ID2212 Network Programming with Java
Lecture 10

Enterprise Java Technologies (Part 1 of 3)

Component Architecture.
Overview of Java EE.
Java Servlets

Leif Lindbäck, Vladimir Vlassov

KTH/ICT/SCS

HT 2016

Outline

- **Component Architecture**
- **Overview of the Java Platform Enterprise Edition, Java EE**
- **Java Servlets**

Component Architecture

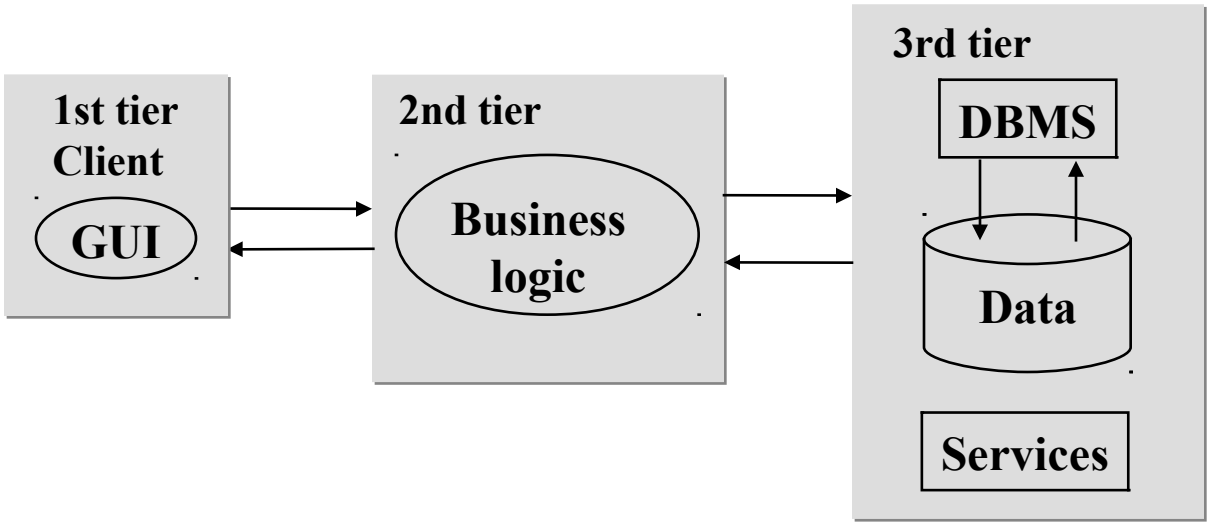
Component Architecture: Why and How

Component Architecture

- Consists of components that live in containers.
- A component is a piece of code, in Java EE usually a class, that solves some of the application's functional requirements.
    - The work done by a component is application specific.
- A container is a framework that solves some of the application's non-functional requirements.
    - The work done by a container is independent of the application. The same container is used for different applications.

The Three-Tier Model

- The *three -tier architecture* allows maintaining state information, improving security, performance, scalability and availability.
- Also gives higher cohesion (bettter separation of concern), since business logic and data storage are separated.
  - Client in the first tier - presentation layer
  - Business logic in 2nd tier - security and personalization of the client
  - System services (and databases) in 3rd tier – services and storage



Why Component Architecture

- **Not using an existing framework means writing new code which means introducing new bugs.**
- **Decrease of the need for in-house expertise**
- **Existing frameworks are thoroughly tested and proven to work well.**
- **It is easy to get help with frequently used frameworks.**

Why Component Architecture (cont'd)

- **Code handling non-functional requirements is very similar in different applications.**
  - **Non-functional requirements include scalability, performance, availability, etc.**
  - **Also, non-functional requirements are difficult to code.**
- **Callback style makes sure all calls to non-functional requirements code are made at the right time.**

How Component Architecture

- *Component*
  - a program building block for an application;
  - presents a manageable, discrete chunk of logic (functionality);
  - implements a set of well-defined interfaces.
  - Examples: pricing component, billing component
- *Container*
  - an application program or a subsystem in which the component lives;
  - Component's context;
  - creates, manages and "glues" components;
  - provides life cycle management, security, deployment, and runtime services for components it contains (component contract).
  - Examples: Web container (for JSF pages and servlets), EJB container (for EJBs)

How Component Architecture (cont'd)

- *Specifications*
  - For components, containers (hosts), and tools (development, deployment)
  - Set of conventions (standards) for
    - Container (Context) Services
    - APIs (classes, interfaces, methods, constructors)
      - Names
      - Semantics

- A well-defined component architecture is a set of standards (specifications) allowing different vendors to write compatible components, containers and tools

Development and Deployment

- ***Development tools***
  - **for developing components. The most used Java EE IDEs are:**
    - **NetBeans (netbeans.org)**
    - **Eclipse (eclipse.org)**
- ***Deployment tools***
  - **for configuring and deploying components. The GlassFish server has the following tools:**
    - **Asant (GlassFish command line interface)**
    - **NetBeans (Integrated with GlassFish)**
    - **Admin console (GlassFish web interface)**

An Application Server

- **Run time environment for component-based applications**
  - **Applications are deployed and run on an application server**
- **Provides containers and services for applications made of components.**
  - **Services: security, management, naming, thread pools, persistence, transactions, etc.**
- **Some Java EE Application Servers:**
  - **GlassFish (Oracle, Java EE reference implementation)**
  - **Payara (open source GlassFish clone)**
  - **WebSphere (IBM)**
  - **WebLogic (Oracle)**
  - **Wildfly (Red Hat)**

Client-Side Components

- **May provide some type of user interface (GUI)**
  - **Get user input and direct to a server**
  - **Present (display) a server response to the user**
  - **Perform client-side (pre- and post-) processing**
  - **Run in browsers or as standalone applications**
- ***HTML pages, using HTTP***
- ***Andriod/iOS apps, using web services***
- ***Standalone programs (Java or other language), using for example web services.***

Server-Side Components

- **Offer services (server-side operations)**
  - **Provide dynamic-content web documents, accessing databases, authentication, transactions, etc.**
- ***Java Servlets, JavaServer Faces (JSF), JavaServer Pages (JSP)***
  - **Provides HTTP interface**
  - **Deployed in a Servlet container**
- ***Enterprise JavaBeans (EJB)***
  - **Provides transaction management and persistence**
  - **Deployed in an EJB container**

Component Architectures, Some Existing Approaches

- **Component Architectures from Microsoft**
  - **.NET, COM, DCOM and COM++**
- **Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG)**
- **Java Enterprise Edition (Java EE) from Oracle**
- **PHP-based servers like Apache and NGINX**
- **Python-based servers like Zope and Django**

Java Platform, Enterprise Edition (Java EE)

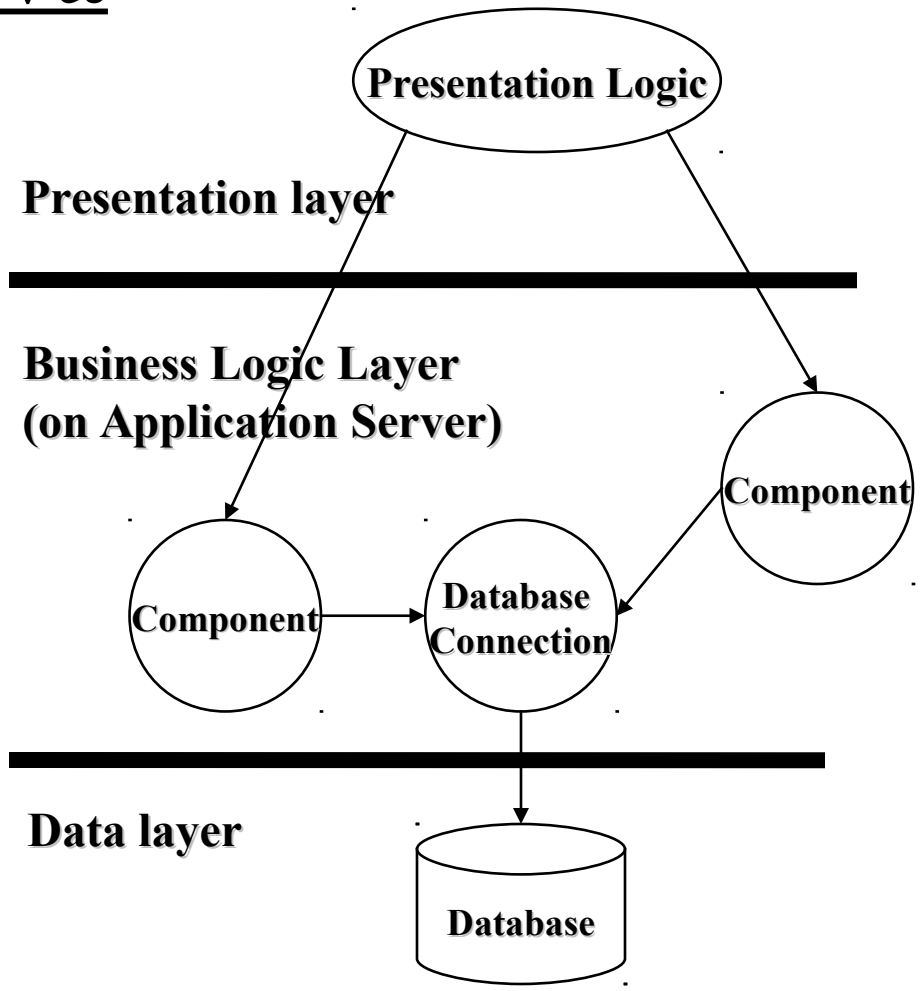
http://www.oracle.com/technetwork/java/javaee/overview/index.html

Some Useful Links

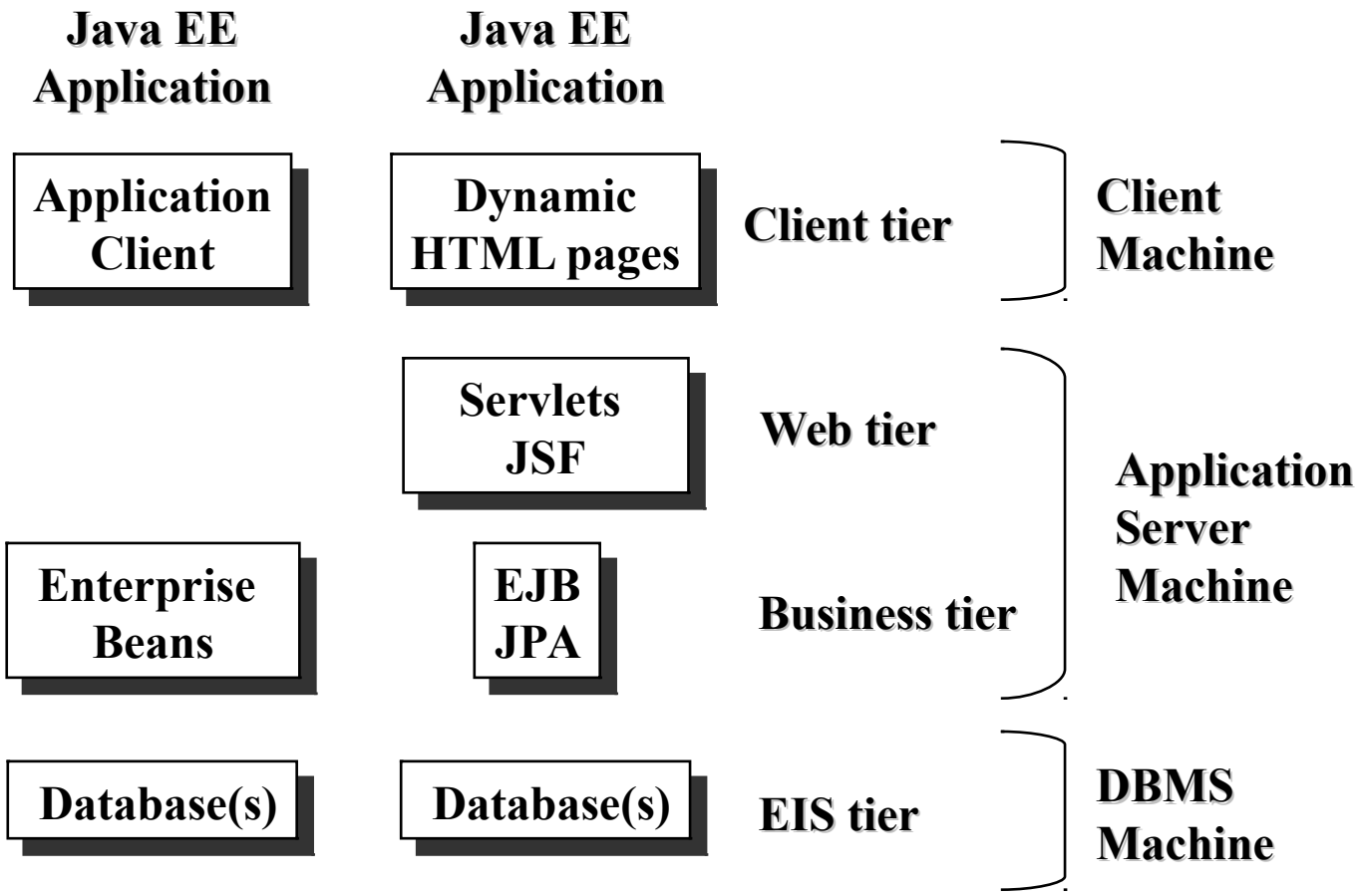
- **Java Platform, Enterprise Edition (Java EE)**
  - `http://www.oracle.com/technetwork/java/javaee/overview/index.html`
- **Java EE Training & Tutorials**
  - `http://www.oracle.com/technetwork/java/javaee/documentation/index.html`
- **The Java EE 7 Tutorial:**
  - `http://download.oracle.com/javaee/7/tutorial/`

Java Platform EE: Enterprise Java

- **Targeted at the development of three-tier architectures:**
  - **Business Logic components are reusable and portable**
  - **Application server must follow the Java EE specification and provide specified set of services**



Multi-Tiered Java EE Applications



The Java EE Technologies

- **Four groups:**
  - **Enterprise Application Technologies**
  - **Web Application Technologies**
  - **Management and Security Technologies**
  - **Web Services Technologies**

Enterprise Application Technologies

- *Enterprise JavaBeans (EJB)*
  - EJBs are the standard building blocks for business logic.
- *Java EE Connector Architecture*
  - An architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems.
- *Java Message Service API (JMS)*
  - A specification of an API for message based communication. To create, send, receive, and read messages.
- *Java Persistence API (JPA)*
  - Provides object-relational mapping.
- *Java Transaction API (JTA)*
  - An API for resource managers and transactional applications. Also used in writing JDBC drivers, EJB containers and hosts.
- *JavaMail*
  - Provides an interface to a mail system.

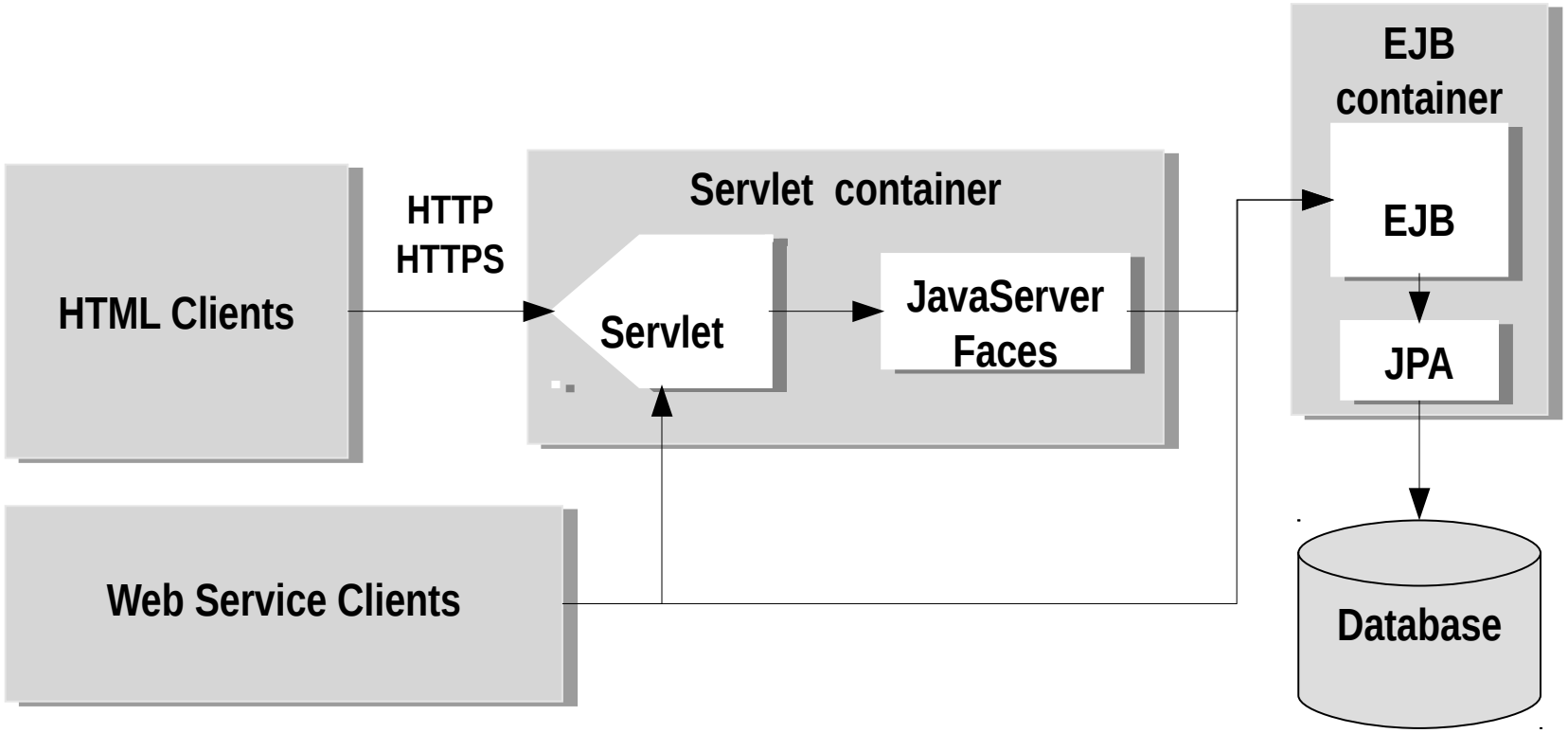
Web Application Technologies

- *Java Servlets*
  - Process requests and construct responses, usually for HTML pages
  - Provides a gateway between Web clients and EJBs
- *JavaServer Faces (JSF)*
  - An API for representing UI components (elements of HTML pages) and managing their state; handling events from components; server-side data validation and conversion.
- *JavaServer Pages Standard Tag Library (JSTL)*
  - Encapsulates core functionality common to many JSF applications, e.g. iterator and conditional tags for handling flow control, tags for manipulating XML documents, internationalization tags, tags for accessing databases using SQL, and commonly used functions.

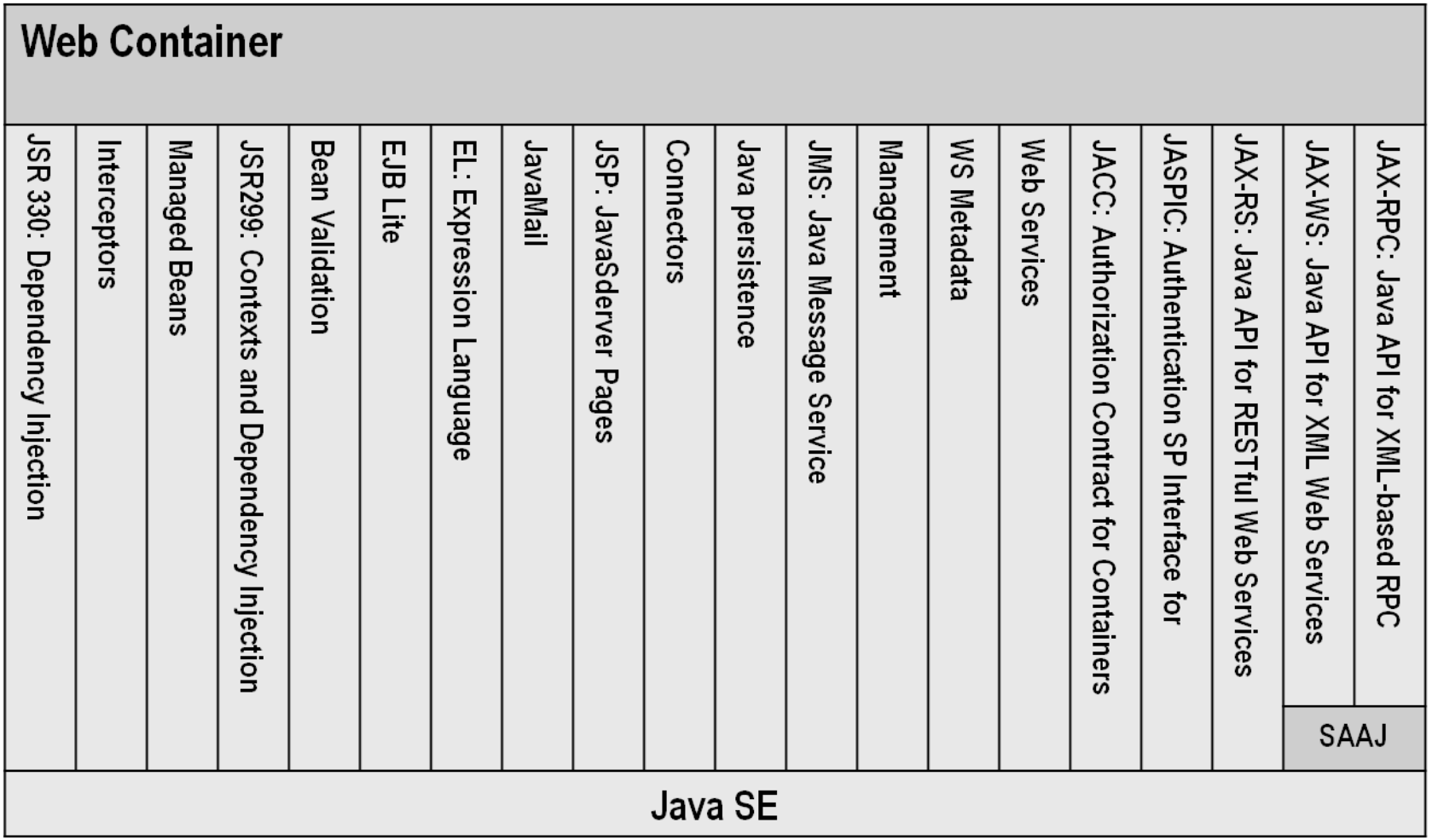
Web Services Technologies

- *Java API for RESTful Services (JAX-RS)*
- *Java API for XML-Based Web Services (JAX-WS)*
- *Java Architecture for XML Binding (JAXB)*
  - **Provides a convenient way to bind an XML schema to a representation in Java code.**
- *SOAP with Attachments API for Java (SAAJ)*
  - **Provides a standard way to send XML documents over the Internet from the Java platform.**

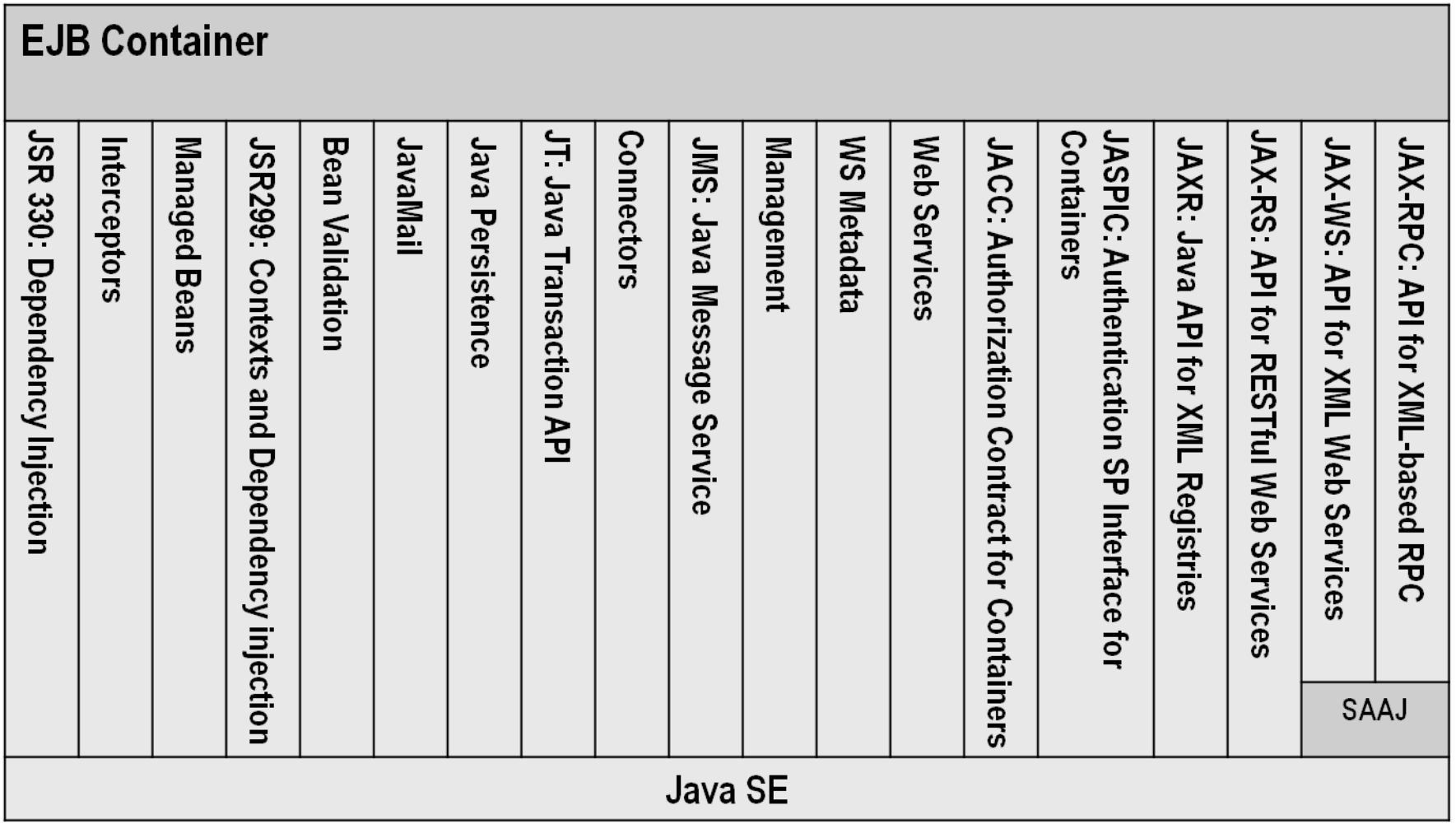
Java EE Containers



Java EE Servlet Container APIs



Java EE EJB Container APIs



A Java EE Application Server

- Hosts components in the middle tier of a three-tier architecture.
- Has containers for server-side components
  - Servlet container for Web components: JSF, Servlets, etc.
  - EJB container for EJBs, etc.
  - Provides services for the components, such as naming, transactions.
  - Connectivity with other tiers (client, databases)
    - Provides access to the components from clients
    - Provides access to the third tier (system services, databases)

Java Servlets

`javax.servlet`

`javax.servlet.http`

home page:
`http://www.oracle.com/technetwork/java/javaee/overview/index.html`

Java Servlets

- A *Java Servlet* is a component in a Java EE servlet container that interacts with clients using a Web protocol, such as HTTP.
  - Executes in server's JVM,
  - Allow providing dynamic Web pages,
  - Main usage is as Controller, calls the model component that shall handle the request and forwards request to the next view,
  - Thus acts as gateway between Web clients and other services, such as databases, EJBs, and JMS,
  - Generates HTTP response.

Java Servlets (cont'd)

- `javax.servlet.Servlet` **interface can be used as a generic interface for any service, not just HTTP.**
  - **Should be a request-response interaction**
- **A** `javax.servlet.http.HttpServlet` **interacts with a client using HTTP protocol.**

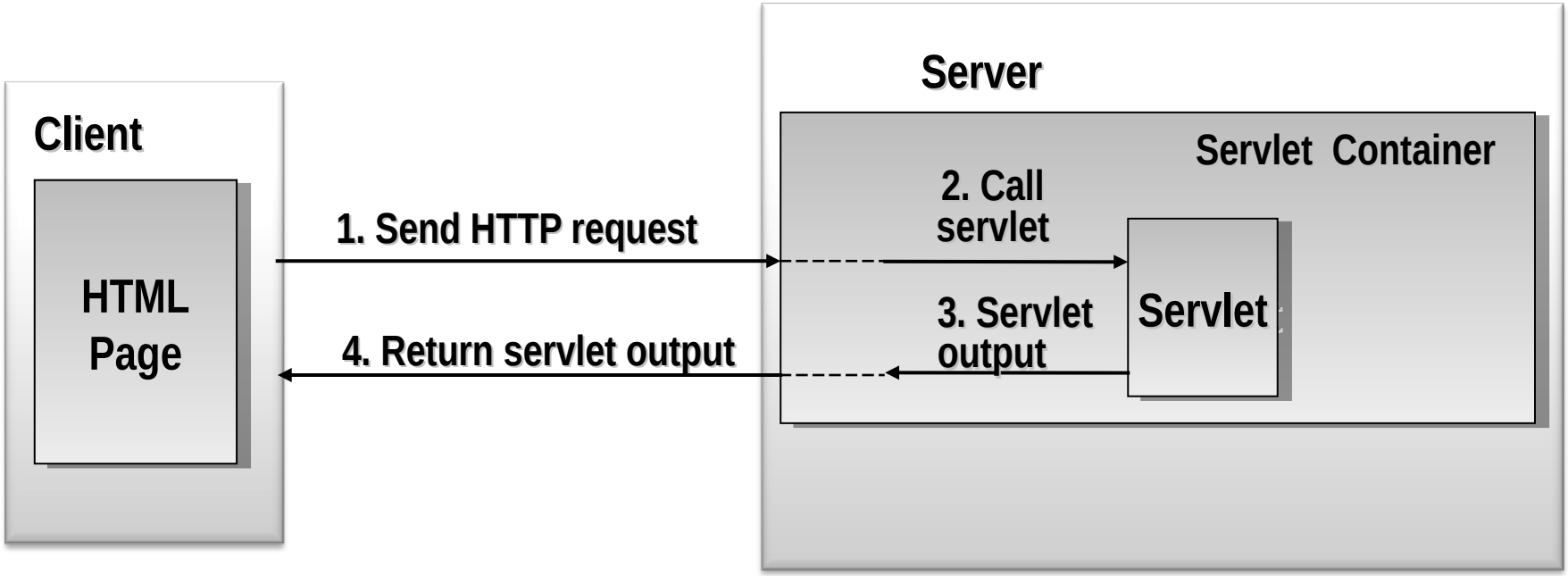
How an HTTP Servlet Executes

- The user submits an HTML form or clicks an HTML link.
- The browser sends the user's query in a GET (POST) request to a servlet pointed to by URL in the request

```
GET /someResource?inputString=3 HTTP/1.1
```

- The Web container instantiates the servlet if it does not exist and invokes the service method, passing request and response objects as parameters.
- The servlet handles the request and writes a result to the output stream of the response object
  - The output stream is directed to the client.

Executing Servlets



Include and Forward

- **A servlet can include an output of another servlet:**

```
RequestDispatcher dispatcher
   = getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null)
   dispatcher.include(request, response);
```

- **A servlet can forward the request (with additional attributes) to another servlet:**

```
request.setAttribute("message", message);
request.setAttribute("account",cashier.readAccount(acctNo));
request.getRequestDispatcher("/account").forward(request,
   response);
```

The Life Cycle of a Servlet

- **Controlled by the container in which the servlet is deployed.**
- **When a request is mapped to a servlet, the container:**
  - **Loads the servlet class (if not loaded yet).**
  - **Creates an instance of the servlet class (if it does not exist) and invokes the `init` method to initialize the servlet instance.**
  - **Invokes the `service` method, passing request and response objects.**
- **If the container needs to remove the servlet, it calls the servlet's `destroy` method.**

Implementing the `Servlet` Interface. Extending the `HTTPServlet` Class

- A servlet class implements the `Servlet` interface either directly, or more commonly, by extending the class `HTTPServlet`
- The `Servlet` interface
  - `init(ServletConfig config)`
    - Initializes the servlet and places it into service.
    - The servlet can get a value of a named init parameter (if any) by name using the `ServletConfig` object.
    - The init parameters for the servlet are specified in the deployment descriptor (i.e., web.xml file).
  - `service(ServletRequest req, ServletResponse res)`
    - Allows the servlet to respond to a request after the servlet has been initialized by the `init` method.
  - `destroy()`
    - Removes the servlet from service after all its threads have exited or a timeout period has passed.

Extending the `HTTPServlet` Class

- Used to create an HTTP servlet suitable for a Web site.
- A subclass of `HttpServlet` must override at least one of the following methods:
  - `doGet` – if the servlet supports HTTP GET requests
  - `doPost` – if the servlet supports POST requests
  - `doPut` – if the servlet supports PUT requests
  - `doDelete` – if the servlet supports DELETE requests
  - `init` and `destroy`
    - to manage resources that are held for the life of the servlet
  - `getServletInfo`
    - provides information about the servlet

Servlet's Request and Response

- A Web container creates and passes to the servlet's service methods (`doGet`, `doPost`, etc.) two objects:
  - An `HTTPServletRequest` object,
  - An `HTTPServletResponse` object.
- The `HTTPServletRequest` interface allows to inspect the request:
  - getters (`getHeader`, `getQueryString`, `getParameter`, etc.),
  - checkers (`isRequestedSessionIdValid`, etc.)
  - methods to pass information between the servlet container and a servlet or between interacting servlets
- The `HTTPServletResponse` interface provides functionality for creating and sending a response (e.g. output stream).

Example: Hello World

```
package se.kth.id2212.lecture10;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloServlet", urlPatterns = {"/HelloServlet"})
public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet HelloServlet at " +
                        request.getContextPath() + "</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Never ever write HTML code in a Servlet!</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

    @Override
    public String getServletInfo() {
        return "This is the HelloWorld Servlet example. ID2212 course. KTH";
    }

}
```

Example, Cont'd

The generated HTML code:

```
<html>
  <head>
    <title>Servlet HelloServlet at /lecture10</title>
  </head>
  <body>
    <h1>Never ever write HTML code in a Servlet!</h1>
  </body>
</html>
```

Monitor and React to Servlet's Life Cycle Events

- **Define listeners to receive and handle life-cycle events issued by the Servlet container (context, session or request events). For example, a context listener:**

```
@WebListener
public final class ContextListener implements
                                    ServletContextListener {
    private ServletContext context = null;
    public void contextInitialized(ServletContextEvent event) {
        context = event.getServletContext();
        try {
            BookDAO bookDB = new BookDAO();
            context.setAttribute("bookDB", bookDB);
        } catch (Exception ex) { e.printStackTrace();}
    }
    public void contextDestroyed(ServletContextEvent event) {
        context = event.getServletContext();
        BookDAO bookDB =
            (BookDAO)context.getAttribute("bookDB");
        bookDB.remove();
        context.removeAttribute("bookDB");
    }
}
```

Servlet Life-Cycle Events and Listeners

Source	Event	Listener Interface
Web context	Initialization and destruction	`javax.servlet.ServletContextListener`
	Attribute added, removed, or replaced	`javax.servlet.ServletContextAttributeListener`
Session	Creation, invalidation, activation, passivation, and timeout	`javax.servlet.http.HttpSessionListener`, `javax.servlet.http.HttpSessionActivationListener`
	Attribute added, removed, or replaced	`javax.servlet.http.HttpSessionAttributeListener`
	A servlet request has started being processed by web components	`javax.servlet.ServletRequestListener`
	Attribute added, removed, or replaced	`javax.servlet.ServletRequestAttributeListener`

Filtering Requests and Responses

- A web resource can be filtered by a chain of filters in a specific order specified on deployment.
- A *filter* is an object that can transform the header and content (or both) of a request or response:
  - Query the request and act accordingly;
  - Block the request-and-response pair from passing any further;
  - Modify the request headers and data;
  - Modify the response headers and data.
- A filter class is defined by implementing the `Filter` interface.
- See the `javax.servlet` package.

Accessing the Web Context

- The context in which web components execute, i.e. the servlet container
- To get the context, call the `getServletContext` method on the servlet.
- The context object implements the `ServletContext` interface.

Accessing the Web Context (cont'd)

- The web context provides methods for accessing:
  - Initialization parameters
  - Resources associated with the web context,
- For example (see Slide 45), retrieving an object attribute (set by the Context listener):

```
public class CatalogServlet extends HttpServlet {
   private BookDBAO bookDB;
   puvblic void init() throws ServletException {
       bookDB =
   (BookDBAO)getServletContext().getAttribute("bookDB");
       if (bookDB == null)
         throw new UnavailableException("Couldn't get database.");
   }
}
```