/JSP definitions
  - Servlet/JSP mappings
  - MIME types (**M**ulti-purpose **I**nternet **M**ail **E**xtensions)
  - Security

Contents Deployment Descriptor

- Prolog
- Alias Paths
- Context and Initialization Parameters
- Event Listeners
- Filter Mappings
- Error Mappings
- Reference to Environment Entries, Resource environment entries, or Resources

web.xml (Cont.)

- Case sensitive
- Order sensitive (in the following order)
  - icon, display-name, description, distributable
  - context-param, filter, filter-mapping
  - listener, servet, servlet-mapping, session-config
  - mime-mapping, welcome-file-list
  - error-page, taglib, resource-env-ref, resource-ref
  - security-constraint, login-config, security-role
  - env-entry, ejb-ref, ejb-local-ref

Prolog (of web.xml)

- Every XML document needs a prolog

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/ webapp _2 _3 .dtd">
```

Alias Paths (of web.xml)

- When a request is received by Servlet container, it must determine which Web component in a which web application should handle the request.
- It does so by mapping the URL path contained in the request to a Web component.

- A URL path contains the context root and alias path
  - http://<host>:8080/context_root/alias_path

- Alias Path can be in the form of either
  - /alias-string (for servlet) or
  - /*.jsp (for JSP)

Alias Paths (of web.xml)

```
<web-app>
<welcome-file-list>
   <welcome-file>test.html</welcome-file>
   </welcome-file-list>

    <servlet>
      <servlet-name>MyServlet</servlet-name>
      <servlet-class>MyServlet</servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name>MyServlet</servlet-name>
      <url-pattern>/servlet1</url-pattern>
   </servlet-mapping>
</web-app>
```

Context and Initialization Parameters

- Represents application context
- Can be shared among Web components in a WAR file

```
<web-app>
...
<context-param>
    <param-name>
                javax.servlet.jsp.jstl.fmt.localizationContext
    </param-name>
    <param-value>messages.BookstoreMessages</param-value>
</context-param>
...
</web-app>
```

Event Listeners (of web.xml)

- Receives servlet life-cycle events

```
<listener>
        <listener-class>listeners.ContextListener</listener-class>
</listener>
```

Filter Mappings

- Specify which filters are applied to a request, and in what order

```
<filter>
        <filter-name>OrderFilter<filter-name>
        <filter-class>filters.OrderFilter<filter-class>
</filter>
<filter-mapping>
        <filter-name>OrderFilter</filter-name>
        <url-pattern>/receipt</url-pattern>
</filter-mapping>
```

Error Mappings

- Maps status code returned in an HTTP response to a Java programming language exception returned by any Web component and a Web resource

```
<error-page>
        <exception-type>exception.OrderException</exception-type>
        <location>/errorpage.html</location>
</error-page>
```

References (of web.xml)

- Need when web components make references to environment entries, resource environment entries, or resources such as databases
- Example: declare a reference to the data source

```
<resource-ref>
        <res-ref-name>jdbc/BookDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
</resource-ref>
```

Example web.xml of hello2

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4“ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>hello2</display-name>
    <servlet>
        <display-name>GreetingServlet</display-name>
        <servlet-name>GreetingServlet</servlet-name>
        <servlet-class>servlets.GreetingServlet</servlet-class>
    </servlet>
    <servlet>
        <display-name>ResponseServlet</display-name>
        <servlet-name>ResponseServlet</servlet-name>
        <servlet-class>servlets.ResponseServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>GreetingServlet</servlet-name>
        <url-pattern>/greeting</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ResponseServlet</servlet-name>
        <url-pattern>/response</url-pattern>
    </servlet-mapping>
</web-app>
```

Summary

- J2EE is the platform of choice for development and deployment of n-tier, web-based, transactional, component based enterprise applications
- J2EE is standard-based architecture
- J2EE is all about community
- J2EE evolves according to the needs of the industry