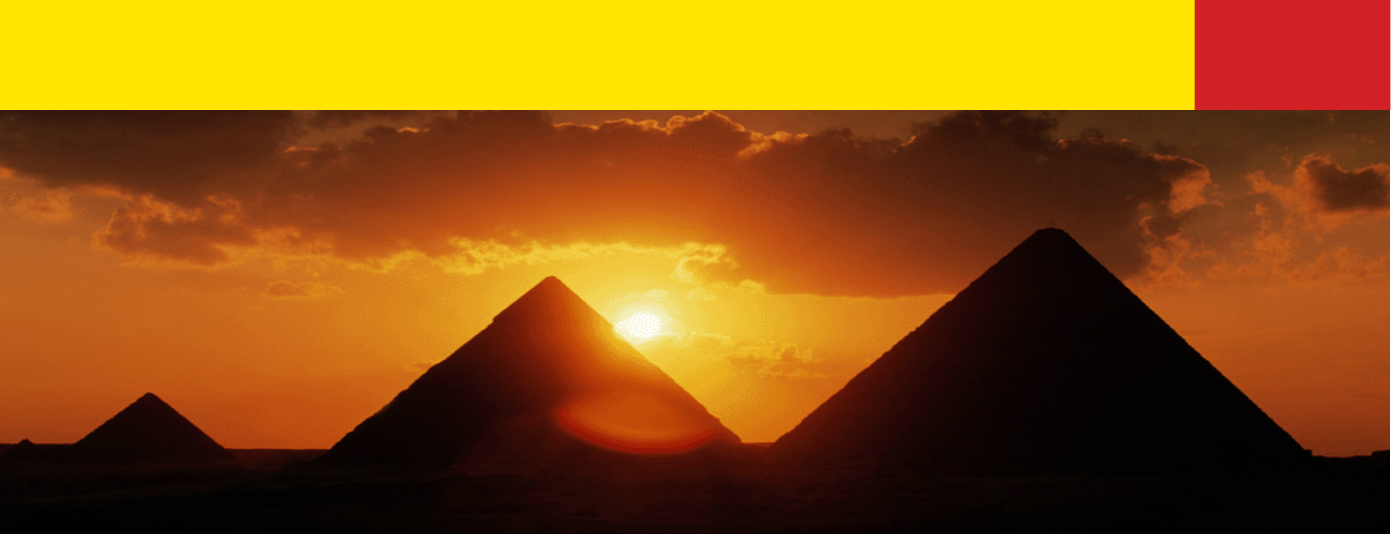
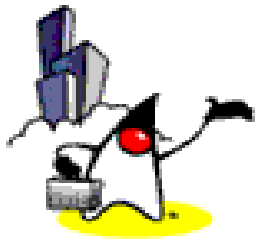


Web Application Architecture (based J2EE 1.4 Tutorial)

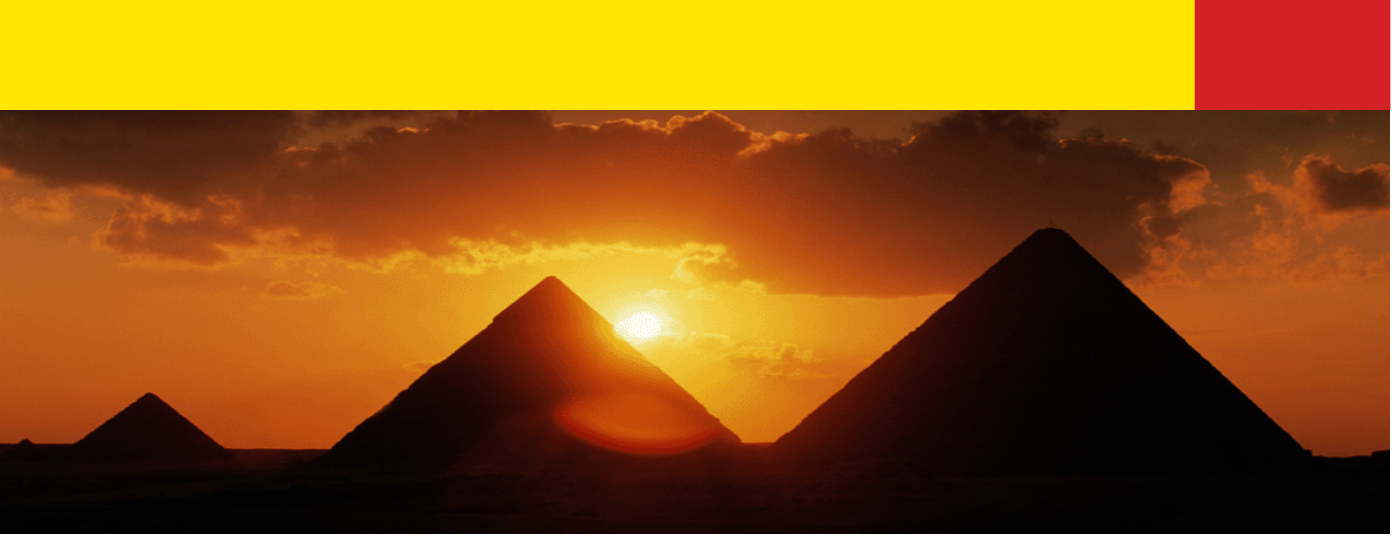


Disclaimer & Acknowledgments

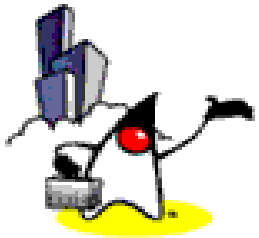
- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgments
  - The slides of this presentation are made from “Web Application” section of J2EE 1.4 tutorial written by Sun Microsystems

Agenda

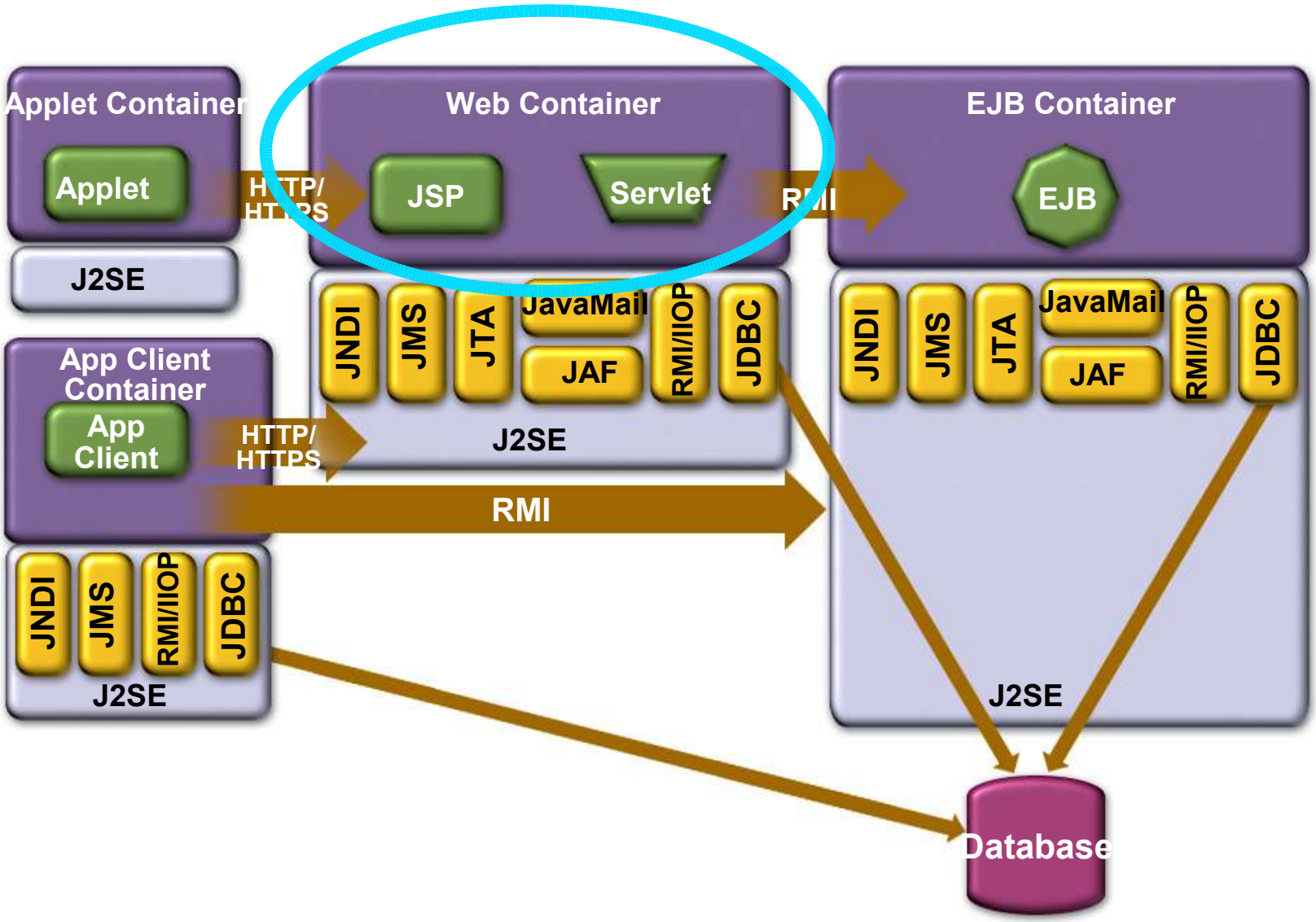
- Web application, components and Web container
- Technologies used in Web application
- Web application development and deployment steps
- Web Application Archive (*.WAR file)
  - *.WAR directory structure
  - WEB-INF subdirectory
- Configuring Web application
  - Web application deployment descriptor (web.xml file)



Web Application & Web Components & Web Container



Web Components & Container

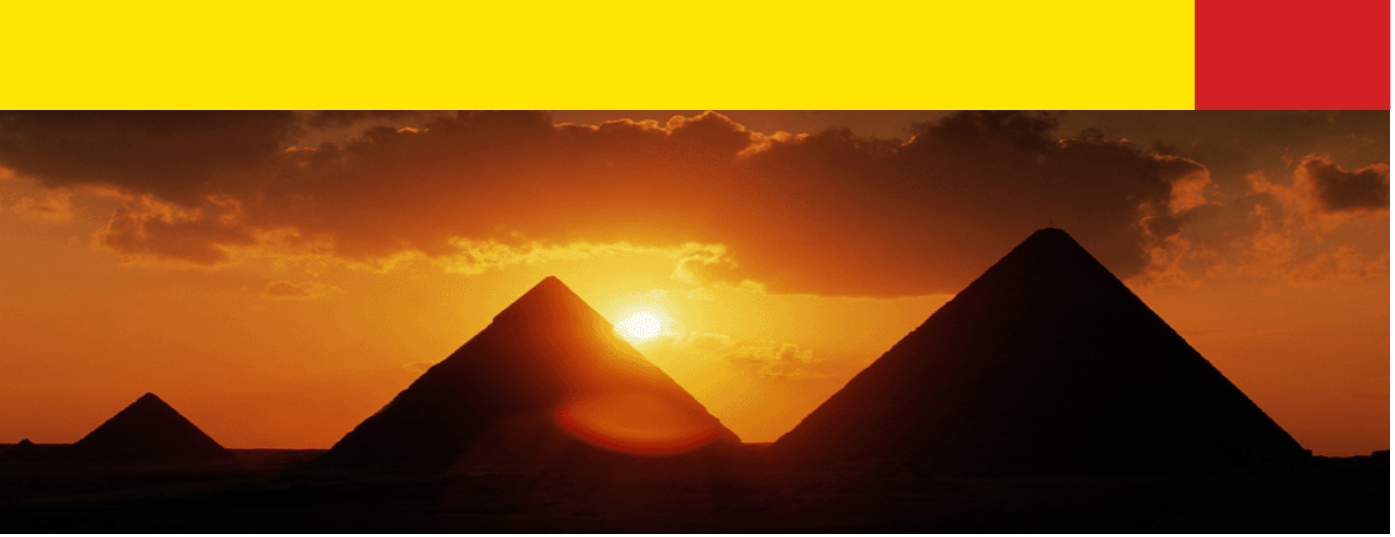


Web Components & Container

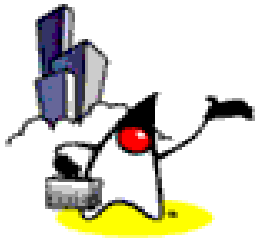
- Web components are in the form of either Servlet or JSP (along with JavaBean's and custom tags)
- Web components run in a Web container
  - Tomcat and Resin are popular web containers
  - All J2EE compliant app servers (Sun Java System App Server) provide web containers
- Web container provides system services to Web components
  - Request dispatching, security, and life cycle management

Web Application & Components

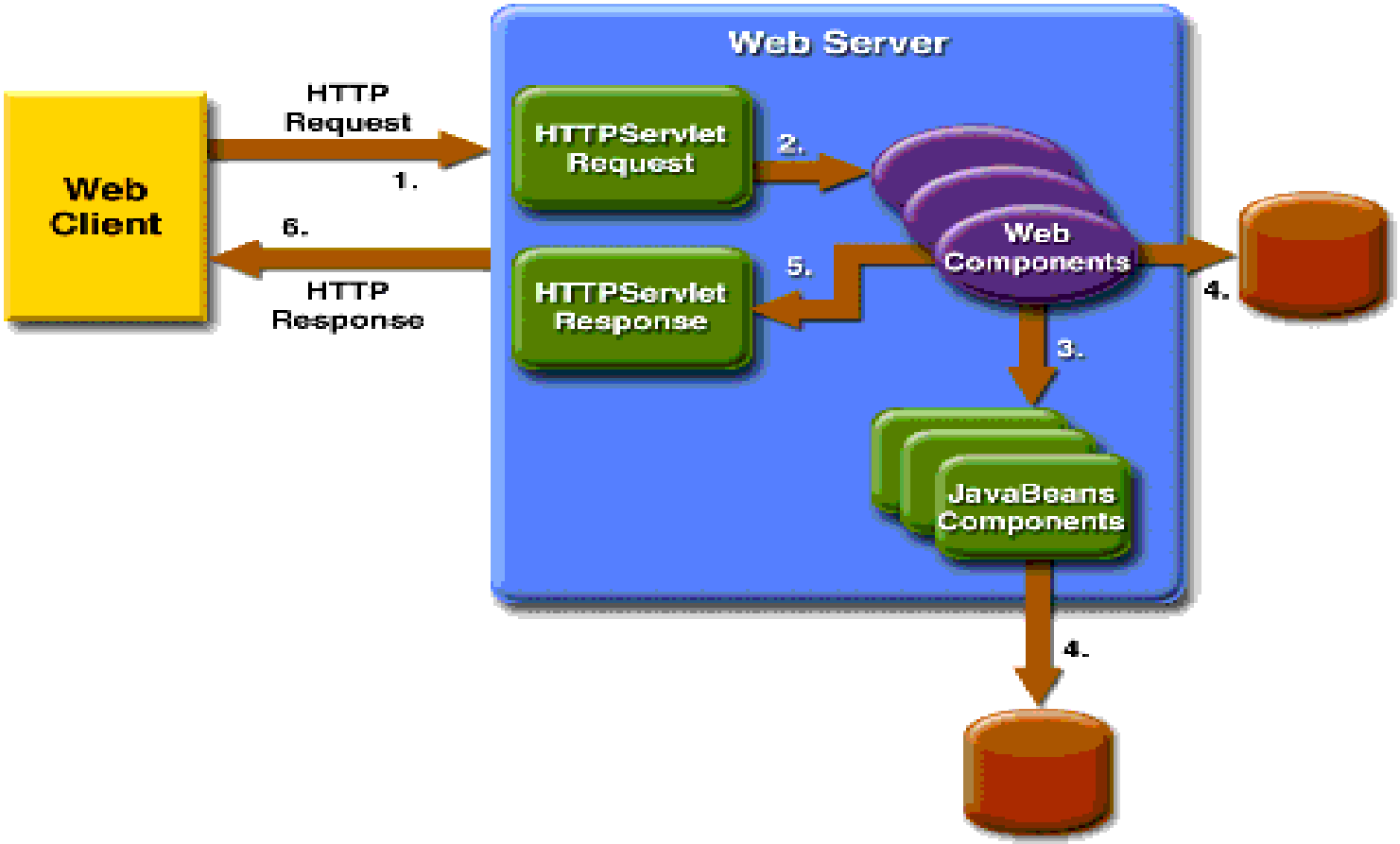
- Web Application is a deployable package
  - Web components (Servlets and JSP's)
  - Static resource files such as images
  - Helper classes
  - Libraries
  - Deployment descriptor (web.xml file)
- Web Application can be represented as
  - A hierarchy of directories and files (unpacked form) or
  - *.WAR file reflecting the same hierarchy (packed form)



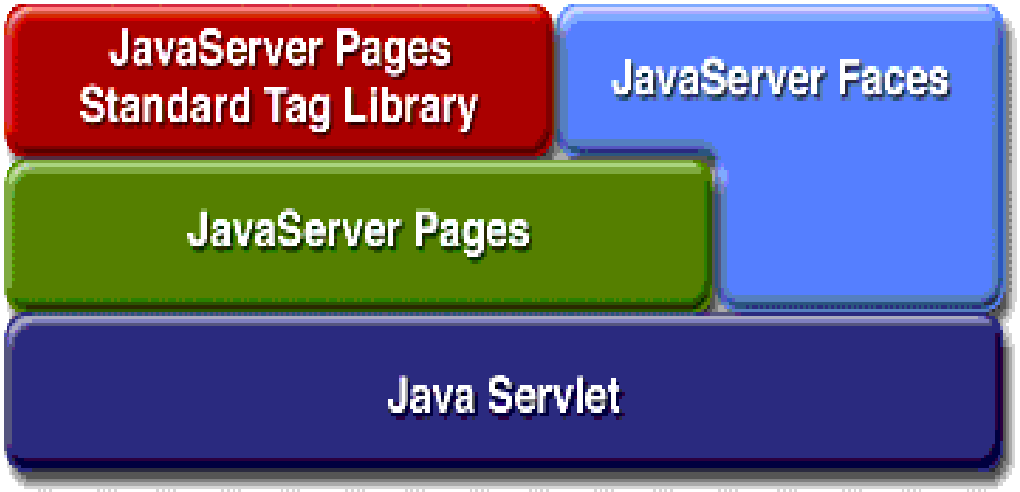
Technologies Used In Web Application

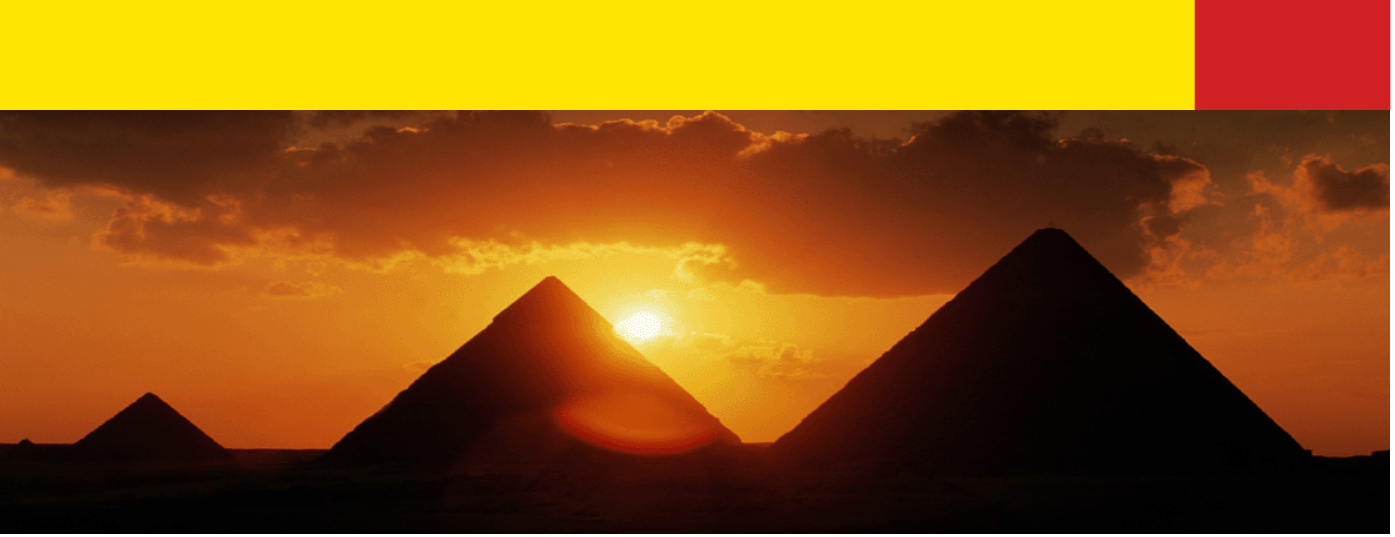


Web Request Handling

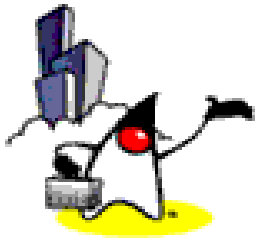


Java Web Application Technologies





Web Application Development and Deployment Steps (using hello2 example under J2EE 1.4 tutorial)



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 build.xml for Ant-based build (and other application development life-cycle management) process
- IDE (i.e. NetBeans) handles all these chores

Development Tree Structure

- Keep Web application source separate from compiled files
  - facilitate iterative development
- Root directory (example from hello2 sample code from J2EE 1.4 tutorial)
  - build.xml: Ant build file
  - src: Java source of servlets and JavaBeans components
  - web: JSP pages and HTML pages, images

Example: hello2 Tree Structure (before “ant build” command)

- Hello2
  - src/servlets
    - GreetingServlet.java
    - ResponseServlet.java
  - web
    - WEB-INF
      - web.xml
    - duke.waving.gif
  - build.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: duke.waving.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

Example: hello2 Tree Structure (after "asant build" command)

- Hello1
  - src
  - web
  - build.xml
  - build
    - WEB-INF
      - classes
        - GreetingServlet.class
        - ResponseServlet.class
      - web.xml
    - duke.waving.gif

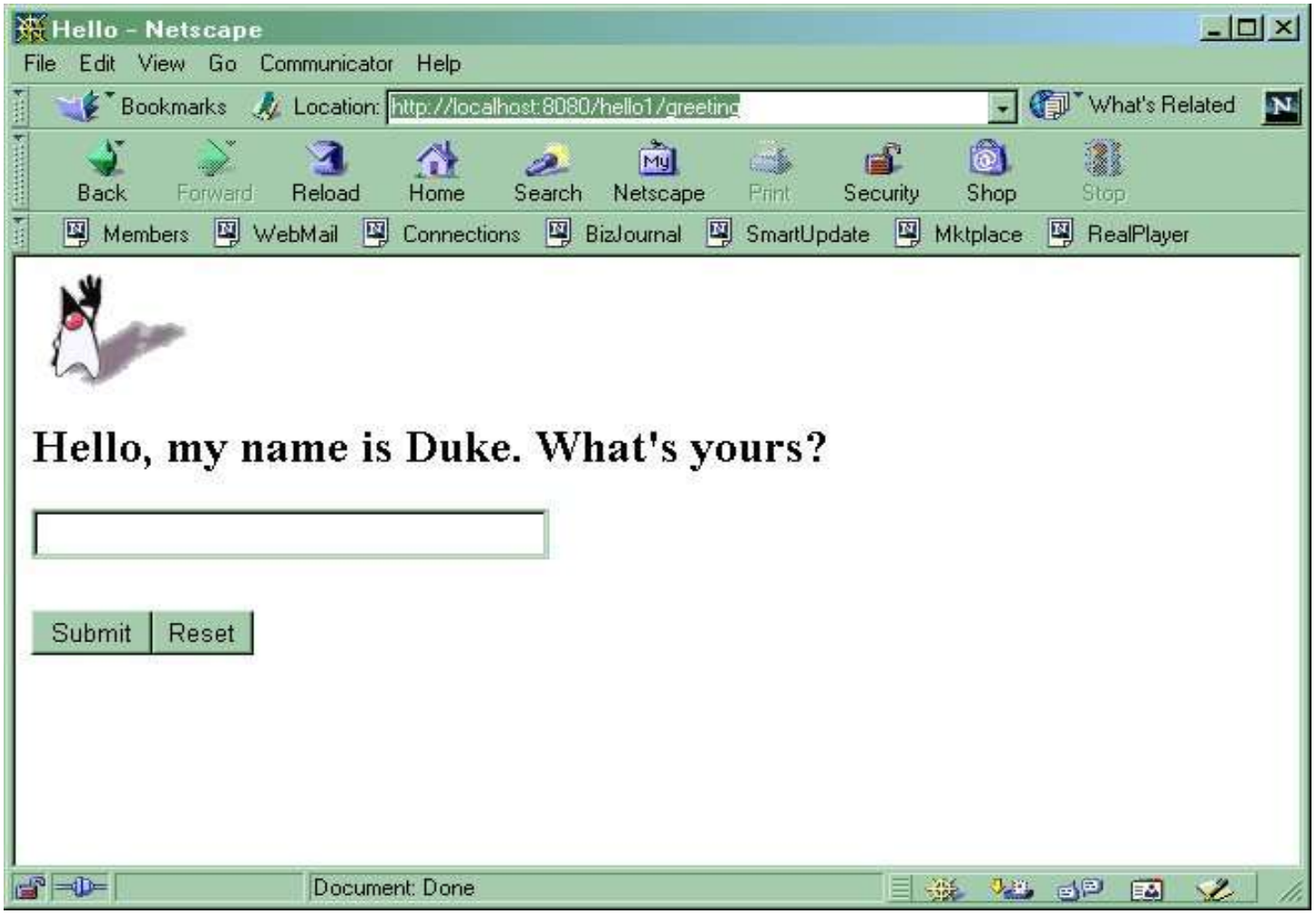
5. Deploy Web application

- 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
  - asadmin deploy --port 4848 --host localhost –passwordfile "c:\j2eetutorial14\examples\common\admin-password.txt" --user admin hello2.war (asant deploy-war)
  - App server admin console
  - NetBeans

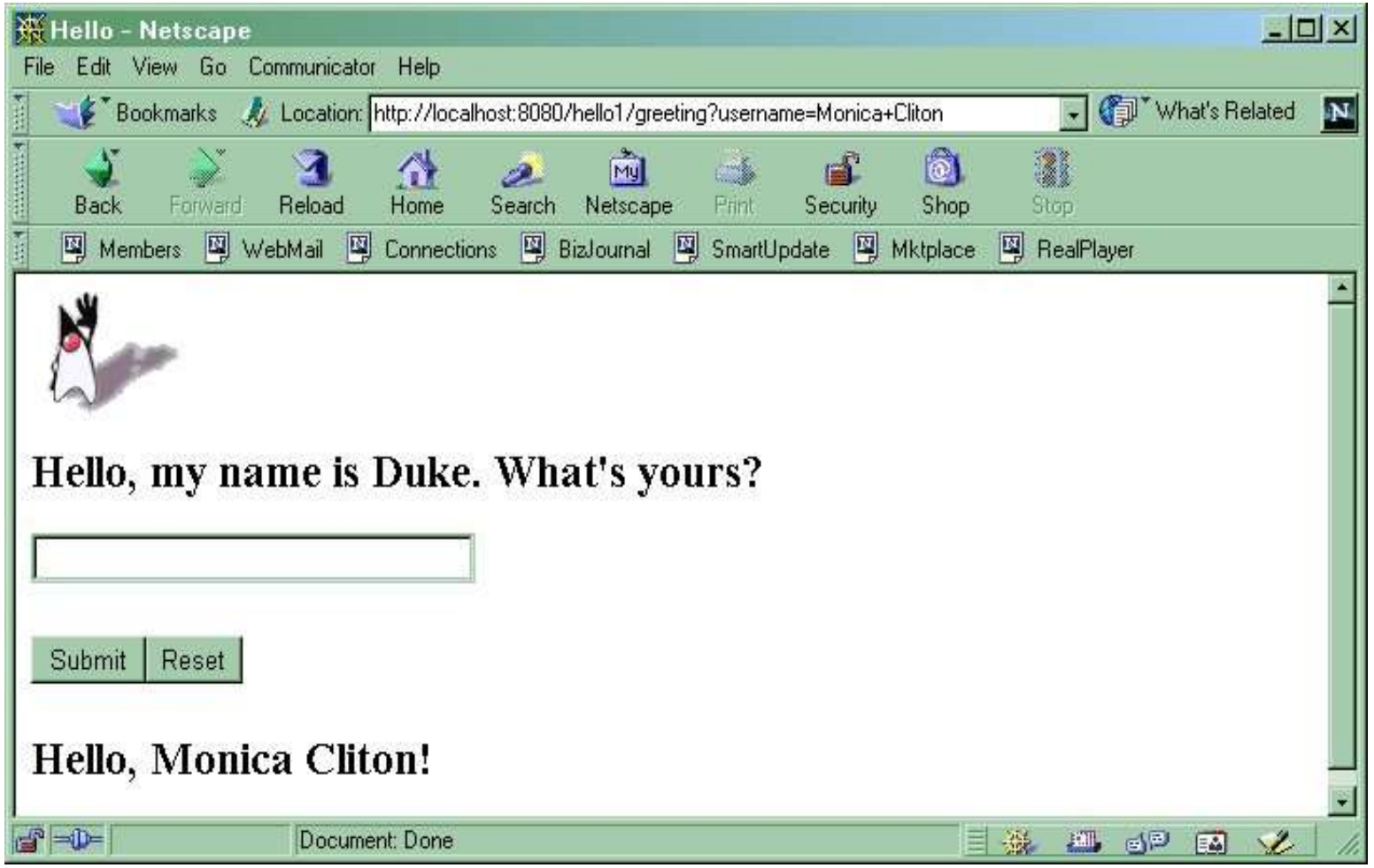
6. Perform Client Access to Web Application

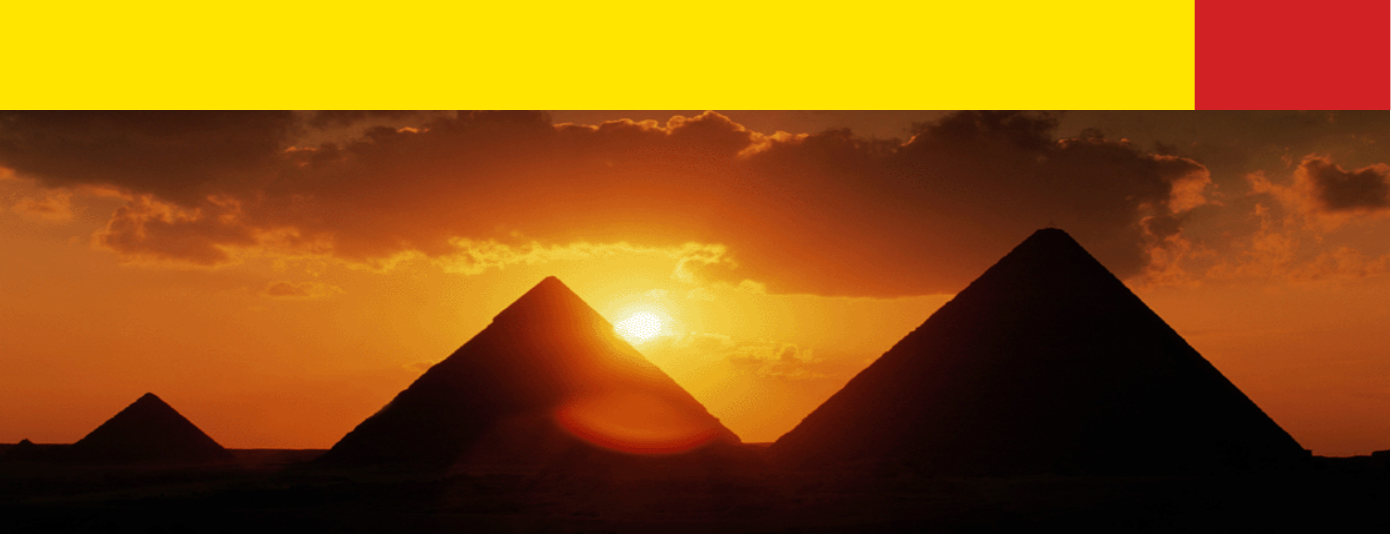
- From a browser, go to URN of the Web application

http://localhost:8080/hello1/greeting

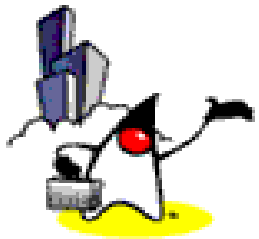


Running Web Application





Web Application Archive (*.WAR)



Web Application

- Web application can be deployed in two different forms
  - a *.war file or
  - an unpacked directory laid out in the same format as a *.war file (build directory)
- Use *.war file when you have to deploy on a remote machine
  - asant deploy-war command

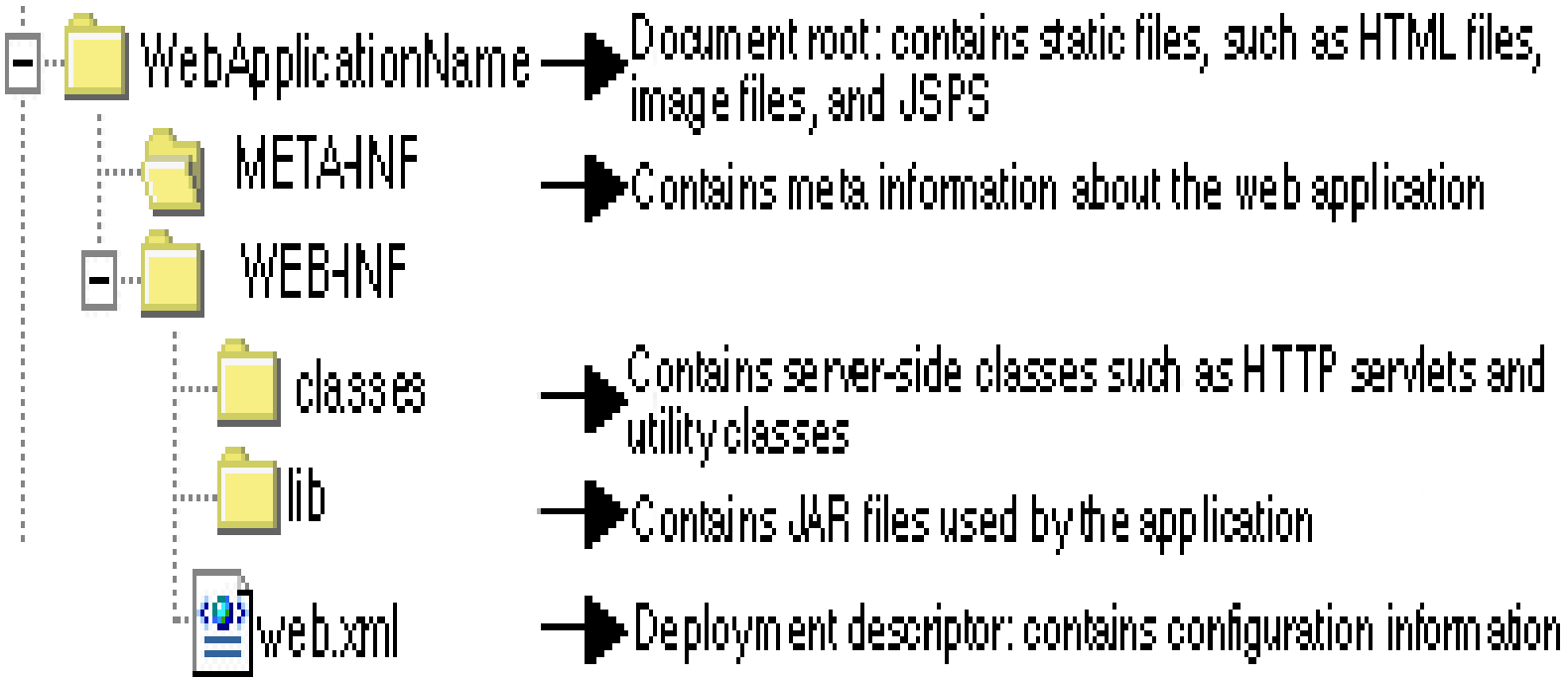
What is *.WAR file?

- Ready to deploy'able package over web container
- Similar to *.jar file
- Contains things to be deployed
  - Web components (servlets or JSP's)
  - Server-side utility classes
  - Static Web presentation content (HTML, image, etc)
  - Client-side classes (applets and utility classes)
- Reflects contents in build directory

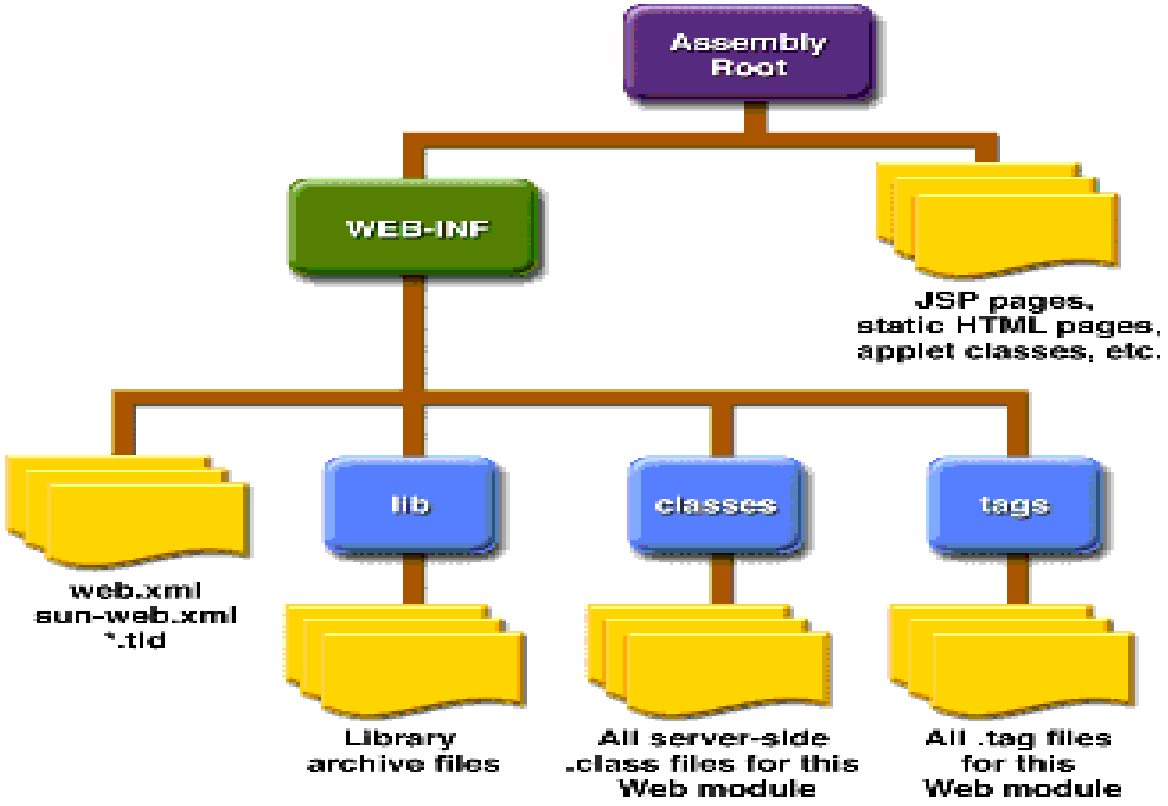
Document Root & Context

- Document Root of the Web application
  - Top-level directory of WAR
  - Contains JSP pages, client-side classes and archives, and static Web resources are stored
  - Also contains WEB-INF directory
- A context is a name that gets mapped to the document root of a Web application
  - /hello1 is context for hello1 example
  - Distinguishes a Web application in a single Web container
  - Has to be specified as part of client URN

Directory Structure of *.WAR file



Directory Structure of *.WAR file



How to Create *.WAR file?

- 3 different ways
  - Use IDE (NetBeans)
  - Use ant tool after putting proper build instruction in build.xml file
    - “asant create-war” (under J2EE 1.4 tutorial)
  - Use “jar cvf <filename>.war .” command under build directory

Example: Creating hello2.war via "asant create-war" command

```
C:\j2eetutorial14\examples\web\hello2>asant create-war
Buildfile: build.xml
...
create-war:
     [echo] Creating the WAR....
   [delete] Deleting:
    C:\j2eetutorial14\examples\web\hello2\assemble\war\hello2.war
   [delete] Deleting directory
    C:\j2eetutorial14\examples\web\hello2\assemble\war\WEB-INF
    [copy] Copying 1 file to
    C:\j2eetutorial14\examples\web\hello2\assemble\war\WEB-INF
    [copy] Copying 2 files to
    C:\j2eetutorial14\examples\web\hello2\assemble\war\WEB-INF\classes
     [war] Building war:
    C:\j2eetutorial14\examples\web\hello2\assemble\war\hello2.war
    [copy] Copying 1 file to C:\j2eetutorial14\examples\web\hello2
```

Example: Creating hello2.war via jar command

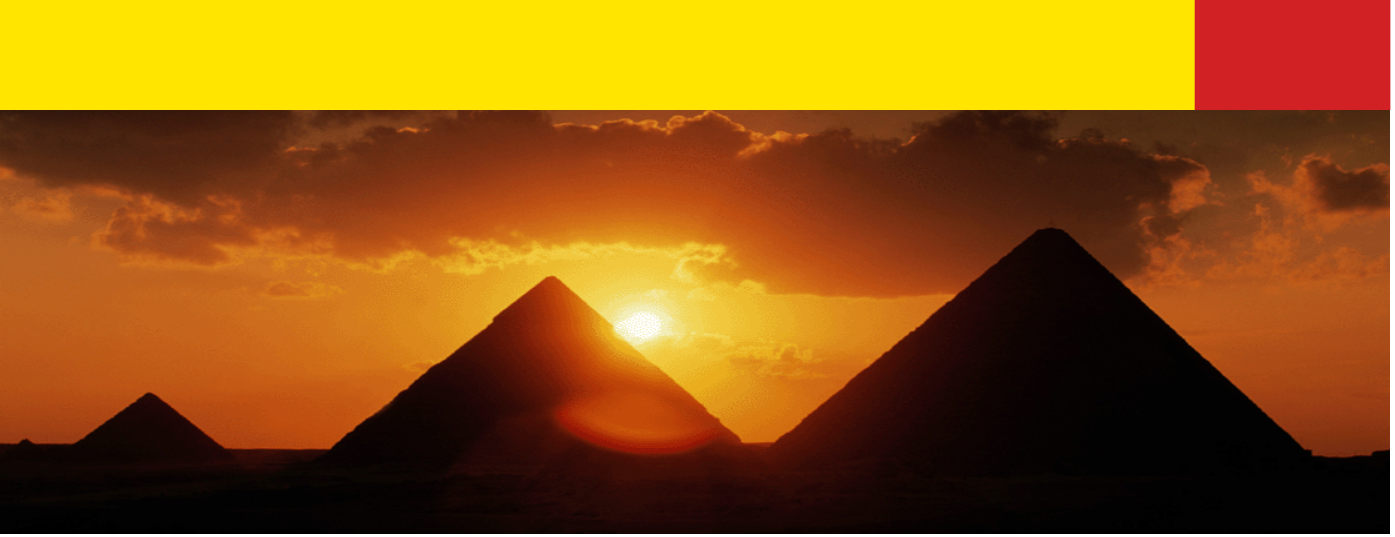
```
C:\j2eetutorial14\examples\web\hello2\build>jar cvf hello2.war .
added manifest
adding: duke.waving.gif(in = 1305) (out= 1295)(deflated 0%)
adding: servlets/(in = 0) (out= 0)(stored 0%)
adding: servlets/GreetingServlet.class(in = 1680) (out= 887)(deflated 47%)
adding: servlets/ResponseServlet.class(in = 1090) (out= 572)(deflated 47%)

C:\j2eetutorial14\examples\web\hello2\build>jar xvf hello2.war
  created: META-INF/
extracted: META-INF/MANIFEST.MF
extracted: duke.waving.gif
  created: servlets/
extracted: servlets/GreetingServlet.class
extracted: servlets/ResponseServlet.class
```

WEB-INF Directory

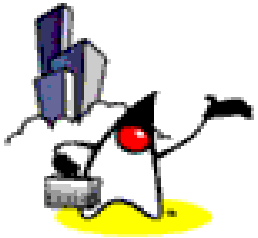
- Subdirectory of Document root
- Contains
  - web.xml : Web application deployment descriptor
  - JSP tag library descriptor files
  - Classes : A directory that contains server-side classes: servlets, utility classes, and JavaBeans components
  - lib : A directory that contains JAR archives of libraries (tag libraries and any utility libraries called by server-side classes)

HTTP request URL & Web component URN (alias) & Context

- Request URL: User specified access point of a web resource
  - http://[host]:[port]/[request path]?[query string]
  - [request path] is made of context and web component's URN
  - http://localhost:8080/hello1/greeting?username=Monica
- Context: Name of the root document of a web application – Identifies a particular application on that server
  - /hello1 is context



Configuring Web Application via web.xml



Configuring Web Application

- Configuration information is specified in web.xml (Web Applications Deployment Descriptor)

Web Applications Deployment Descriptor (web.xml)

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

Web Applications Deployment Descriptor (web.xml)

- 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

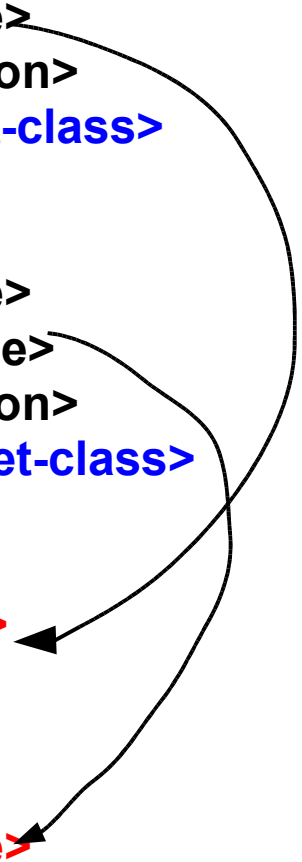
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
    Web Application 2.3//EN" "http://java.sun.com/dtd/web-
    app_2_3.dtd">
```

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)

```
<servlet>
        <servlet-name>greeting</servlet-name>
        <display-name>greeting</display-name>
        <description>no description</description>
        <servlet-class>GreetingServlet</servlet-class>
</servlet>
<servlet>
        <servlet-name>response</servlet-name>
        <display-name>response</display-name>
        <description>no description</description>
        <servlet-class>ResponseServlet</servlet-class>
 </servlet>
 <servlet-mapping>
        <servlet-name>greeting</servlet-name>
        <url-pattern>/greeting</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>response</servlet-name>
        <url-pattern>/response</url-pattern>
 </servlet-mapping>
```



Context and Initialization Parameters (of web.xml)

- 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 (of web.xml)

- 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 (of web.xml)

- 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>
```

Passion!