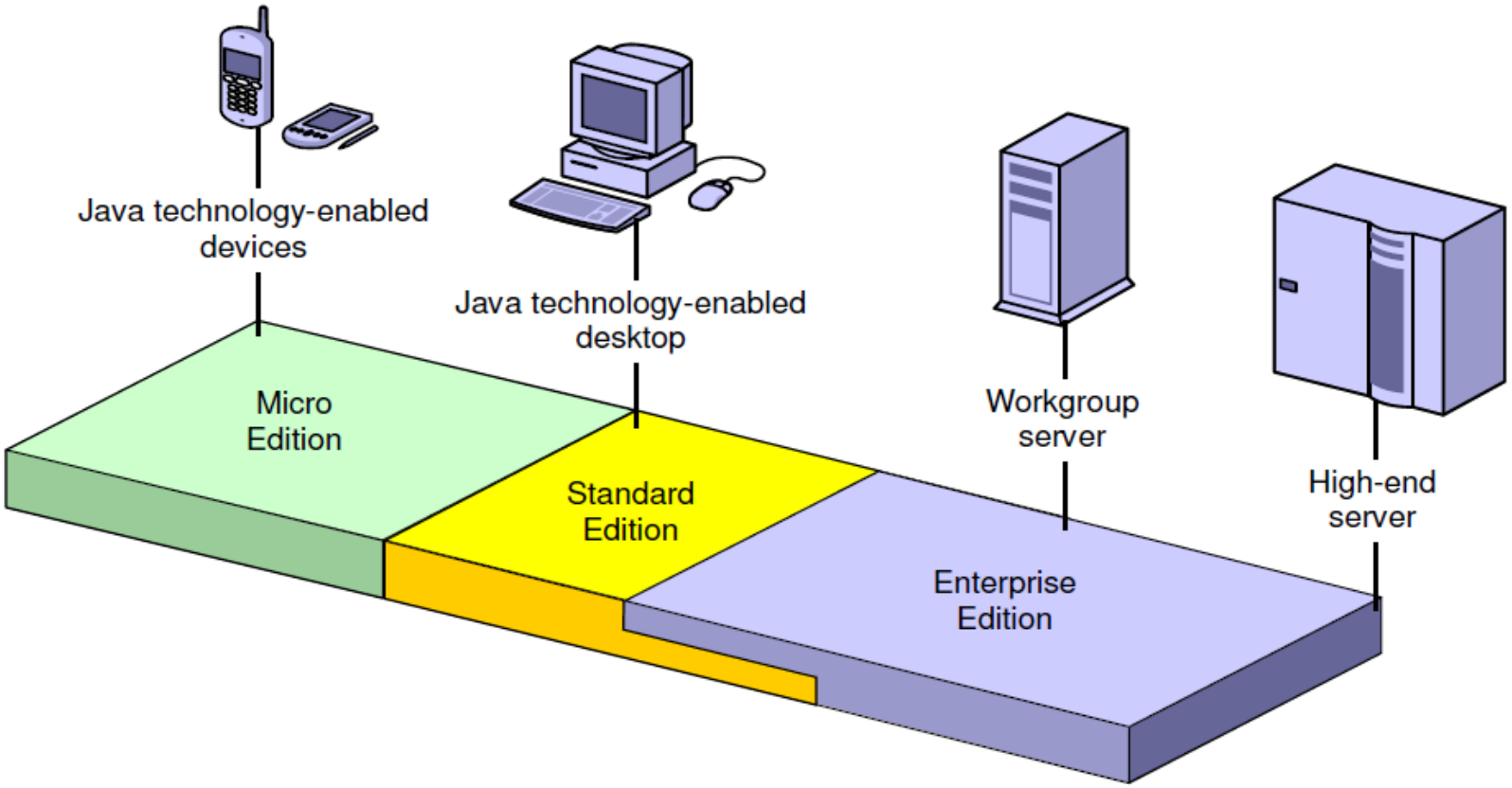
Placing the Java™ EE Model in Context

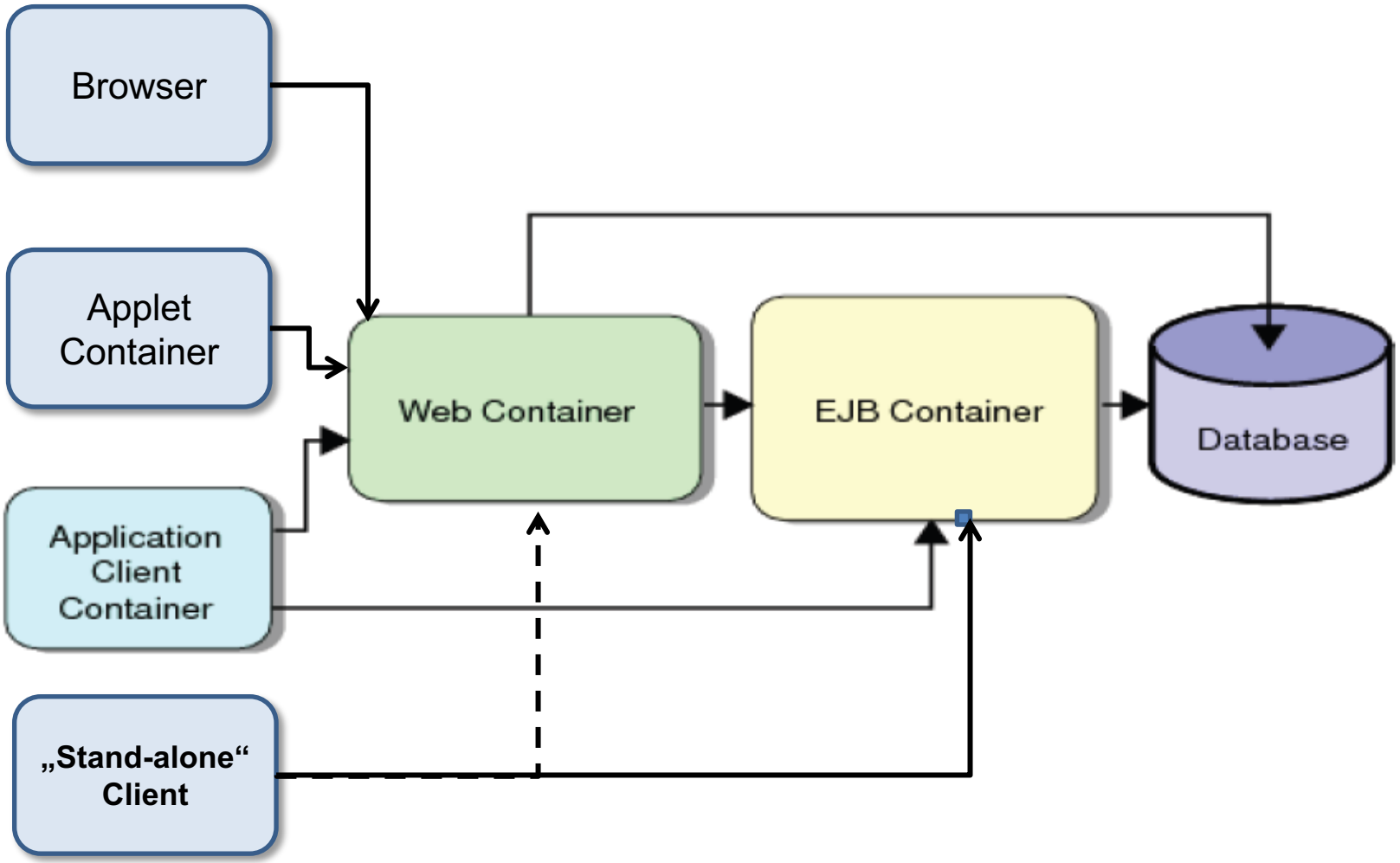
Modul 1 - Java EE Model
Java Technology Platforms



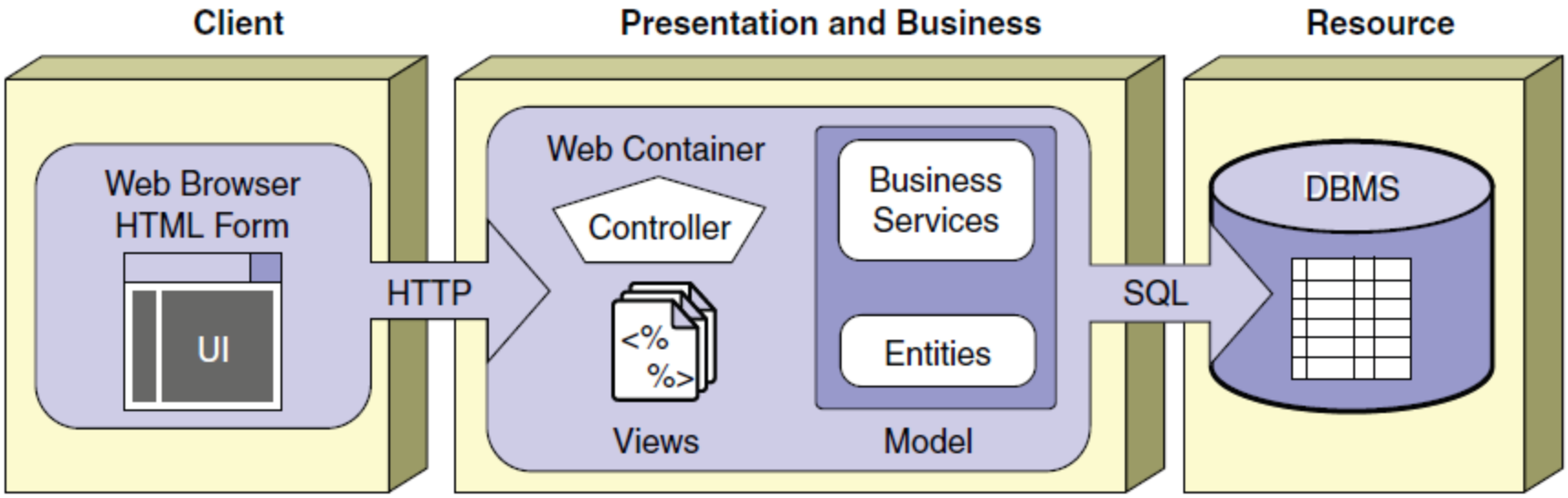
Modul 1 - Java EE Model
JEE APIs

Quelle: www.trivadis.com

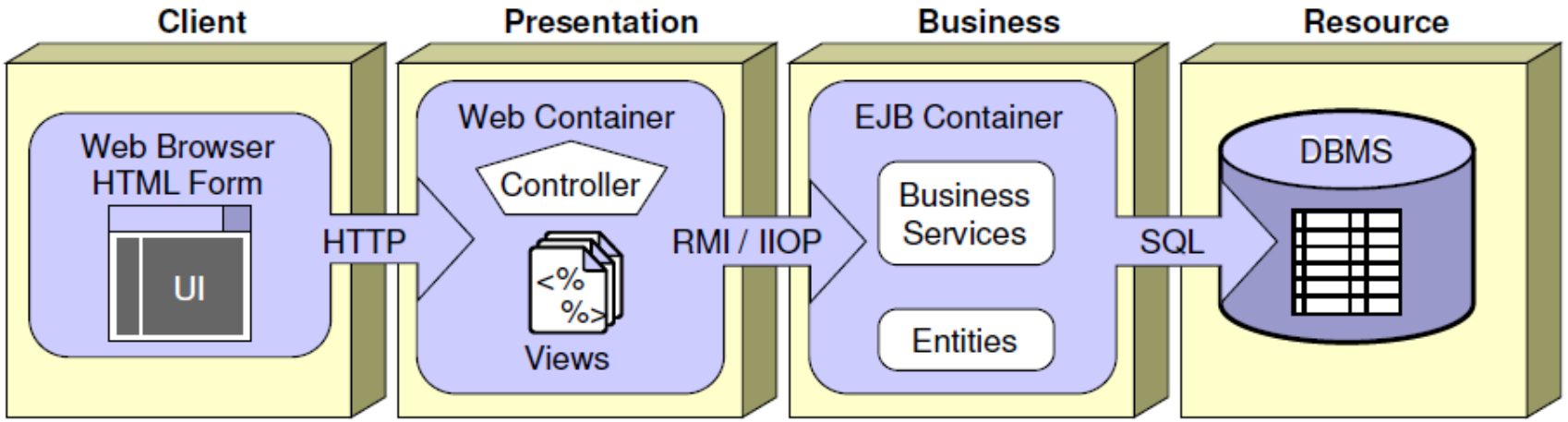
Modul 2 - Java EE Model
Java EE Component Containers



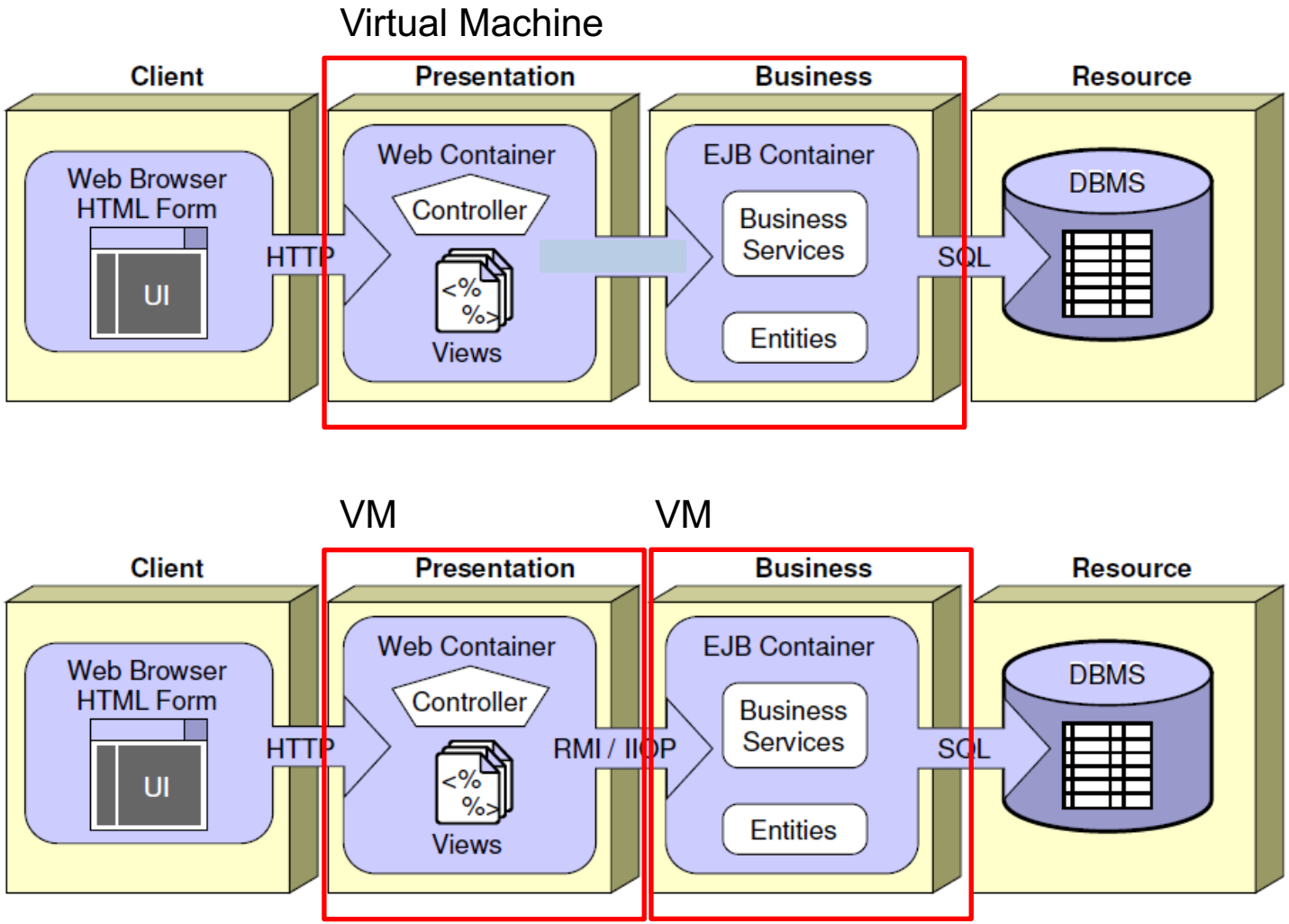
Modul 1 - Java EE Model
Java EE Web-Centric Architecture



Modul 1 - Java EE Model
Java EE EJB Component-Centric Architecture



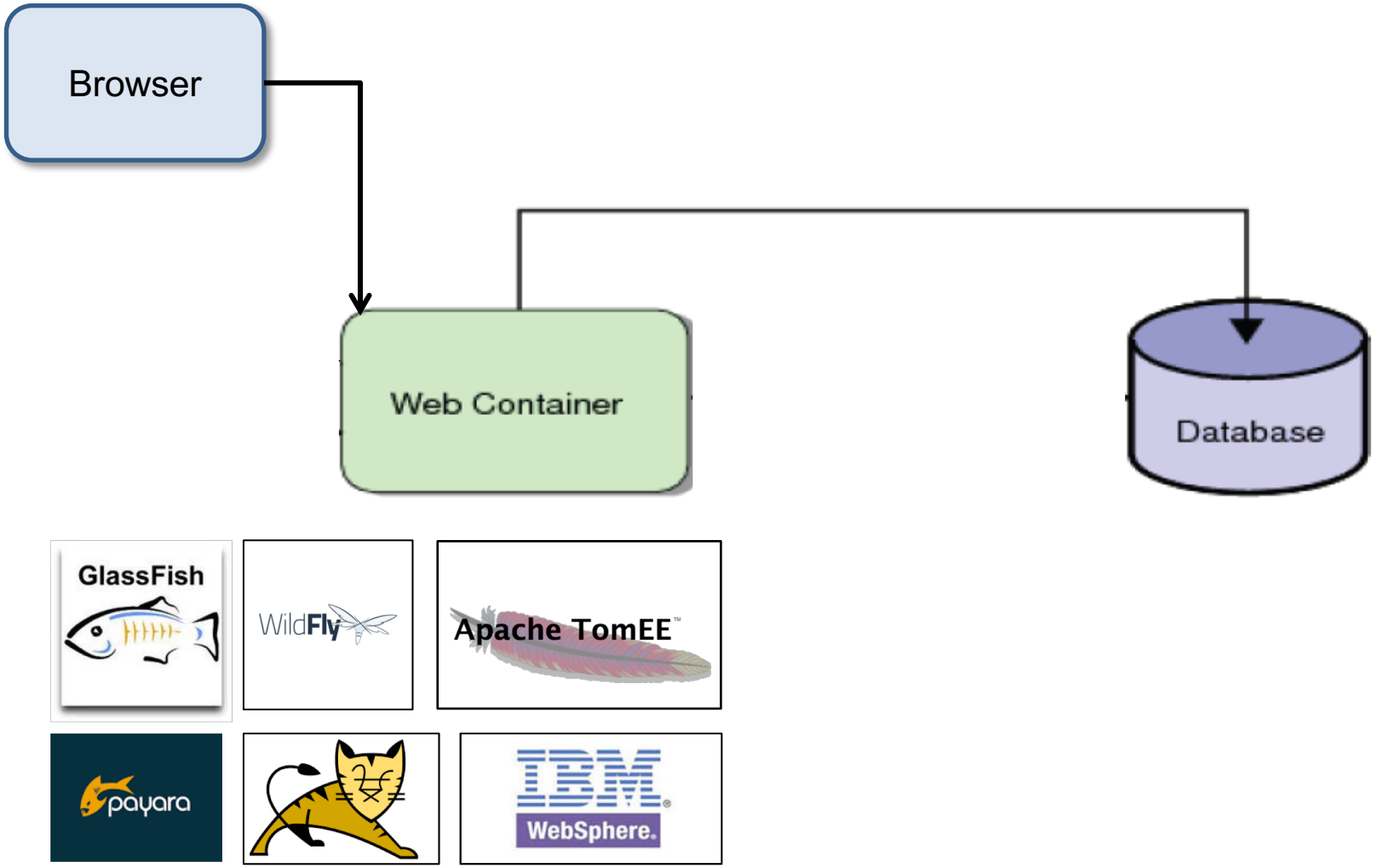
Modul 1 - Java EE Model
EJB Container: Local vs. Remote



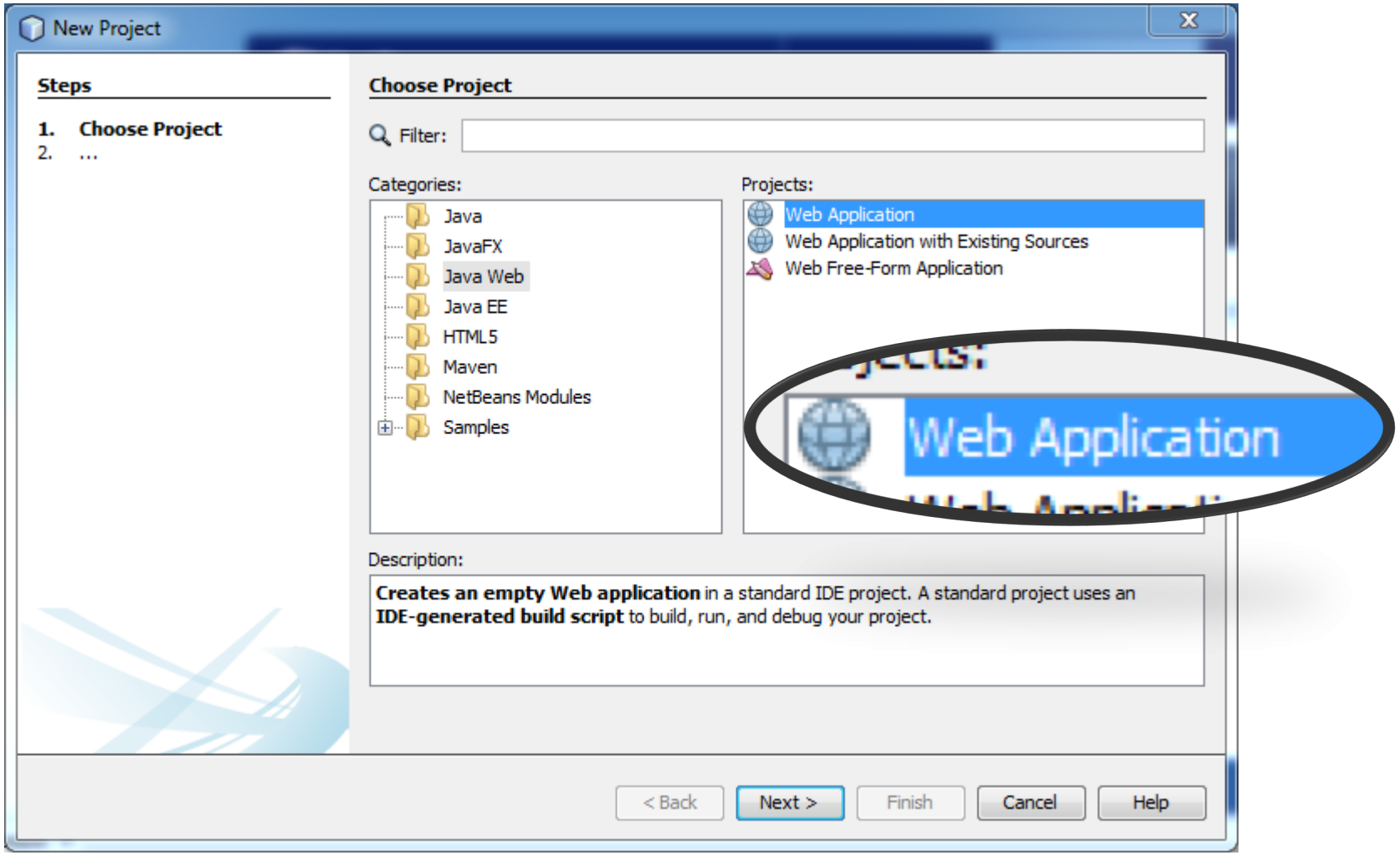
JavaEE Models

JavaEE Web-Application

Modul 2 - Java EE Model
Java EE Web-Application (Web-Centric)

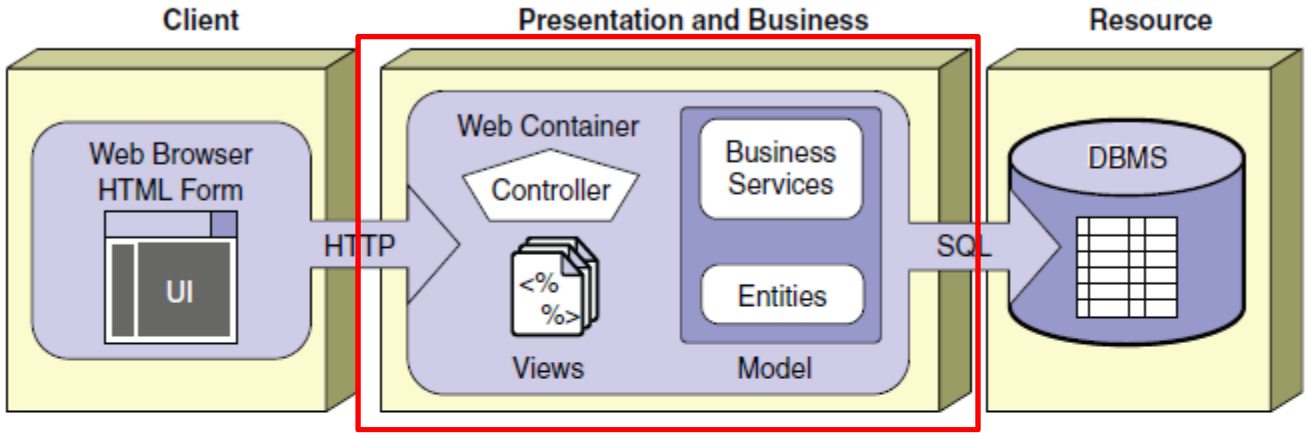


Modul 2 - Java EE Model
Java EE Web-Application (Web-Centric)

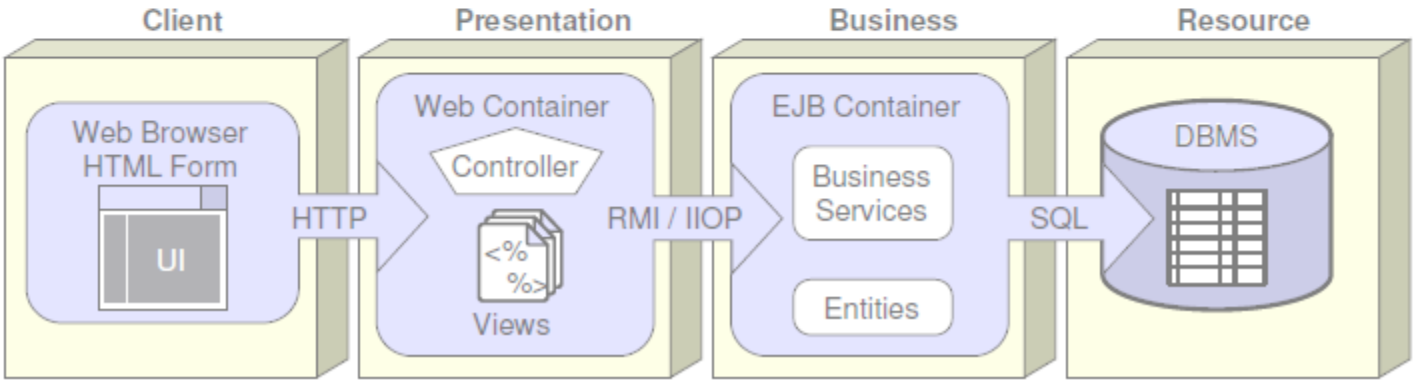


Modul 3
Role of Web Components in a Java EE Application

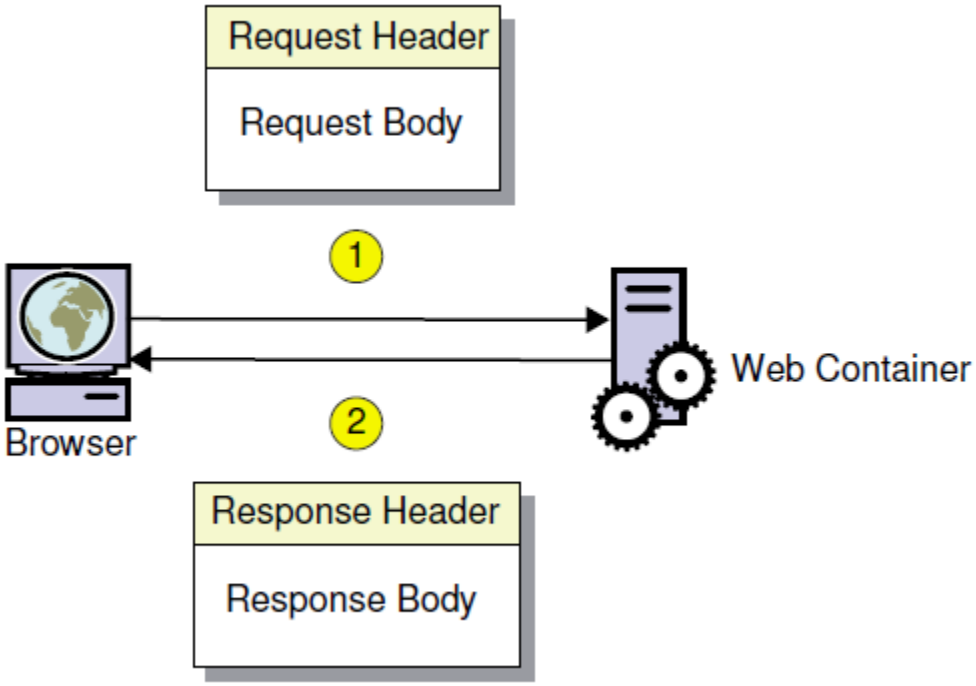
- Web-centric Java EE application architecture:



- EJB component-centric Java EE application architecture:



Modul 3
HTTP Request-Response Model



The GET and POST Requests

	GET **Request**	POST **Request**
Type of Use	Default	Form submission
Method of Sending Form Data	• Sent with the URI • Size is limited	• Sent in the request body • Size is unlimited
Display of Form Data	Browser displays in the URI area	Browser does not normally display with the URI

Modul 3
HTTP Request-Response Model

The GET Method

Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.php?name1=value1&name2=value2
```

The POST Method

Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

Quelle: https://www.w3schools.com/tags/ref_httpmethods.asp

Modul 3
FORM Data

Ziel-Adresse

Request-Art

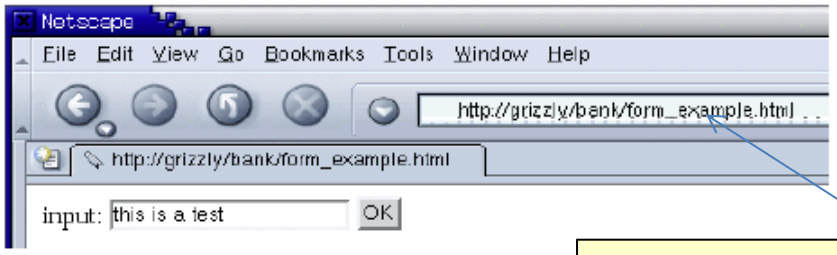
HTML snippet:

```
<FORM ACTION='form_test' METHOD='POST'>
<INPUT NAME='input1' SIZE='20'/>
<INPUT TYPE='SUBMIT' VALUE='OK'/>
</FORM>
```

Request-Parameter-Name

Button, der “wegschickt”

Browser form:

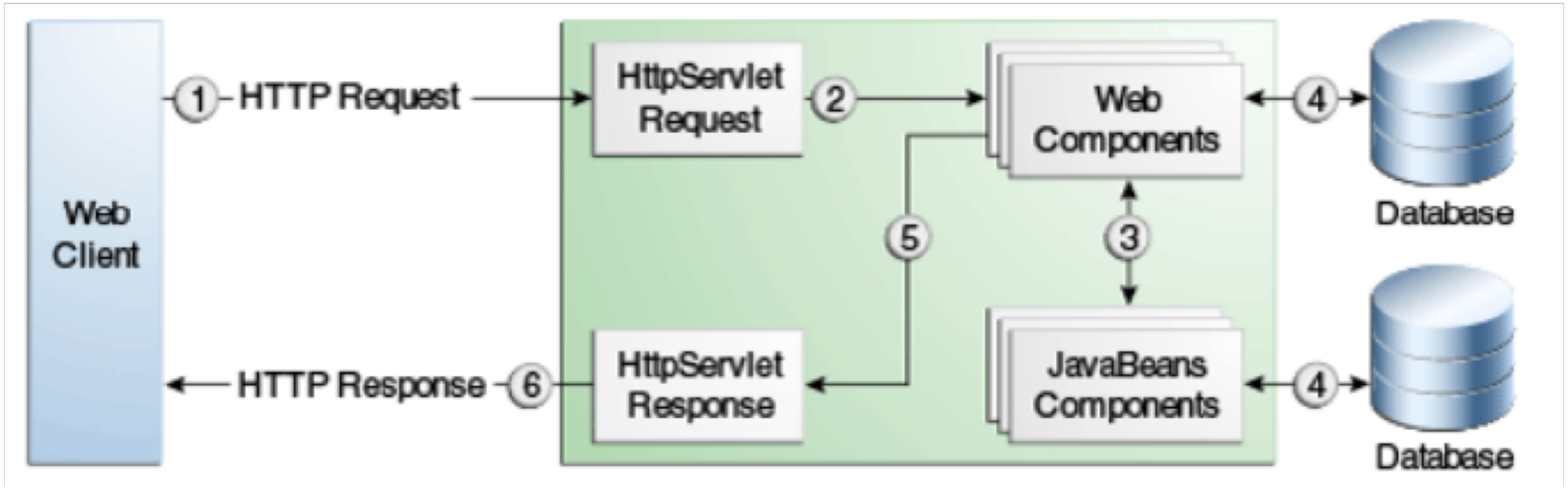


http://www.xyz.de/form_test

Browser request:

```
POST /bank/form_test HTTP/1.1
... request headers...

input1=this+is+a+test
```

Modul 3
HTTP Request-Response Model



1: The request is converted into a HttpServletRequest-Object by the WebContainer

2: The HttpServletRequest-Object is deliverd to a WebComponent (servlet, JSP, WebService)

3: The WebComponent can call Java-Methods

4: The WebComponent can makes DB access

5: The WebComponent generates a HttpServletResponse-Object

6: The WebContainer converts the HttpServletResponse-Object into a HTTP response

Quelle: https://docs.oracle.com/javaee/7/tutorial/webapp001.htm#GEYSJ

Modul 3
Content Type and the Response Header

The server response includes a `Content-Type` header that contains MIME type values including but not limited to:

- `text/html`
- `text/xml`
- `image/jpeg`

Examples of additional response headers include:

- `Content-Encoding`
- `Content-Length`
- `Cache-Control`

Modul 3
Content Type and the Response Header

Wie erstelle ich eine WebSeite, die Text und Bildern enthält?

- Bilder werden per Link “eingebettet”

Wie “bette” ich ein Bild ein?

- `<img src='/images/space.gif' />`

Aber das Bild soll dynamisch erzeugt werden?

- `<img src='www.my-url.de/path/servlet' />`

Was gibt **`www.my-url.de/path/servlet zurück`**?

- Byte-Stream, z.B. Content-Type **`image/jpeg`**

Wie erzeuge ich eine image/jpeg-Seite?

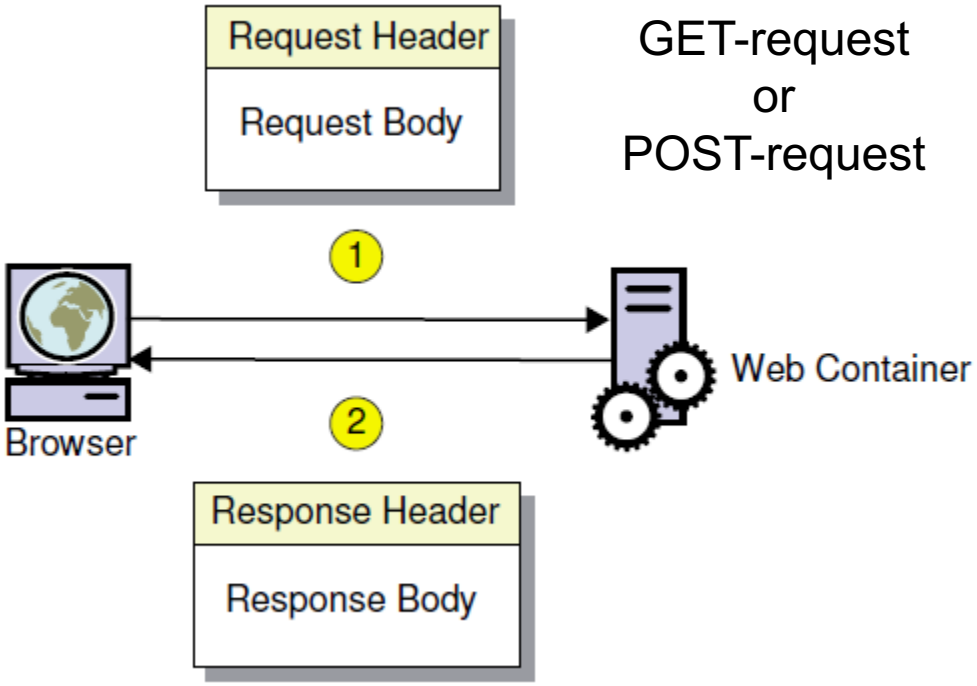
- “Output-Stream” erzeugen und Bits versenden (“streaming servlet”)

Developing Servlets

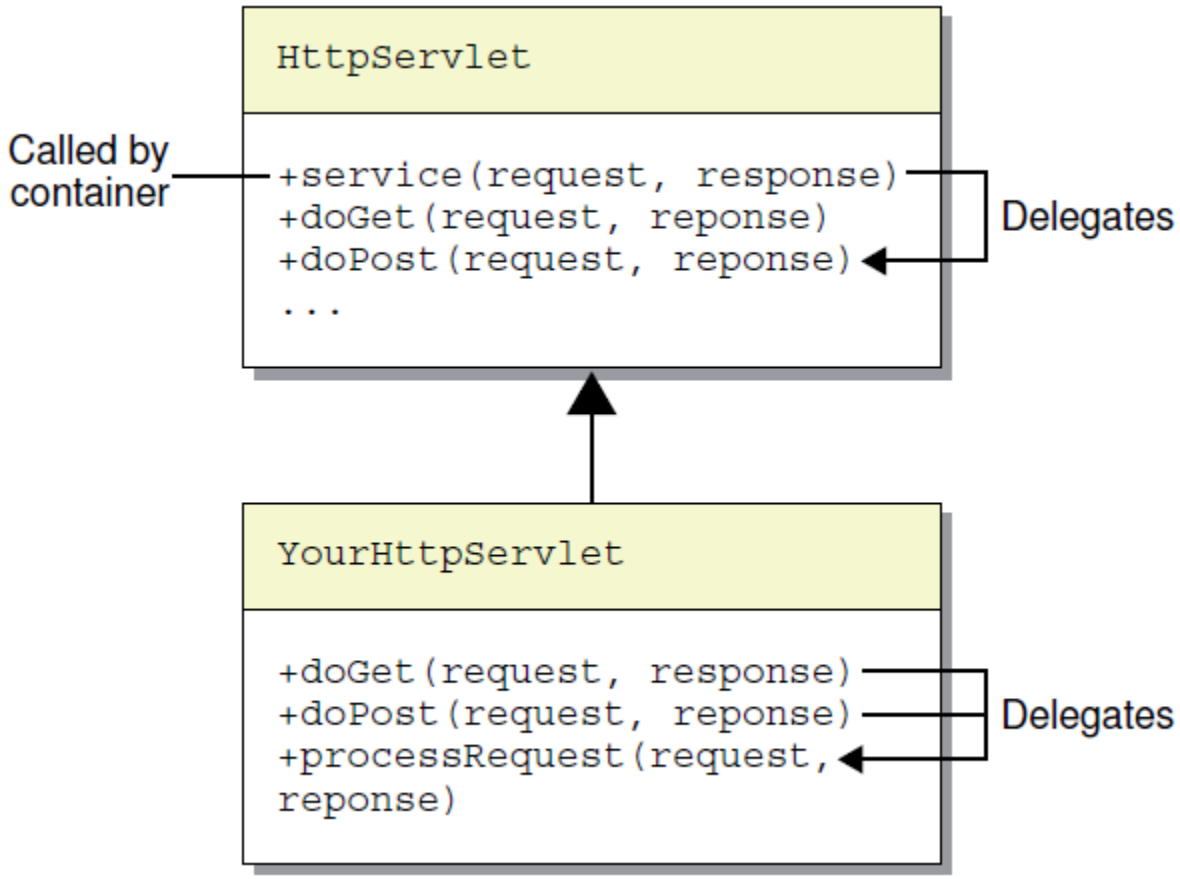
Modul 4
What is a servlet???

```
package edu.hsog;

import ...

@WebServlet(name = "MyServelt")
public class MyServelt extends HttpServlet {

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

    }

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

    }
}
```

Modul 4
HTTP Request-Response Model

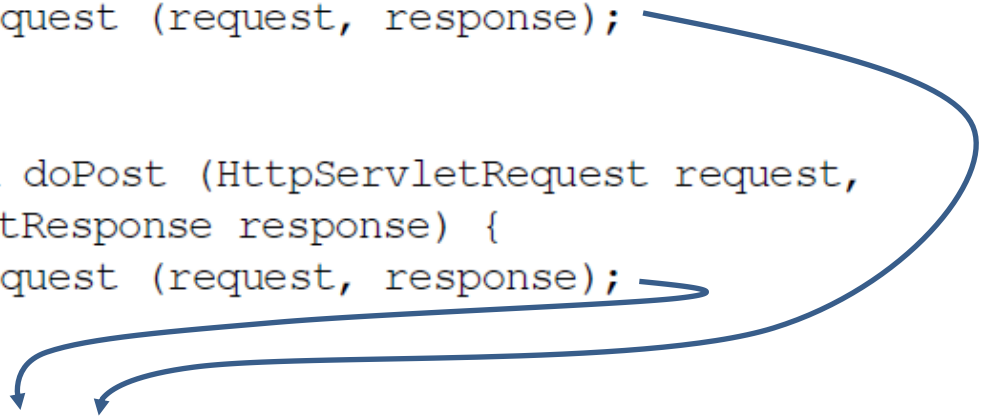


Modul 4
Request Handling Methods



Modul 4
Basic Servlet

```
1   import javax.servlet.http.*;
2   public class MyHttpServlet extends HttpServlet {
3
4     public void doGet (HttpServletRequest request,
       HttpServletResponse response) {
5       processRequest (request, response);
6     }
7
8     public void doPost (HttpServletRequest request,
       HttpServletResponse response) {
9       processRequest (request, response);
10    }
11
12    public void processRequest(HttpServletRequest request,
       HttpServletResponse response) {
13      // Process request and generate response
14    }
15  }
```



Modul 4
Servlet Configuration with **`web.xml`**

Without configuration, a servlet does not have an accessible URL. To configure a servlet and many other aspects of a web application, place a `web.xml` file in the `WEB-INF` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>SomeName</servlet-name>
        <servlet-class>package.MyHttpServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SomeName</servlet-name>
        <url-pattern>/myservlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Modul 4
Servlet Configuration with Annotations

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet(name = "SomeName",
            urlPatterns = {"/myServlet"}
public class MyHttpServlet extends HttpServlet {
    ...
```

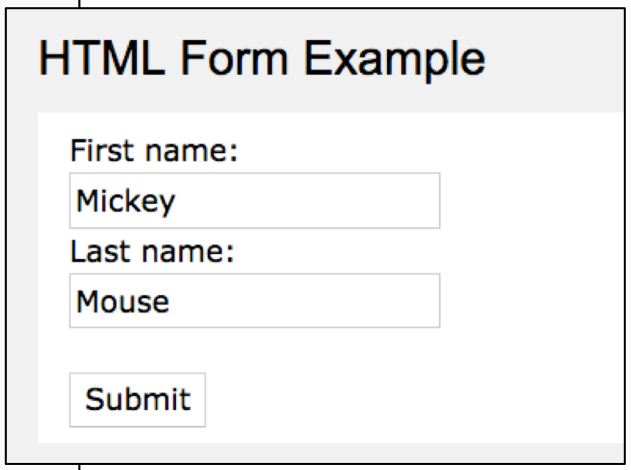
Aufgabe 1

Schreibe ein servlet welches “Hallo World” als HTML-Seite zurücksendet.

Modul 4
Servlet – the request object

How to access form-data in a servlet?

```
<!DOCTYPE html>
<html>
<body>

<form action="/action_page     ">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```



Answer: `String fn = (String) request.getParameter("firstname");`

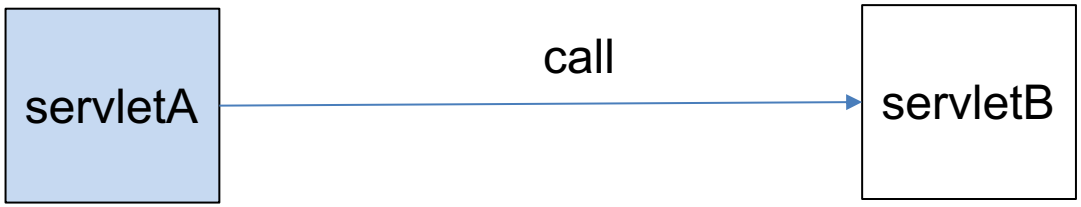
Modul 4
Example of Handling Form Data and Producing Output

```
1   public void processRequest (HttpServletRequest request,
     HttpServletResponse response)
2        throws IOException {
3     response.setContentType (“text/plain”);
4     PrintWriter out = response.getWriter();
5     String name = request.getParameter ( "input1");
6     if (name == null || name.length() == 0)
7       name = “anonymous”;
8     out.println (“Hello, “ + name);
9     out.close();
10    }
```

Aufgabe 2

Modifiziere das servlet, so dass Formulardaten gelesen werden.

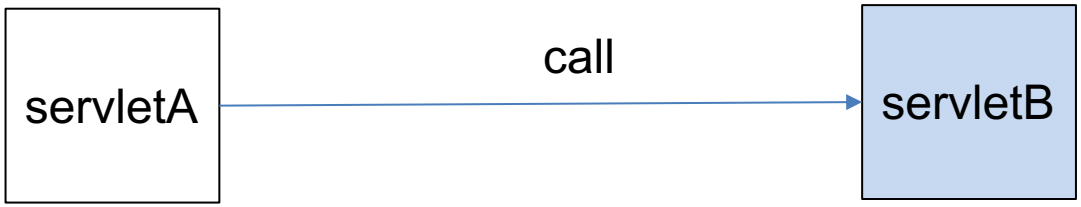
Modul 4
Forwarding Control and Passing Data



RequestDispatcher

```
// servletA calls servletB

   // Add data
   request.setAttribute("tagA", "... Meine Werte ... ");

   RequestDispatcher rd = request.getRequestDispatcher("servletB");
   rd.forward(request, response);
```

Modul 4
Forwarding Control and Passing Data

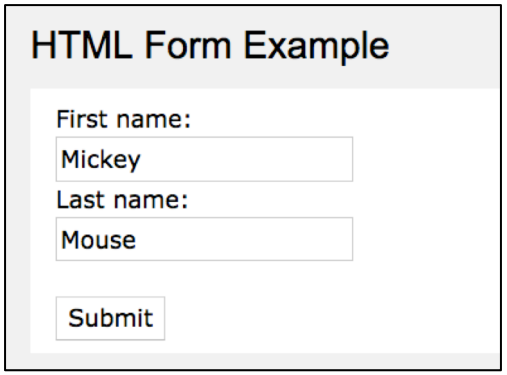



```
// servletB reads data

   // Get data
   String b = (String) request.getAttribute("tagA");
```

Modul 4
Servlet – getParameter() vs getAttribute()

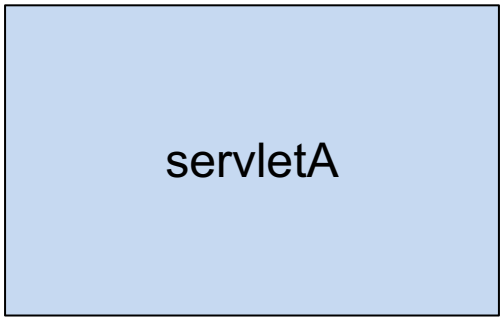
Form-data

`request.getParameter( … )`



„own data“ added with `request.setAttribute()`

`request.getAttrinute( … )`

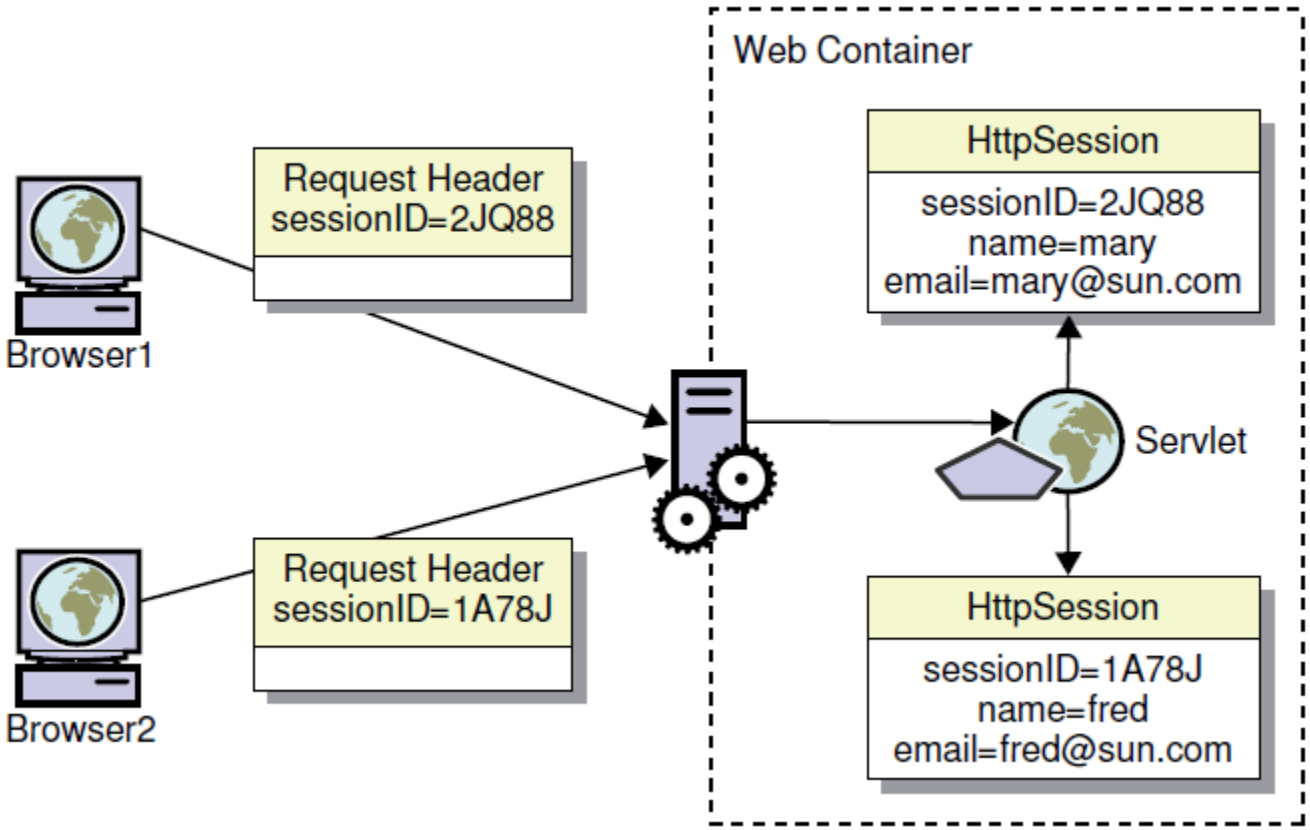


Aufgabe 3

Modifiziere das servlet so, dass es ein anderes servlet aufruft und den String “hallo next servlet” übergibt.

Modul 4
HTTP Session

Modul 4
HTTP Session



Modul 4
Session Binding

Each session generates unique HttPSession object

HttpSession-Object can be used:

- check if session is new: **`session.isNew()`**
- add data to session object:
  `session.setAttribute(„tagA“, „value“)`
- get data from session object: **`session.getAttribute(„tagA“)`**
- delete session: **`session.invalidate()`**

Modul 4
Retrieving a Session Object

```
1   // Get a session object for the current client, creating
2   // a new session if necessary
3   HttpSession session = request.getSession();
4
5   // If this is a new session, initialize it
6   if (session.isNew()) {
7     // Initialize the session attributes
8     // to their start-of-session values
9     session.setAttribute ("account", new Account());
10    // ... other initialization
11  }
12
13  // Get this client's 'account' object
14  Account account = (Account) session.getAttribute("account");
```

Do not use the session object to transfer data between components in place of request attributes or method parameters. Sessions consume memory in the web server.

(Session-Caching-Antipattern)

Modul 4
Logout and Invalidation

Application design and session management can reduce the security risk and memory usage of sessions that remain open after use.

- Provide users with options for logging out or for closing the session:
  - Log out after fixed number of steps
  - Log out with the click of a Logout button
  - Log out through menu operation
- To close the session, call its `invalidate` method:

```
1   if ("logout".equals(request.getParameter("action")) {
2     session.invalidate();
3   }
4
```

After 30 mins (default) session expires automatically

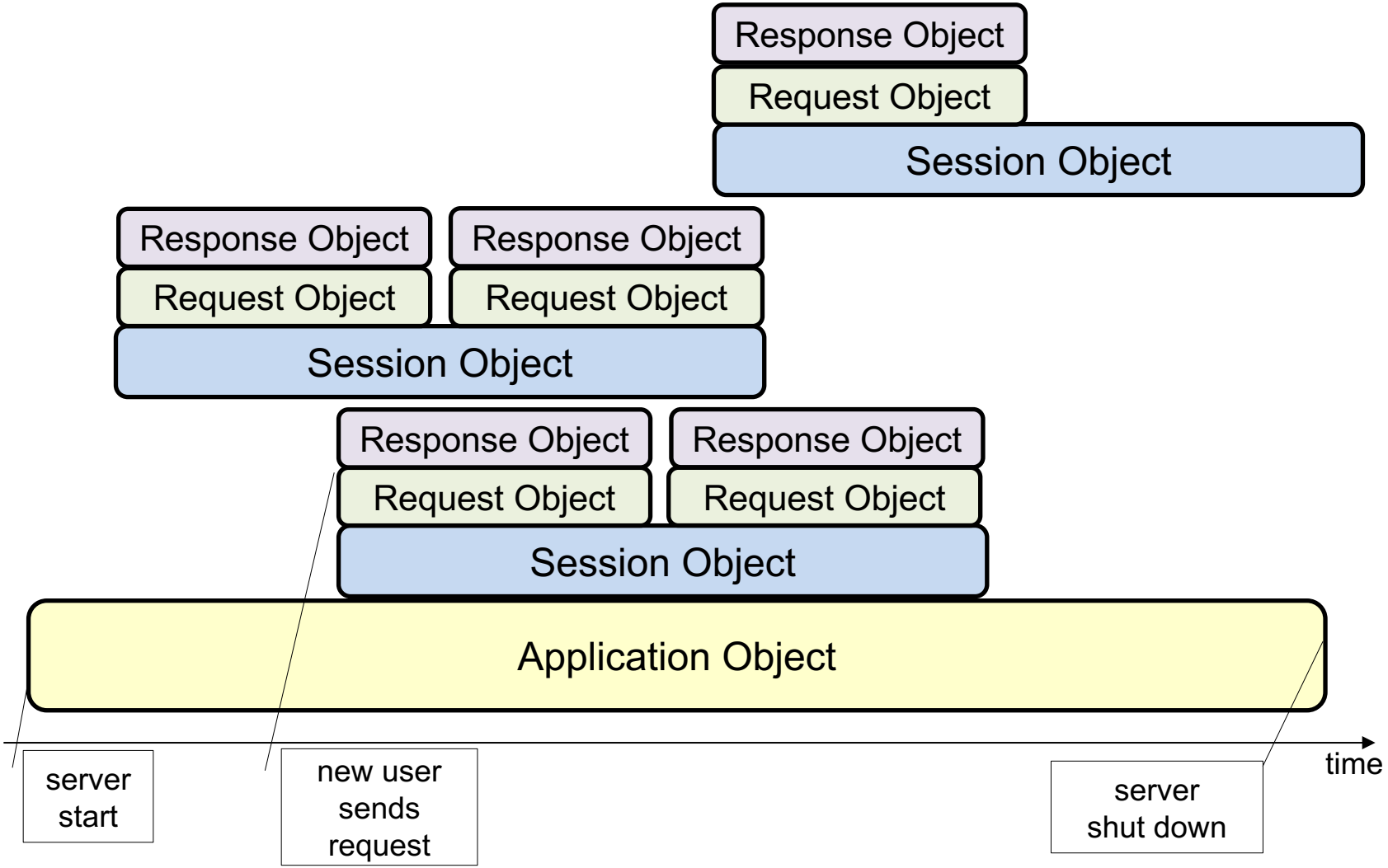
Modul 4
Request Object

Generic Methods	Purpose
`getParameter`	Gets form data elements
`getAttribute` and `setAttribute`	Gets and sets attributes, which are used for passing data between components
`getRequestDispatcher`	Gets a request dispatcher to transfer control to another component
HTTP-Specific Methods	**Purpose**
`getUserPrincipal`	Gets user's identity
`getCookies`	Gets cookies sent by browser
`getSession`	Gets client session

Modul 4
Response Object

Generic Methods	Purpose
`getOutputStream`, `getWriter`	Gets a stream or writer to send data to the browser
`setContentType`	Indicates the MIME type of response body
HTTP-Specific Methods	**Purpose**
`encodeURL`	Adds a session ID to a URL
`addCookie`	Sends a cookie to the browser
`sendError`	Sends an HTTP error code

Modul 4
Implicit Objects



Modul 4
Implicit Objects

Object-Name	Interface / Class
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
…	

Request object: Request object has a request scope that is used to access the HTTP request data, and also provides a context to associate the request-specific data. Request object implements javax.servlet.ServletRequest interface. It uses the getParameter() method to access the request parameter.

Response object : This object has a page scope that allows direct access to the HTTPServletResponse class object.

Modul 4
Implicit Objects

Object-Name	Interface / Class
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
…	

Application object: These objects has an application scope. These objects are available at the widest context level, that allows to share the same information between the JSP page's servlet and any Web components with in the same application.

Session object: Session object has a session scope that is an instance of javax.servlet.http.HttpSession class. Perhaps it is the most commonly used object to manage the state contexts. This object persist information across multiple user connection.

Modul 4
Application object

Application object: These objects has an application scope. These objects are available at the widest context level, that allows to share the same information between the JSP page's servlet and any Web components with in the same application.

…

```
ServletContext appCtx = getServletContext();
appCtx.setAttribute("for_ever", “running");
```

…

```
... = appCtx.getAttribute("for_ever");
```

…


Java
Servlets

&


JSP

Modul 4
What is a Java Server Page???

```
<%@page import="java.util.Date"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <%= new Date() %>
    </body>
</html>
```

Modul 4
Servlet Example

```
1   import javax.servlet.*;
2   import javax.servlet.http.*;
3   import java.util.Date;
4
5   public class Controller extends HttpServlet {
6
7   protected void doGet(HttpServletRequest request,
           HttpServletResponse response)
8               throws ServletException, IOException {
9           response.setContentType("text/html");
10          PrintWriter out = response.getWriter();
11          out.println ("<html><head/><body>");
12          out.println ("<h1>Hello, World!</h1>");
13          out.println ("The date is:" + new Date());
14          out.println ("</body></html>");
15          out.close();
16      }
17  }
```

Modul 4
JSP Component Example

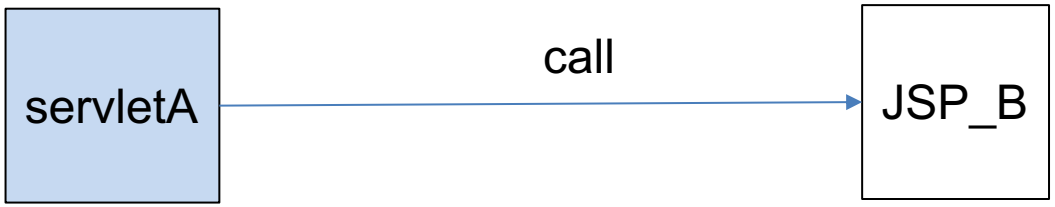
The following code generates the same output as the preceding servlet example:

```
1   <%@ page import="java.util.Date" %>
2   <html>
3   <head/>
4   <body>
5   <h1>Hello, World!</h1>
6   The date is: <%= new Date() %>
7   </body>
8   </html>
```

Modul 4
Comparison of Servlets and JSP™ Components

	Servlets	**JSP Components**
Description	Web components authored in the Java programming language	Presentation content with embedded programmatic elements
Characteristics	Extend generic base classes in the API, typically the `HttpServlet` interface	• Can be enhanced with custom tags • Are translated into servlets by the web container
Created or Managed By	Developers	Content authors

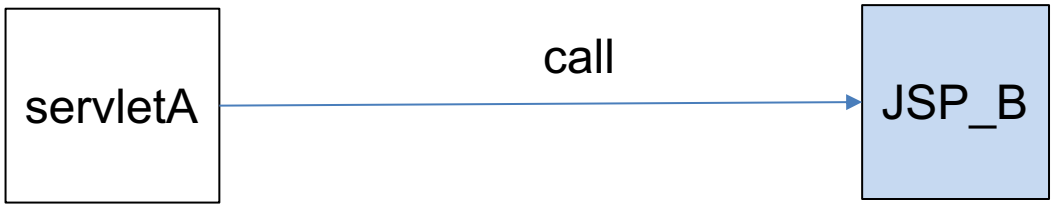
Modul 4
Forwarding Control and Passing Data



RequestDispatcher

```
// servletA calls JSP_B

   // Add data
   request.setAttribute("tagA", "... Meine Werte ... ");

   RequestDispatcher rd = request.getRequestDispatcher("JSP_B.jsp");
   rd.forward(request, response);
```

Modul 4
Forwarding Control and Passing Data




```
 // JSP_B reads data
<!DOCTYPE html>
<html>
    <h1>Hello World from  JSP!</h1>

    // Get data
    <%= request.getAttribute("tagA") %>

</html>
```