

4 Web Programming with Java

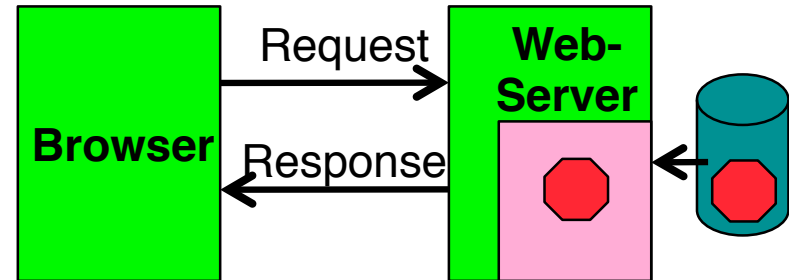
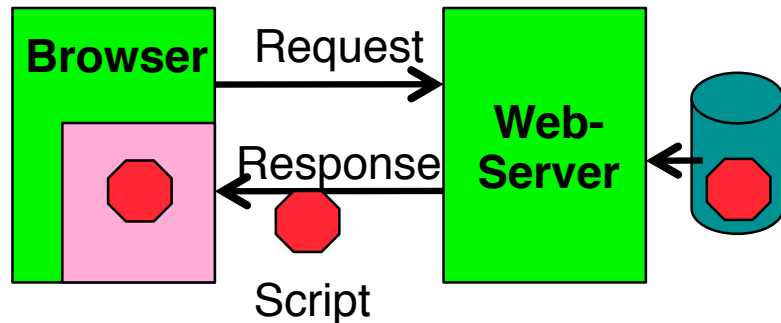
4.1 Client-Side Java: Applets

4.2 Server-Side Java: Servlets

4.3 Java-Based Markup: Java Server Pages (JSP)

4.4 Dynamic Web Applications with Java Server Faces (JSF)

Server-Side vs. Client-Side Realisation



- Client-side realisation:

- Browser contains execution engine for scripts
- Web server does not need to execute scripts
- Script is sent to client as part of server response
- Examples: JavaScript, **Java Applets**

- Server-side realisation:

- Web server contains execution engine for scripts
- Browser does not need to execute scripts
- Script is executed on server and computes response to client
- Examples: PHP, **Java Servlets, Java Server Pages, Java Server Faces**

Applets

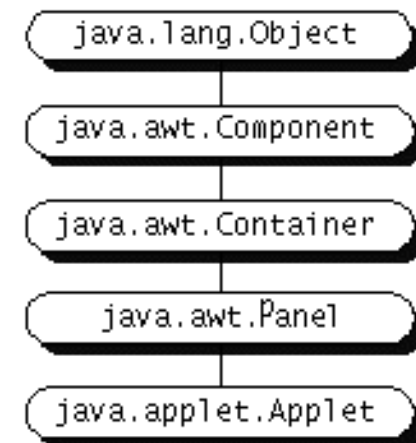
- *Applet*:
 - “application snippet”
 - Java program, embedded in HTML page
 - Executed by browser software
 - » directly or via *plugin*
 - Does not contain a "main()" method!
- *Application*:
 - *Stand-alone* Java program
 - Contains a static "main()" method

Example: Hello-World Applet (1)

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.setFont(new Font("SansSerif", Font.PLAIN, 48));
        g.drawString("Hello world!", 50, 50);
    }
}
```

- Class for applet derived from **Applet**
- **Applet** derived from **Component**
 - Calls `paint` method
 - Redefining the `paint` method means it is executed at display time
- Similar to Java Swing, Java 2D



Example: Hello-World Applet (2) – Old HTML

```
<html>
  <head>
    <title> Hello World </title>
  </head>
  <body>
```

The Hello-World example applet is called: `
`

```
<applet code="HelloWorldApplet.class" width=300>
</applet>
```

```
</body>
</html>
```

java/Applets/HelloWorld.html

This is frequently used but deprecated HTML syntax!

Example: Hello-World Applet (2) – New HTML

```
<html>
  <head>
    <title> Hello World </title>
  </head>
  <body>
```

The Hello-World example applet is called:


```
    <object type="application/x-java-applet"
      height="100" width="400">
      <param name="code" value="HelloWorldApplet" />
    </object>
  </body>
</html>
```

Modern HTML5 syntax

(Note : "classid" not supported in HTML5!)

java/Applets/HelloWorldNew.html

Assuming "HelloWorldApplet.class" exists

Parameter Passing in HTML

Applet:

```
public class HelloWorldAppletParam extends Applet {  
    public void paint(Graphics g) {  
        String it = getParameter("insertedtext");  
        g.setFont(new Font("SansSerif", Font.PLAIN, 48));  
        g.drawString("Hello "+it+" world!", 50, 50);  
    }  
}
```

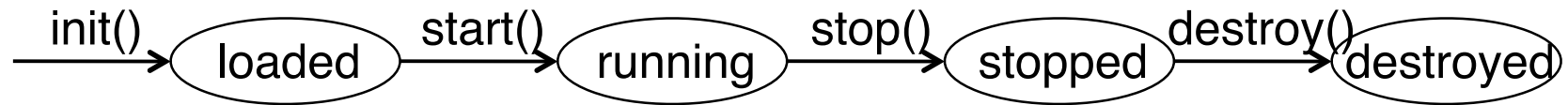
HTML:

This is modern HTML5.

```
<html>  
    ...  
    <br>  
    <object type="application/x-java-applet"  
        height="100" width="800">  
        <param name="code" value="HelloWorldAppletParam" />  
        <param name="insertedtext" value="wonderful" />  
        Java Applets not supported.  
    </object>  
    ...  
</html>
```

Java/Applets/HelloWorldParamNew.html

Applet Life Cycle



Callback methods:

```
public class ... extends Applet {  
    . . .  
    public void init() { . . . }  
    public void start() { . . . }  
    public void stop() { . . . }  
    public void destroy() { . . . }  
    . . .  
}
```


User Interaction in Applets

- Applets are able to react to user input
 - Define an event handler
 - Register during applet initialization (init())
- Applets are executed locally, and therefore have full access to local input
 - Mouse movements, key press, ...
 - This is not possible with server-side code!
- Applets can make use of graphics libraries
 - For instance Java 2D
 - This is not easily possible with server-side code!

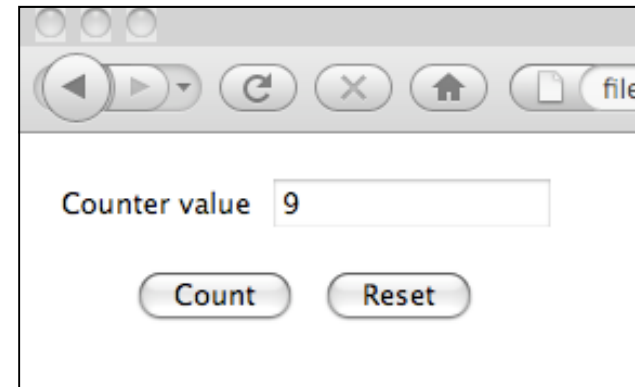
Swing Applets

- Class `javax.swing.JApplet`
 - Derived from `Applet`
 - Is a top level Swing Container
- All Swing GUI components can be used
- Particularities of Swing Applets:
 - Add panels, layout managers etc as with `JFrame`
 - Default layout manager is `BorderLayout`
 - Direct drawing into a Swing applet is not recommended!
 - Redefine method `paintComponent()`
 - Call parent method:

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    . . .
}
```

Example: Counter as Swing-Applet (1)

```
public class CounterSwingApplet extends JApplet {  
  
    CounterPanel counterPanel;  
  
    public void init() {  
        counterPanel = new CounterPanel();  
        add(counterPanel);  
    }  
}  
  
// The View  
class CounterPanel  
    extends JPanel implements Observer {  
  
    private Counter ctr;  
  
    JPanel valuePanel = new JPanel();  
    JTextField valueDisplay = new JTextField(10);  
  
    JButton countButton = new JButton("Count");  
    JButton resetButton = new JButton("Reset");  
    JPanel buttonPanel = new JPanel();  
  
    . . .
```



Counter.html

Example: Counter as Swing Applet (2)

```
public CounterPanel () {                                class CounterPanel (contd.)

    ctr = new Counter();
    valuePanel.add(new Label("Counter value"));
    .
    .
    .
    add(valuePanel, BorderLayout.NORTH);

    countButton.addActionListener(new ActionListener() {
        public void actionPerformed (ActionEvent event) {
            ctr.count();
        }
    });
    .
    .
    .
    ctr.addObserver(this);
}

public void update (Observable o, Object arg) {
    valueDisplay.setText(String.valueOf(ctr.getValue()));
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
}
}

class Counter extends Observable { . . . }
```

Organisation of Bytecode Files

- `<object>` and `<applet>` tags allow
 - Declaration of a "codebase" directory (attribute `codebase`)
 - Declaration of a Java archive (JAR) file (attribute `archive`)
- Advantages of codebase:
 - Java bytecode concentrated at one location
 - Fits with Java file conventions
- Advantages of archives:
 - Less files, less HTTP connections, better performance
 - Lower bandwidth requirements due to (LZW) compression

Applets and Security

- "Sandbox security":
An applet is not allowed to
 - Open network connections (except of the host from which it was loaded)
 - Start a program on the client
 - Read or write files locally on the client
 - Load libraries
 - Call "native" methods (e.g. developed in C)
- "Trusted" Applets
 - Installed locally on the client, or
 - Digitally signed and verified
 - Such applets may get higher permissions, e.g. for reading/writing files
- Execution of applets from locally loaded files is restricted
 - Recent addition
 - Therefore, avoid local loading for tests!

Advantages and Disadvantages of Java Applets

- Advantages:
 - Interaction
 - Graphics programming
 - No network load created during local interactions
 - Executed decentrally – good scalability
- Disadvantages:
 - Dependencies on browser type, browser version, Java version
 - » Persisting problem, leading to many incompatibilities, including recent Java 7 problems
 - Debugging is problematic
 - Java-related security problems (sandbox breaches)

Flashback Trojan botnet infects 600,000 Macs

05.04.2012

siliconrepublic.com

Typical Security Precautions

The image shows two overlapping windows from a Java-enabled browser. The background window is the 'Java' settings page, which allows users to configure which websites can use Java. It lists 'Configured Websites' with entries for 'Local documents', 'localhost', 'java.com', and 'ptolemy.eecs.berkeley.edu', each with an 'Allow Always' button. A warning at the bottom states: '"Java" is set to run in unsafe mode for some websites. In unsafe mode, Java can access your documents and data.' Below this, a dropdown menu is set to 'Ask' for 'When visiting other websites:'. The foreground window is the 'Security' tab of the Java settings, showing 'Enable Java content in the browser' checked and the 'Security Level' slider set to 'Medium'. The 'Very High' level is locked. A note below the slider reads: 'Least secure setting - All Java applications will be allowed to run after presenting a security prompt.' At the bottom of this window are buttons for 'Restore Security Prompts' and 'Manage Certificates...'. Overlaid on the bottom right is a security warning dialog titled 'Do you want to run this application?'. It features a yellow warning triangle icon and text: 'An unsigned application from the location below is requesting permission to run. Location: http://localhost'. A red warning message states: 'Running unsigned applications like this will be blocked in a future release because it is potentially unsafe and a security risk.' Below this is a link for 'More Information'. At the bottom of the dialog are 'Run' and 'Cancel' buttons.

Future of Client Plug-Ins?

- “Companies in general are migrating to pure HTML5 based applications and moving away from plugins such as Flash, Silverlight and Java,” Kandek said. “Java will continue to grow on the server side, where its powerful processing capabilities are absolutely needed.”

Wolfgang Kandek, Qualys Inc. (Vulnerability Management Software)

Source:

<http://www.pcworld.com/article/2030778/researchers-javas-security-problems-unlikely-to-be-resolved-soon.html>

4 Web Programming with Java

4.1 Client-Side Java: Applets

4.2 Server-Side Java: Servlets

4.3 Java-Based Markup: Java Server Pages (JSP)

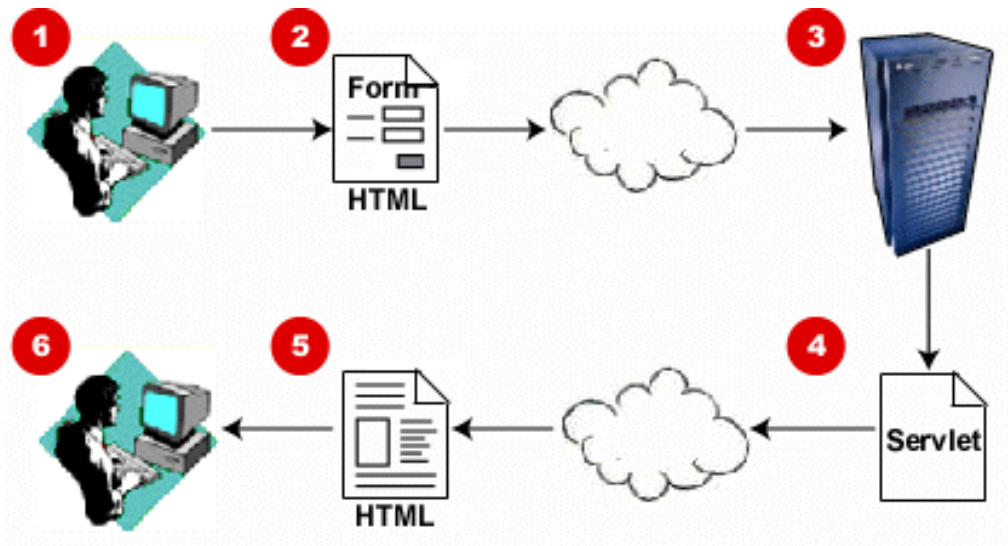
4.4 Dynamic Web Applications with Java Server Faces (JSF)

Literature:

<http://java.sun.com/products/servlet/docs.html>

<http://glassfish.java.net/>

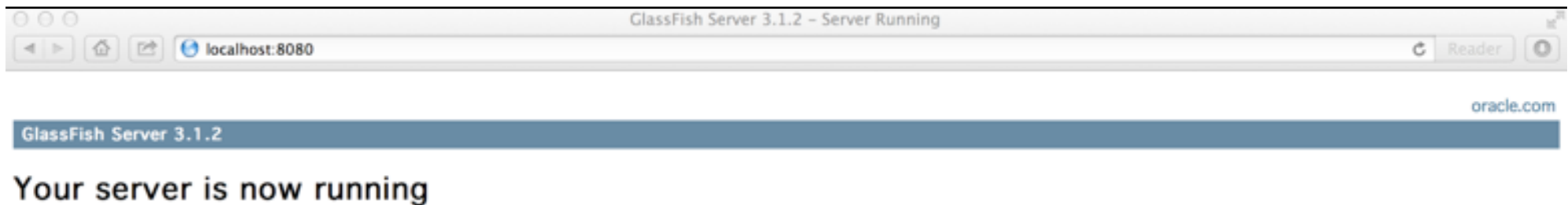
Basic Principle: Server-Side Execution



1. User fills form
2. Form is sent as HTTP request to server
3. Server determines servlet program and executes it
4. Servlet computes response as HTML text
5. Response is sent to browser
6. Response, as generated by servlet, is displayed in browser

Java-Enabled Web Server

- Servlets are part of Java *Enterprise* Edition (Java EE)
- Prerequisite:
 - Web server must be enabled for Java *servlets*
 - » Recognize servlet requests
 - » Administer servlets
 - » Execute servlets (*servlet container*)
- Before doing any experiments:
 - Install Servlet Container software
 - E.g. Apache Tomcat or Oracle GlassFish



Java Servlets

- Java Servlet Specification (JSS):
 - Part of Java Enterprise Edition (EE)
 - First version: 1996 (Java: 1995)
 - Current version: 3.0 (with Java EE 6)
 - Java Server Pages: 1997–1999
 - Java Server Faces: 2001 – ... (Version 2.3: April 2013)
- Reference implementation for a “servlet container”:
 - “GlassFish” (Oracle)
- Other well-known Java EE servers:
 - Apache Tomcat (Catalina), BEA Weblogic (now Oracle), JBoss, jetty
- Basic principle very similar to PHP:
 - Web server calls (Java) servlet code on request from client
 - Servlet determines response to client
 - » Most obvious usage: Produces a HTML page
 - » Other usages: Acts as server-side partner in AJAX-like technologies

Servlet-API: Basics

- **abstract class javax.servlet.GenericServlet**
 - Declares method `service()`
- **abstract class javax.servlet.http.HttpServlet**
 - Subclass of `GenericServlet` for HTTP servlets
 - Defines standard implementation for method `service()`, calls
 - » `doPost()`, `doGet()`, `doPut()`

```
protected void doGet(HttpServletRequest req,  
    HttpServletResponse resp)  
protected void doPost(HttpServletRequest req,  
    HttpServletResponse resp)
```
- **interface javax.servlet.http.HttpServletRequest**
 - Provides information about request, method examples:
`getAttribute()`, `getParameter()`, `getReader()`
- **interface javax.servlet.http.HttpServletResponse**
 - Access to response construction, method examples:
`setContentType()`, `getWriter()`

Example: Hello-World Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Example: Very Simple Dynamic Servlet

HTML page showing current date and time

```
public class myDate extends HttpServlet {  
  
    private static final long serialVersionUID = 11L;  
  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
  
        String title = "Date Servlet Page";  
        response.setContentType("text/html");  
  
        PrintWriter out = response.getWriter();  
        out.println("<html><head><title>");  
        out.println(title);  
        out.println("</title></head><body>");  
        out.println("<h1>" + title + "</h1>");  
        out.print("<p>Current time is: ");  
        out.println(new java.util.GregorianCalendar().getTime());  
        out.println("</body></html>");  
        out.close();  
    }  
}
```

Date Servlet Page

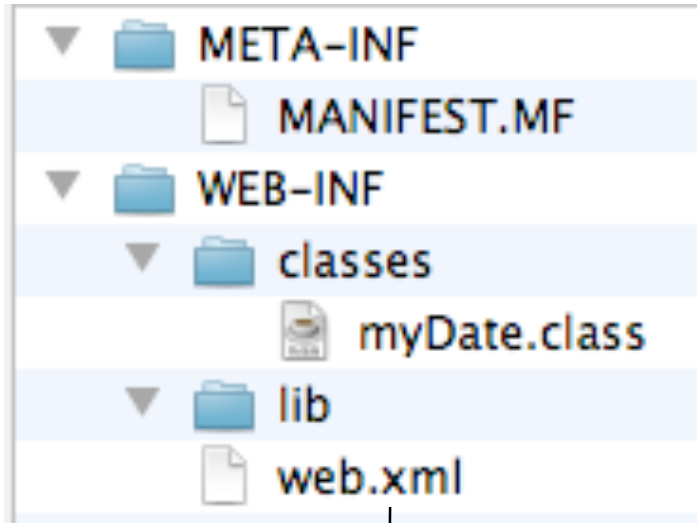
Current time is: Wed Nov 21 17:12:50 CET 2012

<http://localhost:8080/myDateServlet/>

Deployment of Servlet Application

- Servlet is a Java code file (myDate.java)
 - Needs to be compiled (myDate.class)
 - Needs to be made known to the Servlet Container
- *Deployment:*
 - Installation of new server-side java code in the server software
 - Provide a location (directory), called *context path*
 - Provide metadata on the new application
- Usually a Dynamic Web application is archived (with jar) in a single archive file with “.war” extension (Web application archive)
- Several ways for deployment exist
 - E.g. administrative Web interface of GlassFish
 - E.g. through development environments
 - » e.g. NetBeans, Eclipse (with GlassFish Plugin)

File Structure for Deployment



Meta information

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://java.sun.com/dtd/web-app_2_3.dtd">  
<web-app>  
  <display-name>My little Date Application</display-name>  
  <description>  
    Small demo example, by Heinrich Hussmann, LMU.  
  </description>  
  <context-param>  
    <param-name>webmaster</param-name>  
    <param-value>hussmann@ifi.lmu.de</param-value>  
    <description>  
      The EMAIL address of the administrator.  
    </description>  
  </context-param>  
  <servlet>  
    <servlet-name>myDate</servlet-name>  
    <description>  
      Example servlet for lecture  
    </description>  
    <servlet-class>myDate</servlet-class>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>myDate</servlet-name>  
    <url-pattern>/</url-pattern>  
  </servlet-mapping>  
  <session-config>  
    <session-timeout>30</session-timeout>    <!-- 30 minutes -->  
  </session-config>  
</web-app>
```

Administration Interface for Server


User: admin | Domain: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Deploy Applications or Modules

Specify the location of the application or module to deploy. An application can be in a packaged file or specified as a directory.

Location: Packaged File to Be Uploaded to the Server

 myDateServlet.war

Local Packaged File or Directory That Is Accessible from GlassFish Server

Type: *

Context Root:

Path relative to server's base URL.

Application Name: *

4 Web Programming with Java

4.1 Client-Side Java: Applets

4.2 Server-Side Java: Servlets

4.3 Java-Based Markup: Java Server Pages (JSP)

4.4 Dynamic Web Applications with Java Server Faces (JSF)

Literature:

<http://java.sun.com/products/jsp>

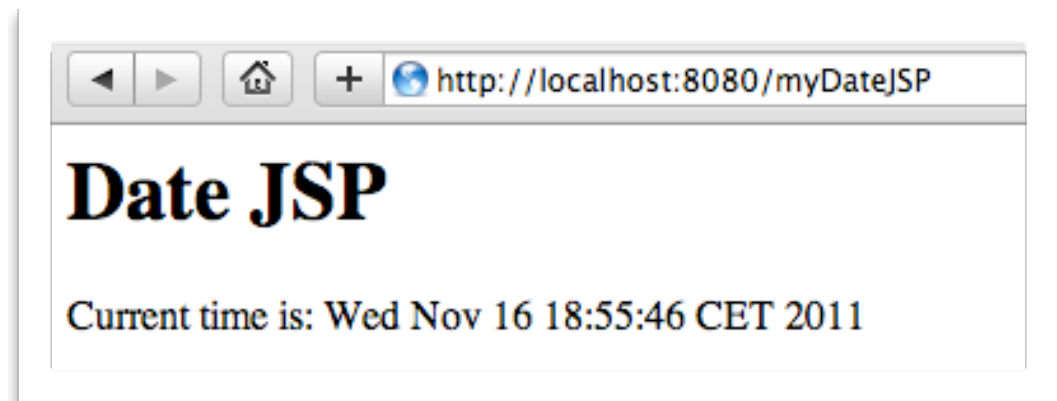
<http://courses.coreservlets.com/Course-Materials/csajsp2.html>

Introductory Example: Java Server Page (JSP)

HTML page with current date/time

```
<html>
<%! String title = "Date JSP"; %>
<head><title> <%=title%> </title></head>
<body>
<h1> <%=title%> </h1>
<p>Current time is:
<% java.util.Date now = new GregorianCalendar().getTime(); %>
<%=now%></p>
</body></html>
```

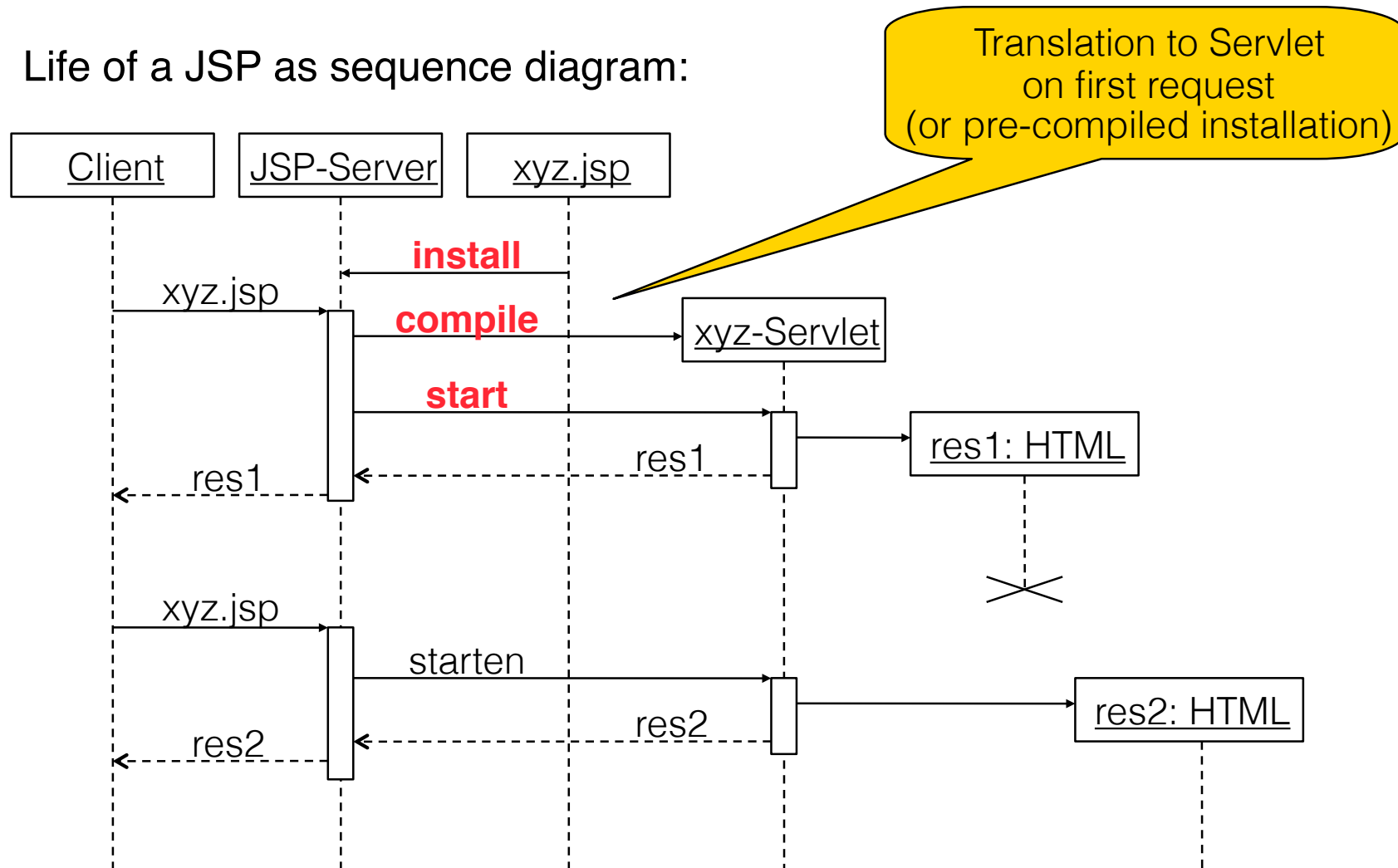
- Basic idea for Java Server Pages:
 - Scripts embedded in HTML ("*Scriptlets*")
 - Automatic translation into Java Servlet code



Java HTML

Java Server Pages und Servlets

Life of a JSP as sequence diagram:



JSP Language Elements

Note: JSP (current version: 2.2) is “deprecated”, to be superseded by JSF!
Java Server Pages specification provides markup language to be compiled into Java Servlets

JSP can be used to generate arbitrary texts, not only HTML

Interesting target language: XML

Language elements:

- Script elements
Embedding of Java code
- Implicit objects
Access to important parts of servlets
- Directives
Global instructions for compilation
- Actions
Standard elements for runtime behavior

Embedding of Scriptlets in HTML

Two options for embedding:

- JSP-specific syntax: Tags with special symbols
 <% , <%! , <%= , <%@ , %> , <%-- , --%>
 – Not elegant, but very practical
- XML-Syntax with *name spaces*
 - » XML name space (xmlns) prefix, e.g. "jsp"
 - » Prefix definition is bound to a URL
 - » Tags take the form <jsp: xyz>
 - » Used for JSP actions in particular

JSP Script Elements

- Declarations
 - Syntax: `<%! declarations %>`
`<jsp:declaration> declarations </jsp:declaration>`
 - Example: `<%! String title = "Date JSP"; %>`
 - Is translated into instance variable of generated class,
i.e. visible in all methods of the class.
- Anweisungen (*Scriptlets*)
 - Syntax: `<% commands %>`
`<jsp:scriptlet> commands </jsp:scriptlet>`
 - Example: `<% java.util.Date now = new
GregorianCalendar().getTime(); %>`
 - Local variables are not visible in other methods.
- Expressions
 - Syntax: `<%= expression %>`
`<jsp:expression> expression </jsp:expression>`
 - Example: `<%= now %>`
 - Equivalent to `<% out.print(now); %>`

Implicit Objects in JSP Scripts

The most important implicit objects:

- **request** (`javax.servlet.http.HttpServletRequest`)
 - To read HTTP headers, parameters, cookies etc. from request
- **response** (`javax.servlet.http.HttpServletResponse`)
 - To write HTTP headers, cookies etc. into the response
- **session** (`javax.servlet.http.HttpSession`)
 - Tracking of associated interactions ("sessions")
- **out** (`javax.servlet.jsp.JspWriter`)
 - Output stream (result test)
 - Standard `print()` and `println()` commands

- Example:

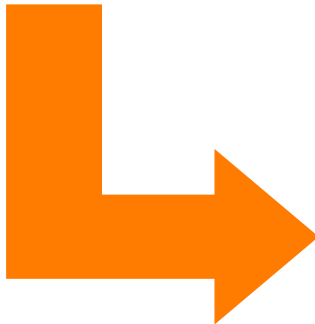
```
<% if (request.getParameter("CountButton") !=null) {  
    counter.count();  
}; %>
```

Generated Servlet Code (Excerpt)

```
<html>
  <%! String title = "Date JSP"; %>
  <head>
    <title> <%=title%> </title>
  </head>
  <body>
    <h1> <%=title%> </h1>
    <p>Current time is:
      <% java.util.Date now = new GregorianCalendar().getTime(); %>
      <%=now%>
    </p>
  </body>
</html>
```

...

```
out.write("\r\n");
out.write("\t<body>\n");
out.write("\t\t<h1> ");
out.print(title);
out.write(" </h1>\n");
out.write("\t\t<p>Current time is:\n");
out.write("\t\t\t");
java.util.Date now = new GregorianCalendar().getTime();
out.write("\n");
out.write("\t\t\t");
out.print(now);
out.write("\n");
```



Cleaning Up the JSP Code

- Mixture between Java scriptlets and HTML markup
 - Is confusing
 - Is difficult to maintain
- Approaches to a better structure of the JSP:
 - Use JavaBeans
 - Use Tag Libraries (Markup tags associated with Java implementation)
 - Use Standard Tag Library (JSTL)
 - Use Expression Language (JSP-EL)
(`${...}` and `#{...}`)
 - Use JSPX (XML syntax, easier to handle for editing tools)
- JSP has evolved into its own, rather complex, programming language

What Is a JavaBean?

- JavaBeans is a *software component* model for Java
 - Not to be confused with Enterprise Java Beans (EJBs)!
- Software components:
 - Units of software which can be stored, transmitted, deployed, configured, executed without knowing the internal implementation
 - Main usage: Tools for composing components
- Driver for JavaBeans technology: User Interfaces
 - AWT and Swing components are JavaBeans
 - GUI editing tools instantiate and configure JavaBeans
- Main properties of a JavaBean:
 - Has a simple constructor without parameters
 - Provides public getter and setter methods for its properties:
getProp, ***setProp*** (no setter = read-only)
 - Is serializable
 - Supports listener mechanism for property changes
 - » *Bound* properties: provide listener for changes
 - » *Constrained* properties: allow listeners to *veto* on changes

JavaBeans in JSP: Action useBean

- Syntax of useBean Aktion:

```
<jsp:useBean id=localName class=className  
             scope=scopeDefn />
```

scope: "page" (current page), "request" (current request);
"session" (current session), "application" (full application)

- Reading properties:

```
<jsp:getProperty name=localName  
                property=propertyName/>
```

- Writing properties:

```
<jsp:setProperty name=localName  
                property=propertyName/  
                value=valueAsString
```

```
<jsp:getProperty name="counter" property="current"/>  
is equivalent to:  
<%=counter.getCurrent(); %>
```

Counter as JavaBean (1)

```
package counter;
```

```
public class CounterBean implements java.io.Serializable {
```

```
    private static final long serialVersionUID = 12L;
```

```
    private int count;  
    private int startValue;  
    private int incrValue;  
    private boolean enabled;
```

```
    /** Creates new CounterBean */  
    public CounterBean() {  
        startValue = 0;  
        incrValue = 1;  
        reset();  
        enabled = true;  
    } ...
```



Counter as JavaBean (2)

```
public void count () {
    if (enabled) {
        count += incrValue;
    }
}

public void reset () {
    count = startValue;
}

public int getCount() {
    return count;
}

public void setCount(int count) {
    this.count = count;
}
... (getters/setters for all local properties) ...
}
```


Counter JSP with JavaBeans: HTML Source

```
<%@ page contentType="text/html" session="true"%>
<html>
<head><title>Counter Demo Page</title></head>
<body>
  <jsp:useBean id="counter" scope="session" class="counter.CounterBean"/>
  <% if (request.getParameter("CountButton")!=null) {
    counter.count();
  };
  if (request.getParameter("ResetButton")!=null) {
    counter.reset();
  }
  %>
  <h2>Counter Demo</h2>
  <p>
    Current counter value =
    <jsp:getProperty name="counter" property="count" /></p>
    <form method="POST" action="CounterJSP.jsp">
      <input name="CountButton" type="submit" value="Count" />
      <input name="ResetButton" type="submit" value="Reset" />
    </form>
  </body>
</html>
```

4 Web Programming with Java

4.1 Client-Side Java: Applets

4.2 Server-Side Java: Servlets

4.3 Java-Based Markup: Java Server Pages (JSP)

4.4 Dynamic Web Applications with Java Server Faces (JSF)

Literature:

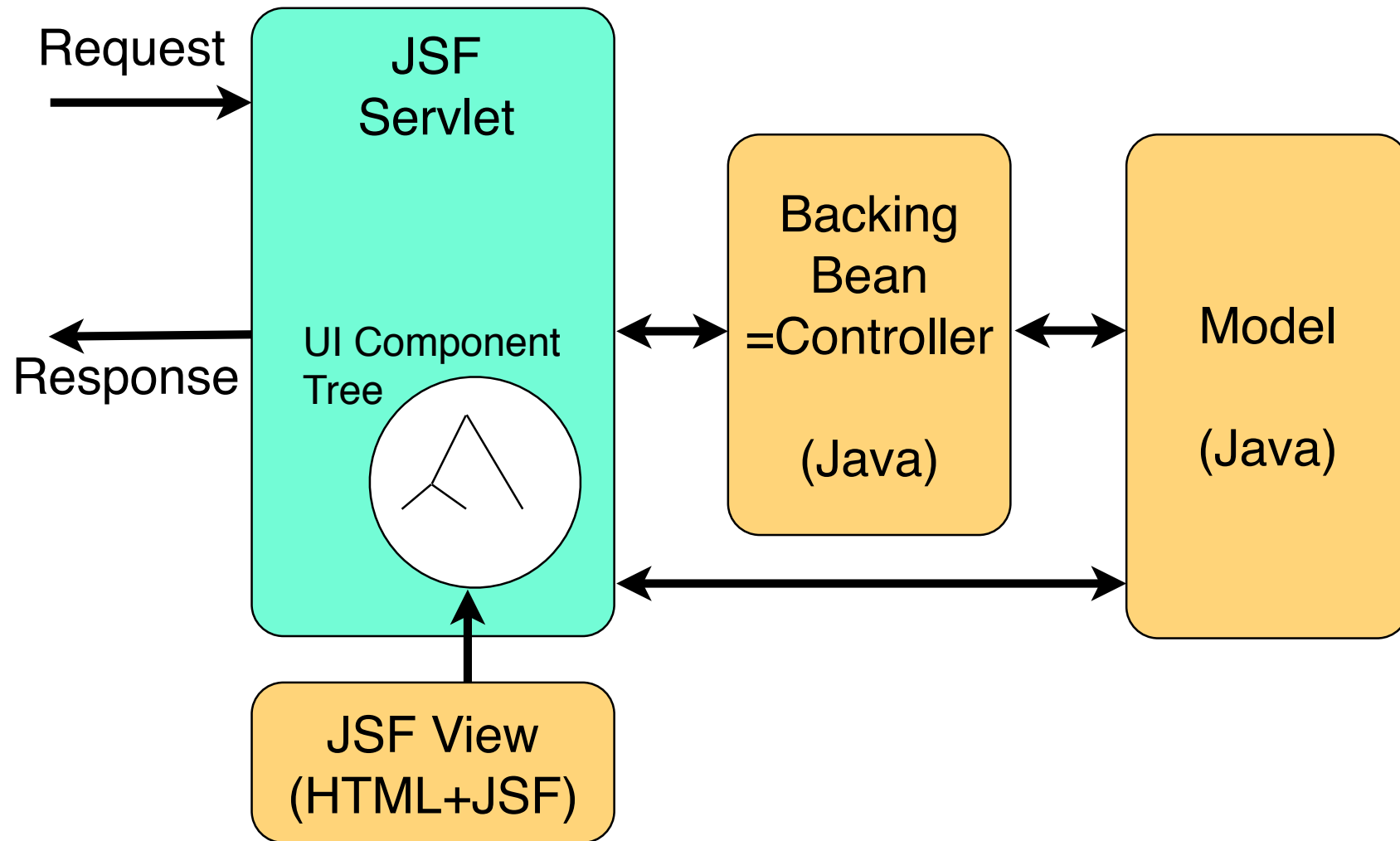
B. Müller: Java Server Faces 2.0, 2. Auflage, Hanser 2010

<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Java Server Faces (JSF)

- Java framework for building Web applications
 - Latest version 2.2 (2013)
 - » 2.0 was a heavy update to previous version
- JSF can be used together with JSP (and other technologies), but also as a separate tool for creating dynamic Web applications
 - JSF is likely to replace JSP in the future
- One single servlet: FacesServlet
 - loads view template
 - builds component tree mirroring UI components
 - processes events
 - renders response to client (mostly HTML)
- JSF follows a strict Model-View-Controller (MVC) architecture

Counter with JSF's MVC Architecture



Example: Model for Counter

- Counter as Java Bean
 - We can use **exactly** the same code as in the JSP example
- Serializable
- All properties exposed as getters and setters

```
▼ counter
  ▼ CounterBean.java
    ▼ CounterBean
      serialVersionUID
      count
      enabled
      incrValue
      startValue
      CounterBean()
      count() : void
      getCount() : int
      getIncrValue() : int
      getStartValue() : int
      isEnabled() : boolean
      reset() : void
      setCount(int) : void
      setEnabled(boolean) : void
      setIncrValue(int) : void
      setStartValue(int) : void
```

Example: View for Counter

```
<!DOCTYPE html>
<html lang="en"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets">
<h:head>
  <title>Counter with Java Server Faces</title>
</h:head>
<h:body>
  <h2>Counter with JSF</h2>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputLabel for="ctrvalue">Counter value = </h:outputLabel>
      <h:outputText id="ctrvalue" value="#{counterController.counterBean.count}"/>
      <h:commandButton value="Count" action="#{counterController.submitCount}" />
      <h:commandButton value="Reset" action="#{counterController.submitReset}" />
    </h:panelGrid>
  </h:form>
</h:body>
</html>
```

JSF Expression Language
(Extension of JSP-EL)

counter.jsf (or .xhtml)

Example: Controller for Counter (1)

```
package counter;
```

```
import javax.faces.bean.ManagedBean;  
import javax.faces.bean.ViewScoped;
```

```
@ManagedBean
```

```
@ViewScoped
```

```
public class CounterController implements java.io.Serializable {
```

```
    private static final long serialVersionUID = 11L;
```

```
    private CounterBean counter;
```

```
    public CounterController() {  
        counter = new CounterBean();  
    }  
}
```

```
...
```

Bean instance name is automatically created from class name (by making lowercase the first letter)

Example: Controller for Counter (2)

```
...  
    public void submitCount() {  
        counter.count();  
    }  
  
    public void submitReset() {  
        counter.reset();  
    }  
  
    public CounterBean getCounterBean() {  
        return counter;  
    }  
}
```



Advantages of JSF

- Clean separation of concerns
 - View (JSF/HTML) just specifies appearance
 - » View contains markup only (+references to beans)
 - Model is clearly separated
 - » Usually, access to persistence layers, databases etc.
 - Usage of controller is enforced
 - » Controller is simple Java Bean
- JSF is fully integrated into Java EE architecture
 - Supports load sharing, transactions etc.

JSF Tag Libraries

- HTML Library (`xmlns:h="http://java.sun.com/jsf/html"`)
 - Forms, input and output
 - Grouping, tables
 - Commands (buttons, links)
 - messages
- Core Library (`xmlns:f="http://java.sun.com/jsf/core"`)
 - Views, subviews
 - Listeners
 - Data converters
 - Validators
 - Internationalization
- Facelets Library (`xmlns:ui="http://java.sun.com/jsf/facelets"`)
 - Component aliasing: Page is written in traditional XHTML
 - Special attribute in HTML tags: Triggers replacement with JSF component
 - Classical HTML editors and other tools can be used!

AJAX Support in JSF

- Special tag in JSF Core Library:

```
<f:ajax>
```

- Ajax tag modifies reaction of components in which it is embedded
 - XMLHttpRequest instead of browser-global request updating the view

- Example:

```
<h:inputText ...>  
    <f:ajax event="valueChange"  
           execute="handler"  
           render= ... />  
</h:inputText>
```

- On value change event (input into text field), specified handler is called
 - on the server, of course (= asynchronous handling of input)
- The *render* attribute can be used to specify the components to be updated after event processing
 - For instance by specifying document parts using “id”s

AJAX + JSF Example

- JSF form with text field and asynchronous reaction to text input:

```
<h:form>
  <h:panelGrid>
    <h:inputText value="#{bean.text}" >
      <f:ajax event="keyup" render="output" />
    </h:inputText>
    <h:outputText id="output" value="#{bean.info}" />
  </h:panelGrid>
</h:form>
```

- Java Bean accepting text and computing info to be displayed:

```
@ManagedBean(name = "bean")
public class Bean {
  private String text;
  private Info info;
  public void setText (String text) {...}
  public Info getInfo() {
    compute info from text
  }
}
```