Your host today is – *Troels C. Damgaard*

Department of Theoretical Computer Science (ITU)

6th of September / IWSJ Fall 2005
A BRIEF “WHO AM I”

2 Introduction

3 Requests and Responses

4 Running Web Applications
   • Deployment

5 Servlet Contexts and Shared State

6 Sessions

7 Advanced concepts

8 Servlets – Pro’s and Con’s

9 Important information

10 Exercises

11 Literature / links
Troels C. Damgaard

- B.Sc. from RUC
- Main interests: High-level programming languages, and calculi and models for mobility and concurrency. (Machine learning and adaptivity in general.)
- Works on Bigraphical Programming Languages (BPL) research project.
- Implemented SML modules for BPL project. See BPL homepage: http://www.itu.dk/research/theory/bpl/
- Homepage: www.itu.dk/people/tcd
A Brief “Who Am I”

INTRODUCTION

Requests and Responses

Running Web Applications
  • Deployment

Servlet Contexts and Shared State

Sessions

Advanced concepts

Servlets – Pro’s and Con’s

Important information

Exercises

Litterature / links
Servlets are written in pure Java using the Servlet API.

- like CGI scripts, Servlets follow the request-response pattern from HTTP.
- allows us to generate responses dynamically based on client request.
- the API provides full access to the HTTP protocol.
- divided into two layers: generic and HTTP-specific – corresponding to the two packages `javax.servlet` and `javax.servlet.http`. 
**Hello World in Servlets**

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

public class HelloWorld extends HttpServlet{
public void doGet(HttpServletRequest req, HttpServletResponse resp)
  throws IOException, ServletException {
  resp.setContentType("text/html");
  PrintWriter out = resp.getWriter();
  out.println("<html><head><title>Servlet</title></head>");
  out.println("<body><h1>Hello World!</h1>");
  out.println("This page was last updated: " + new Date());
  out.println("</body></html>");
}
}
```

**Run Hello World**
• a servlet is a sub-class of HttpServlet.
• the `doGet` method defines the request handler for GET requests.
• the `HttpServletRequest` object contains all information about the request
• the `HttpServletResponse` object is used for generating the response
• a server makes only one instance of each Servlet class, but runs many threads
Useful methods in `HttpServlet` to be implemented in sub-classes:

- `void init()`
- `void doGet(HttpServletRequest, HttpServletResponse)`
- `void doPost(HttpServletRequest, HttpServletResponse)`
- `String getServletName(), String getServletInfo()`
- `void destroy()`
Useful predefined methods:

- `void log(String)`, `void log(String, Throwable)`
- `ServletConfig getServletConfig()` - reads server configuration for this Servlet
- `ServletContext getServletContext()` - explained later...

Exceptions: `ServletException`
OUTLINE

A Brief “Who Am I”

Introduction

REQUESTS AND RESPONSES

Running Web Applications
  • Deployment

Servlet Contexts and Shared State

Sessions

Advanced concepts

Servlets – Pro’s and Con’s

Important information

Exercises

Litterature / links
REQuests

The full request is available in the given `HttpServletRequest`:

- `String getHeader(String), Enumeration getHeaders(String)` - reads HTTP request headers.
- `String getParameter(String)` - parses and decodes querystring (for GET) or body (for POST)
- `InputStream getInputStream()` - for reading raw POST request body
- `String getRemoteHost()` - returns client IP address
- ...

Troels C. Damgaard (ITU)
public class Requests extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp) 
        throws IOException, ServletException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html><head><title>Requests</title></head><body>");
        out.println("<h1>Hello, visitor from " + req.getRemoteHost() + "</h1>");
        String useragent = req.getHeader("User-Agent");
        if (useragent!=null)
            out.println("You seem to be using " + useragent + "<p>");
        String name = req.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>");
    }
    ...
}
The response is generated using the given `HttpServletResponse`:

- `void addHeader(String name, String value)` - add name/value pair to header.
- `void setStatus(int sc)` - set status code (default: `SC_OK = 200`).
- `void sendError(int sc, String msg)` - send error reply.
- `void sendRedirect(String url)` - redirect browser to given page.
- `ServletOutputStream getOutputStream()` - output stream for response body.

- ...

– always make the header *before* sending the response body!
CONSTRUCTING A RESPONSE

public class Responses extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException{
        long expires = new Date().getTime() + 1000*60*60*24;
        resp.setContentType("text/html");
        resp.addDateHeader("Expires", expires);
        ServletOutputStream out = resp.getOutputStream();
        out.println("<html><head><title>Servlet Response</title></head><body>");
        out.println("<h1>Todays News</h1>");
        out.println(getNews());
        out.println("<p><hr>");
        out.println("<i>This news item expires " + new Date(expires));
        out.println("</body></html>");
    }
    ...
}
**ADDNG doGet**

```java
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException{
    resp.sendError(resp.SC_METHOD_NOT_ALLOWED, "Que? - No habla POST!");
}
```

**Run Responses**

- `setContentType` and `addDateHeader` are convenience methods on top of `setHeader` and `addHeader`.
- **Warning**: Browsers usually cache responses to GET request - always set the `Expires` header appropriately.
- Use `response.addDateHeader("Expires", 0)` to disable caching.
A Brief “Who Am I”

Introduction

 Requests and Responses

4 Running Web Applications
   • Deployment

5 Servlet Contexts and Shared State

6 Sessions

7 Advanced concepts

8 Servlets – Pro’s and Con’s

9 Important information

10 Exercises

11 Literature / links
Servers

- Apache’s **Tomcat 5.5** - the official reference implementation for Servlets 2.4 and JSP 2.0 (and Open Source!)
- Sun **Java System Web Server 6.1** (Sun’s commercial alternative)
- Macromedia’s **JRun**
- New Atlanta’s **ServletExec**
- Caucho’s **Resin**
- Gefion Software’s **LiteWebServer** (somewhat old, but might be a lightweight alternative if you feel like playing around)
- Trifork’s **Enterprise Application Server** (Danish, **yay**!)
- …and any more (by now, you should know how to do a:
  
  `http://www.google.dk/search?q=java+application+server`)

Installation and server **configuration** are of course implementation dependent, but application **deployment** is essentially the same.
A Servlet Web application is structured in directories as follows:

**DIRECTORY STRUCTURE**

- `myapp/` contains auxiliary files (e.g. HTML, GIF, CSS, JSP files), can be in sub-dirs.
- `myapp/WEB-INF/` contains deployment descriptor, `web.xml`.
- `myapp/WEB-INF/classes/` contains Servlet class files (in appropriate sub-dirs, if non-default package names).
- `myapp/WEB-INF/lib/` contains extra jar files.

Using the normal `jar` tool, a complete Web application can be wrapped into a portable Web Archive (`.war`).
The deployment descriptor: `web.xml` provides control of – among other:

- URL mapping (from URLs to files)
- initialization parameters
- error pages
- security constraints, client authentication
- filters and listeners (*more later*)
AN EXAMPLE DEPLOYMENT DESCRIPTOR FOR HelloWorld

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>TCD’s web-apps</display-name>
    <description>
        This directory contains various example web-apps for the course IWSJ (Fall 2005).
    </description>
    <context-param>
        <param-name>some-servlet-context-param</param-name>
        <param-value>Hip</param-value>
    </context-param>
    <servlet>
        <servlet-name>helloworld</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
        <init-param>
            <param-name>some-http-servlet-param</param-name>
            <param-value>Hop</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>helloworld</servlet-name>
        <url-pattern>/helloworld</url-pattern>
    </servlet-mapping>
</web-app>
```
For smaller applications, the default deployment descriptor is probably sufficient. The default mapping from URLs to files is:

- **Servlets:***
  
  http://HOST:PORT/myapplication/servlet/package.servletclass
  (omit “package.” if default package)

- **auxiliary files:*** http://HOST:PORT/myapplication/file

**Warning:**

- **initialization parameters** are not available to servlets accessed by their default URL.
On installing and deploying your first program in Apache Tomcat Server:

- if you haven’t already gotten it up and running – see the excellent step-by-step HowTo by Henning at:
  
  
  (URL is posted at course-website).
- **Note**: I can also recommend the Windows installer, but you need administrator rights to use it, so you cannot use at ITU(!)
- **Note**: If anybody wishes so, I’ll make a zip-file with the HelloWorld program with the correct directory layout and this descriptor available...?
OUTLINE

1. A Brief “Who Am I”
2. Introduction
3. Requests and Responses
4. Running Web Applications
   - Deployment
5. **Servlet Contexts and Shared State**
6. Sessions
7. Advanced concepts
8. Servlets – Pro’s and Con’s
9. Important information
10. Exercises
11. Literature / links
Each Web application has a `ServletContext` object (returned by `getServletContext()`).

- contains a server log (accessible via `log(String msg)` method)
- access to initialization parameters set in the deployment descriptor (via `getInitParameter(String name)` and `getInitParameterNames()` which returns an `Enumeration`)
- and access to other (global) server data (e.g. `getMajorVersion()` and `getServerInfo()`).
- other Web application’s servlet contexts are accessible (can be restricted)
- allows data to be shared using attributes

The last property allows us to use the `getServletContext()` for shared state – both within an application and across several applications (although we might also like to restrict the latter).
**ContextObject** ATTRIBUTES

- void setAttribute(String name, Object object)
- Object getAttribute(String name)
- any object can be stored - and freely modified.
- need to downcast when get’ing...

**Alternatives to ServletContext:**

- **JDBC** - API for connecting to SQL databases
- **JDOM/JAXP** - APIs for manipulating XML documents
- If you’re JDOM and JAXP knowledge is rusty, the tutorial http://www.brics.dk/~amoeller/XML/programming/ is recommended (this material has later been adapted for Part I of the book on which your compendium is based).
Let’s run through the interaction patterns of one the larger example in the book:

**DEPLOYMENT DESCRIPTOR**

```xml
<web-app>
  ...
  <servlet>
    <servlet-name>ask</servlet-name>
    <servlet-class>QuickPollAsk</servlet-class>
  </servlet>
  ...
  <servlet-mapping>
    <servlet-name>ask</servlet-name>
    <url-pattern>/ask</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```
REGISTER A QUESTION (quickpoll/index.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>

Run: Register a question
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>");
    }
}
Ask the question (quickpoll/ask)

```java
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) request.getAttribute("question");
        out.print(question + "?
";
        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">
" +
        
```

Run: Ask the question
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));
        }
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<html><head><title>QuickPoll</title></head><body>"
                + "<h1>QuickPoll</h1>" +
                "Thank you for your vote!" +
                "</body></html>");
    }
}

Run vote by itself
The Quick Poll Service (cont.)

See the results of the vote (quickpoll/results)

```java
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();
        ...
        // produce output
        ...
    }
}
```

Run: See the results of the vote
## State in Servlets

<table>
<thead>
<tr>
<th>State type</th>
<th>… in Servlets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>ContextObject attributes</td>
</tr>
<tr>
<td>Shared (appl.)</td>
<td>ContextObject attributes(*)</td>
</tr>
<tr>
<td>Session</td>
<td>HttpSessionObject attributes</td>
</tr>
<tr>
<td>Transient</td>
<td>Local vars in do-methods</td>
</tr>
</tbody>
</table>

(*): We can also use **servlet class fields** to save shared application state, but
- we have to handle possible synchronization issues manually, and
- one instance of a class is loaded pr. instance of a servlet (using the deployment descriptor).
Session state is maintained in an HttpSession object:

- `request.getSession(true)` – returns the current HttpSession object or generates a new session object
- `boolean isNew()` – returns true if session is new
- `Object getAttribute(String name)` – reads per-session state
- `void setAttribute(String name, Object value)` – writes per-session state
- `String getId()` – session unique identifier
- `long getCreationTime()`
- `long getLastAccessedTime()`
- `void setMaxInactiveInterval(int seconds)`
- ...(see the javadoc)

– we note that this resembles the management of shared state.
A TYPICAL EXAMPLE...

```java
... HttpSession session = request.getSession(true);
ShoppingCart cart;
if (session.isNew()) {
    cart = new ShoppingCart();
    session.setAttribute("shoppingcart", cart);
}
cart = (ShoppingCart) session.getAttribute("shoppingcart");
addItemToCart(item, cart);
...```

Session flow control is based on

- `isNew()` method checks
- registering state in `HttpSession` attributes
Under the hood: Uses **cookies** or **URL rewriting** (see section 8.2 in the compendium).

Since URL rewriting may be used, URLs to our own Servlets should always be passed through `response.encodeURL(String)`(!)

Cookies can also be controlled manually – using `addCookie` and `getCookie` of `HttpServletRequest` and `HttpServletResponse` respectively).

This is a rather low-level approach - the session flow is not explicit in the code! (We shall return to this subject when considering **JWIG**).
Listeners

- also called event handlers
- based on the Design Pattern concept Observer’s.

Overall idea:
- register a particular class to handle events like initialization attribute updates
- decouples place of event occurrence from actual action to be taken
- provides more easily refactorable code

For web applications registering listeners is done in the deployment descriptor (i.e. in web.xml).
**LISTENERS (cont.)**

**Events**

- **Servlet context events** (*ServletContextListener*):
  - initialize
  - destroy

- **Servlet context attribute events** (*ServletContextAttributeListener*):
  - set
  - add
  - remove

- **Session events** (*HttpSessionListener*):
  - create
  - invalidate
  - time out

- **Session attribute events** (*HttpSessionListener*):
  - set
  - add
  - remove
Basic idea: Implement the appropriate interface, and register your listener, then you’re up and running.

**Example Listener**

```java
public class DataListener implements ServletContextAttributeListener{
    public void attributeReplaced(ServletContextAttributeEvent event) {
        if (event.getName().equals("some_data"))
            updateSomeOtherData();
    }
    ...
}
```

**...and Deployment Declaration**

```xml
<listener>
    <listener-class>DataListener</listener-class>
</listener>
```
Overall, listeners can be said to be useful for

- implementing dependencies between data (but only if data is modified explicitly via `setAttribute`
  - and not via some methods/fields in the data)
- monitoring the running service

(For another nice example, which illustrates easy refactoring, see the compendium.)
FILTERS

- inserting **hooks** before and after requests are processed
- **wrappers** modify the request and the response

A **Filter** can modify
- the incoming request, e.g.
  - redirect
  - check security requirements
  - perform logging
- the outgoing response, e.g.
  - data compression or encryption
  - XSLT transformation
  - perform logging
**FILTERS (cont.)**

**SMALL EXAMPLE FILTER ...**

```java
public class TraceFilter implements Filter {
    private ServletContext context;

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

    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
            throws IOException, ServletException {
        context.log("["+request.getRemoteHost()+"] request: "+
                     ((HttpServletRequest) request).getRequestURL());
        chain.doFilter(request, response);
        context.log("["+request.getRemoteHost()+"] done");
    }
}
```

...and deployment declaration (on next slide).
AND DEPLOYMENT DECLARATION

<filter>
  <filter-name>myfilter</filter-name>
  <filter-class>TraceFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>myfilter</filter-name>
  <url-pattern>/ */</url-pattern>
</filter-mapping>
Filter objects: multiple filters are processed in order of deployment (in the descriptor) in a stack discipline with the actual Servlet in the bottom. Supplementary, HttpServletRequestWrapper and HttpServletResponseWrapper provide wrappers to modify the request/response (— again a classic design pattern).

Idea:

- Construct a wrapper class with the original request/response as an argument,
- and implement methods for the wrapper class possibly using or disregarding methods or data in the original request/response.
Request Dispatchers

Services are often composed of many Servlets (and JSP pages) working together. We can forward a request to another Servlet using a RequestDispatcher.

**Typical Example of Forwarding**

```java
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/my_other_app/servlet/SomeServlet");
dispatcher.forward(request, response);
```

As usual, use `request.setAttribute(String, String)` to supply extra data to the new handler:

- **forward** - complete transfer of control to another Servlet
- **include** - insert result from running another Servlet (another kind of "wrapping")
A Brief Look at Security

Authentication and encryption can be controlled

- **declaratively** - using the deployment descriptor (*web.xml*)
- **operationally** - by explicit service code
- or a combination

Basic idea for implementing security:

1. make file with usernames, passwords, and roles
2. design login and login-failure pages
3. specify URLs that require authentication (and optionally, also SSL)
**A Brief Look at Security (cont.)**

**Authentication**
- form-based
- HTTP Basic

**SSL**
- requires the JSSE package
- requires a public-key server certificate (details are server specific)
HttpServletRequest contains useful security methods, for programmatic support to security:

**PROGRAMMATIC SUPPORT**

- `String getRemoteUser()` - who has logged in?
- `boolean isUserInRole(String role)` - what is the user’s abstract role?
- `boolean isSecure()` - does the connection use SSL?
OUTLINE

1. A Brief “Who Am I”
2. Introduction
3. Requests and Responses
4. Running Web Applications
   • Deployment
5. Servlet Contexts and Shared State
6. Sessions
7. Advanced concepts
8. SERVLETS – PRO’S AND CON’S
9. Important information
10. Exercises
11. Literature / links
So what’s all the fuss about?

Pro’s

Lot’s of features added comparing with the socket-based approach. We have:

- multi-threading,
- declarative configuration of web applications
- convenience methods for request parsing
- shared state (servlet contexts)
- higher-level session management
- direct support for employing nice design patterns for structuring our code
So what’s all the fuss about? (cont.)

Con’s

But couldn’t we do better?

1. We still construct HTML based on appending java String’s to the outputstream – very low-level – not easy to see or guarantee that either
   ▶ wellformed HTML, or
   ▶ valid (according to some schema)
   is produced.

2. Quite hard to separate design issues from programming issues – can be alleviated to some extent by employing MVC-pattern (which we shall return to).

3. Session flow is not obvious from code – as a consequence:
   ▶ Connection between code generating HTML form and code handling form field data is not explicit. Form field data read using getParameter might not be present!
   ▶ Using attributes for registering state is vulnerable. Can become necessary to check whether some attributes have been set in an earlier interaction.
We shall return to some of these issues, when discussing JWIG later on.
A Brief “Who Am I”
Introduction
Requests and Responses
Running Web Applications
  • Deployment
Servlet Contexts and Shared State
Sessions
Advanced concepts
Servlets – Pro’s and Con’s

**IMPORTANT INFORMATION**

Exercises

Literature / links
As both Henning and I are unfortunately away on a conference on the 25th there will be no (regular) lecture that day.

Further info at the next lecture.
INSTRUCTORS

- Abdul Khader (khabf04@itu.dk)
- Priyadarsini Seetharaman (itspri@itu.dk)

Remember exercise class-rooms are: 4A56 and 4A58.
EXERCISE 2

In recommended order:

- 9.1, 9.2, 9.3 (exc. (a)), 9.6, 9.8 (all in IWSJ compendium)
- Code examples from the book are accessible at the companion website for the book (see course homepage for link)
- **but** the code-example for the exercises are *not* available. Hence, we have the possibility for an extra pedagogical exercise:

**In exc. 9.2 we ask you to code a servlet** `scope.java`, which stores values in the `ServletContext`, `SessionContext` and a local variable – and then consider the question in the original exercise.
OUTLINE

1. A Brief “Who Am I”
2. Introduction
3. Requests and Responses
4. Running Web Applications
   - Deployment
5. Servlet Contexts and Shared State
6. Sessions
7. Advanced concepts
8. Servlets – Pro’s and Con’s
9. Important information
10. Exercises
11. LITTERATURE / LINKS
These slides are an adapted and updated version of the slides used for the course IWSJ E2004, taught by guest teacher Michael Schwartzbach from BRICS, Aarhus, and Henning Niss (ITU).

The slides from last year are available as an online developed by Michael Schwartzbach and Anders Møller (also BRICS) at [http://www.brics.dk/~amoeller/WWW/index.html](http://www.brics.dk/~amoeller/WWW/index.html).

The material in the tutorial have been adapted and used as a basis for the course book that we use.

Though slightly outdated here and there, the webpages can still be recommended as they contain a number of links to other interesting material (e.g. other tutorials by the authors).
If you’re JDOM and JAXP knowledge is rusty, the tutorial http://www.brics.dk/~amoeller/XML/programming/ is recommended (this material has later been adapted for Part I of the book on which your compendium is based).