Politecnico
di Milano

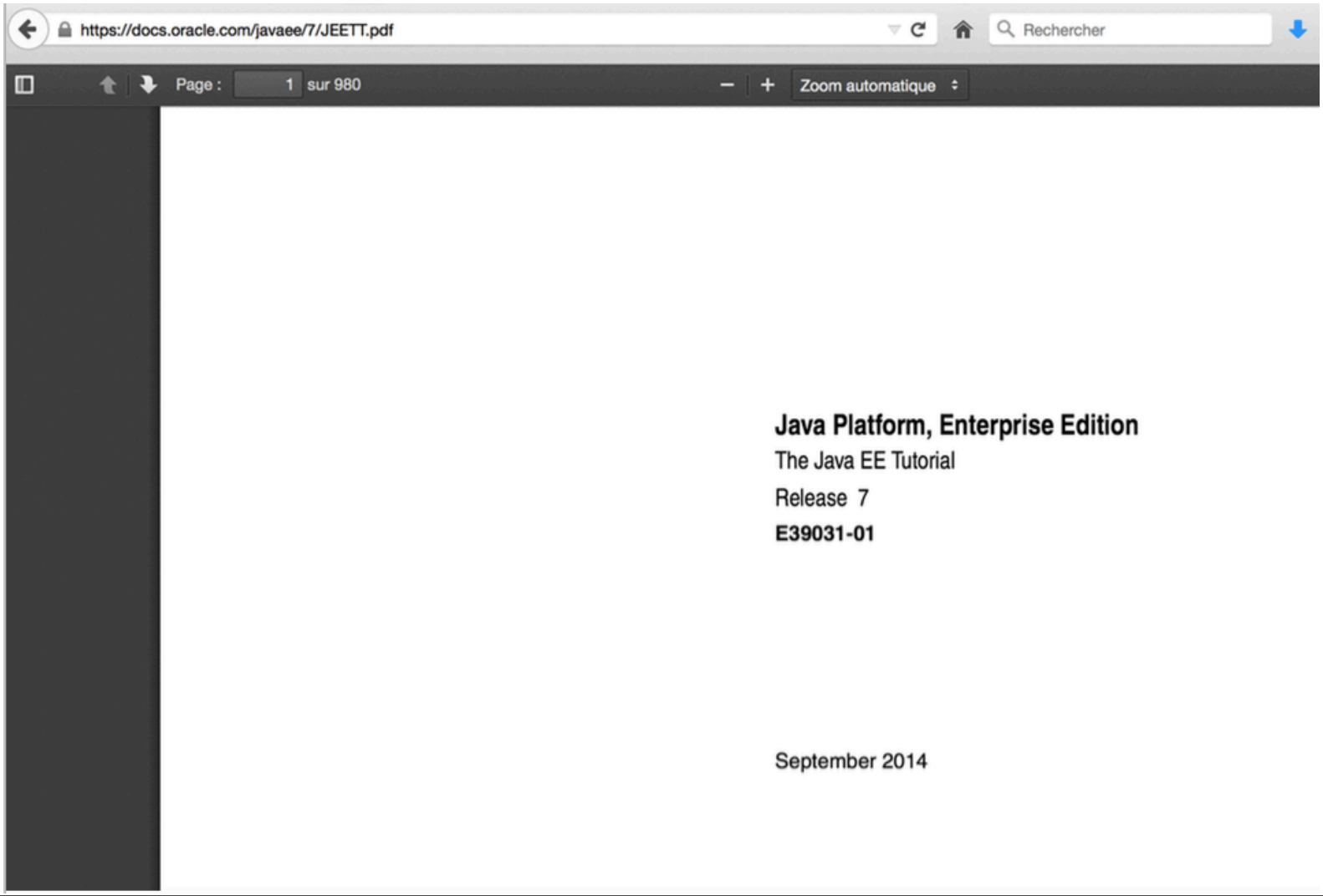
Java Enterprise Edition 7 (JEE 7)
Overview

Michele Guerriero
michele.guerriero@polimi.it

JEE & Testing Sessions

- Sessions 1 & 2: Java EE introduction and main services
- Session 3: software testing
- Session 4: Java EE advanced topics
- Java EE Lab: apply what you have seen in sessions 1 & 2

Reference Documentation



Focus of This Lecture

- What is Java EE?
- The Java EE platform development model
  - a multitier architecture
  - components and containers
  - web components overview
  - business components overview
  - resource management

Java?

- A **language**: general-purpose, concurrent, strongly typed, class-based object-oriented
- A **platform** (JRE): Java Virtual Machine (JVM), API and libraries
- **Programming tools** (JDK): compiler, debugger and monitoring tools

Java Platforms

- **Software Platform**: a software supporting the development and/or execution of applications.

- Java Platforms
  - **Java SE:** desktop applications, small number of users. Basic set of APIs for: file system access, networking, security, database access, GUI
  - **Java EE:** enterprise applications, lots of users, lots of requirements. Built on top of Java SE.
  - Others (Java Card, Java ME, Java FX)

Enterprise Applications

- Software applications for organizations: companies, schools, governments, etc.
- Distributing services to multiple clients simultaneously.
- Satisfying many and evolving functional requirements.
- Satisfying several non-functional requirements: reliability, performance, security, scalability, availability, extensibility, interoperability.
- …a lot of coding using Java SE (try to think at how you would do that…)

The Java EE Platform

- Main objective: provide developers with a powerful set of APIs to speed up the development process, reduce the application complexity, and improve the application performance
- Developers can focus on essential aspects of a business applications (business logic, user interfaces)
- We will refer to Jave EE **7**

How it Works?

- Extending the Java SE API
- **Abstract** from low level problems such as:
  - Transaction management
  - State management
  - Multi-threading
  - Connection pool management
- Reduce development time
  - **convention over configuration**
  - **annotation** for configuration

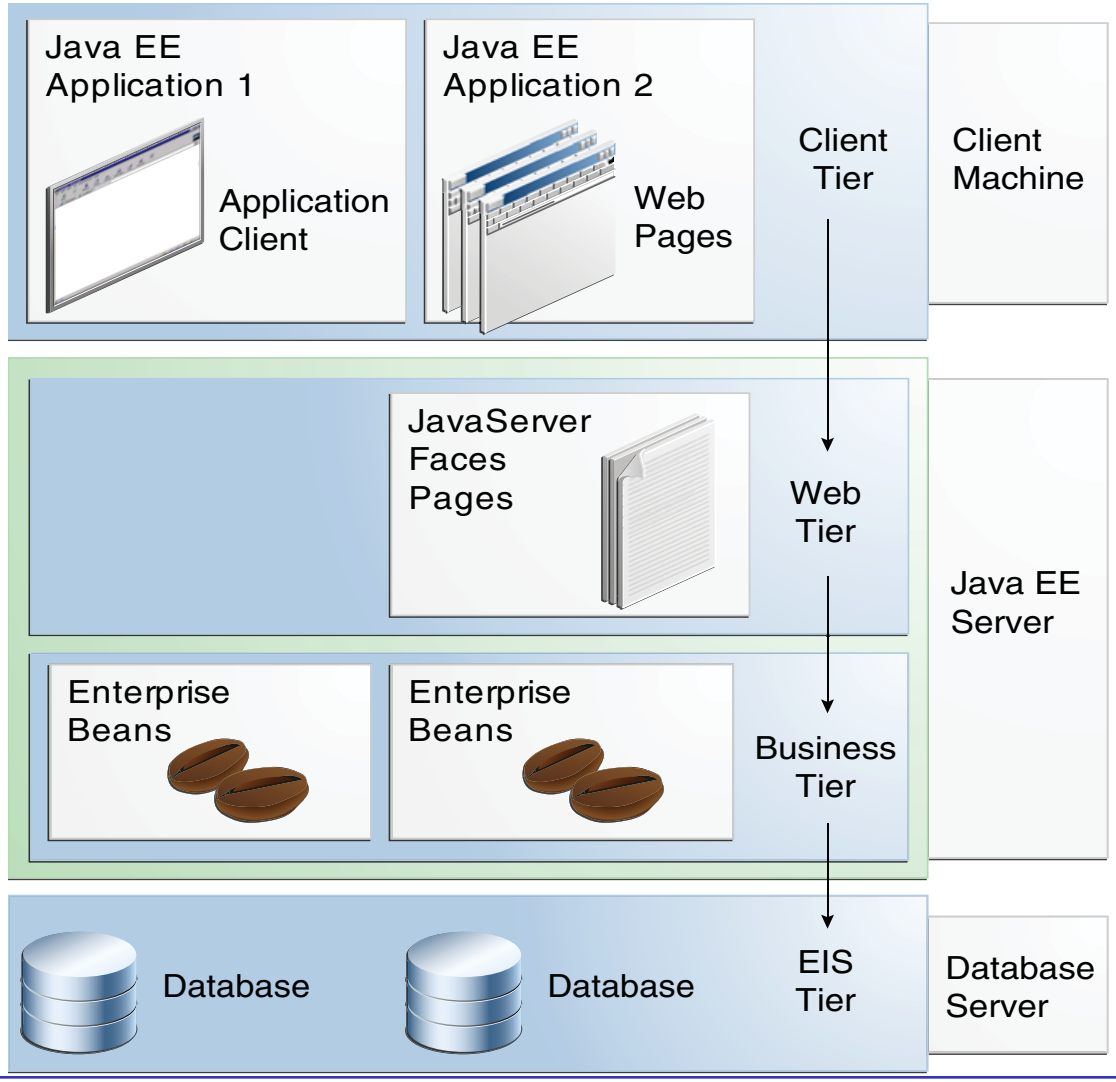
How it Works?

- Reduce application complexity
  - **multitiered** model
  - **containers / components**
  - **dependency injection**
- Provide mechanisms to support **interoperability** with non Java systems (e.g., JAX-WS, JAX-RS)
- Follow the philosophy **Write Once, Run Anywhere**
  - Define a contract that makes it possible to use platforms from various vendors (e.g., Glassfish, Wildfly, TomEE)

The Approach

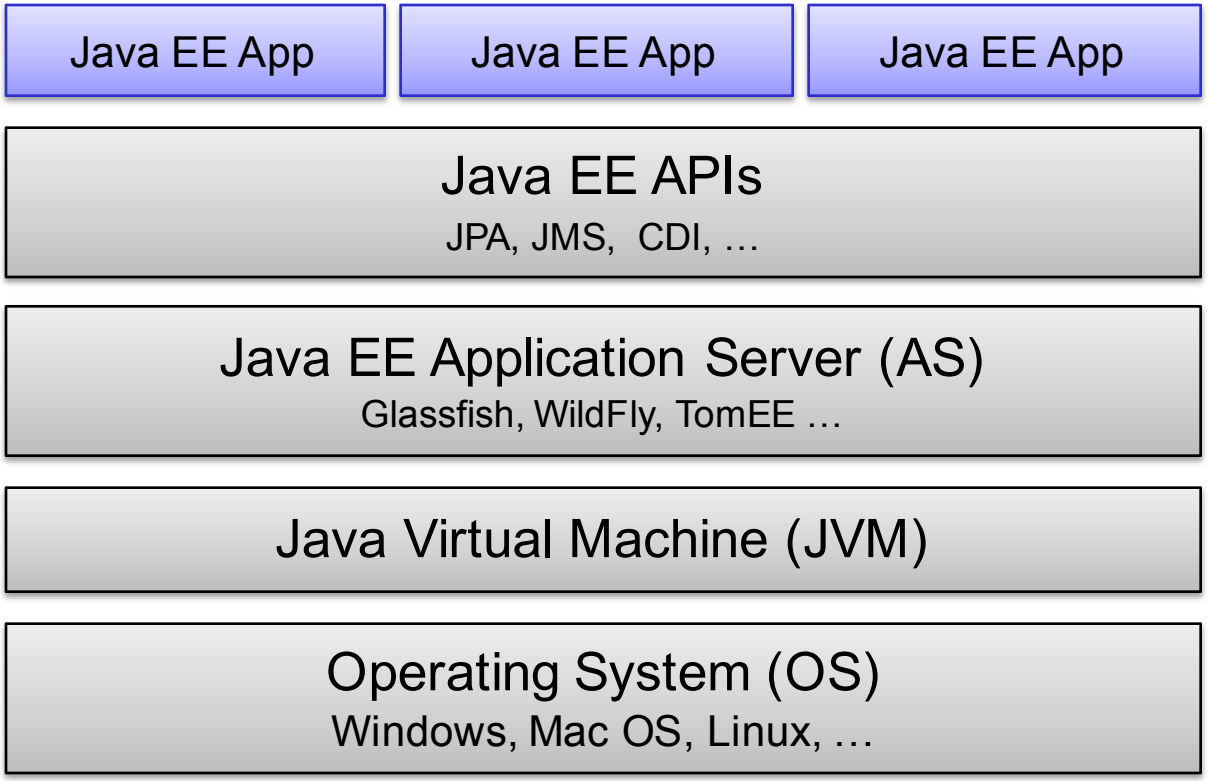
- A **multitier architectural model**
  - the client-tier running on the client machine
  - the web-tier, running on the Java EE Server
  - the business-tier running on the Java EE Server
  - the enterprise information system (EIS) consisting of databases or other external applications
- A predefined set of **components** (belonging to and hosted on the different tiers) that offer various functions
- A set of **containers** that are the interface between components and low-level platform-specific functionalities

The architectural model of Java EE



From Jendrock et al. *The Java EE 7 Tutorial*

The Java EE platform stack



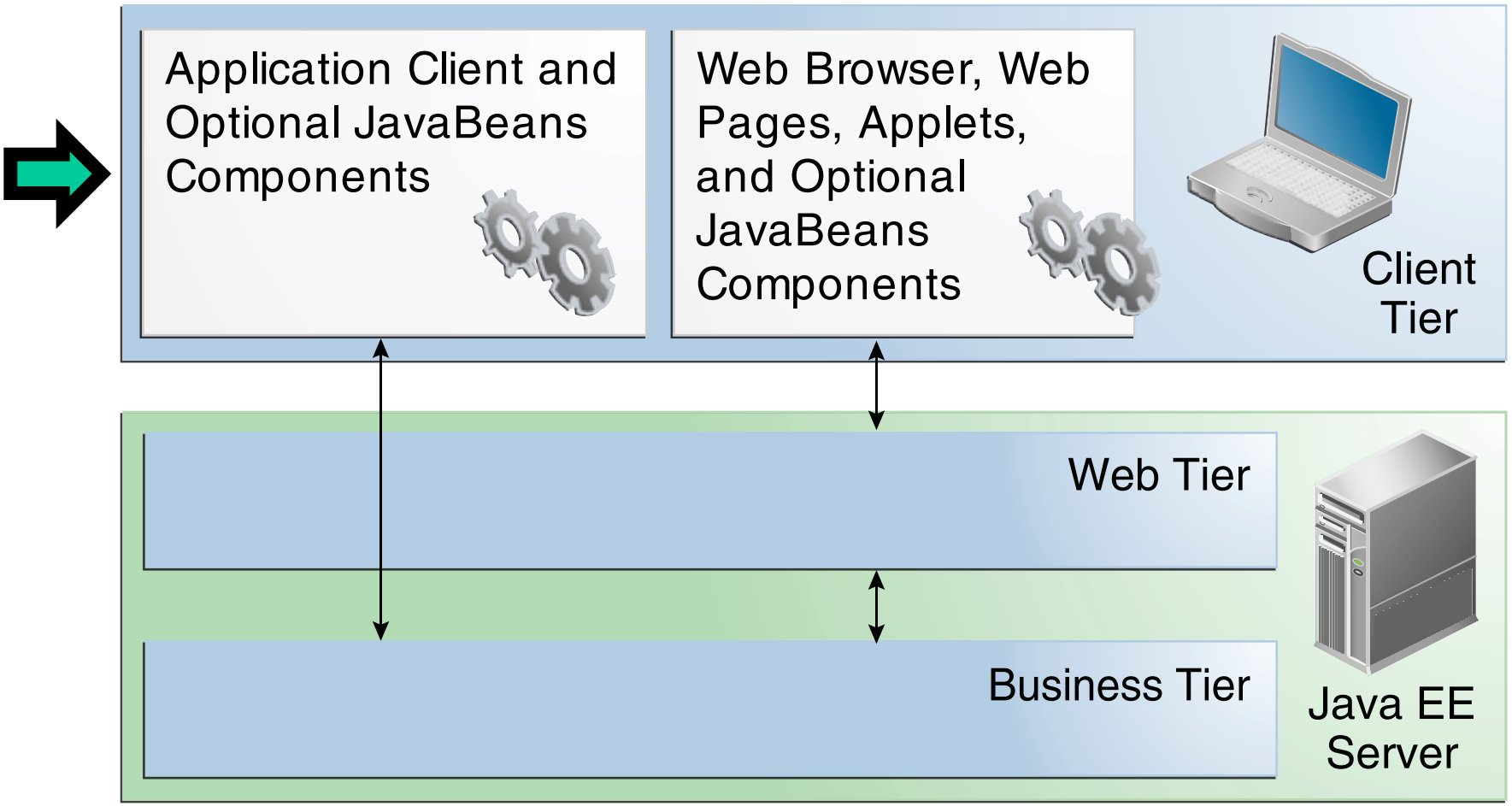
Components and component models: general definitions

- ***Component:*** software element that
  - encapsulates the reusable implementation of some functionality
  - can be composed without modifications
  - respects the constraints imposed by *a component model*
- ***Component model***: is a combination of standards governing
  - how to build individual components
  - how to organize (compose) components to build an application
  - how component can communicate and interact among each other

Java EE components

- A Java EE component
  - is a self-contained software functional unit
  - can be assembled in a Java EE application
  - is able to communicate with other components
- Available Java EE components
  - running on the **client**
    - Web clients, application clients, and applets
  - running on the **server**
    - web components: Java Servlet, JavaServer Pages (JSP), Java Server Faces (JSF)
    - business components: Enterprise JavaBean (EJB), JPA Entities

Java EE Clients



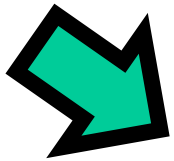
From Jendrock et al. *The Java EE 7 Tutorial*

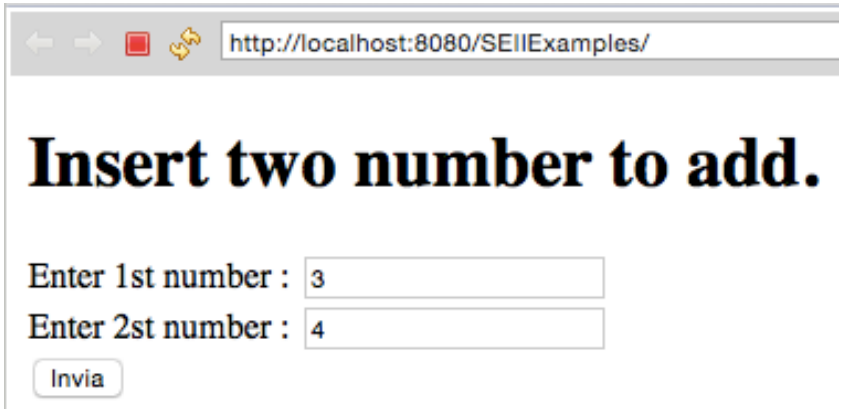
Java EE Clients (1)

- A **Web client** consists of:
  - **dynamic** web pages (e.g. HTML, XHTML) **generated by** web components (see later…)
  - a **Web browser** that renders pages received by the server

HTML Example

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
    <h1>Insert two number to add.</h1>
    <form action="AddServlet" method="post">
    Enter 1st number : <input type="text" name="t1"> <br>
    Enter 2st number : <input type="text" name="t2"> <br>
    <input type="submit">
    </form><br>
</body>
</html>
```





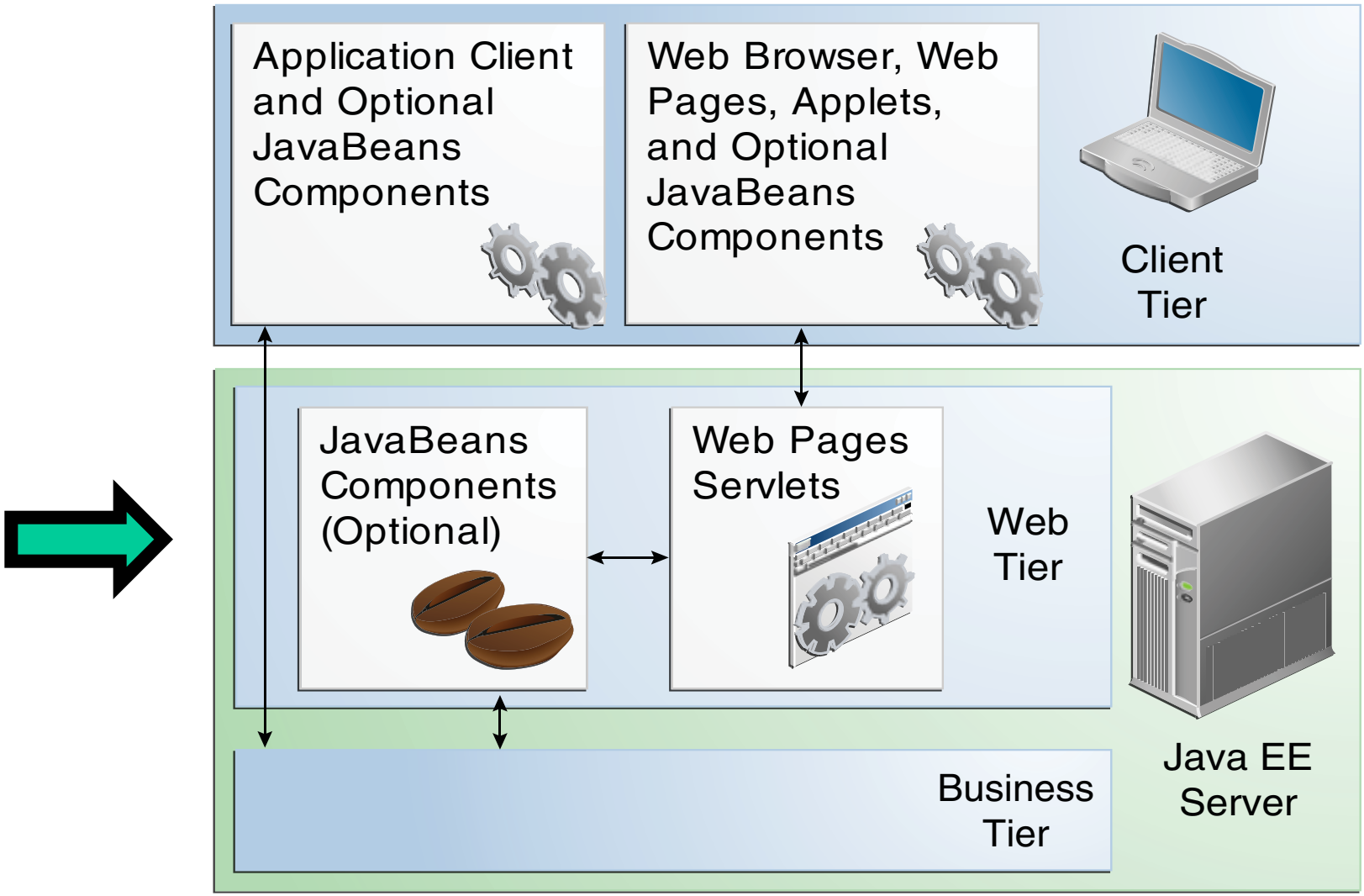
Java EE Clients (2)

- **Application clients**
  - run directly on the client machine
  - useful when the user interface is particularly rich and difficult to develop with a markup language
  - usually interact directly with the business tier
  - it can be written in any programming language enabling the Java EE platform to interoperate with legacy systems and non-java languages
- Note: web client vs. application client
  - The web client simplifies the distribution, deployment, and update of the system
  - The application client allows to build a more complex and powerful client

Java EE Clients (3)

- **Applets**
  - small clients written in Java
  - downloaded from the web tier as part of a web page
  - executed on the JVM installed in the web browser
  - require Java plug-in enabled on the web browser and a proper security policy file
- Note: usually the web client solution is preferable to the applet
  - less requirements on the client configuration (e.g. applets require security policies)
  - possibility to delegate the development of web pages to a web design group not expert of Java

Web Components



From Jendrock et al. *The Java EE 7 Tutorial*

Web Components - Servlets

- **Servlets**: Java classes that
  - run on the web server
  - accessed by means of a **request-response** programming model
  - JEE defines **HTTP-specific servlet** (see the *javax.servlet.http* package)
  - **dynamically** process requests and generate responses
  - can manage **sessions** to relate multiple requests from the same client
  - are executed within a **web container**
  - the lifecycle is controlled by the web container where they are deployed

HttpServlet Example

```
public class AddServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public AddServlet() {
        super();
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        response.setContentType("text/html");
        response.setBufferSize(8192);

        PrintWriter out = response.getWriter();

        String first = request.getParameter("t1");

        String second = request.getParameter("t2");

        if(!StringUtils.isNumeric(first) || !StringUtils.isNumeric(second)){
            RequestDispatcher dispatcher =
                    getServletContext().getRequestDispatcher("/error");

            dispatcher.include(request, response);

        } else{
            int i = Integer.parseInt(first);
            int j = Integer.parseInt(second);
            int k = i+j;

            out.println("<html><body><h1>Here is the sum!</h1>"
                    + "<br>The sum is: " + k + "<br>"
                    + "</body></html>"
                    + "<form action=\"AddServlet\" method=\"get\">"
                    + "Do another sum"
                    + "<input type=\"submit\">"
                    + "</form><br></body>");

        }
    }
}
```

Called when an HTTP POST request is issued toward the URL mapping this servlet

An output stream to send data to the clients.

Transferring Control to Another Web Component

hello2 example from Java EE 7 Tutorial

Another Presentation Technology – JSP Pages

- **JSP pages**: text-based documents that
  - allow more natural approach in creating static content (HTML or XML tags)
  - JSP tags to generate dynamic
  - finally compiled as servlets

JSP Version of “AddServlet” (1)

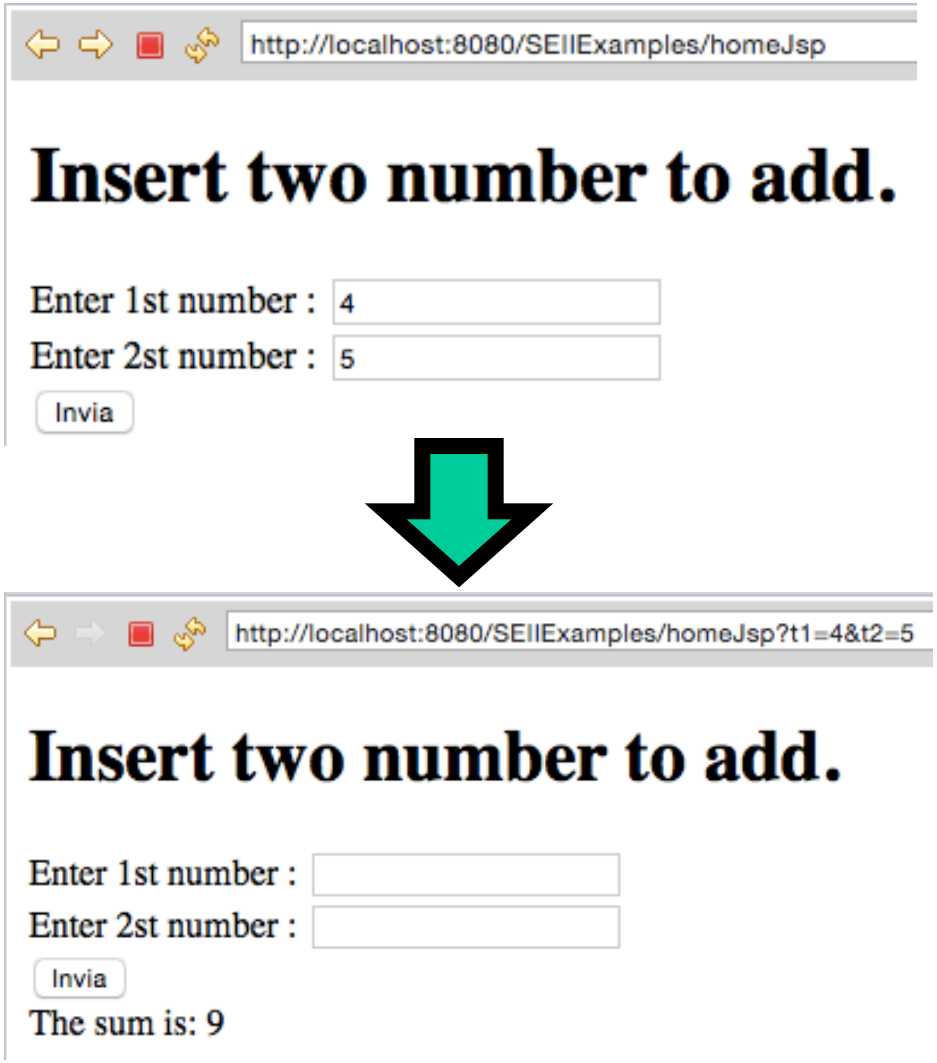

```jsp
<%@ page import="org.apache.commons.lang3.StringUtils"%>
<!DOCTYPE html>
<%!private static Integer k;%>
<%
    String first = request.getParameter("t1");

    String second = request.getParameter("t2");

    if (StringUtils.isNumeric(first) && StringUtils.isNumeric(second)) {
        int i = Integer.parseInt(first);
        int j = Integer.parseInt(second);
        k = i + j;
    }
%>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
    <h1>Insert two number to add.</h1>
    <form>
        Enter 1st number : <input type="text" name="t1"> <br>
        Enter 2st number : <input type="text" name="t2"> <br> <input type="submit">
        <%
            if (k != null) {
                out.println("<br> The sum is: " + k + "<br>");
            }
        %>
    </form>
    <br>
</body>
</html>
```

Specific tags to manipulate Java code while writing the HTML page.

JSP Version of “AddServlet” (2)



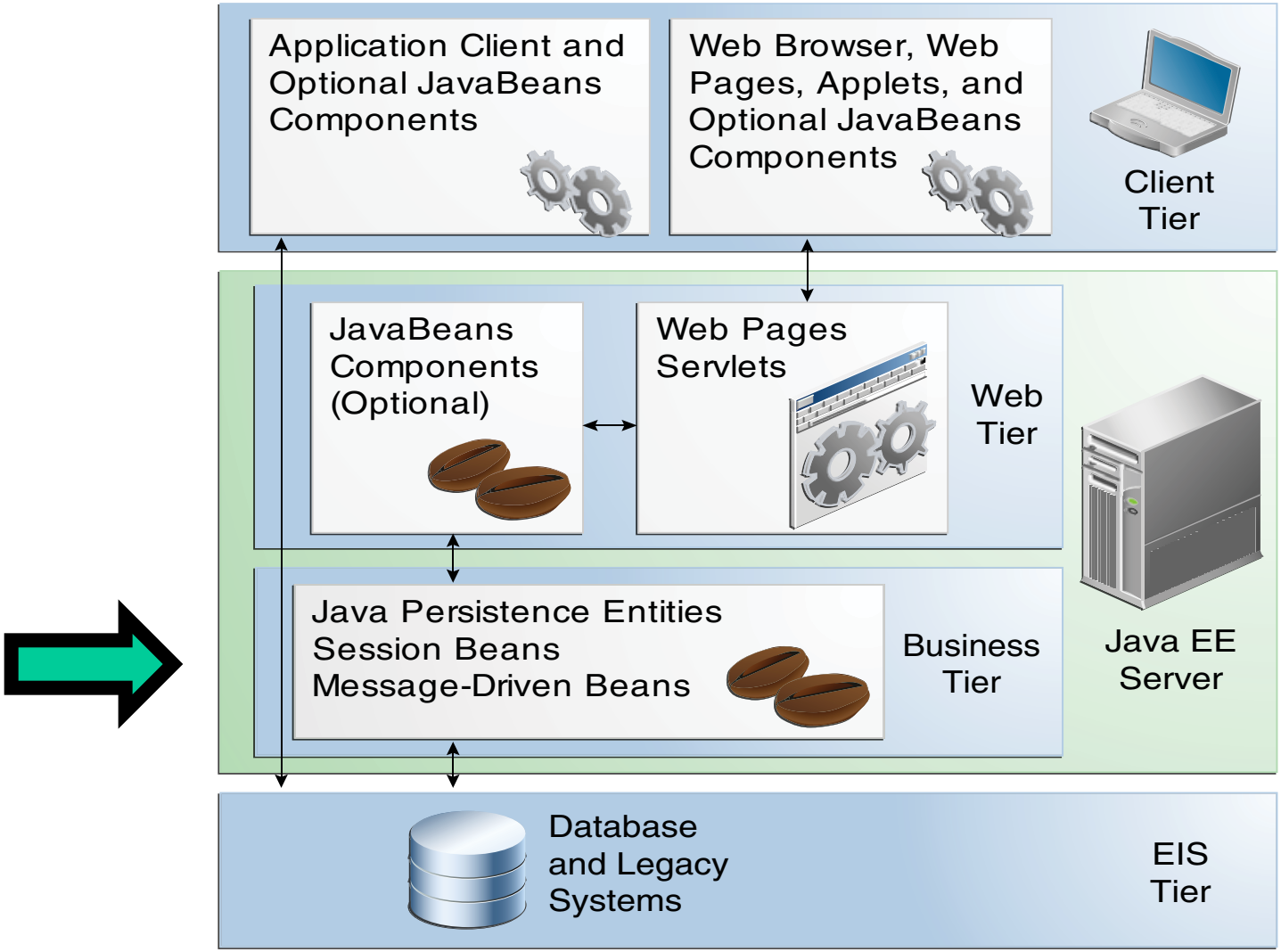
web.xml: Deployment Descriptor

- Provides information about *configuration and deployment of Web components*

```
<web-app>
  <display-name>SEIIExamples</display-name>
  <welcome-file-list>
    <welcome-file>home.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>error</servlet-name>
    <jsp-file>/error.html</jsp-file>
  </servlet>
  <servlet-mapping>
    <servlet-name>error</servlet-name>
    <url-pattern>/error</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>AddServlet</servlet-name>
    <servlet-class>it.polimi.seiiexample.servlets.AddServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AddServlet</servlet-name>
    <url-pattern>/AddServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Can be implicitly injected by the @WebServlet("/AddServlet") annotation

Business Components



From Jendrock et al. *The Java EE 7 Tutorial*

Business Tier - Enterprise Beans

- Enterprise Beans implement the logic that addresses the needs of a specific business domain (banking, retail, finance)
  - The developer can focus on the actual business logic
  - Portability across Java EE applications
  - Useful when the application must be scalable

- Types
  - **Session beans**
  - **Message-Driven Beans**

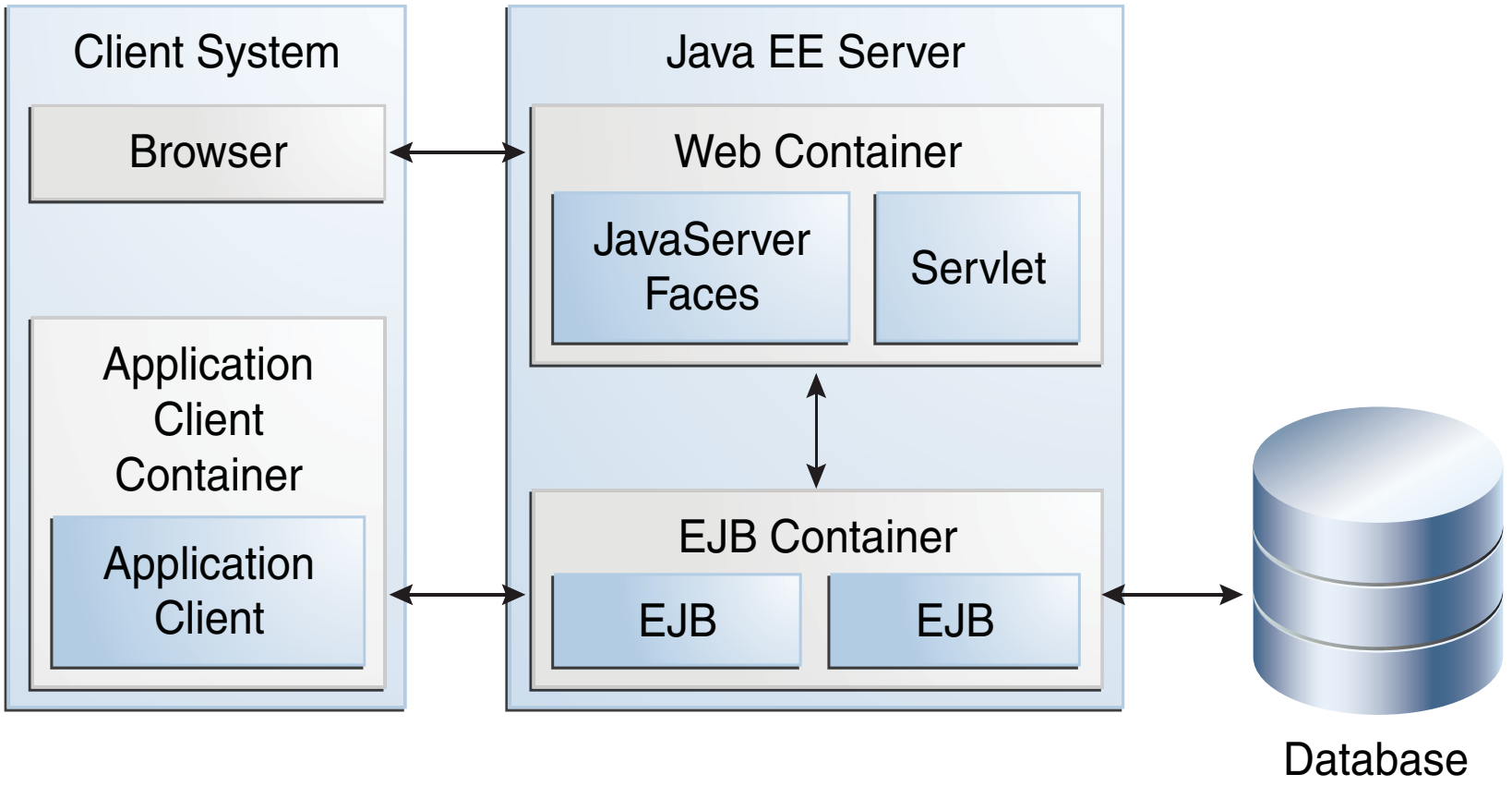
Java EE Containers

- The Java EE containers offer services to manage
  - the component lifecycle
    - Deployment
    - Activation/instantiation
    - Configuration
    - Execution scope
    - Termination
  - transactions
  - security
  - lookup of other components
  - communication between components
  - ….
- They are the interface between components and the low level functionalities (primary services) offered by the platform
- Allow the developer to focus on the application problem, not on the "details"

Java EE Container Types

- EJB container:
  - Manages the execution of enterprise beans
  - Responsible for system-level services (e.g. transaction management, security authorization)
- Web container:
  - Manages the execution of presentation components (e.g. web pages, servlets)
- Application client container:
  - For application client components.
- Applet container:
  - For apples: consists of a web browser and a Java plug-in running on the client

Java EE containers types



From Jendrock et al. *The Java EE 7 Tutorial*

EJB Container and Resource Management (1)

- **Goal**: allow application resource access to a high number of clients
- **Solution 1**: EJB available in more than one instance
  - one per active client
- **Advantage**: every client sees a dedicated service
- **Problems**:
  - Needs to instantiate and destroy a large number of components
  - Often the interaction with the client is very short
  - Thus, the instantiation and destroy time generates a significant overhead

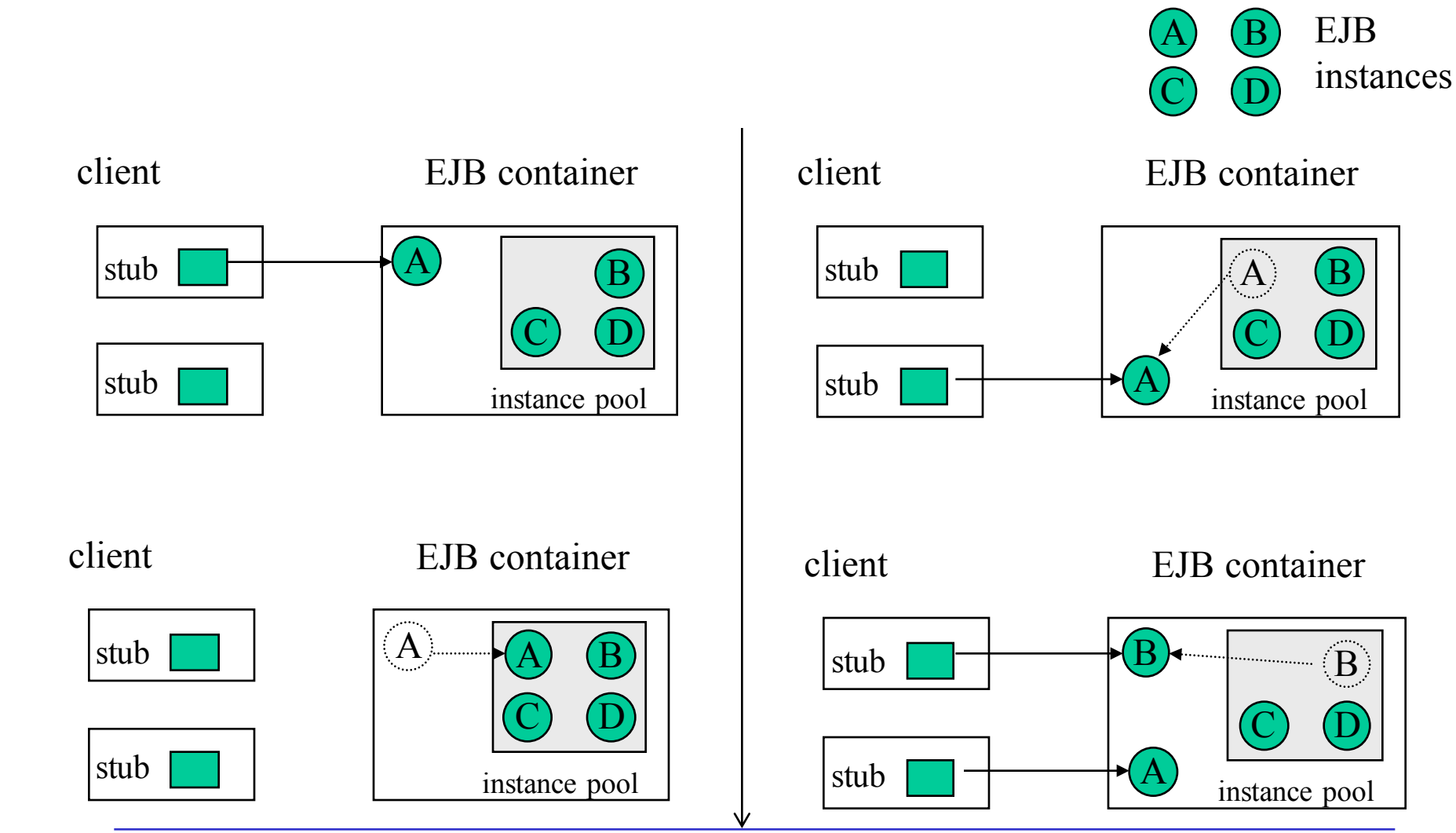
EJB Container and Resource Management (2)

- **Actual Solution**: instance pooling
  - The client sends a request to the EJB container EJB (never directly to the EJB)
  - The container selects an EJB instance from a pool and sends the request to it
  - After satisfying the request, the instance goes back to the pool

Session Beans

- **Stateless session bean**:
  - do not maintain any notion of state between different method invocation - stateless session beans inside a pool are equivalent
- **Stateful session bean**:
  - The state of an object consists of the values of its instance variables. They represent the state of a client session. When the client/session terminates, the bean is no longer associated with that client
- **Singleton**:
  - Is instantiated once per application and exists for the whole application lifecycle. A single bean instance is shared across and **concurrently** accessed by clients

Instance Pooling and Stateless Session Beans



Stateless EJB Example

```java
@Stateless
public class AdditionBean {

    private int i,j,k;

    public int getI() {
        return i;
    }

    public void setI(int i) {
        this.i = i;
    }

    public int getJ() {
        return j;
    }

    public void setJ(int j) {
        this.j = j;
    }

    public int getK() {
        return k;
    }

    public void add(){
        k=i+j;
    }
}
```

Bean declaration

```java
public class AddServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB
    private AdditionBean addBean;

    public AddServlet() {
        super();
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {

        response.setContentType("text/html");
        response.setBufferSize(8192);

        PrintWriter out = response.getWriter();
        String first = request.getParameter("t1");
        String second = request.getParameter("t2");

        if(!StringUtils.isNumeric(first) || !StringUtils.isNumeric(second)){
            RequestDispatcher dispatcher =
                    getServletContext().getRequestDispatcher("/error");
            dispatcher.include(request, response);
        } else{
            int i = Integer.parseInt(first);
            int j = Integer.parseInt(second);
            addBean.setI(i);
            addBean.setJ(j);
            addBean.add();

            out.println("<html><body><h1>Here is the sum!</h1>"
                    + "<br>The sum is: " + addBean.getK() + "<br>"
                    + "</body></html>"
                    + "<form action=\"AddServlet\" method=\"get\">"
                    + "Do another sum"
                    + "<input type=\"submit\">"
                    + "</form><br></body>");
        }
    }
}
```

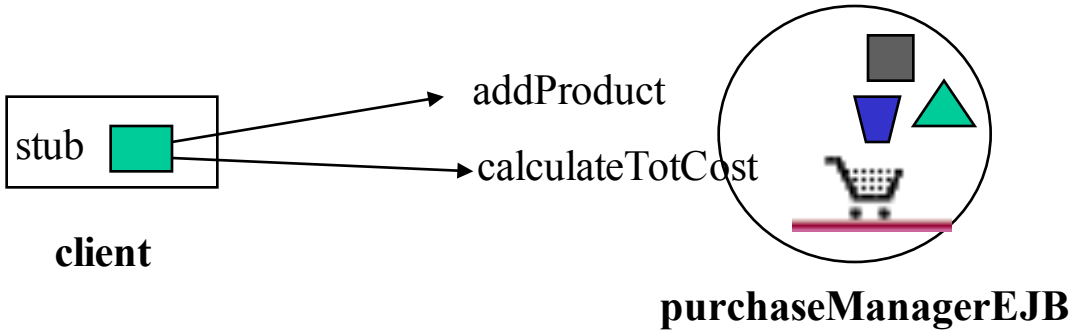
Dependency injection

Stateless EJB Example

- **Dependency injection** is the simplest way of obtaining an enterprise bean reference.
- Clients that run within a Java EE server, such as servlets or other enterprise beans, support dependency injection using the ***javax.ejb.EJB*** annotation.
- Applications that run outside a Java EE server, such as Java SE applications, must perform an explicit lookup. *Java Naming and Directory Interface* JNDI supports a global syntax for identifying Java EE components to simplify this explicit lookup.
- We will look at JNDI in the las session…

Stateful Session Bean and Passivation/Activation mechanisms

- Stateful session bean: owns a *conversational state*
- Example:

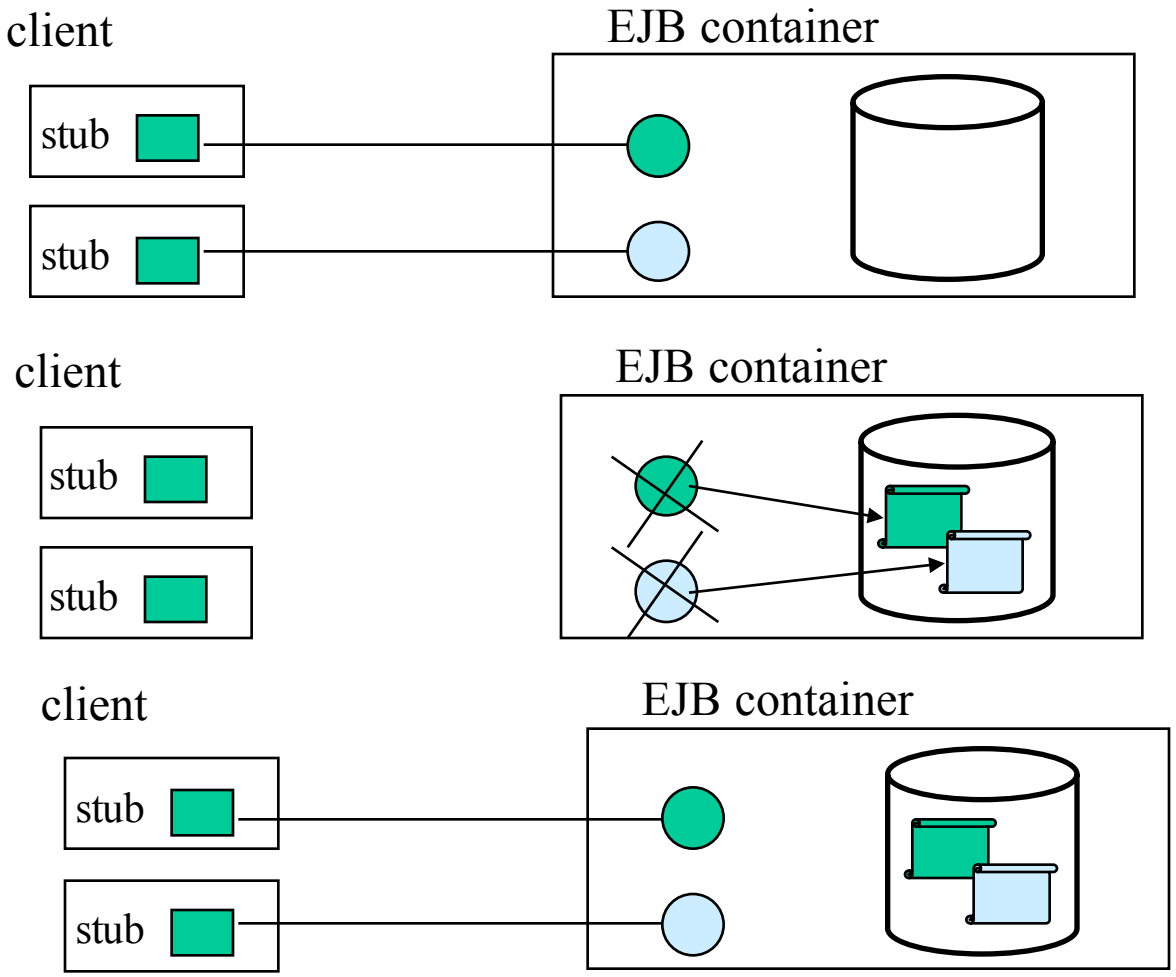


- The same EJB has to serve a client till the end of a conversation
- Stateful session bean do not participate to the istance pooling
- They are passivated when “idle” and activated when needed

Passivation and Activation (1)

- Passivation
  - The container serializes and stores the conversational state of the EJB
  - The EJB is removed from the memory
- Activation
  - It is executed when a method is invoked on the EJB
  - The container creates a new instance of the EJB
  - Assign to this instance the state of the passivated instance
- Passivation and activation is transparent to clients

Passivation and Activation (2)



Passivation and Activation (3)

- Developers can define, as part of stateful EJBs, methods that are called
  - Right before passivation, e.g., to close a file
  - Right after activation, e.g., to assign values to transient variables
- The container rebuilds at the activation the remote references to the other beans

Stateful EJB Example


```
@Stateful
public class ListElement{

    List<Integer> values = new ArrayList<Integer>();

    public void addElement(int i) {
        values.add(i);
    }

    public void removeElement(int i) {
        values.remove(new Integer(i));
    }

    public List<Integer> getElements() {
        return values;
    }

}
```

Using the ListElement Stateful EJB


```
<%@ page import="it.polimi.seiiexamples.beans.ListElement"%>
<%@ page import="javax.naming.InitialContext"%>
<%@ page import="java.util.List"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%!private static ListElement values;

    public void jspInit() {
        try {
            InitialContext ic = new InitialContext();
            values = (ListElement) ic.lookup("java:global/EJBStateful/ListElement");

        } catch (Exception e) {
            System.out.println(e);
        }
    }%>
<%
    if (request.getParameter("addNum") != null) {
        values.addElement(Integer.parseInt(request.getParameter("t1")));
    }

    if (request.getParameter("remNum") != null) {
        values.removeElement(Integer.parseInt(request.getParameter("t1")));
    }
%>
```

Using the ListElement Stateful EJB


```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Addition list elements JSP page</title>
</head>
<body>
    <h1>Welcome</h1>
    <form name="abc" method="post">
        <input type="text" name="t1"><br> <input type="submit"
            value="Add" name="addNum"><br> <input type="submit"
            value="Remove" name="remNum"><br>
        <%
            if (values != null) {
                List<Integer> nums = values.getElements();
                for (int value : nums) {
                    out.println("<br>" + value);
                }

                out.println("<br> Size=" + nums.size());
            }
        %>
    </form>
</body>
</html>
```

Persistence Service

- Persistence allows the entities to be **durable**
  - Their methods and public attributes can be accessed at any time
  - Information is not lost in case of a system failure
- Relational DBMSs are very common
- Java EE defines a mechanism to **map objects (entities) to relations** in a relational DBMS (**O/R mapping**) using Java Persistence API (**JPA**)

Japa Persistence API

- **JPA entities**
  - Object-oriented view of the database
  - A class represents a table in a database
  - Each class instance (object) is a row in the table
  - Class attributes are mapped to columns
  - Entities are automatically materialized in and retrieved from the database at runtime by JPA implementations (e.g. Hibernate, EclipseLink)

Persistence – JPA Entity Example


```
@Entity
@Table(name = "Users")
public class User {

    private static final String PERSISTENCE_UNIT_NAME = "UserDatabase";
    private static final EntityManagerFactory factory = Persistence.
            createEntityManagerFactory(PERSISTENCE_UNIT_NAME);

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private String id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String password;
}
```

Primary key

Field validation

persistence.xml

As used when creating the EntityManagerFactory!

```
<persistence>
    <persistence-unit name="UserDatabase" transaction-type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>JDBCRESOURCE</jta-data-source>
        <class>it.polimi.seiiexamples.entities.User</class>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://localhost:3306/example_database" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="root" />
        </properties>
    </persistence-unit>
</persistence>
```

Persistence – the EntityManager

- The **EntityManager** instance
  - associated to a persistence context (set of entities)
  - creates and removes entities, finds entities by their primary keys, and run queries on entities
  - Container-managed
    - shared and injected by the container
    - container manages its lifecycle and transactions context
  - Application-managed
    - Lifecycle as well as transactions must be managed by the application

Persistence – the EntityManager Example


```
public static Boolean validate(String userName, String password) {
    boolean result = false;
    EntityManager em = factory.createEntityManager();
    Query q = em.createQuery("SELECT u FROM User u WHERE u.Login = :login"
            + " AND u.Password = :pass");
    q.setParameter("login", userName);
    q.setParameter("pass", password);
    try {
        User user = (User) q.getSingleResult();
        if (userName.equalsIgnoreCase(user.getName())
                && password.equals(user.getPassword())) {
            result = true;
        }
    } catch (NoResultException e) {
        return null;
    } finally {
          em.close();
    }
    return result;
}
```

Persistence – Mappings

- Entities lifecycle states:
  - Managed (or attached): in sync with DB
  - Transient: EntityManager doesn't know about its existence yet
  - Detached: not in sync with DB
- Entities are created (or removed) by means of explicit *persist* (or *remove)* invocation or as a result of a cascading operation
- One entity can map on more than one table (multiple joins and updates)
- … and it can maintain relationships (with specific multiplicity) with other entities
  - One to one
  - One to many
  - Many to many