



# Java Technologies

## Java EE - Introduction

# First of All – Course Information

- The Goal
- The Motivation
- Teaching / Learning
- Bibliography
- Evaluation
  - Lab: problems, personal projects, essays → **easy**
  - Exam: written test / quiz → **hard**

# The Context

- We are in the situation of developing a **complex, large-scale, portable, scalable, reliable, secure, transactional, distributed system**.
- *Who is the customer?* A HUGE bank, a chain of hypermarkets, a Fortune 500 company, etc.
- *What do we know?* **Programming languages** (Java, Groovy, etc.), **various protocols** (TCP, UDP, HTTP, SOAP, etc.), specifications, etc.
- *What do we want?* **A framework** that will make our lives as easy as possible.
- *When do we want it?*



# Java Enterprise Edition (Java EE)

*“The aim of the Java EE platform is to provide developers with a powerful set of APIs while shortening development time, reducing application complexity, and improving application performance.”*

- **Specifications**

Java EE API describes how an enterprise application should be created, what components should contain, etc.

- **Implementations** → Application Servers

- Oracle GlassFish (reference implementation)
- WebLogicServer, JBoss Application Server / WildFly, IBM WebSphere, Apache Tomcat / Geronimo / TomEE, etc.

- **Portability**

# Downloads

- **NetBeans IDE**

- Bundle: *Java EE* or *All*
- Includes Tomcat and GlassFish Application Servers
- Free

- **Eclipse IDE for Java EE Developers**

- An application server must be installed separately
- Free


- **IntelliJ IDEA Ultimate Edition**

- Free 30 day trial / Free for students and teachers

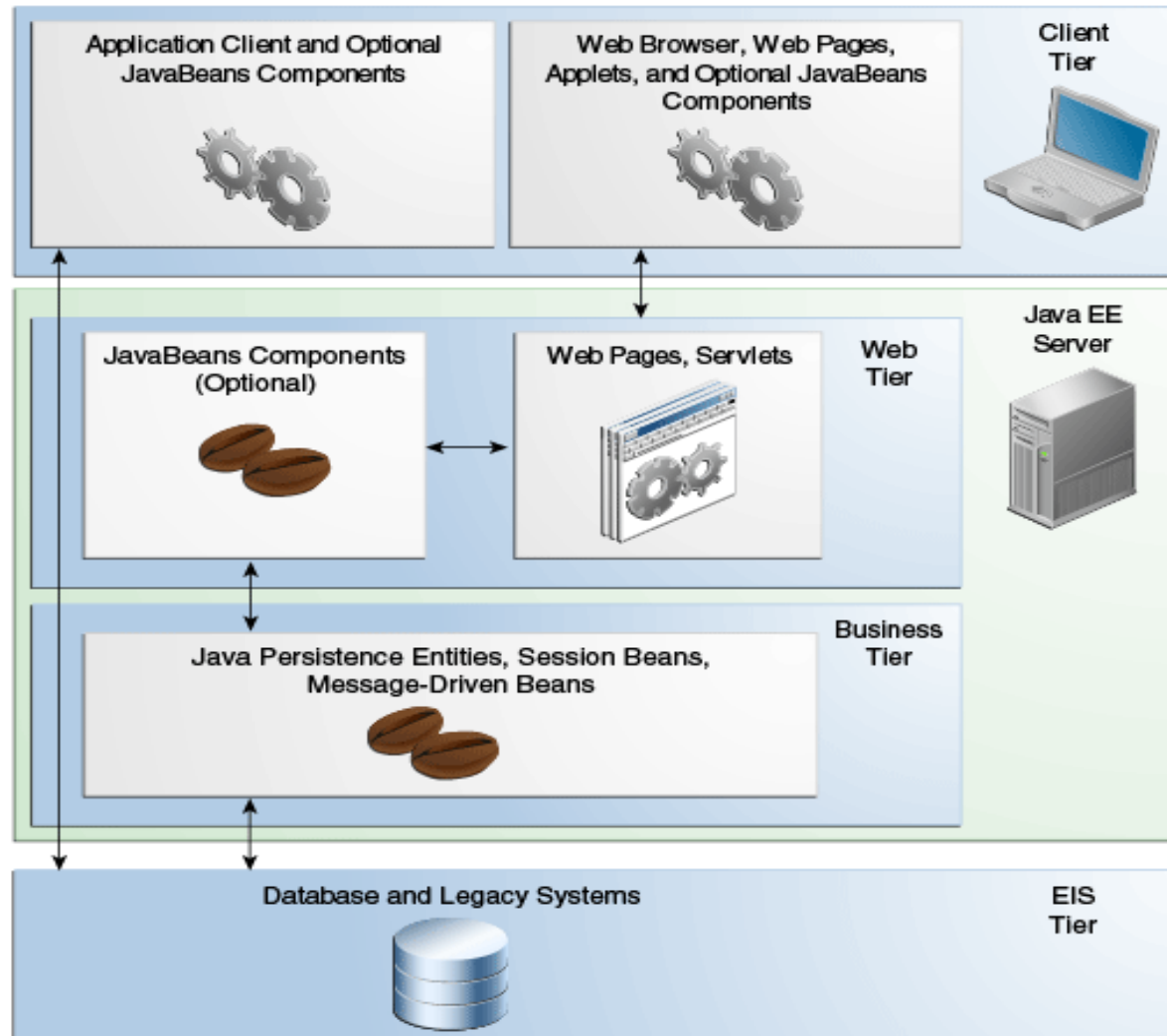
# Java EE Technologies

- **Servlets**
- **JSP** Java Server Pages
- **JSF** Java Server Faces
- **JNDI** Java Naming and Directory Interface
- **JPA** Java Persistence API
- **EJB** Enterprise Java Beans
- **JAX-WS, JAX-RS** Web Services
- **CDI** Context and Dependency Injection
- **JMS, JTA, ...**

# Other EE Alternatives

- Microsoft ASP.NET
  - OK 
- PHP, Python, Perl, Ruby etc. frameworks
  - These are Web Frameworks
  - What about security, transactions, scalability, etc?
- “Do It Yourself” framework
  - What about standards, maintainability, interoperability?

# Distributed Multitiered Applications



Application logic is divided into **components** according to function, and the application components that make up a Java EE application are installed on various machines depending on the tier in the multitiered Java EE environment to which the application component belongs.



# Java EE Components

- Java EE applications are made up of components.
- A Java EE component is a **self-contained functional software unit** that is assembled into an application at a **specific tier** and **communicates with other components**.
- **Client Tier:** application clients, applets
- **Web Tier:** Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) technology components, HTML, XHTML, CSS, etc
- **Business Tier:** EJB components (enterprise beans)
- **Model Tier:** The entity beans or any other beans :)

# Containers

***“Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container.”***

- **Web containers** manages the execution of web pages, servlets, etc. Implemented in most application servers, such as Tomcat: “Apache Tomcat™ is an open source software implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications”
- **Java EE containers** manages the execution of EJB, JMS, etc. and are implemented in the “heavy” Java EE servers, such as Glassfish: “GlassFish is the reference implementation of Java EE and as such supports Enterprise JavaBeans, JPA, JavaServer Faces, JMS, RMI, JavaServer Pages, servlets, etc.”

# Application Lifecycle

- **Develop** the components code and the application deployment descriptors, if necessary.
- **Compile** the application components and helper classes referenced by the components.
- **Package** the application into a deployable unit.  
→ **war, ear**
- **Deploy** the application into dedicated containers, using the application server tools.
- **Access a URL** that references the web application.

# Organizing the Components

- **Source Level: *Java EE blueprints***
- **Build Level**

**\MyApplication**

Web Pages

Resources

**\WEB-INF**

*web.xml*

Configuration files

web.xml = the deployment descriptor file:  
determines how URLs map to components,  
which URLs require authentication, etc.

**\classes**

.class, .properties

**\lib**

.jar

A "web application" is a collection of  
servlets and content installed under a  
specific subset of the server's URL  
namespace such as /MyApplication and  
installed via a .war file

# Sample *web.xml* File

## **<web-app>**

```
<display-name>HelloWorld Application</display-name>
<description>
    This is a complex, large-scale web application
</description>

<session-timeout>30</session-timeout>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>

<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>examples.Hello</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

## **</web-app>**

Some of these metadata may be specified using annotations. Sometimes, the web.xml file may not be required.

# Bibliography

- The Java EE Tutorial
- The Java EE API
- ...