# Servlets

## Web Applications

- Web servers
  - return files
  - run programs
- Web application: collection of servlets, JSP pages, HTML pages, GIF files, ...
- Servlets: programmed using the servlet API, which is directly based on HTTP
- Lifecycles

| | |
|---|---|
| – application | (shared state) |
| – session | (session state) |
| – interaction | (transient state) |

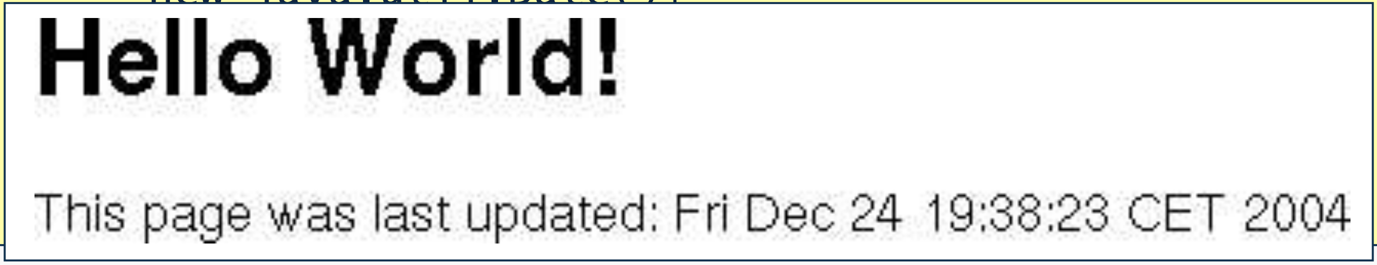
## An Example Servlet

```
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><head><title>ServletExample</title></head>"+
                "<body><h1>Hello World!</h1>"+
                "This page was last updated: "+
                new java.util.Date()+

  }
}
```



# Hello World!

This page was last updated: Fri Dec 24 19:38:23 CET 2004

## Overview

- **Requests and responses**
- Data storage
  - ServletContext
  - Session
- Example: QuickPoll
- Advanced Servlets:
  - Listeners
  - Filters
- Deploying Servlet applications

## Recap: HTTP Requests

```
GET /search?q=Introduction+to+XML+and+Web+Technologies HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.2)
↳  Gecko/20040803
Accept: text/xml,application/xml,application/xhtml+xml,
↳  text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: da,en-us;q=0.8,en;q=0.5,sw;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com/
```

- Request line  (methods: GET, POST, ...)
- Header lines
- Request body (empty here)

# Request methods in Servlets

```
public class Requests extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    // Handle GET requests
  }

  public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    // Handle POSTrequest
  }

  public void doPut(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    // Handle PUT request
  }
}
```

# Requests

- Represented as **`HttpServletRequest`** objects:
  - Full representation of all request data
  - Directly modeled on the HTTP protocol

- Methods in **`HttpServletRequest`**
  - `getHeader`
  - `getParameter`
  - `getInputStream`
  - `getRemoteHost, getRemoteAddr, getRemotePort`
  - ...

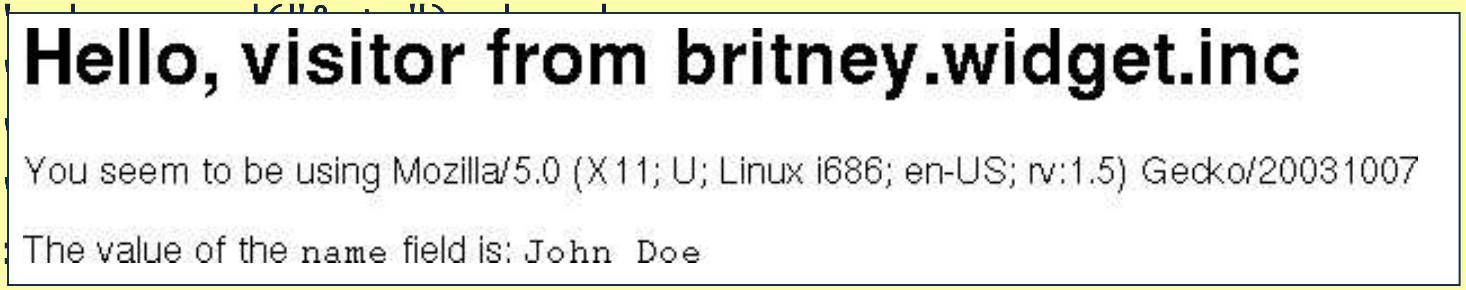
## Example: HttpServletRequest (1/2)

```
public class Requests extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Requests</title></head><body>");
    out.println("<h1>Hello, visitor from "+request.getRemoteHost()+"</h1>");
    String useragent = request.getHeader("User-Agent");
    if (useragent!=null)
      out.println("You seem to be using "+useragent+"<p>");
    String name = request.getParameter("name");
    if (name==null)
      out.println("No <tt>name</tt> field was given!");
    else
      out.println("The value of the <tt>name</tt> field is: <tt>" +
                  htmlEscape(name) + "</tt>");
    out.println("</body></html>");
  }
```

## Example: HttpServletRequest (2/2)

```
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws IOException, ServletException {
   doGet(request, response);
  }

  private String htmlEscape(String s) {
    StringBuffer b = new StringBuffer();
    for (int i = 0; i<s.length(); i++) {
      char c = s.charAt(i);
      switch (c) {
        case '<': b.append("&lt;"); break;
        case '>
        case '"
        case '\
        case '&
        default:
    } }
    return b.toString();
} }
```



Hello, visitor from britney.widget.inc

You seem to be using Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.5) Gecko/20031007

The value of the name field is: John Doe

## Recap: HTTP Responses

```
HTTP/1.1 200 OK
Date: Fri, 17 Sep 2009 07:59:01 GMT
Server: Apache/2.0.50 (Unix) mod_perl/1.99_10 Perl/v5.8.4
↳  mod_ssl/2.0.50 OpenSSL/0.9.7d DAV/2 PHP/4.3.8 mod_bigwig/2.1-3
Last-Modified: Tue, 24 Feb 2009 08:32:26 GMT
ETag: "ec002-afa-fd67ba80"
Accept-Ranges: bytes
Content-Length: 2810
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>...</html>
```

- Status line
- Header lines
- Response body

## Responses

- Represented as **`HttpServletResponse`** objects:
  - getOutputStream/getWriter to write response body
  - No templates, no validations, little help

- Methods in **`HttpServletResponse`**
  - `setStatus`
  - `addHeader, setHeader`
  - **`getOutputStream, getWriter`**
  - `setContentType`
  - `sendError, sendRedirect`
  - ...

## Overview

- Requests and responses
- **Data storage**
  - **ServletContext**
  - **Session**
- Example: QuickPoll
- Advanced Servlets:
  - Listeners
  - Filters
- Deploying Servlet applications

# Servlet Contexts

- One `ServletContext` object for each Web application
- `getServerInfo`
- `getInitParameter`
- ...
- Shared state:
  - `setAttribute(“`*name*`”, `*value*`)`
  - `getAttribute(“`*name*`”)`
  - *don’t use for mission critical data!*

# Sessions

- One `HttpSession` object for each client
  - obtained by `getSession` in the `HttpServletRequest` object
- Session state:
  - `setAttribute(”`*`name`*`”, `*`value`*`)`
  - `getAttribute(”`*`name`*`”)`
- Hides the technical details of tracking users with URL rewriting / cookies / SSL sessions

## Clicker question

- You need to store the name of the client that is currently logged in. Where would you do that?
  - ServletContext
  - [✓ HttpSession]
  - HttpServletRequest
  - HttpServletResponse

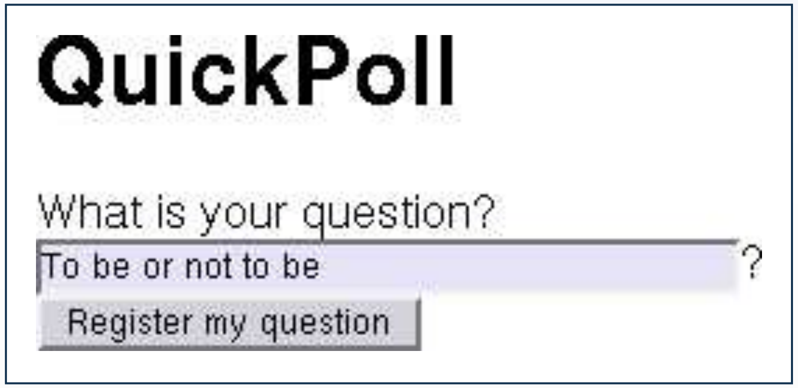
## Overview

- Requests and responses
- Data storage
  - ServletContext
  - Session
- **Example: QuickPoll**
- Advanced Servlets:
  - Listeners
  - Filters
- Deploying Servlet applications

## Example: A Polling Service

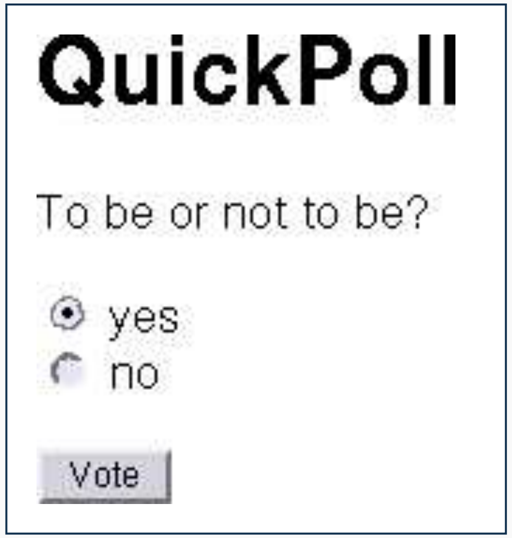
A Web application consisting of

- `QuickPollQuestion.html`
- `QuickPollSetup.java`
- `QuickPollAsk.java`
- `QuickPollVote.java`
- `QuickPollResults.java`


QuickPoll

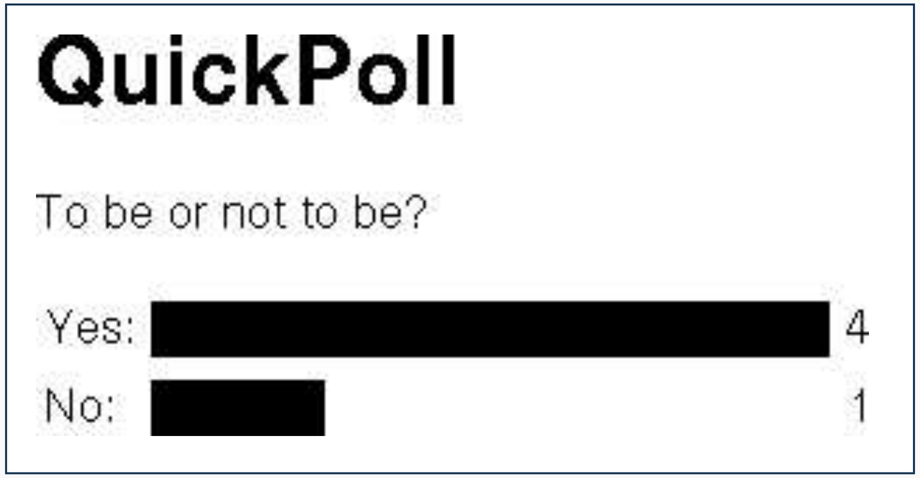
What is your question?

To be or not to be ?


QuickPoll

To be or not to be?

Yes [✓ yes] no


QuickPoll

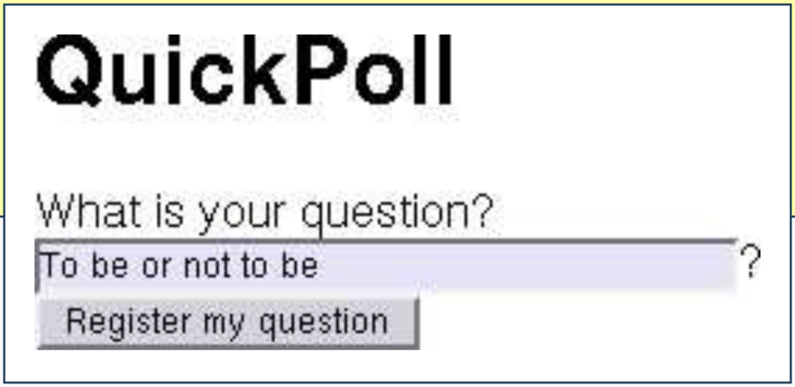
To be or not to be?

Yes: 4

No: 1

## Example: `QuickPollQuestion.html`

```
<html>
<head><title>QuickPoll</title></head>
<body>
<h1>QuickPoll</h1>
<form method=post action=setup>
What is your question?<br>
<input name=question type=text size=40>?<br>
<input type=submit name=submit
       value="Register my question">
</form>
</body>
</html>
```



QuickPoll

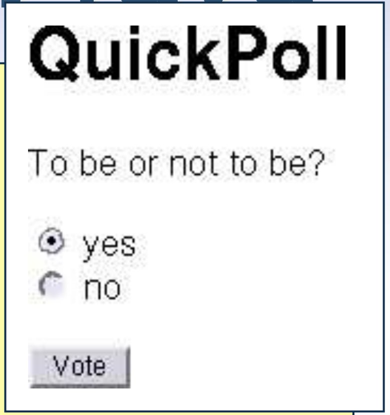
What is your question?

To be or not to be ?

## Example: QuickPollSetup.java

```
public class QuickPollSetup extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws IOException, ServletException {
    String q = request.getParameter("question");
    ServletContext c = getServletContext();
    c.setAttribute("question", q);
    c.setAttribute("yes", new Integer(0));
    c.setAttribute("no", new Integer(0));
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>"+
              "Your question has been registered. "+
              "Let the vote begin!"+
              "</body></html>");
} }
```

## Example: QuickPollAsk.java


QuickPoll

To be or not to be?

[✓ yes] no

```
public class QuickPollAsk extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>"+
              "<form method=post action=vote>");
    String question =
      (String)getServletContext().getAttribute("question");
    out.print(question+"?<p>");
    out.print("<input name=vote type=radio value=yes> yes<br>"+
              "<input name=vote type=radio value=no> no<p>"+
              "<input type=submit name=submit value=Vote>"+
              "</form>"+
              "</body></html>");
} }
```

## Example: `QuickPollVote.java` (1/2)

```
public class QuickPollVote extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws IOException, ServletException {
    String vote = request.getParameter("vote");
    ServletContext c = getServletContext();
    if (vote.equals("yes")) {
      int yes = ((Integer)c.getAttribute("yes")).intValue();
      yes++;
      c.setAttribute("yes", new Integer(yes));
    } else if (vote.equals("no")) {
      int no = ((Integer)c.getAttribute("no")).intValue();
      no++;
      c.setAttribute("no", new Integer(no));
    }
```

## Example: `QuickPollVote.java` (2/2)

```
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.print("<html><head><title>QuickPoll</title></head><body>"+
               "<h1>QuickPoll</h1>"+
               "Thank you for your vote!"+
               "</body></html>");
   }
}
```

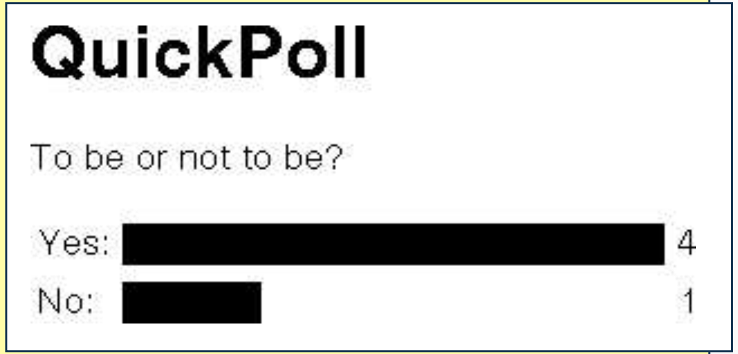
## Example: QuickPollResult.java (1/2)

```
public class QuickPollResults extends HttpServlet {
  public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
      throws IOException, ServletException {
    ServletContext c = getServletContext();
    String question = (String)c.getAttribute("question");
    int yes = ((Integer)c.getAttribute("yes")).intValue();
    int no = ((Integer)c.getAttribute("no")).intValue();
    int total = yes+no;
    response.setContentType("text/html");
    response.setDateHeader("Expires", 0);
    response.setHeader("Cache-Control",
                       "no-store, no-cache, must-revalidate");
    response.setHeader("Pragma", "no-cache");
    PrintWriter out = response.getWriter();
```

## Example: `QuickPollResult.java` (2/2)

```
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>");
    if (total==0)
      out.print("No votes yet...");
    else {
      out.print(question + "?<p>"+"<table border=0>"+
       "<tr><td>Yes:<td>"+drawBar(300*yes/total)+"<td>"+yes+
       "<tr><td>No:<td>"+drawBar(300*no/total)+"<td>"+no+
       "</table>");
    }
    out.print("</body></html>");
  }

  String drawBar(int length) {
    return "<table><tr><td bgcolor=black height=20 width="+
           length+"></table>";
} }
```



QuickPoll

To be or not to be?

| | | |
|---|---|---|
| Yes: | ■■■■■■■■■■ | 4 |
| No: | ■■■ | 1 |

## Problems in QuickPoll

- Need **access control** to `QuickPollSetup`
- No **escaping of special characters**
- Need to **check right order of execution**
- Need to check that expected **form field data** is present
- No **synchronization** in `QuickPollVote`
- Should store state in **database**
- **Redundancy** in HTML generation

# Clicker question

- Are Servlet web applications (as presented here) client-based or server-based applications?
  - Client-based applications
  - [✓ Server-based applications]
  - A hybrid of both

## Overview

- Requests and responses
- Data storage
  - ServletContext
  - Session
- Example: QuickPoll
- **Advanced Servlets:**
  - **Listeners**
  - **Filters**
- Deploying Servlet applications

## Listeners

– also called *observers* or *event handlers*

- `ServletContextListener`
  - Web application initialized / shut down
- `ServletRequestListener`
  - request handler starting / finishing
- `HttpSessionListener`
  - session created / invalidated
- `ServletContextAttributeListener`
  - context attribute added / removed / replaced
- `HttpSessionAttributeListener`
  - session attribute added / removed / replaced

## Example: SessionMonitor (1/2)

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

public class SessionMonitor
    implements HttpSessionListener, ServletContextListener {
  private int active = 0, max = 0;

  public void contextInitialized(ServletContextEvent sce) {
    store(sce.getServletContext());
  }

  public void contextDestroyed(ServletContextEvent sce) {}

  public void sessionCreated(HttpSessionEvent se) {
    active++;
    if (active>max)
      max = active;
    store(se.getSession().getServletContext());
  }
```

## Example: SessionMonitor (2/2)

```
public void sessionDestroyed(HttpSessionEvent se) {
    active--;
    store(se.getSession().getServletContext());
  }

  private void store(ServletContext c) {
    c.setAttribute("sessions_active", new Integer(active));
    c.setAttribute("sessions_max", new Integer(max));
  }
}
```

## Filters

- Code being executed before and after the servlet
  - executed in stack-like fashion with servlet at the bottom
- Can **intercept** and **redirect** processing
  - security
  - auditing
- Can **modify requests and responses**
  - data conversion (XSLT, gzip, ...)
  - specialized caching

– *all without changing the existing servlet code!*

## Example: LoggingFilter (1/2)

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

public class LoggingFilter implements Filter {
  ServletContext context;
  int counter;

  public void init(FilterConfig c) throws ServletException {
    context = c.getServletContext();
  }

  public void destroy() {}
```

## Example: `LoggingFilter` (2/2)

```
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
      throws IOException, ServletException {
    String uri = ((HttpServletRequest)request).getRequestURI();
    int n = ++counter;
    context.log("starting processing request #"+n+" ("+uri+")");
    long t1 = System.currentTimeMillis();
    chain.doFilter(request, response);
    long t2 = System.currentTimeMillis();
    context.log("done processing request #"+n+", "+(t2-t1)+" ms");
  }
}
```

## Clicker question

- How would you implement a component that ensures that a client has logged in?
  - As a Listener
  - As a Servlet
  - [✓ As a Filter]
  - As a static HTML file

## Overview

- Requests and responses
- Data storage
  - ServletContext
  - Session
- Example: QuickPoll
- Advanced Servlets:
  - Listeners
  - Filters
- **Deploying Servlet applications**

## Option 1: Deployment Descriptors

An XML file `web.xml` describing

- mapping from URIs to application resources
- initialization parameters
- security constraints
- registration of listeners and filters

## Example web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee”
         version="3.0">

  <display-name>A Small Web Application</display-name>

  <servlet>
   <servlet-name>MyFirstServlet</servlet-name>
   <servlet-class>HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>MyFirstServlet</servlet-name>
    <url-pattern>/hello/*</url-pattern>
  </servlet-mapping>

</web-app>
```

## Registration of Filters in `web.xml`

```
<web-app ...>
  ...
  <filter>
    <filter-name>My Logging Filter</filter-name>
    <filter-class>LoggingFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>My Logging Filter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  ...
</web-app>
```

# Option 2: (Auto)Magic!

- Using Java annotations:

```
@WebServlet(name="mytest",
    urlPatterns={"/myurl"})
  public class MyServlet extends HttpServlet {
    ....
  }
```

- Only classes in WEB-INF/classes will be discovered

## Option 2: (Auto)Magic!

- Also for Filter and Listener classes:

```
@WebFilter ("/*")
public class FooFilter implements Filter {

    // This filter is applied for requests to /*

}
```

```
@WebServletContextListener
public class Listener implements
    ServletContextListener{

    // Listener implementation
}
```

## Web Applications

A Web app is structured as a directory:

- *`myapp/`*
  - contains HTML/CSS/GIF/... files
- *`myapp/`*`WEB-INF/`
  - contains the ***deployment descriptor*** `web.xml`
- *`myapp/`*`WEB-INF/classes/`
  - contains servlet class files
    (in subdirs corresponding to package names)
- *`myapp/`*`WEB-INF/lib/`
  - contains extra jar files

## The Tomcat Server

- Reference Implementation, Open Source
- `lib/servlet-api.jar`
- `bin/startup.sh`, `bin/shutdown.sh`
- `conf/server.xml`
- `webapps/`*`myapp`*

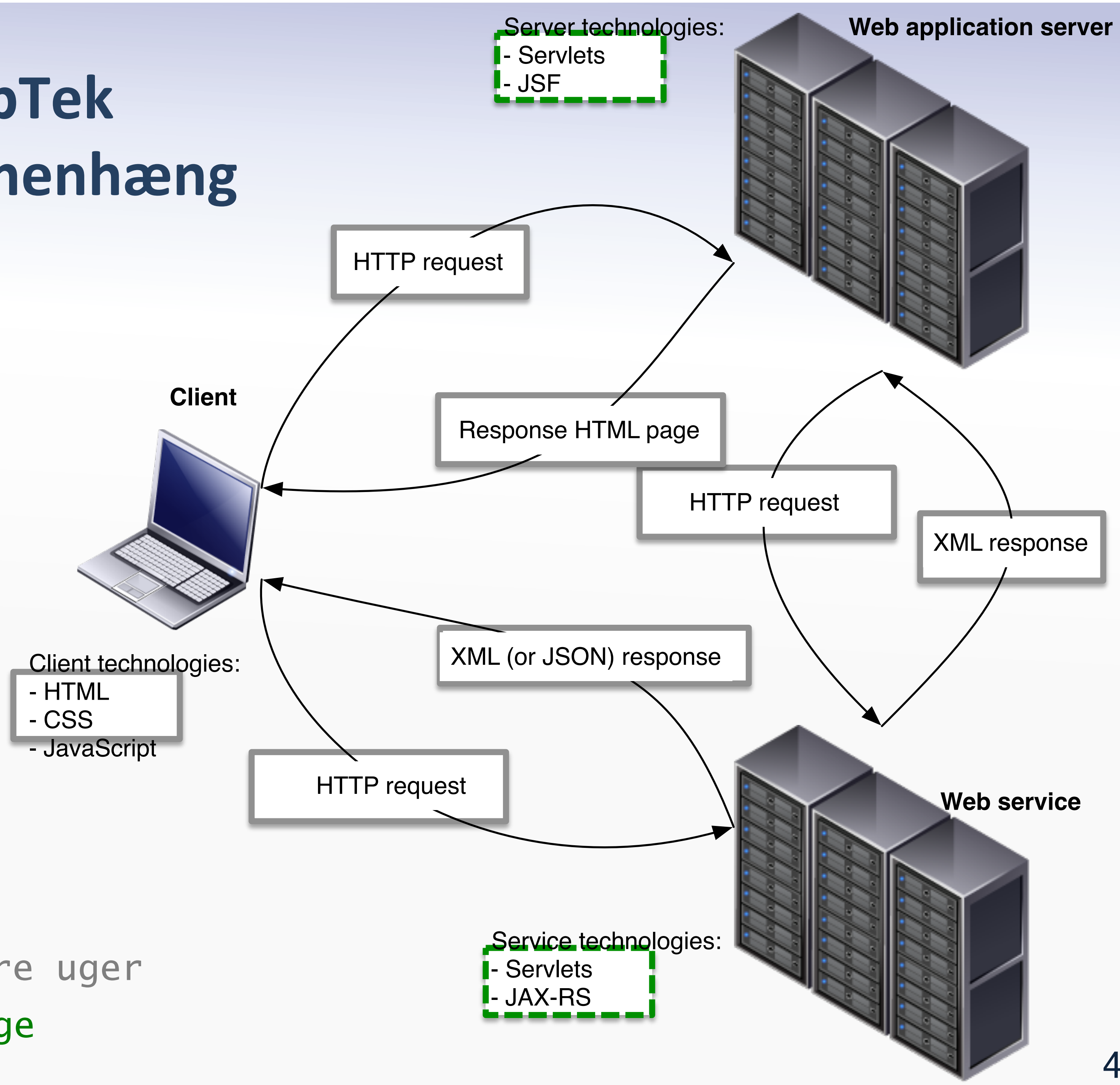
## Overview

- Requests and responses
- Data storage
  - ServletContext
  - Session
- Example: QuickPoll
- Advanced Servlets:
  - Listeners
  - Filters
- Deploying Servlet applications

## Summary

- Servlets closely follow the **request-response** pattern and the structure of HTTP
- Features:
  - Multi-threading
  - Declarative configuration
  - Request parsing, including decoding of form data
  - Shared state
  - Session management
  - Advanced code structuring:  listeners, filters, wrappers
  - Client authentication, SSL

# dWebTek sammenhæng



Tidligere uger

Denne uge

## Online resources

- Oracle Servlet website: http://www.oracle.com/technetwork/java/index-jsp-135475.html (including specification an tutorials)
- Tomcat website: http://tomcat.apache.org/