NEW RELEASE

Jakarta EE 10

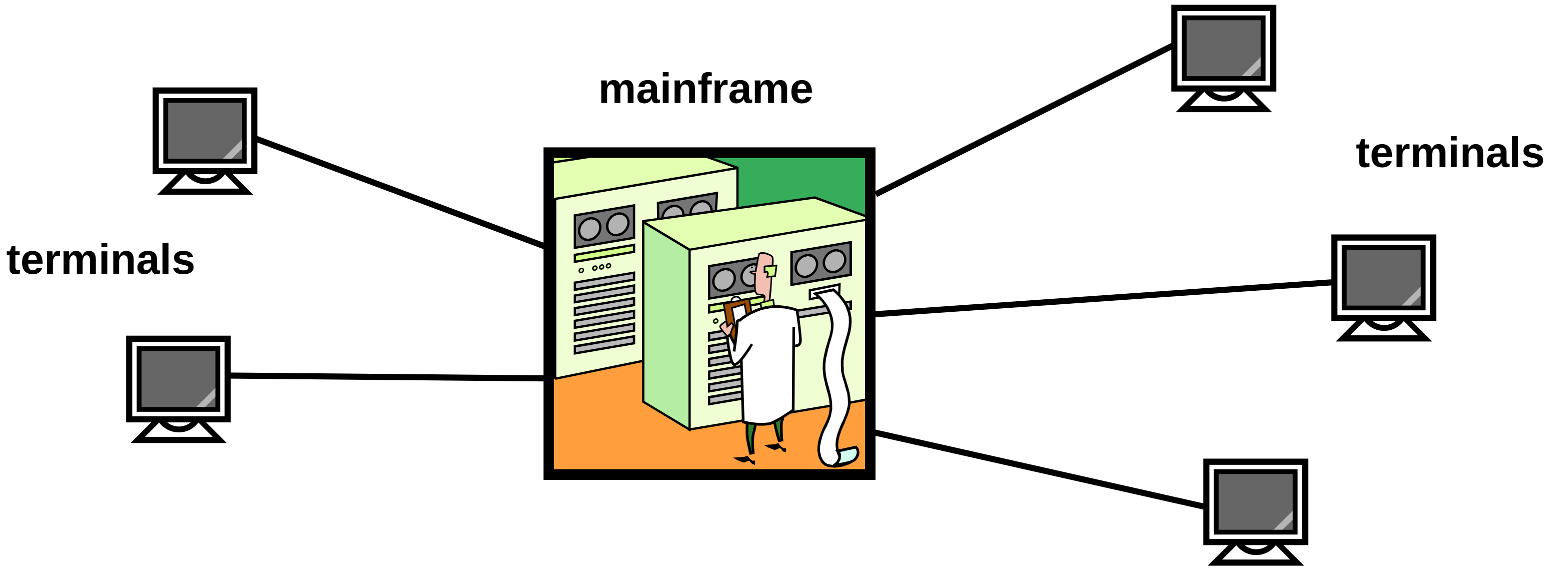
Community-Driven

Modernized **Simplified** **Lightweight**

JAKARTA® EE

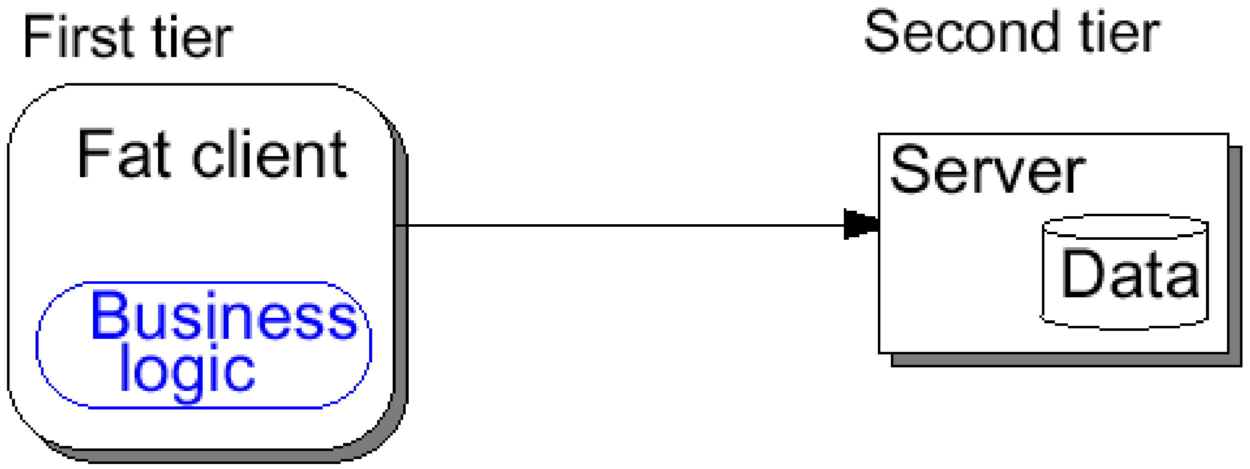
In the beginning,

- *In the beginning, there was darkness and cold. Then, …*

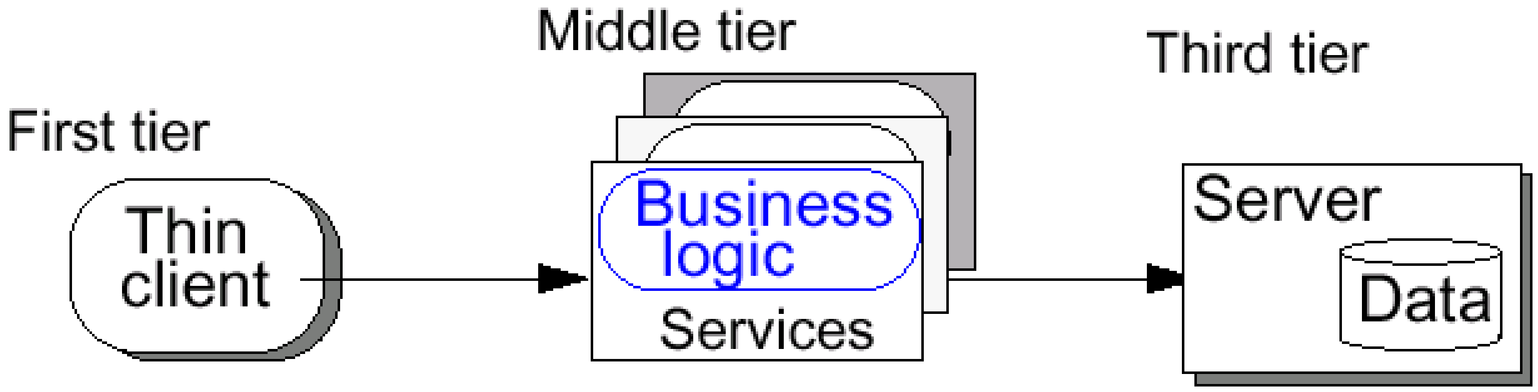


Centralized, non-distributed

- In the 90’s, systems should be *client-server*

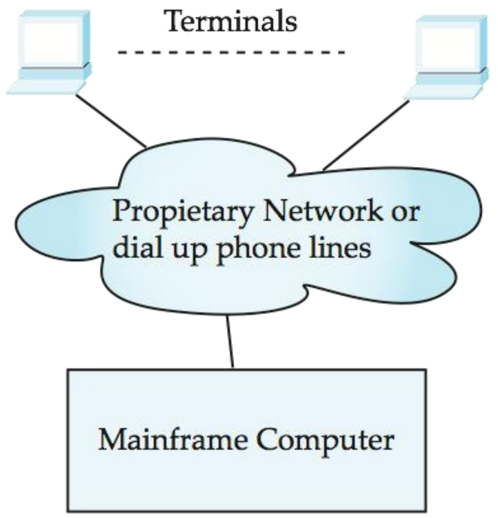


- Today, enterprise applications use the *multi-tier* model

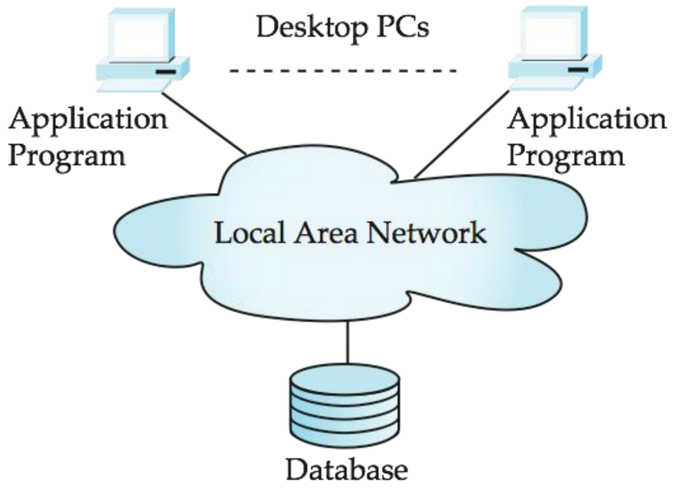


Application Architecture Evolution

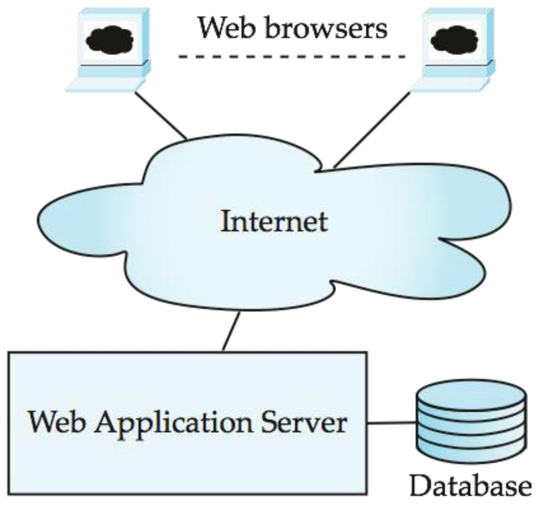
- Three distinct era’s of application architecture
  - mainframe (1960’s and 70’s)
  - personal computer era (1980’s)
  - We era (1990’s onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era

The World Wide Web

- The Web is a distributed information system based on hypertext
- Most Web documents are hypertext documents formatted via the Hypertext Markup Language (HTML)
- HTML documents contain
  - text along with font specifications, and other formatting instructions
  - hypertext links to other documents, which can be associated with regions of the text.
  - forms, enabling users to enter data which can then be sent back to the Web server

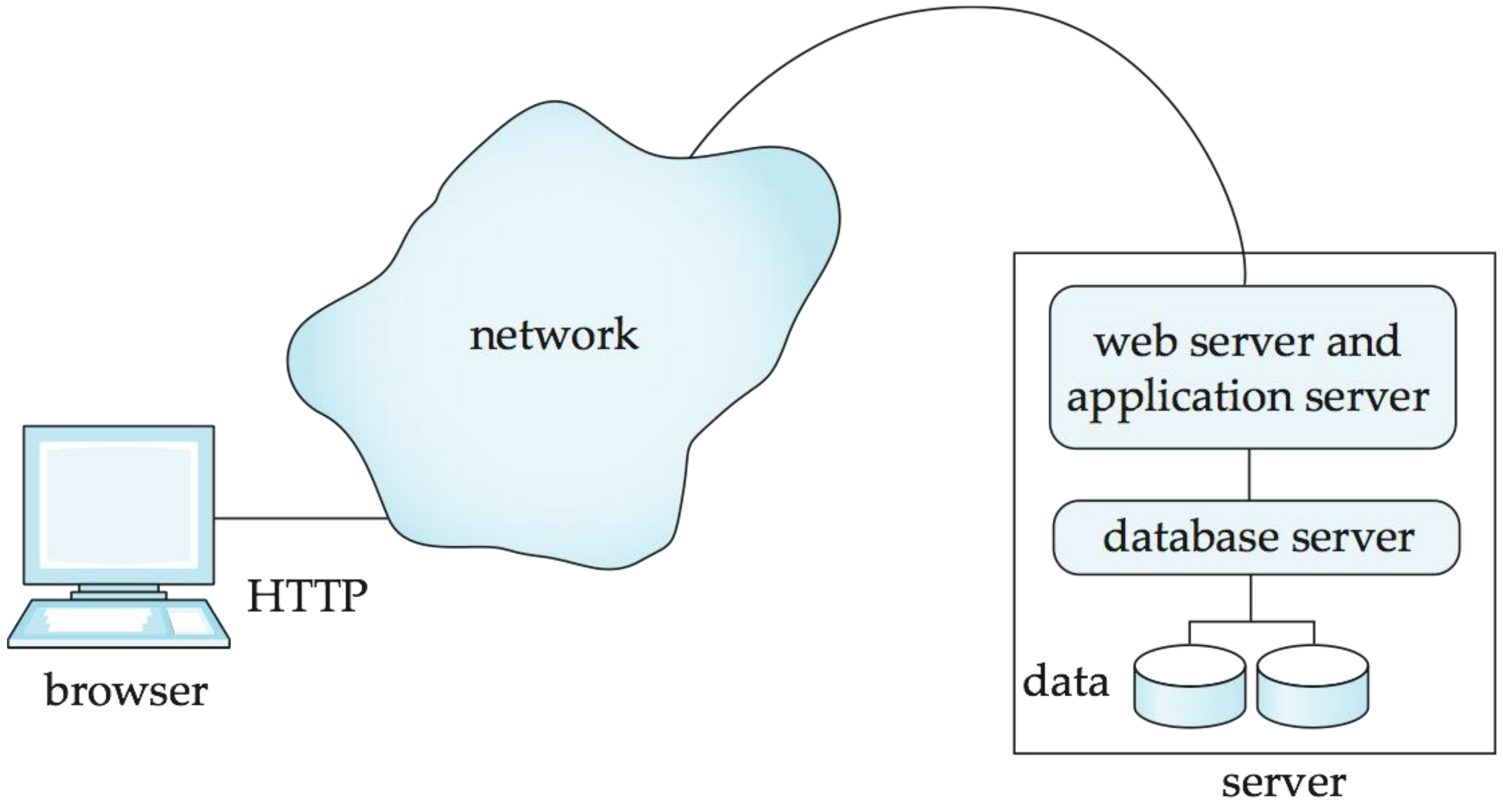
URL, HTML and HTTP

- URL – Universal Resource Locator – unique identifier of the resources on the server
- HTML provides formatting, hypertext link, and image display features
  - including tables, stylesheets (to alter default formatting), etc.
- HTML also provides input features
    - Select from a set of options
      - Pop-up menus, radio buttons, check lists
    - Enter values
      - Text boxes
  - Filled in input sent back to the server, to be acted upon by an executable at the server
- HyperText Transfer Protocol (HTTP) used for communication with the Web server

Web Servers

- A Web server can easily serve as a front end to a variety of information services.
- The document name in a URL may identify an executable program, that, when run, generates a HTML document.
  - When an HTTP server receives a request for such a document, it executes the program, and sends back the HTML document that is generated.
  - The Web client can pass extra arguments with the name of the document.
- To install a new service on the Web, one simply needs to create and install an executable that provides that service.
  - The Web browser provides a graphical user interface to the information service.
- Common Gateway Interface (CGI): a standard interface between web and application server

Two-Layer Web Architecture



Application Servers

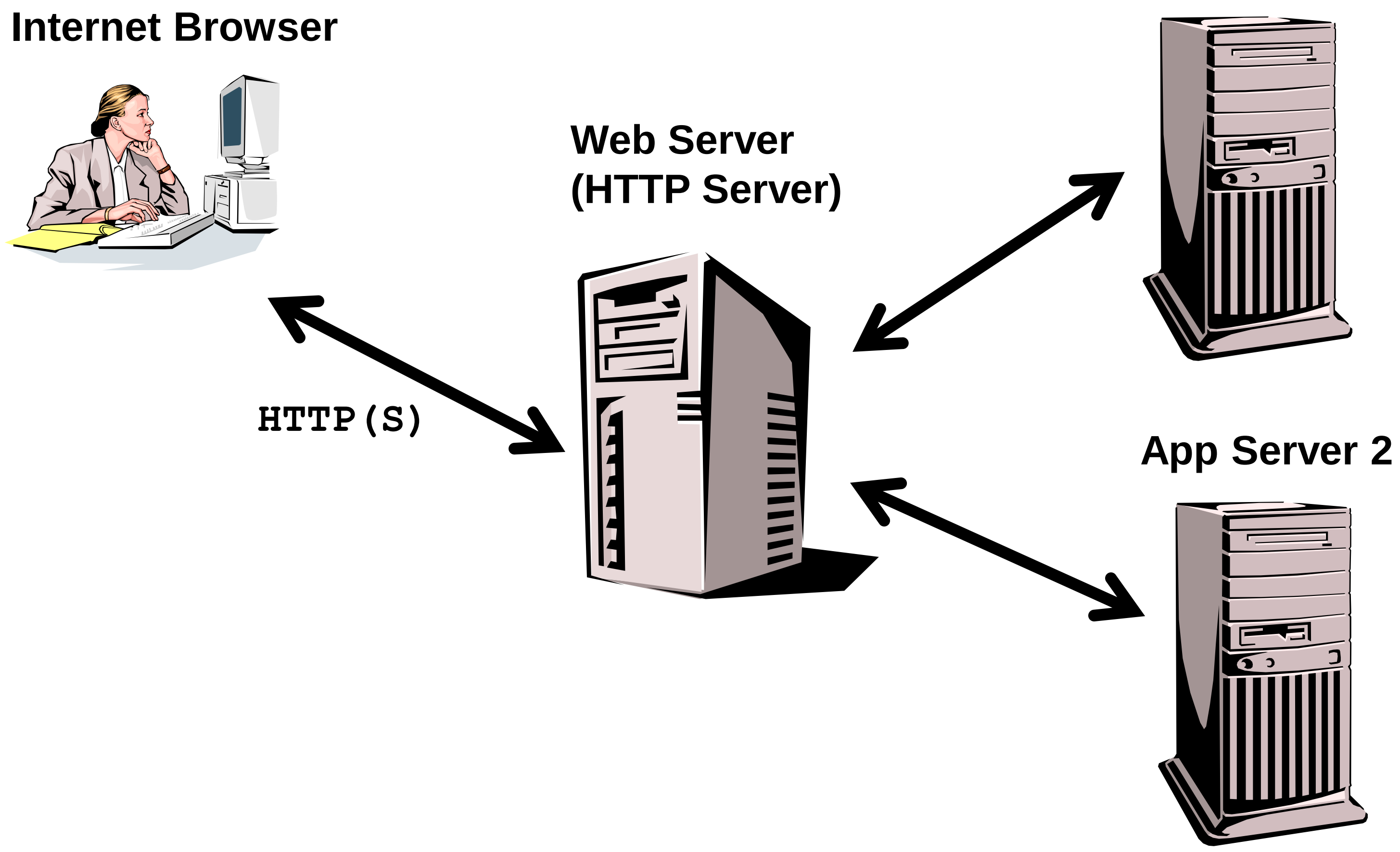
- “Multi-tier applications” have several independent components

- An *application server* provides the infrastructure and services to run such applications

J2EE Application Servers

- Major J2EE products:
  - BEA WebLogic
  - IBM WebSphere
  - Sun iPlanet Application Server
  - Oracle 9iAS
  - HP/Bluestone Total-e-Server
  - Borland AppServer
  - Jboss (free open source)

Web Server and Application Server



The **Java EE** stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as Jakarta EE.

What is Java EE (or J2EE)

- Short for ***J**ava Platform, **E**nterprise **E**dition*.
- Java EE is a platform-independent, Java-centric environment from Oracle for developing, building and deploying Web-based enterprise applications online.
- The Java EE is a set of specifications, extending Java SE with specifications for enterprise features such as distributed computing and web services..

What is Java Platform, Enterprise Edition?

The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

Current version is Jakarta EE 10 - released recently

Java SE

- When most people think of the Java programming language, they think of the Java SE API.

- Java SE's API provides the core functionality of the Java programming language.

- It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.

- In addition to the core API, the Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java technology applications.

Java EE

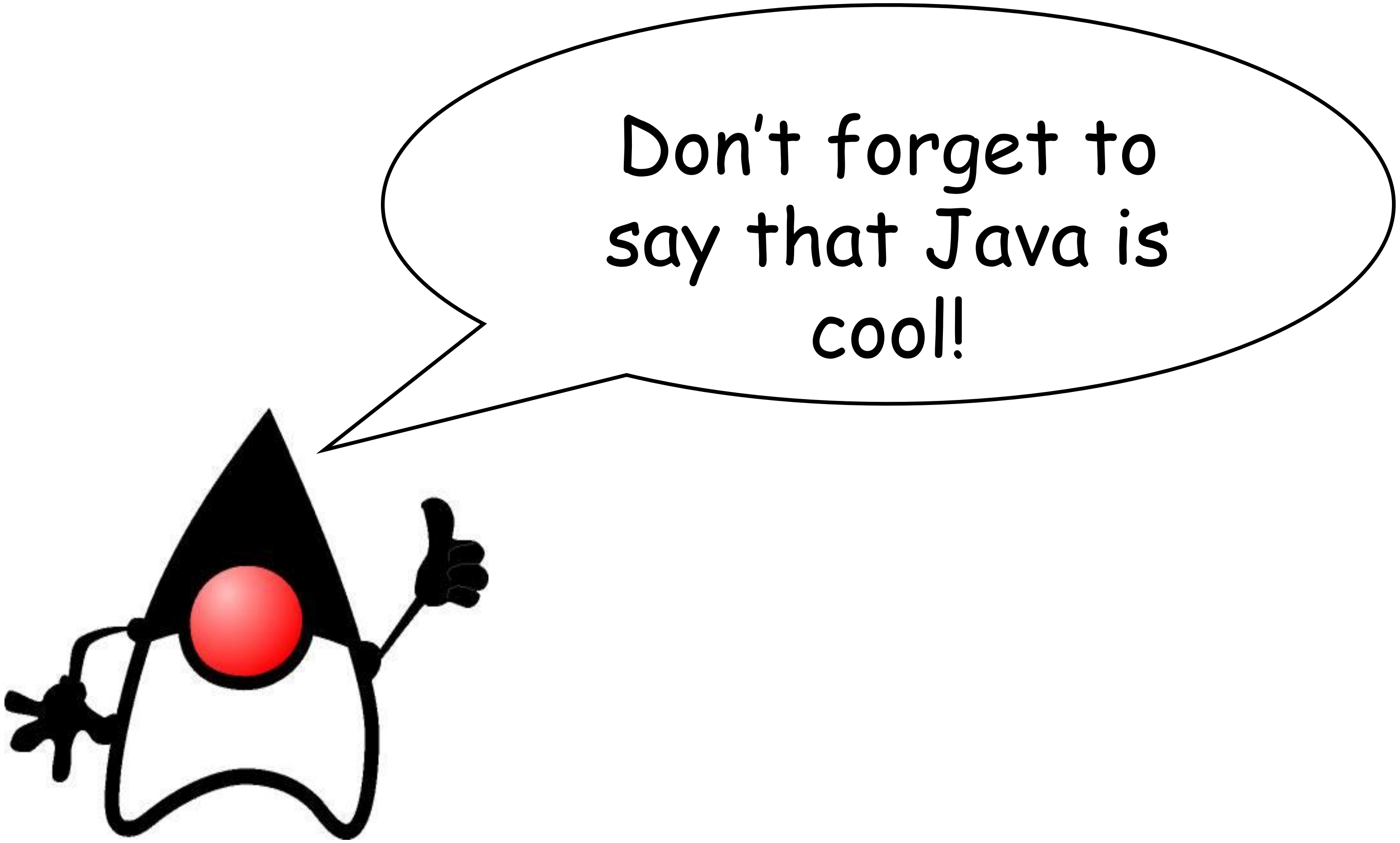
- The Java EE platform is built on top of the Java SE platform.
- The Java EE platform provides an API and runtime environment for developing and running
  - large-scale
  - multi-tiered
  - scalable
  - reliable
  - secure network applications

Java EE Benefits

- Flexibility of scenarios and support to several types of clients

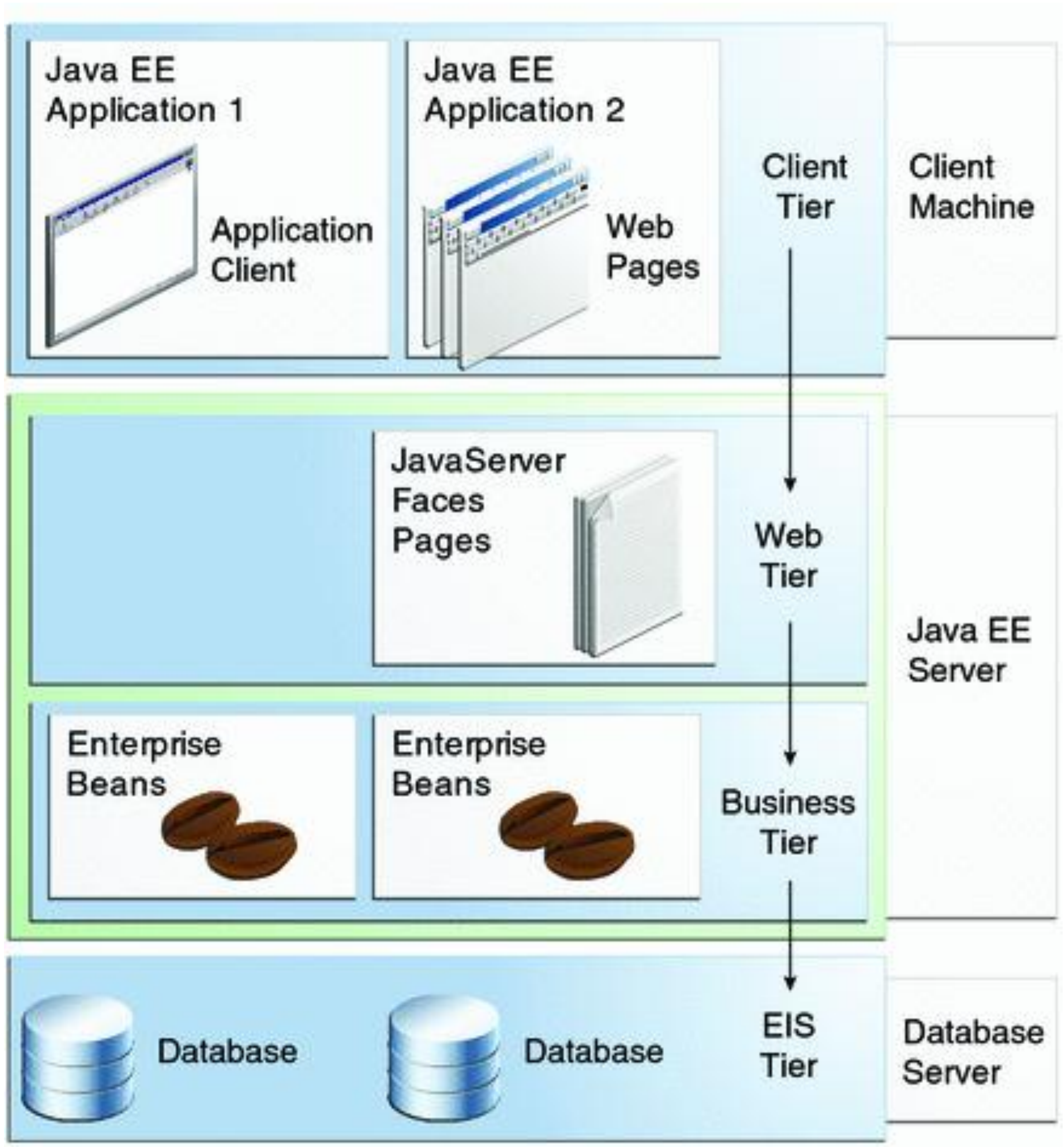
- Programming productivity:
  - Services allow developer to focus on business
  - Component development facilitates maintenance and reuse
  - Enables deploy-time behaviors
  - Supports division of labor

Java EE Benefits



J2EE Application Model

- This model partitions the work needed to implement a multitier service into the following parts:
  - The business and presentation logic to be implemented by the developer
  - The standard system services provided by the Java EE platform



Multi-tiered Applications

J2EE Components

- Java EE applications are made up of components.
- A **Java EE component** is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components.
- The Java EE specification defines the following Java EE components:
  - Application clients and applets are components that run on the client.
  - Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) technology components are web components that run on the server.
  - Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server.

Java EE Clients

- **A Java EE client is usually either a web client or an application client**.

- A **web client** consists of two parts:
  - Dynamic web pages containing various types of markup language (HTML, XML, and so on), which are generated by web components running in the web tier
  - A web browser, which renders the pages received from the server

- A web client is sometimes called a **thin client**. Thin clients usually do not query databases, execute complex business rules, or connect to legacy applications.

- When you use a thin client, such heavyweight operations are off-loaded to enterprise beans executing on the Java EE server, where they can leverage the security, speed, services, and reliability of Java EE server-side technologies.

J2EE Clients

Application Clients:

- An **application client** runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language.

- An application client typically has a graphical user interface (GUI) created from the Swing or the Abstract Window Toolkit (AWT) API, but a command-line interface is certainly possible.

- Application clients directly access enterprise beans running in the business tier.

- Application clients written in languages other than Java can interact with Java EE servers, enabling the Java EE platform to interoperate with legacy systems, clients, and non-Java languages.

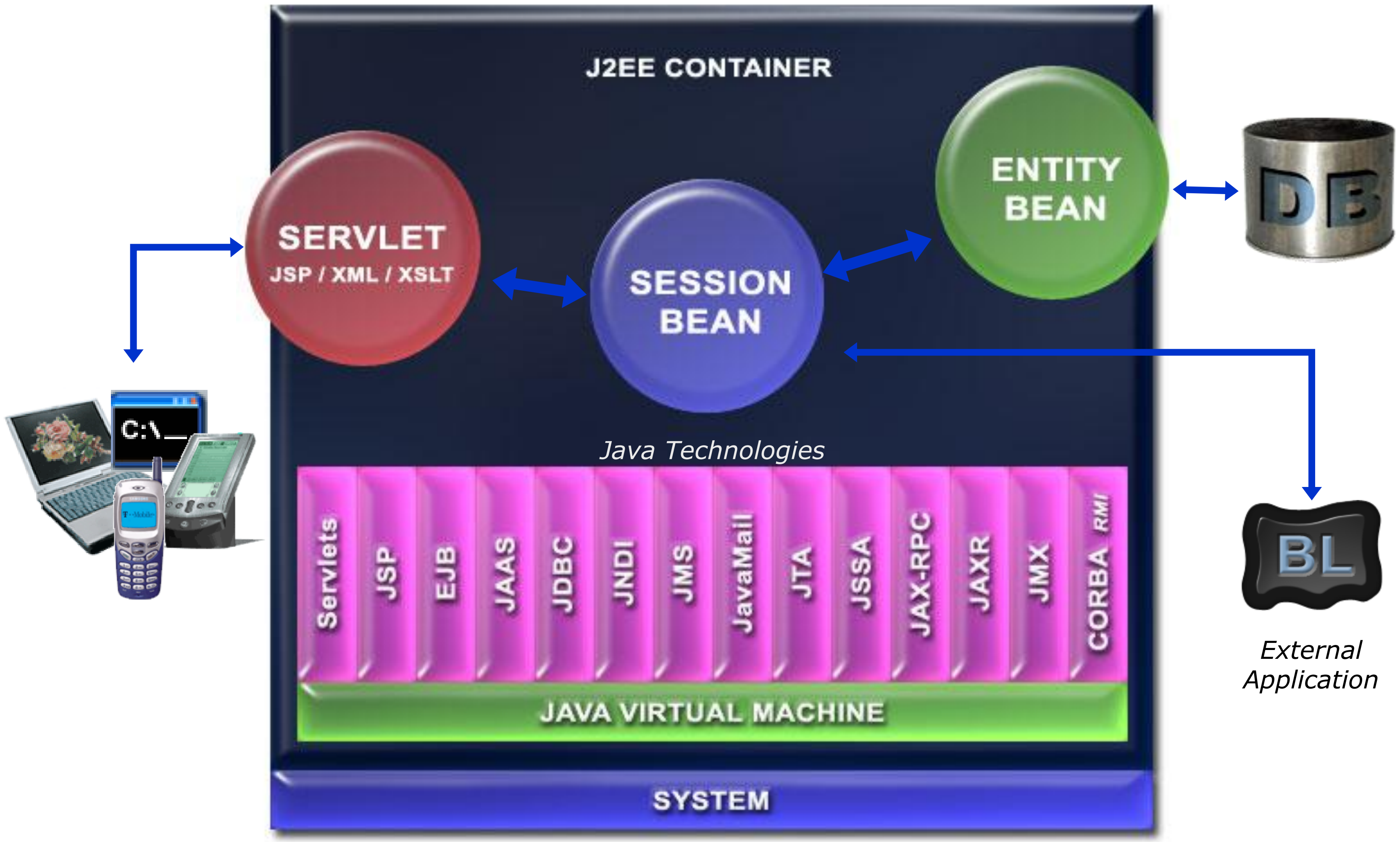
Web Components

- Java EE web components are either servlets or web pages created using JavaServer Faces technology and/or JSP technology (JSP pages).
  - **Servlets** are Java programming language classes that dynamically process requests and construct responses.
  - **JSP pages** are text-based documents that execute as servlets but allow a more natural approach to creating static content.
  - **JavaServer Faces technology** builds on servlets and JSP technology and provides a user interface component framework for web applications.

J2EE Containers

- Web components are supported by the services of a runtime platform called a **web container**.
- A web container provides such services as request dispatching, security, concurrency, and lifecycle management. A web container also gives web components access to such APIs as naming, transactions, and email.
- Normally, thin-client multi-tiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details.
- The component-based and platform-independent Java EE architecture makes Java EE applications easy to write because business logic is organized into reusable components.

The J2EE Framework



Web Applications

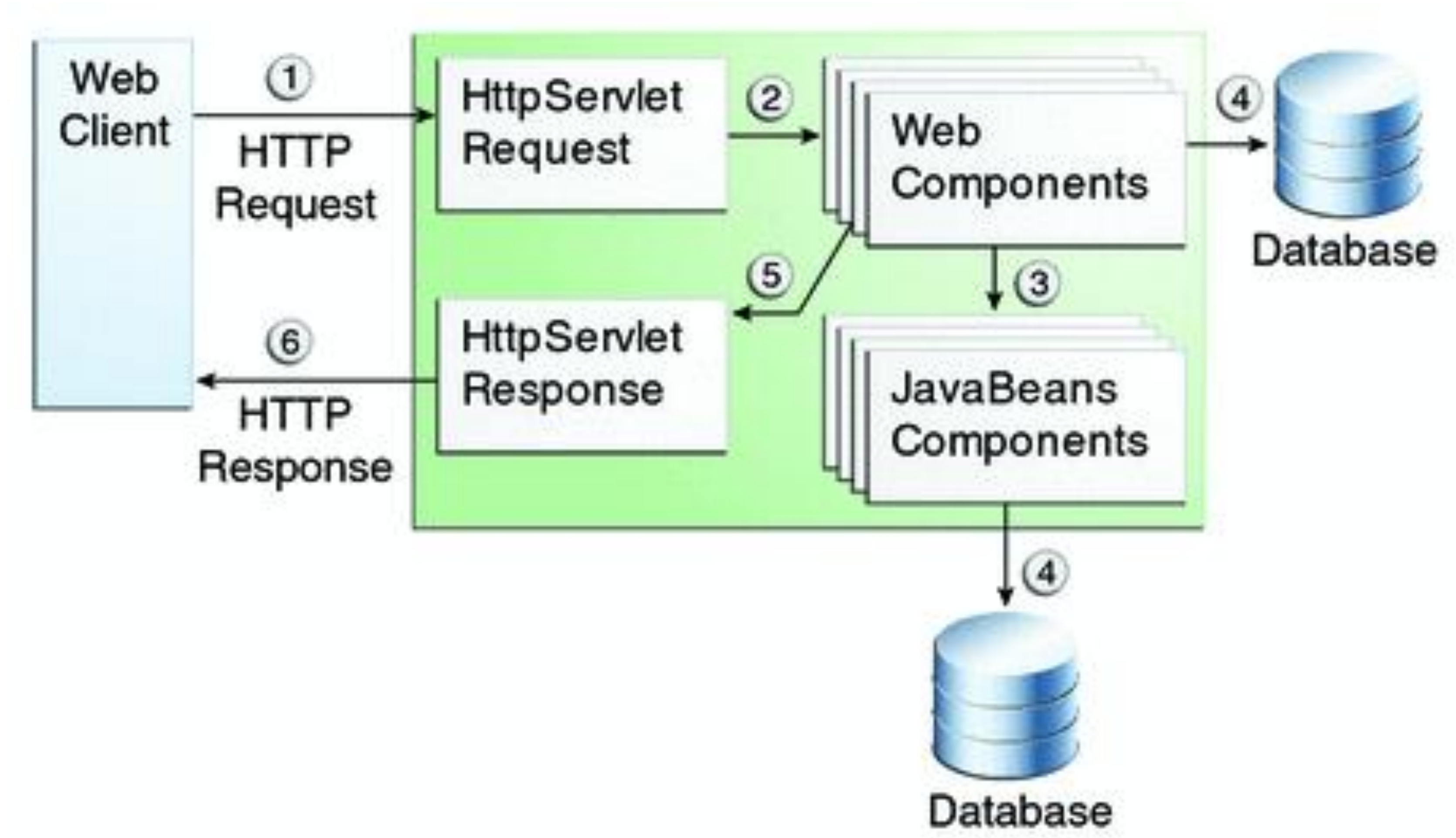
- A **web application** is a dynamic extension of a web or application server.

Web applications are of the following types:

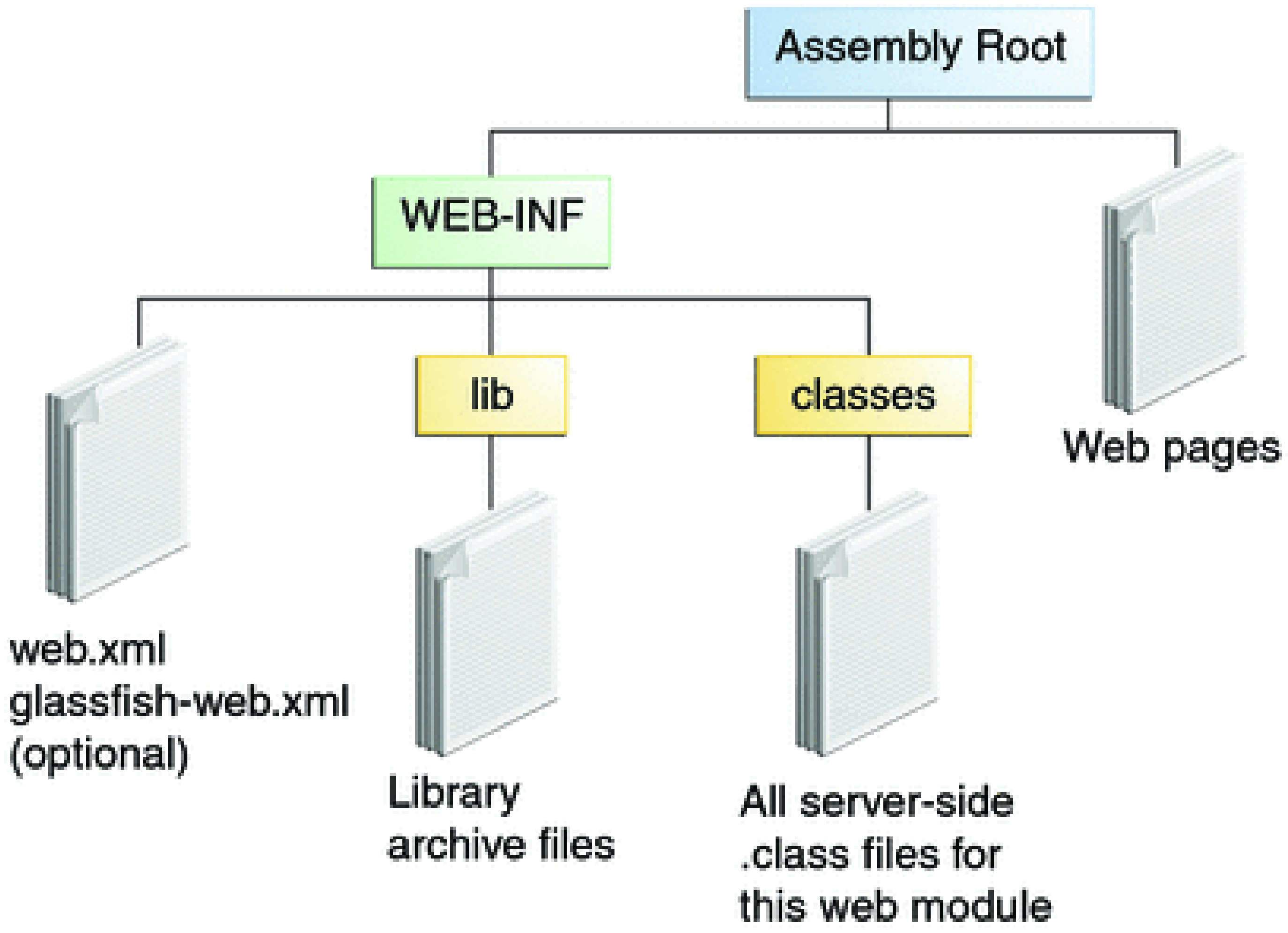
- **Presentation-oriented**: A **presentation-oriented web application** generates interactive web pages containing various types of markup language (HTML, XHTML, XML, and so on) and dynamic content in response to requests.

- **Service-oriented**: A **service-oriented web application** implements the endpoint of a web service. Presentation-oriented applications are often clients of service-oriented web applications.

Java Web Application Request Handling



Web Module Structure



Java Servlets

What are Servlets?

- Basically, a java program that runs on the server
- Creates dynamic web pages

What Do They Do?

- Handle data/requests sent by users (clients)
- Create and format results
- Send results back to user

History

- Dynamic websites were often created with CGI
- CGI: Common Gateway Interface
- Poor solution to today’s needs
- A better solution was needed

Servlets vs. CGI

- Servlet Advantages
  - Efficient
    - Single lightweight java thread handles multiple requests
    - Optimizations such as computation caching and keeping connections to databases open
  - Convenient
    - Many programmers today already know java
  - Powerful
    - Can talk directly to the web server
    - Share data with other servlets
    - Maintain data from request to request
  - Portable
    - Java is supported by every major web browser (through plugins)
  - Inexpensive
    - Adding servlet support to a server is cheap or free

Servlets vs. CGI

- CGI Advantages
  - CGI scripts can be written in any language
  - Does not depend on servlet-enabled server

What Servlets Need

- Java Server Web Development Kit
- Servlet capable server
- Java Server Pages (JSP)
- Servlet code

Servlet Code

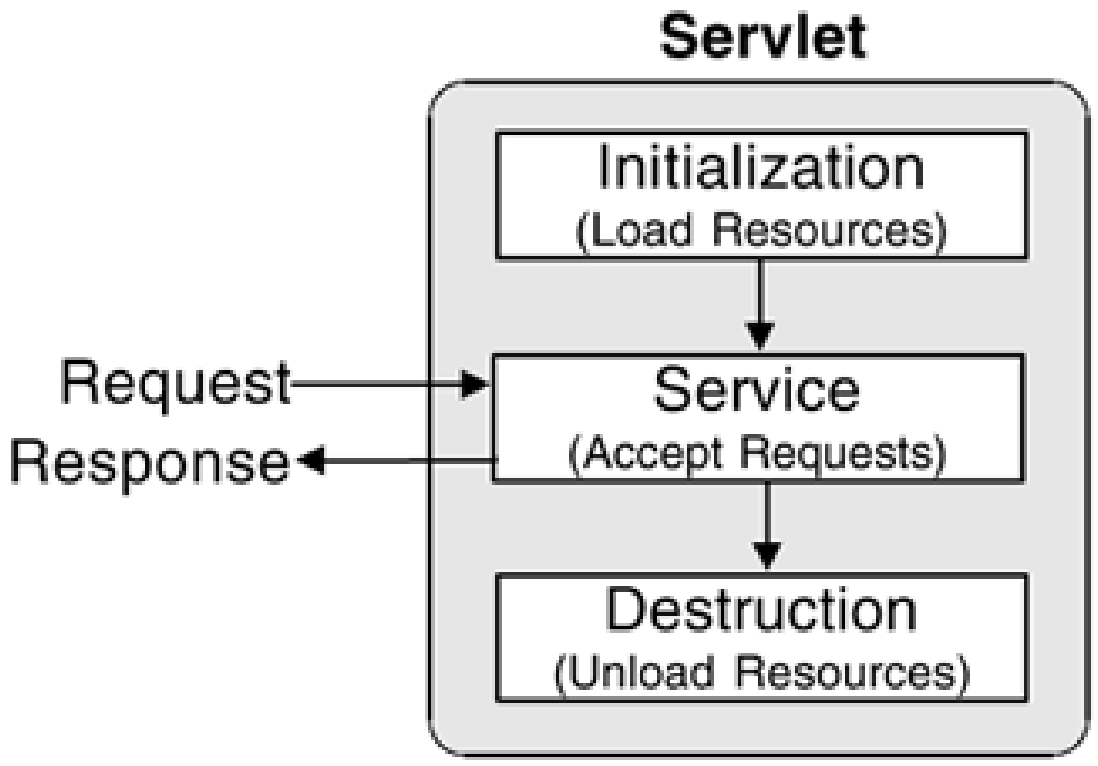
- Written in standard Java
- Implement the javax.servlet.Servlet interface

Main Concepts of Servlet Programming

- Life Cycle
- Client Interaction
- Saving State
- Servlet Communication
- Calling Servlets
- Request Attributes and Resources

Life Cycle

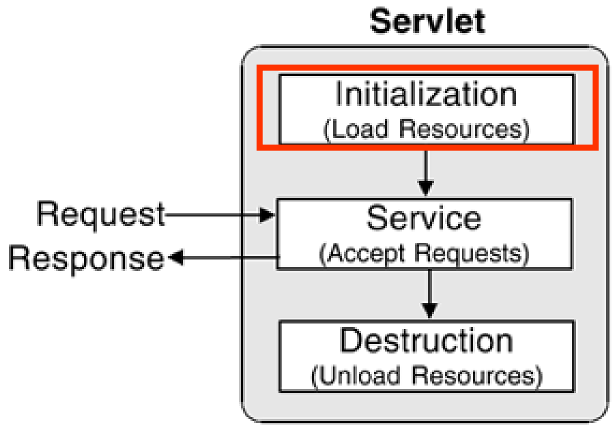
- Initialize
- Service
- Destroy



Life Cycle: Initialize

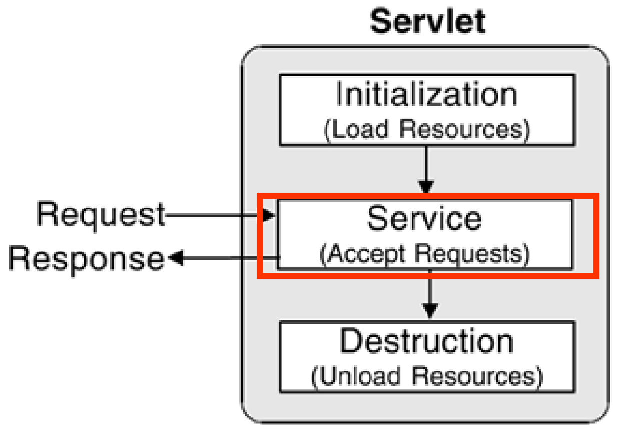
- Servlet is created when servlet container receives a request from the client

- Init() method is called only once



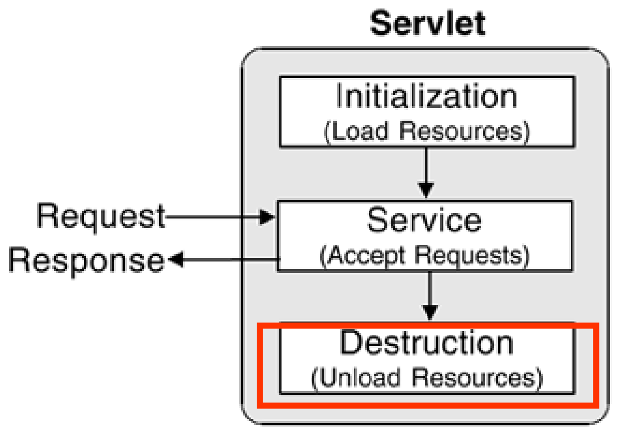
Life Cycle: Service

- Any requests will be forwarded to the service() method
  - doGet()
  - doPost()
  - doDelete()
  - doOptions()
  - doPut()
  - doTrace()



Life Cycle: Destroy

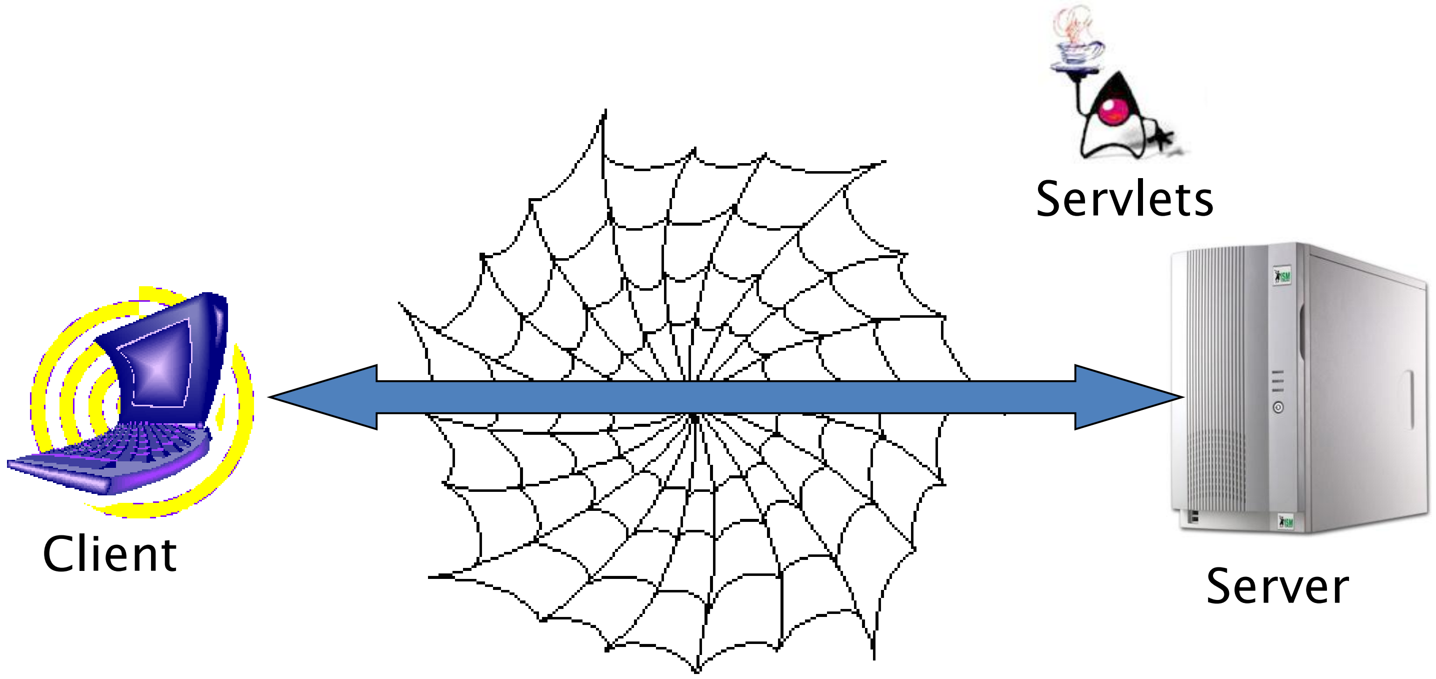
- destroy() method is called only once
- Occurs when
  - Application is stopped
  - Servlet container shuts down
- Allows resources to be freed



Client Interaction

- Request
  - Client (browser) sends a request containing
    - Request line (method type, URL, protocol)
    - Header variables (optional)
    - Message body (optional)
- Response
  - Sent by server to client
    - response line (server protocol and status code)
    - header variables (server and response information)
    - message body (response, such as HTML)

- Thin clients (minimize download)
- Java all “server side”



Saving State

- Session Tracking
  – A mechanism that servlets use to maintain state about a series of requests from the same user (browser) across some period of time.
- Cookies
  – A mechanism that a servlet uses to have clients hold a small amount of state-information associated with the user.

Servlet Communication

- To satisfy client requests, servlets sometimes need to access network resources: other servlets, HTML pages, objects shared among servlets at the same server, and so on.

Calling Servlets

- Typing a servlet URL into a browser window
  - Servlets can be called directly by typing their URL into a browser's location window.
- Calling a servlet from within an HTML page
  - Servlet URLs can be used in HTML tags, where a URL for a CGI-bin script or file URL might be found.

Request Attributes and Resources

- Request Attributes
  - getAttribute
  - getAttributeNames
  - setAttribute
- Request Resources - gives you access to external resources
  - getResource
  - getResourceAsStream

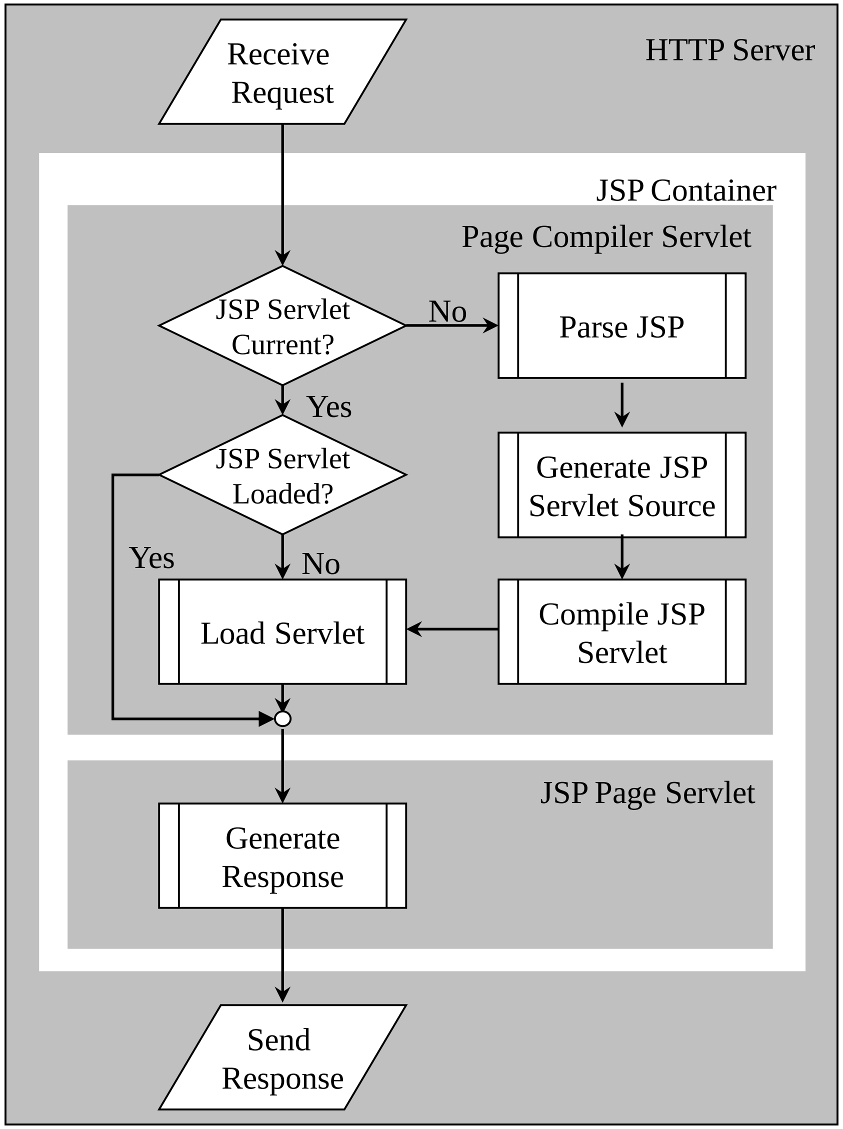
Java Server Pages (JSP)

- Java Server Pages are HTML pages embedded with snippets of Java code.
  - It is an inverse of a Java Servlet
- Four different elements are used in constructing JSPs
  - Scripting Elements
  - Implicit Objects
  - Directives
  - Actions

Java Server Pages (JSP)

Architecture

- JSPs run in two phases
  - **Translation Phase**
  - **Execution Phase**
- In translation phase JSP page is compiled into a servlet
  - called JSP Page Implementation class
- In execution phase the compiled JSP is processed



Scripting Elements

Types

- There are three kinds of scripting elements
  - **Declarations**
  - **Scriptlets**
  - **Expressions**

Declarations

Basics

- Declarations are used to define methods & instance variables
  - Do not produce any output that is sent to client
  - Embedded in **<%! and %>** delimiters

  Example:

```
<%!
      Public void jspDestroy() {
            System.out.println("JSP Destroyed");
      }
      Public void jspInit() {
            System.out.println("JSP Loaded");
      }
      int myVar = 123;
%>
```

  - The functions and variables defined are available to the JSP Page as well as to the servlet in which it is compiled

Scriptlets

Basics

- Used to embed java code in JSP pages.
  - Contents of JSP go into _JSPpageservice() method
  - Code should comply with syntactical and semantic constuct of java
  - Embedded in **<% and %>** delimiters

Example:

```
<%
      int x = 5;
      int y = 7;
      int z = x + y;
%>
```

Expressions

Basics

- Used to write dynamic content back to the browser.
  - If the output of expression is Java primitive the value is printed back to the browser
  - If the output is an object then the result of calling toString on the object is output to the browser
  - Embedded in **<%= and %>** delimiters

Example:

- <%="Fred"+ " " + "Flintstone %>

  prints "Fred Flintstone" to the browser
- <%=Math.sqrt(100)%>

  prints 10 to the browser

Java Implicit Objects

Scope

- Implicit objects provide access to server side objects
  - e.g. request, response, session etc.
- There are four scopes of the objects
  - Page: Objects can only be accessed in the page where they are referenced
  - Request: Objects can be accessed within all pages that serve the current request.

    (Including the pages that are forwarded to and included in the original jsp page)
  - Session: Objects can be accessed within the JSP pages for which the objects are defined
  - Application: Objects can be accessed by all JSP pages in a given context

Java Implicit Objects

List

- **request:** Reference to the current request
- **response:** Response to the request
- **session:** session associated woth current request
- **application:** Servlet context to which a page belongs
- **pageContext:** Object to access request, response, session and application associated with a page
- **config:** Servlet configuration for the page
- **out:** Object that writes to the response output stream
- **page:** instance of the page implementation class (this)
- **exception:** Available with JSP pages which are error pages

Java Implicit Objects

Example

```
<html>
  <head>
    <title>Implicit Objects</title>
  </head>
  <body style="font-family:verdana;font-size:10pt">
    <p>
      Using Request parameters...<br>
      <b>Name:</b> <%= request.getParameter("name") %>
    </p>
    <p>
      <% out.println("This is printed using the out implicit
            variable"); %>
    </p>
    <p>
      Storing a string to the session...<br>
      <% session.setAttribute("name", “ABCj"); %>
      Retrieving the string from session...<br>
      <b>Name:</b> <%= session.getAttribute("name") %>
    </p>
    <p>
      Storing a string to the application...<br>
      <% application.setAttribute("name", “ABCj"); %>
      Retrieving the string from application...<br>
      <b>Name:</b>
      <%= application.getAttribute("name") %>
    </p>
    <p>
      Storing a string to the page context...<br>
      <% pageContext.setAttribute("name", “ABCj"); %>
      Retrieving the string from page context...</br>
      <b>Name:</b>
      <%= pageContext.getAttribute("name") %>
    </p>
  </body>
</html>
```

Directives

Basics & Types

- Messages sent to the JSP container
  - Aids the container in page translation
- Used for
  - Importing tag libraries
  - Import required classes
  - Set output buffering options
  - Include content from external files
- The jsp specification defines three directives
  - **Page:** provder information about page, such as scripting language that is used, content type, or buffer size
  - **Include** – used to include the content of external files
  - **Taglib** – used to import custom actions defined in tag libraries

Page Directives

Basics & Types

- Page directive sets page properties used during translation
  - JSP Page can have any number of directives
  - Import directive can only occur once
  - Embedded in <%@ and %> delimiters
- Different directives are
  - **Language:** (Default Java) Defines server side scripting language (e.g. java)
  - **Extends:** Declares the class which the servlet compiled from JSP needs to extend
  - **Import:** Declares the packages and classes that need to be imported for using in the java code (comma separated list)
  - **Session:** (Default true) Boolean which says if the session implicit variable is allowed or not
  - **Buffer:** defines buffer size of the jsp in kilobytes (if set to none no buffering is done)

Page Directives

Types con't.

- Different directives are (cont'd.)
  - **autoFlush:**When true the buffer is flushed when max buffer size is reached (if set to false an exception is thrown when buffer exceeds the limit)
  - **isThreadSafe:** (default true) If false the compiled servlet implements SingleThreadModel interface
  - **Info:** String returned by the getServletInfo() of the compiled servlet
  - **errorPage:** Defines the relative URI of web resource to which the response should be forwarded in case of an exception
  - **contentType:** (Default text/html) Defines MIME type for the output response
  - **isErrorPage**: True for JSP pages that are defined as error pages
  - **pageEncoding:** Defines the character encoding for the jsp page

Page Directives

Example

```
<%@
        page language="java"
        buffer="10kb"
        autoflush="true"
        errorPage="/error.jsp"
        import="java.util.*, javax.sql.RowSet"
%>
```

Include Directive

Basics

- Used to insert template text and JSP code during the translation phase.
  - The content of the included file specified by the directive is included in the including JSP page
- Example
  - <%@ include file="included.jsp" %>

JSP Actions

Basics & Types

- Processed during the request processing phase.
  - As opposed to JSP directives which are processed during translation
- Standard actions should be supported by J2EE compliant web servers
- Custom actions can be created using tag libraries
- The different actions are
  - Include action
  - Forward action
  - Param action
  - useBean action
  - getProperty action
  - setProperty action
  - plugIn action

JSP Actions

Include

- Include action used for including resources in a JSP page
  - Include directive includes resources in a JSP page at translation time
  - Include action includes response of a resource into the response of the JSP page
  - Same as including resources using RequestDispatcher interface
  - Changes in the included resource reflected while accessing the page.
  - Normally used for including dynamic resources
- Example
  - <jsp:include page=“inlcudedPage.jsp”>
  - Includes the the output of includedPage.jsp into the page where this is included.

JSP Actions

Forward

- Forwards the response to other web specification resources
  - Same as forwarding to resources using RequestDispatcher interface
- Forwarded only when content is not committed to other web application resources
  - Otherwise an IllegalStateException is thrown
  - Can be avoided by setting a high buffer size for the forwarding jsp page
- Example
  - <jsp:forward page=“Forwarded.html”>
  - Forwards the request to Forwarded.html

JSP Actions

Param

- Used in conjunction with Include & Forward actions to include additional request parameters to the included or forwarded resource

- Example

```
<jsp:forward page="Param2.jsp">
    <jsp:param name="FirstName" value="Sanjay">
</jsp:forward>
```

  – This will result in the forwarded resource having an additional parameter FirstName with a value of Sanjay

JSP Actions
useBean

- Creates or finds a Java object with the defined scope.
  - Object is also available in the current JSP as a scripting variable
- Syntax:

```
<jsp:useBean id="name"
scope="page | request | session | application"
class="className" type="typeName" |
bean="beanName" type="typeName" |
type="typeName" />
```

  - At least one of the type and class attributes must be present
  - We can't specify values for bith the class and bean name.
- Example

```
<jsp:useBean id="myName" scope="request" class="java.lang.String">
        <% firstName="Sanjay"; %>
</jsp:useBean>
```

JSP Actions

get/setProperty

- getProperty is used in conjunction with useBean to get property values of the bean defined by the useBean action
- Example (getProperty)
  - <jsp:getProperty name=“myBean” property=“firstName” />
  - Name corresponds to the id value in the useBean
  - Property refers to the name of the bean property
- setProperty is used to set bean properties
- Example (setProperty)
  - <jsp:setProperty name=“myBean” property=“firstName” value=“Sanjay”/>
  - Sets the name property of myBean to SanjayExample (setProperty)
  - <jsp:setProperty name=“myBean” property=“firstName” param=“fname”/>
  - Sets the name property of myBean to the request parameter fname
  - <jsp:setProperty name=“myBean” property=“*”>
  - Sets property to the corresponding value in request

JSP Actions

plugIn

- Enables the JSP container to render appropriate HTML (based on the browser type) to:
  - Initiate the download of the Java plugin
  - Execution of the specified applet or bean
- plugIn standard action allows the applet to be embedded in a browser neutral fashion
- Example

```
<jsp: plugin type="applet" code="MyApplet.class" codebase="/">
    <jsp:params>
        <jsp:param name="myParam" value="122"/>
    </jsp:params>
    <jsp:fallback><b>Unable to load applet</b></jsp:fallback>
</jsp:plugin>
```

Thank You!