# N-Tier Architectures

- Distributed Web-Applications are today implemented as *Multi-Tier Applications*
- Each tier (= layer) has it‘s own functionality
- Advantages
  - Separate components are less complex
  - Distribution of implementation tasks
  - Flexibility
  - Scalaibility
  - Security

# N-Tier Architectures

- Separation of functionality into three conceptual layers
- Presentation Tier
  - interacts with user, presents information and accepts requests
  - has usually a graphical user interface
- Business Tier
  - implements application logic
  - often divided into
    - Web Tier
    - Application Server
- Resource Management Tier
  - manages the data sources of the information system (DBMS, file system, ERP-system)

## Presentation Layer

- User Interface
- Nowadays is usually realized as thin client, implemented with a Web-Browser
- Functionallity
  - accept user requests
  - Present results of server side computations
- Common technologies
  - **HTML** (forms)
  - JSP
  - …

## Business Layer

- This layer is where most of the appliction logic is implemented, e.g.
  - shopping cart (in an e-business application)
  - price computation, tax calculation
- This layer is for scalibility and safety reasons often further divided into
  - Web-Server
  - Application Server
- Common technologies
  - **Servlets**, JSP, ASP, .NET, Corba, **EJB**

## Resource Management Layer

- Often referred to as Data Layer
- Function is to manage the business data of an application
  - customer information
  - product data
  - orders

- Common technologies
  - Data Base Systems like DB2, Oracle, MySql, …
  - Enterprise Resource Planning Systems like SAP

# 1-Tier Architecture

- All conceptual layers are combined in a single tier
- Very popular on mainframe computer architectures => Monolithic systems
  - Clients are usually dumb terminals
  - No other entry points from outside
  - Focus on efficient usage of hardware resources
  - Prototyp of a legacy system
- Advantages
  - Optimizes performance, no communication overhead required
- Disadvantages
  - Maintanance is difficult, qualified Cobol programmers necessary

## 2-Tier Architecture

- Most popular approach: Client / Server System
- Parts of the presentation layer (and application logic) is transferred on a PC or workstation
- Clients correspond to presentation layer, servers correspond to application logic and data layer or
- Clients include presentation and application layer, servers are used as database-servers
  - thin client / fat server versus fat client and thin server

# 3-Tier Architecture

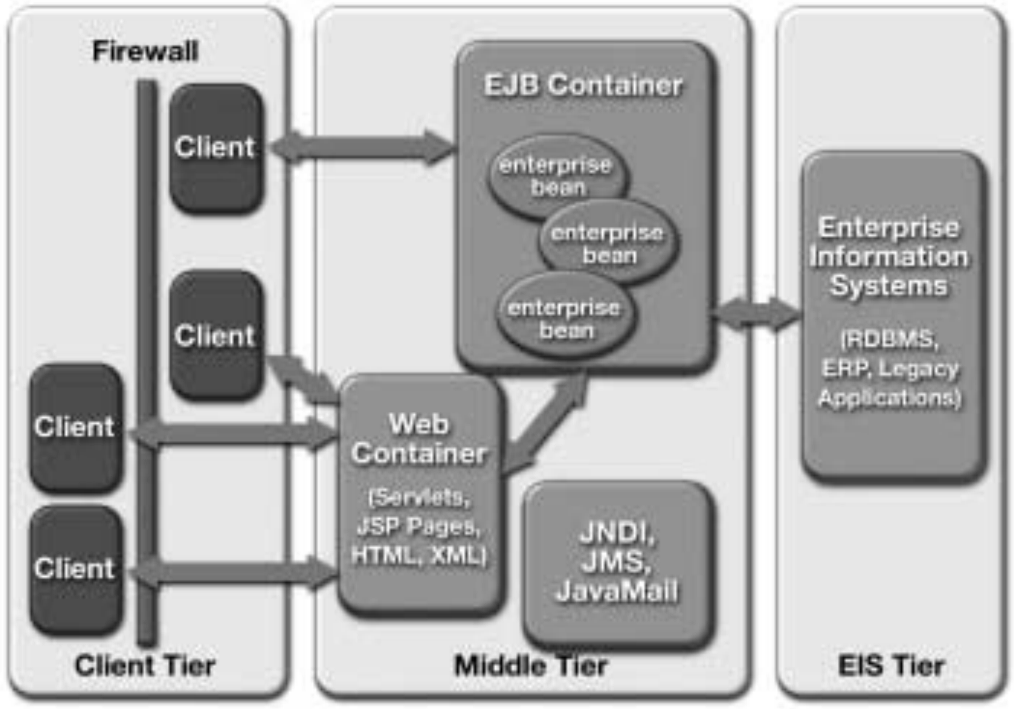
- Based on a clear separation of three layers
  - Client tier with presentation layer
  - Middle tier implements application logic
  - Resource management layer is composed of one (or many) database-servers
- Scalable, application layer can be distributed on different computers in a cluster
- Supports integration of multiple resource managers (ERP-Systems, DBMSs etc.) **but**
- increased communication between layers

# N-Tier Architectures

- Further generalizes 3-Tier architecture
  - Resource layer consists of different tier, focus on intergration of different systems
  - Presentation layer is realised in 2 separate tiers
    - Clients are using browsers for HTML
    - Web-Servers generate dynamic HTML-pages
    - Usually results in 4-Tier architecture (see J2EE part of lecture)
- N => N+1
  - adds flexibility and distribution options, **but**
  - introduces performance, complexity and management issues

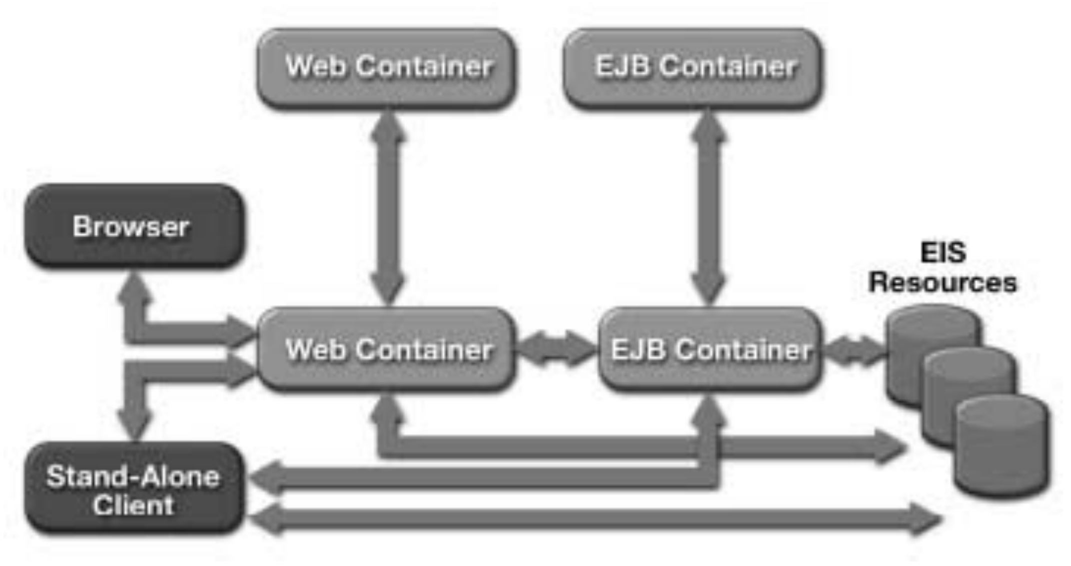
# Java 2 Enterprise Edition (J2EE)

- Sun‘s view of a N-Tier Arcitecture



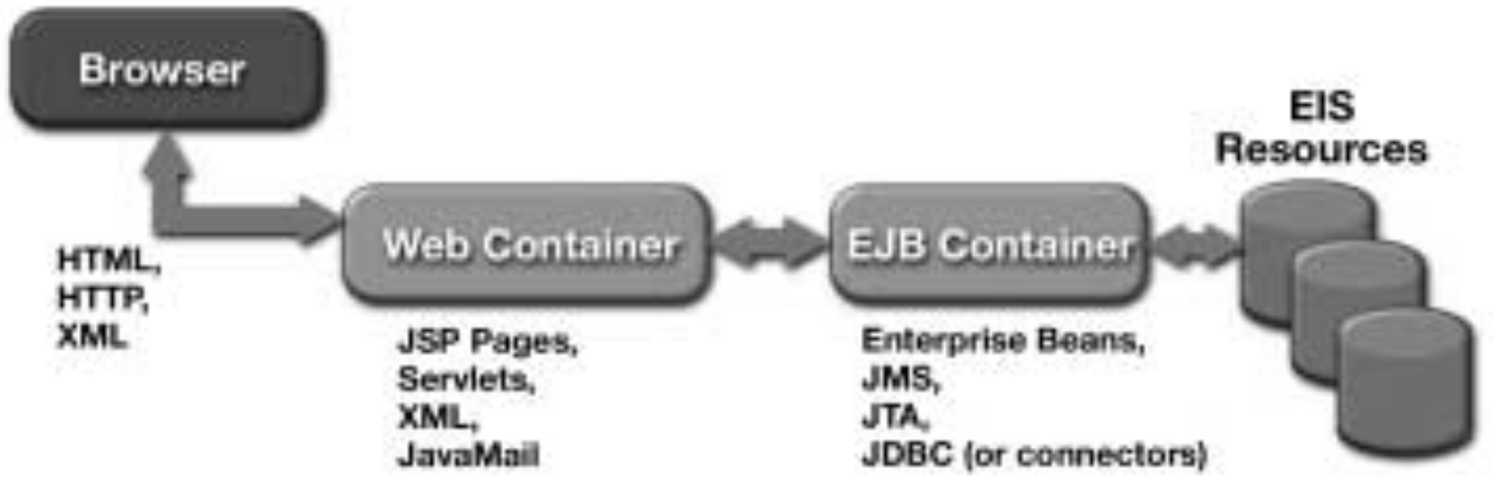
# J2EE Application Scenarios

- J2EE provides a flexible programming model and supports a variety of software architectures:



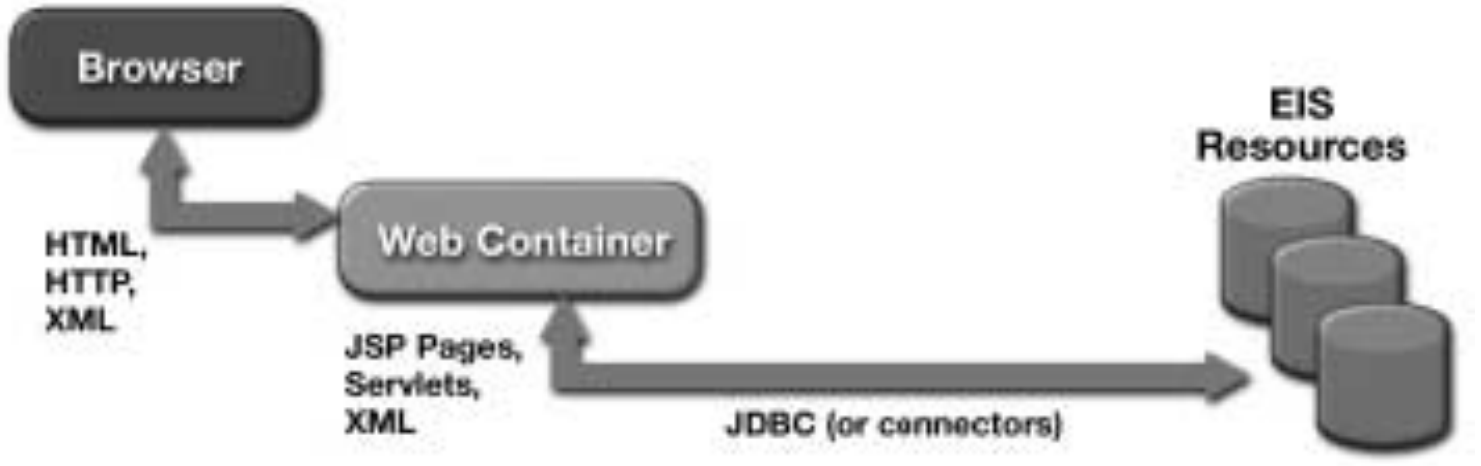
# J2EE Application Scenarios

- A typical 4 – Tier architecture of an J2EE-application with a thin client, web server, application server and database:


Browser
HTML, HTTP, XML
Web Container
JSP Pages, Servlets, XML, JavaMail
EJB Container
Enterprise Beans, JMS, JTA, JDBC (or connectors)
EIS Resources

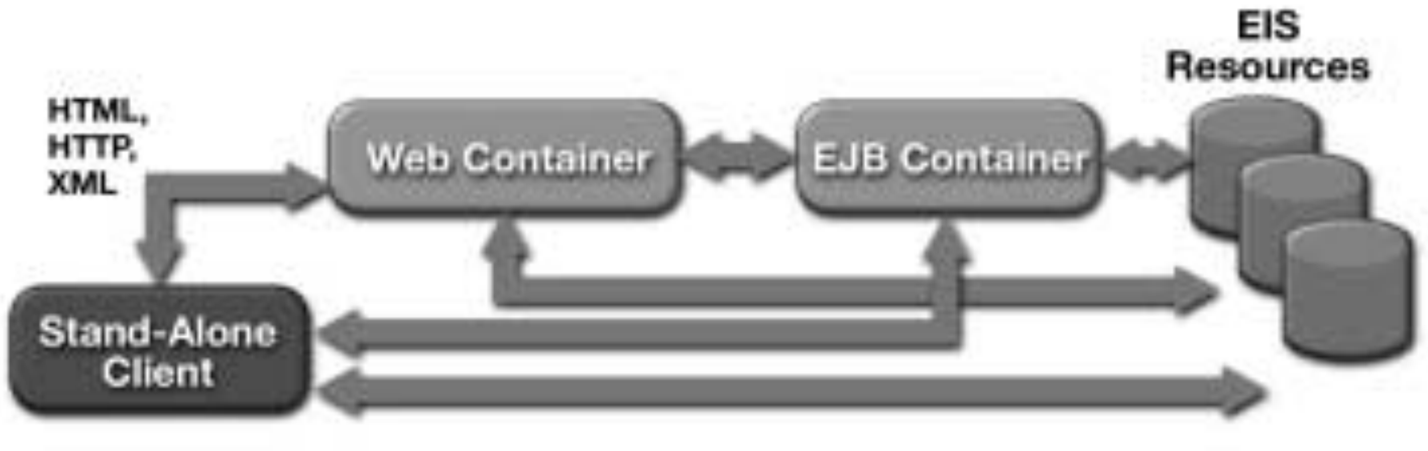
# J2EE Application Scenarios

- Web - centric 3 – Tier architecture
- Web-Container implements presentation- and application logic



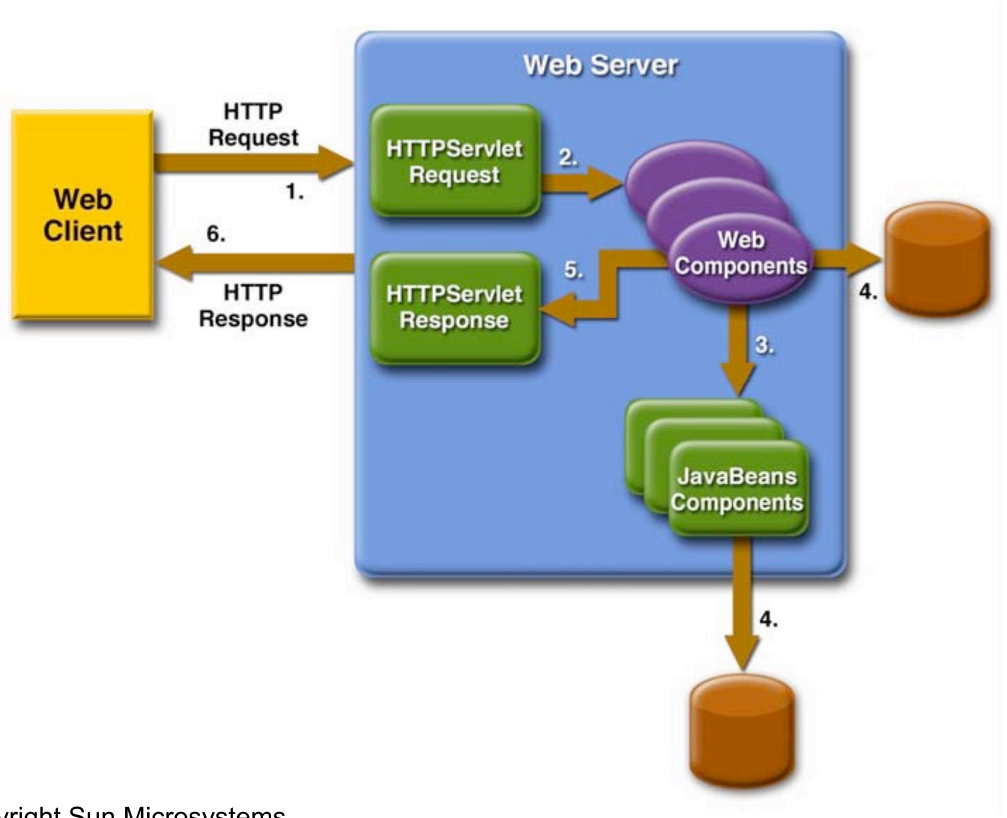
# J2EE Application Scenarios

- J2EE guidelines even allow classical C/S architectures
- 2-Tier applications with stand-alone clients



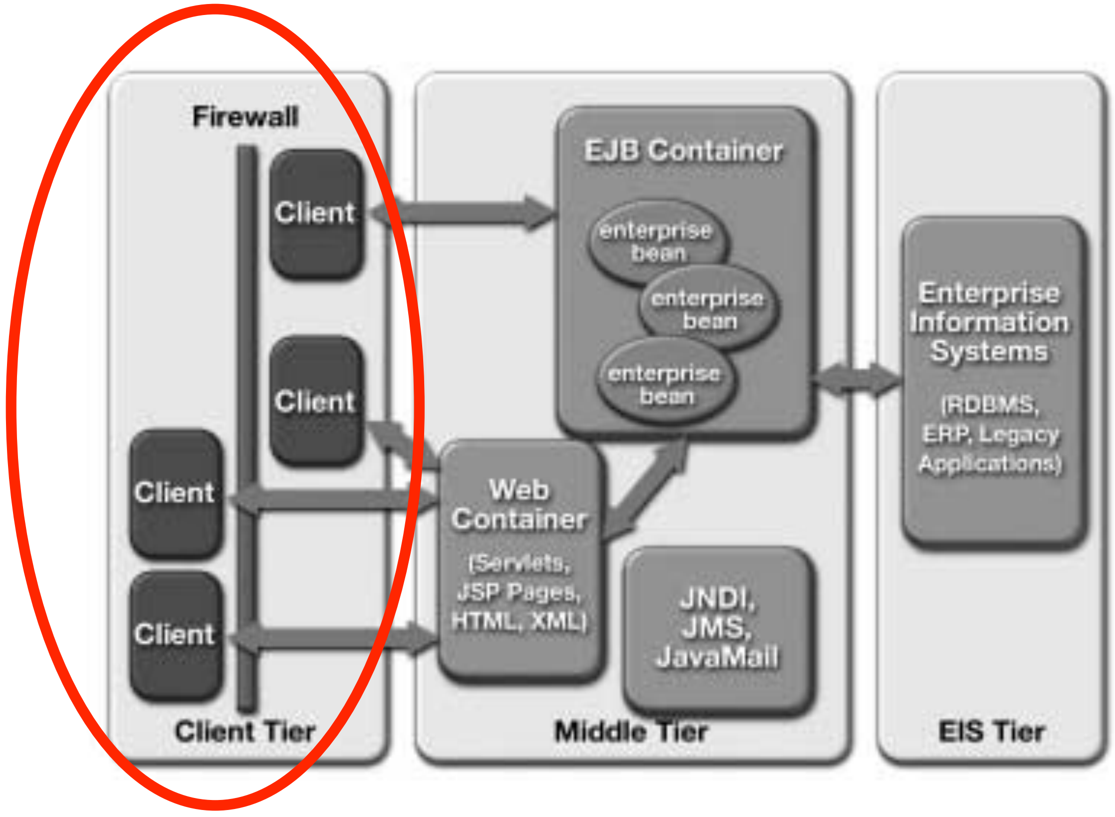
# J2EE at work

- Putting all pieces together:



# Client - Technologies

- Where we are

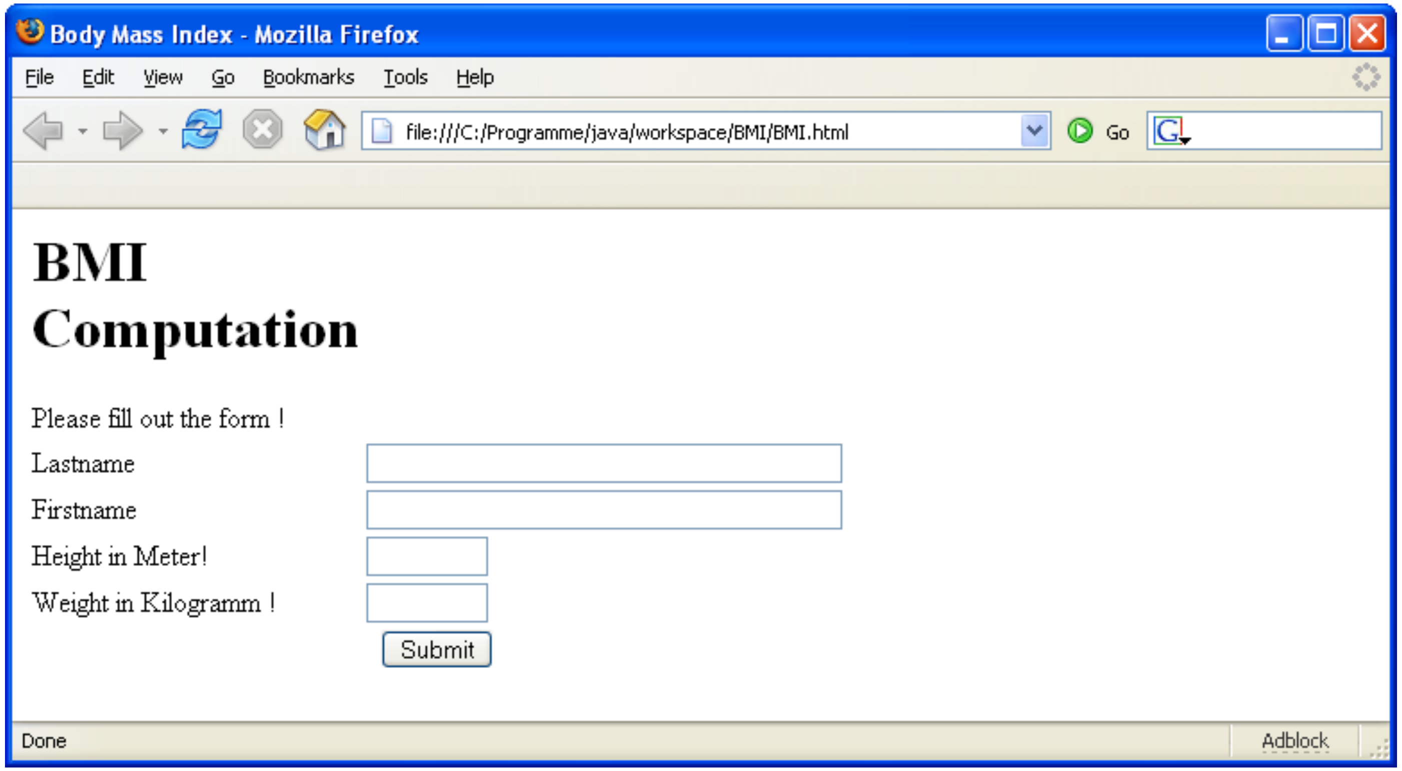


## Web - Clients

- Run on a browser (e.g. Firefox, …)
- User Interface (content) is generated dynamicaly on the server-side
- Generated documents are presented by the browser
- Communication protocoll is HTTP (or HTTPS)
- Documents are written in HTML
- „runs everywhere“

# Web – Client (Example)

- Calculate your Body-Mass-Index


Body Mass Index - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///C:/Programme/java/workspace/BMI/BMI.html
Go
Done
Adblock

**BMI Computation**

Please fill out the form !

| | |
|---|---|
| Lastname | |
| Firstname | |
| Height in Meter! | |
| Weight in Kilogramm ! | |
| | Submit |

## Web – Client (Example)

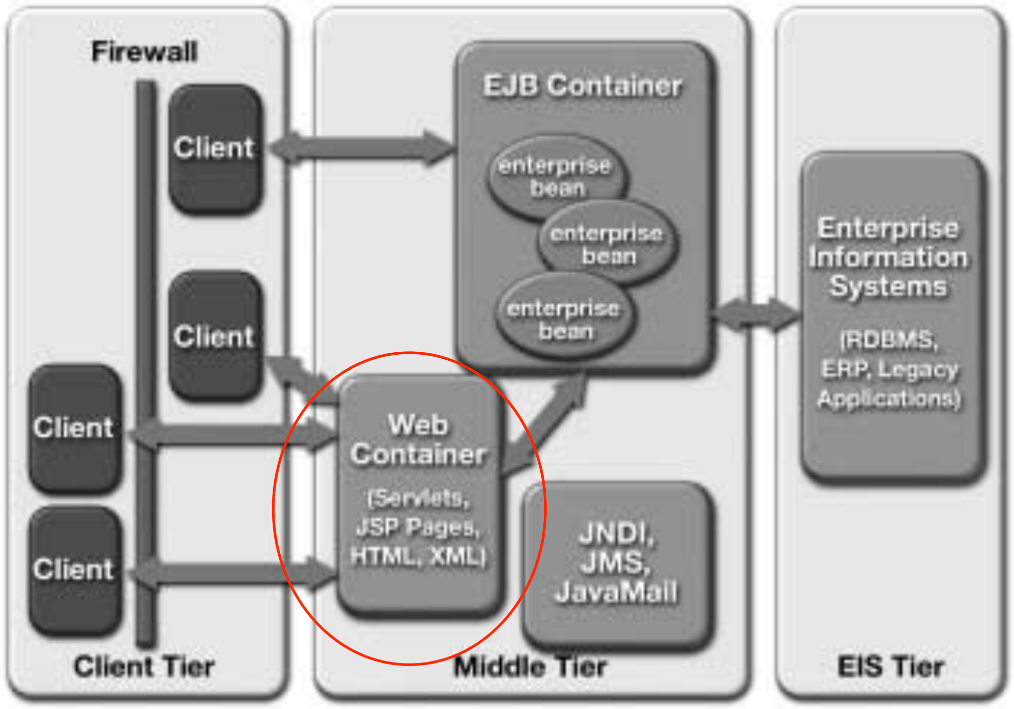
- HTML - Code

# Web – Client (Example)

- Functionality of the presentation tier
  - Presentation of results of server side computations
  - Accept user inputs
- What we need:
  - HTML – elements for user input (HTML – forms)
  - Transport mechanism for data from client to server (HTTP GET and HTTP POST methods)
- We can use URLs to submit data as parameters
  - Format: http://host:port/path/program?p1=v1&p2=v2
  - http://sfoproject.informatik.fh-fulda.de/CareeOnline/BMI/height=1.73&weight=73.0

# Web - Tier

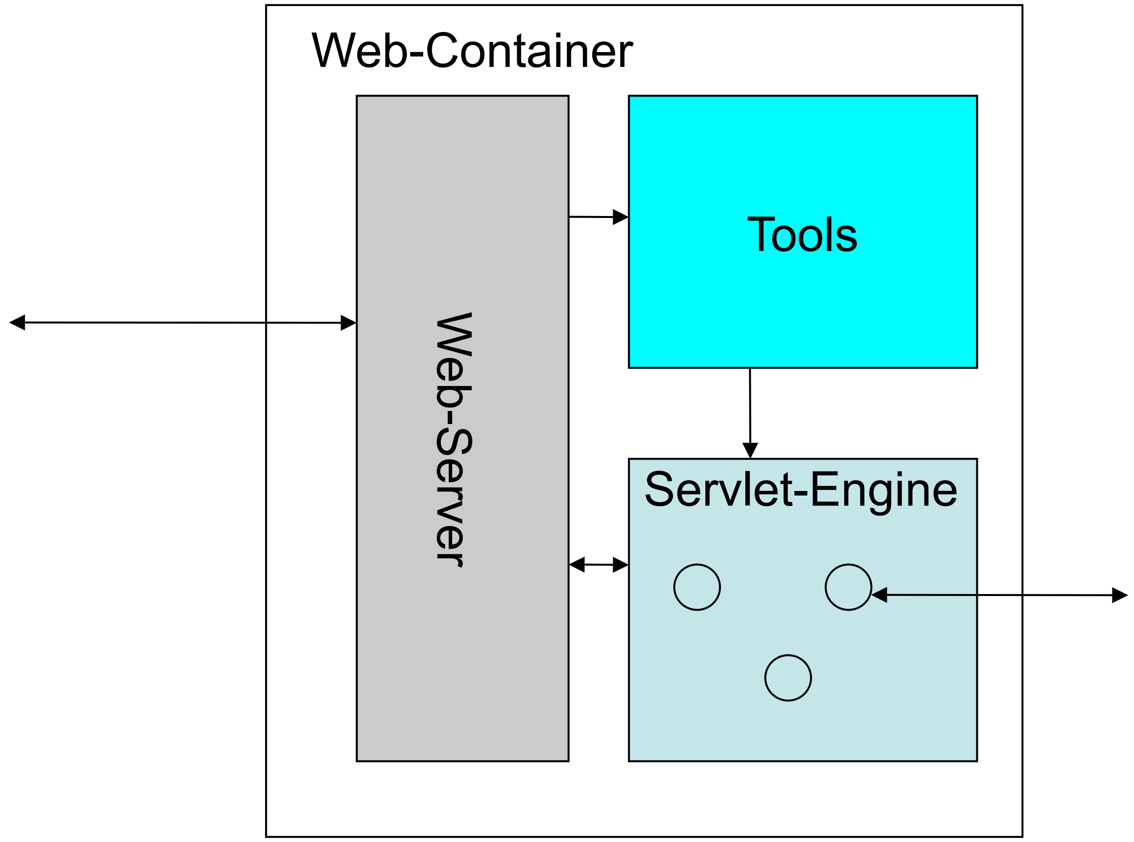
- Where we are



## Web - Tier

- The purpose of the Web-Tier
  - The server in the Web-Tier processes HTTP requests
  - Manages interaction between web client and application logic
  - Generates HTML (or XML) content
  - Business logic may be implemented entirely within the Web-Tier (3-Tier architecture) but is often implemented in another tier using EJB-technologies
  - For greater detail see
    - Designing Enterprise Applications with the J2EE ™ Platform, Second Edition, by  Inderjeet Singh, Beth Stearns,Mark Johnson, and the Enterprise Team [1]
    - http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/

# Web - Container



## HttpServletRequest / Response

- See online API documentation:

  http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/http/HttpServletRequest.html

# Servlet – Simple Example

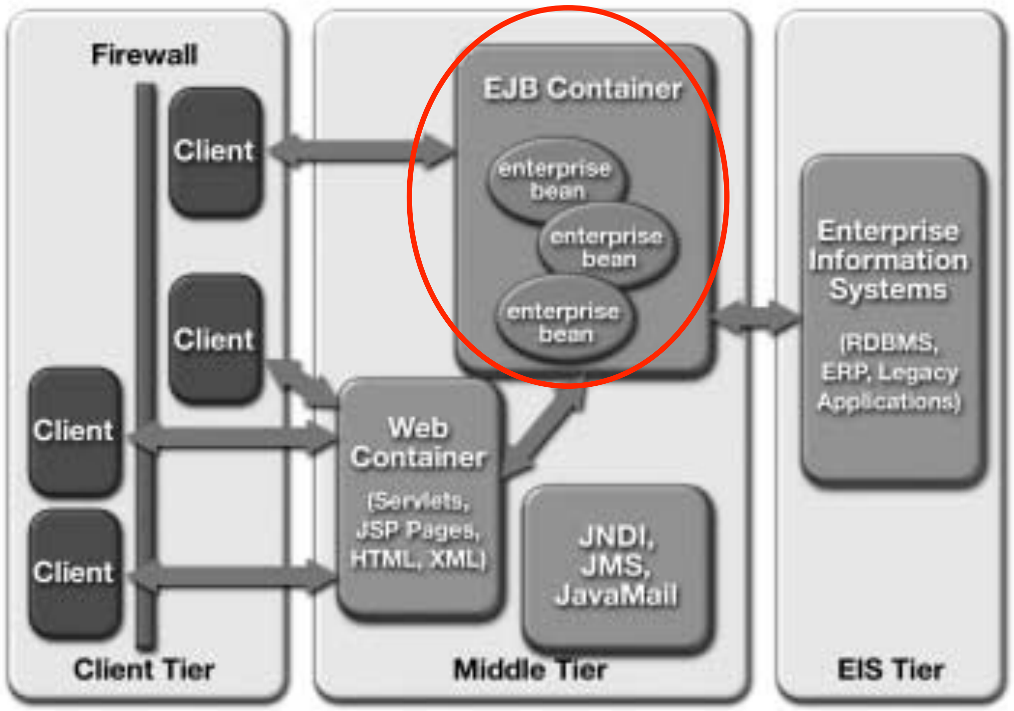
```
import java.io.*;

import javax.servlet.http.*;
import javax.servlet.*;

// Im Browser: http://localhost:8080/BMI/beispiel

public class BMIServlet extends HttpServlet {
     public void doPost(HttpServletRequest req, HttpServletResponse res)
               throws ServletException, IOException {

          String name = req.getParameter("name");
          String vorname = req.getParameter("firstname");
          String groesse = req.getParameter("groesse");
          String gewicht = req.getParameter("gewicht");

          Double heigth = new Double(groesse);
          Double weight = new Double(gewicht);

          double h = heigth.doubleValue();
          double g = weight.doubleValue();
          double bmi = g / (h * h);

          Double BMI = new Double(bmi);
```

## Servlet – Simple Example (cont.)

```
                    res.setContentType("text/html");
                    PrintWriter out = res.getWriter();
                    out.println("<html>");
                    out.println("<body>");
                    out.println("<head>");
                    out.println("<title>Ergebniss dieser BMI -
                                 Berechnung</title>");
                    out.println("</head>");
                    out.println("<body>");
                    out.println("<h1>Hello " + vorname + " " + name + "</h1>");
                    out.println("<h1>Ihr BMI ist " + BMI + "</h1>");
                    out.println("</body>");
                    out.println("</head>");
                    out.close();
          }
}
```

# Enterprise Java Beans (EJB)

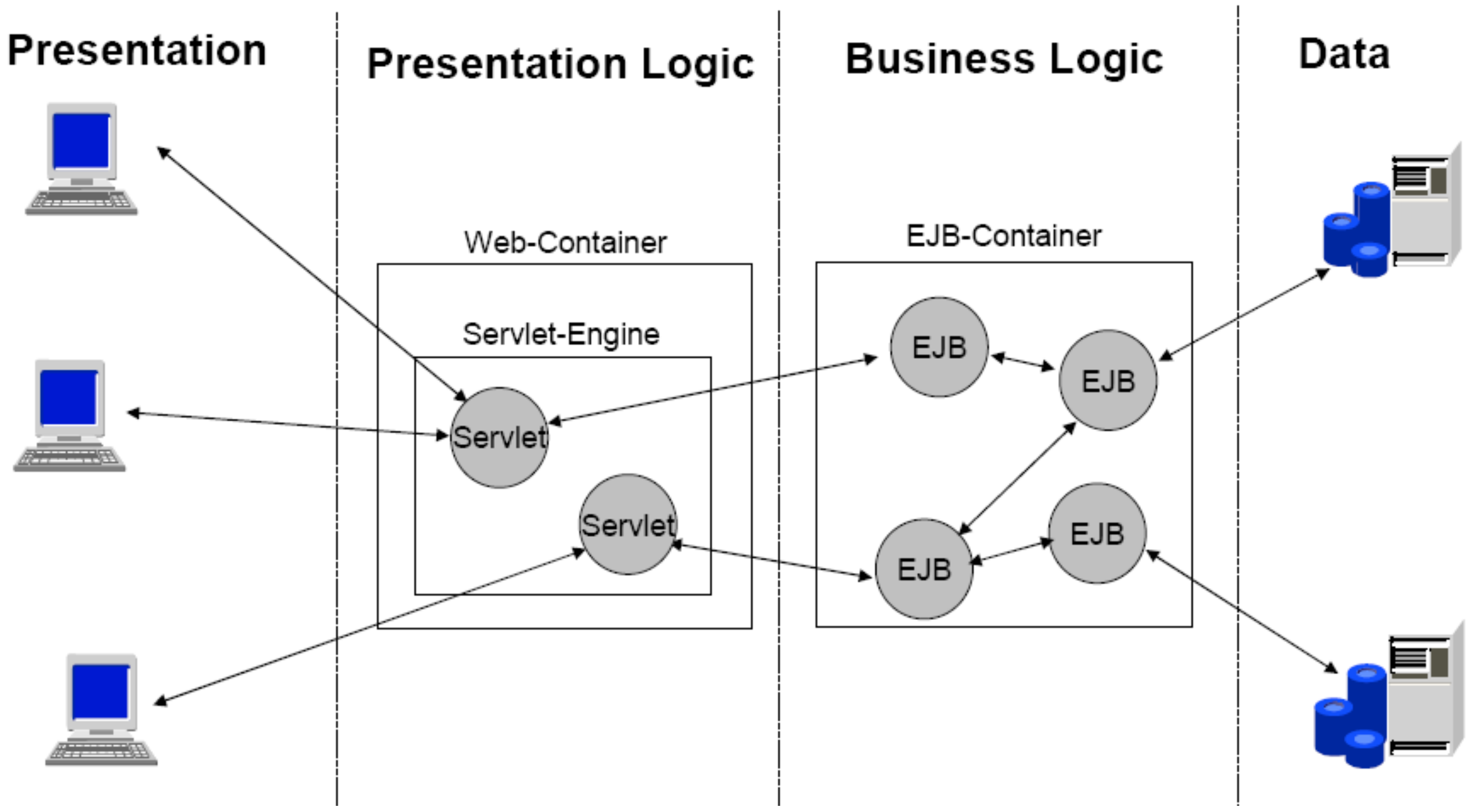
- Where we are



## Enterprise Beans - Overview

- In multi-tier applications an enterpise bean is a server-side component
- Implements (and encapsulates) business logic of an application
- provides system-level services such as transaction management, concurrency control, and security
- EJB technology provides a distributed component model that enables developers to focus on solving business problems
- J2EE platform helps to handle complex systemlevel issues.

# 4-Tier Architecture with EJBs

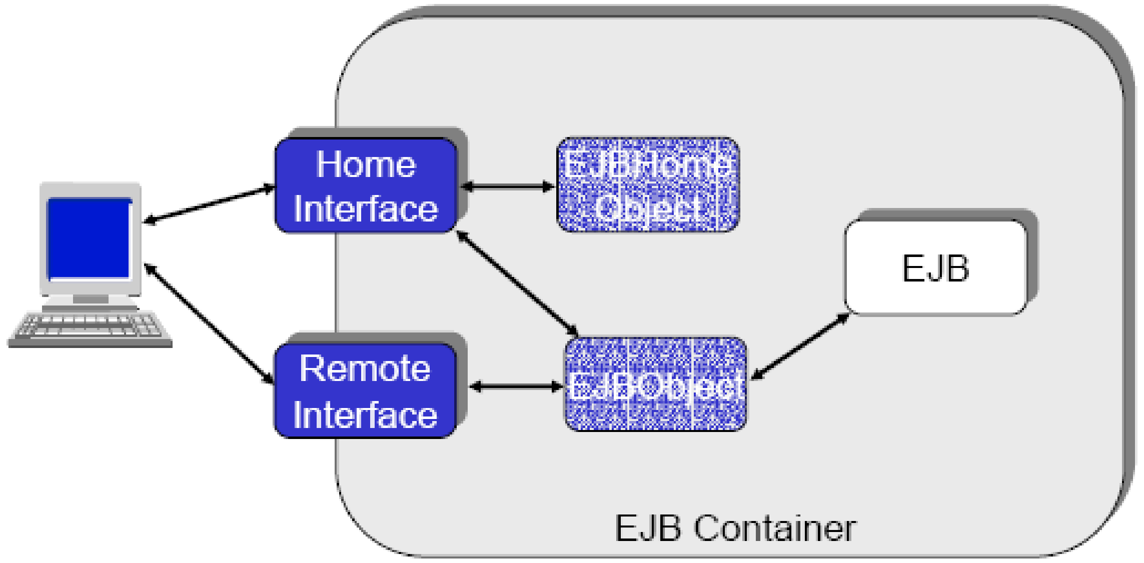


# EJB - Container

- EJBs are created and managed at runtime by a container
- Features of an EJB container
  - Lifecycle management of bean instances
  - Ressource management
  - Remote access to instances
  - Transaction management
  - Security
  - Persistence
  - **Deployment Tools**

# EJB - Components

- Each EJB consists of several components



## Home - Interface

- Manages the lifecycle of EJB instances
- Method types
  - Create – methods, find – methods, remove methods
- Example

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface BMIHome extends EJBHome {
  BMI create() throws RemoteException, CreateException;
}
```

# Remote - Interface

- **Definition** of all business methods (only method signatures, not implementation)
- Example

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface BMI extends EJBObject {
   public double calculateBMI(double weight, double height)
      throws RemoteException;

}
```

# Enterprise Bean Class

- Implements the methods in the Home – Interface
  - For each **create-method** in home interface we need a corresponding **ejbCreate-method**
- Implements each (business-) method defined in the Remote – Interface
- Implements several administrative callback-methods (EJB-Container needs these methods, `ejbActivate()`)
- All these methods are never called by clients directly, but allways via the home- or remote-interface (delegation)

## EJB - Example

```
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class BMIBean implements SessionBean {

    private static final long serialVersionUID = 1L;

    public double calculateBMI(double groesse, double gewicht) {
        return (gewicht / (groesse * groesse));
    }

    public BMIBean() {
    }

    public void ejbCreate() {}
    public void ejbActivate() throws EJBException, RemoteException {}
    public void ejbPassivate() throws EJBException, RemoteException {}
    public void ejbRemove() throws EJBException, RemoteException {}
    public void setSessionContext(SessionContext arg0) throws EJBException,
            RemoteException {}

}
```

## Types of Enterprise Beans

- Three different types of Enterprise Beans
- Session Beans
  - Performs a task (e.g. computations) for a client, often used to model processes
- Entity Beans
  - Represents a business object (e.g. customer) that exists in a persistent storage
- Message Driven Beans
  - Processes messages asynchronously
  - Acts as a listener for the Java Message Service API

# Application clients

- Coding the (web-) client
  - Locating the home interface of an EJB
  - Creating an enterprise bean instance
  - Invoking a business method
- Compiling the client (automatically done by you favorite IDE)
- Packaging the web client
- Specify and ap the client‘s EJB reference
- Deploy everthing
- Run the application

# Application Client (Example)

```
import …

public class BMIEJBServlet extends HttpServlet {

     private static final long serialVersionUID = 1L;

     public void doPost(HttpServletRequest req, HttpServletResponse res)
               throws ServletException, IOException {

          String name = req.getParameter("name");
          String vorname = req.getParameter("firstname");
          String groesse = req.getParameter("groesse");
          String gewicht = req.getParameter("gewicht");
          String fehler = "Fehlermeldung:";

          Double dgroesse = new Double(groesse);
          Double dgewicht = new Double(gewicht);

          double diegroesse = dgroesse.doubleValue();
          double dasgewicht = dgewicht.doubleValue();
          double bmi = 0.0;        // Ausgelagert in die Bean
```

```
      try {

            Context initial = new InitialContext();
            Context myEnv = (Context)initial.lookup("java:comp/env");
            Object objref = myEnv.lookup("ejb/BMIBean");
            fehler = objref.toString();
            BMIHome home =
                        (BMIHome) PortableRemoteObject.narrow(objref,
                        BMIHome.class);
            fehler = fehler + home.toString();

            BMI myBMI = home.create();
            fehler = fehler + myBMI.toString();
            bmi = myBMI.calculateBMI(diegroesse, dasgewicht);
      }
      catch(NamingException e) {fehler = fehler + "NamingException";}
      catch(CreateException e) {fehler = fehler + "NamingException";}



      /* Ausgabe in HTML Seite */

      Double bmiValue  = new Double(bmi);
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<html>");
      out.println("<body>");
      out.println("<head>");
      out.println("<title>Ergebniss dieser BMI - Berechnung</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Hello " +fehler + "..."+ vorname + " " + name +
            " " + dgroesse +" " + dgewicht +"</h1>");
      out.println("<h1>Ihr BMI ist " + bmiValue + "</h1>");
      out.println("</body>");
      out.println("</head>");
      out.close();
}
```

## Summary

- Objectoriented Analysis and Design of an application
- Implementaion of the components (make or buy) by using J2EE technologies
- Testing (components)
- Packging
- Deployment
- „Einsatz“
- Maintenance