

# Web 2

week 2: Servlet and JSP

# AGENDA

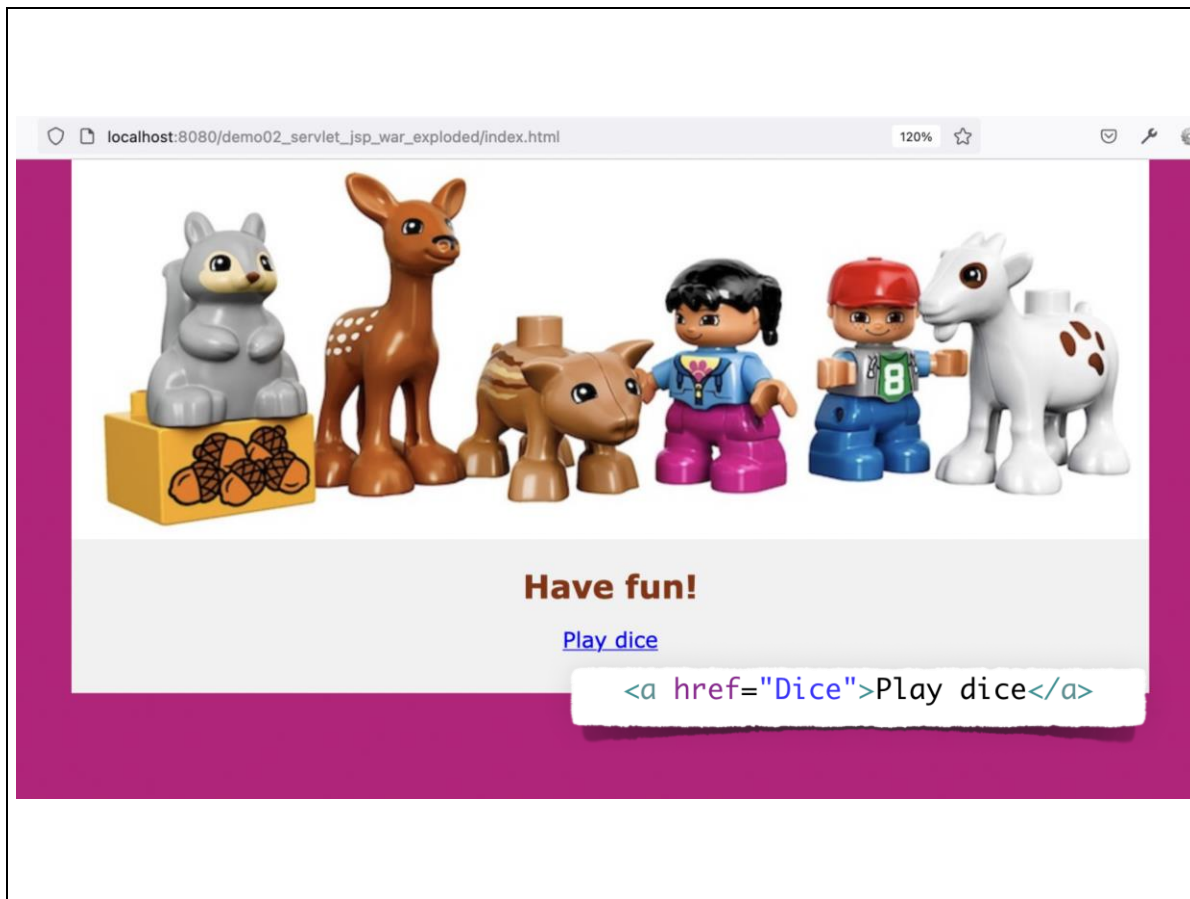
- ❑ HTTP Server vs. Web Container
- ❑ Servlet schrijven
- ❑ Request handling
- ❑ JSP
- ❑ Foutopsporing

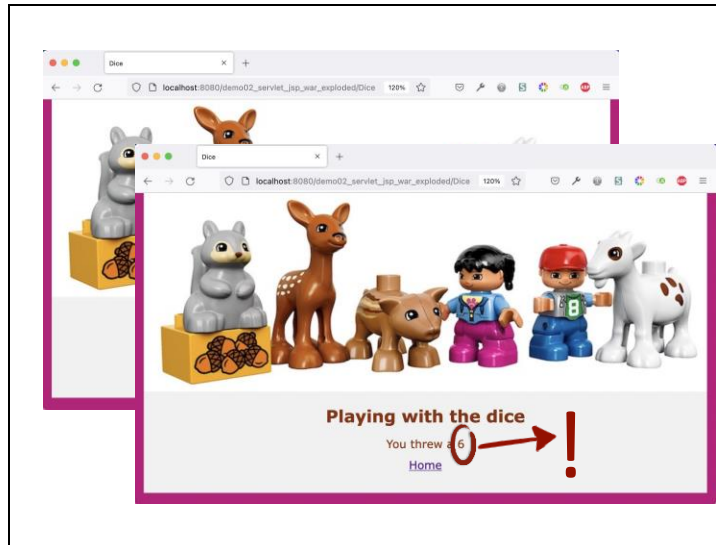


# PLAYING DICE



Maak een **WEB** applicatie die bij klik op de link de dobbelsteen laat rollen.

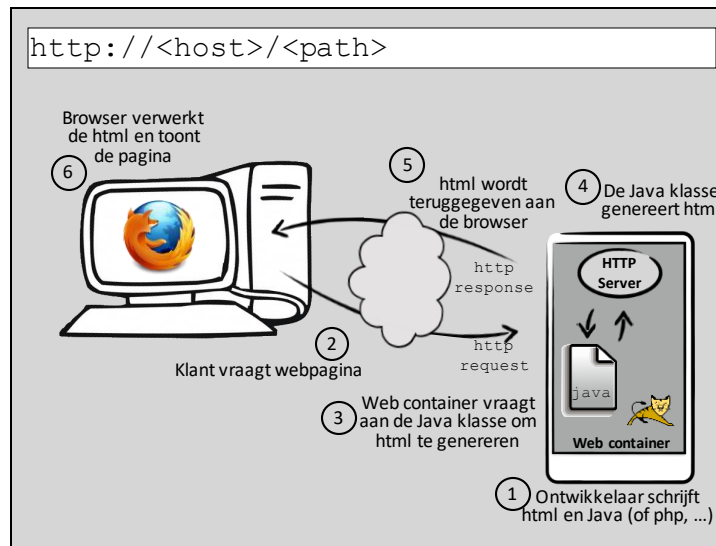




Deze pagina simuleert het gooien met een dobbelsteen

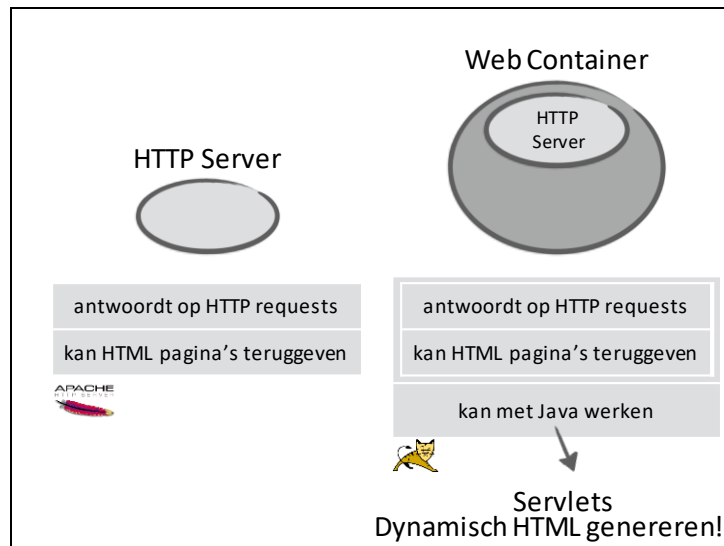
—> statische HTML pagina niet mogelijk!

—> html zal dynamisch moeten gegenereerd worden, op het moment dat de pagina opgevraagd wordt



Dynamische website:

- http server volstaat niet meer
- web container nodig die met Java kan werken (of php interpreter om met php te werken, ...)



Web Container kan alles wat een HTTP server kan, en meer: kan Java code uitvoeren.

De klasse die HTML genereert wordt servlet genoemd

Een servlet is een 'gewone' klasse, maar wordt door de web container herkend en beheerd

# AGENDA

- ✓ HTTP Server vs. Web Container
- Servlet schrijven
- Request handling
- JSP
- Foutopsporing





1. ONTWIKKELAAR SCHRIJFT  
**JAVA SERVLET**

2-6. WEB CONTAINER  
DOET AL DE REST

```

@WebServlet("/Dice")
public class DiceServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Dice</title>");
        out.println("<link rel='stylesheet' href='style/style.css' type='text/css'>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Playing dice with Web 2</h1>");
        out.println("<main style='text-align: center'>");
        out.println("<h2>Playing with the dice</h2>");
        int eyes = new Random().nextInt(6) + 1;
        out.println("<p>You threw a " + eyes + "</p>");
        out.println("<p><a href='index.html'>Home</a></p>");
        out.println("</main>");
        out.println("</body>");
        out.println("</html>");
    }
}

```


SERVLET

Voorbeeld servlet

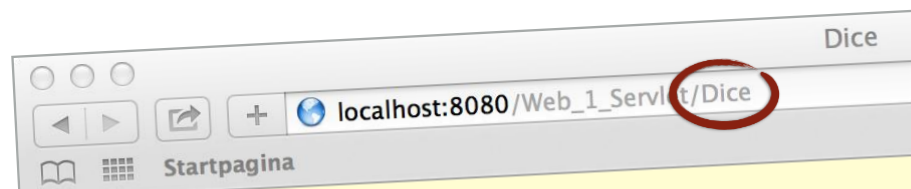
zie demo op GitHub, Demo1\_Dice, klasse ui.view.DiceServlet

# SERVLET

Als een request voor deze URL binnenkomt,  
leest de container deze URL-mapping en  
weet hij dat hij bij deze servlet moet zijn




```
@WebServlet("/Dice")  
public class DiceServlet extends HttpServlet {  
  
    ...  
}
```



# ANNOTATIONS

De container overloopt  
alle Java klassen en  
kijkt of er een annotatie  
`@WebServlet` staat

De container checkt  
of de URL bij de annotatie  
overeenkomt met  
de URL uit de request



```
@WebServlet("/Dice")  
public class DiceServlet ...
```

# SERVLET

OM HTML TE GENEREREN

'Gewone'  
Java klasse...

...die al vanalles meekrijgt  
van een klasse die al bestaat

(zie OOP)

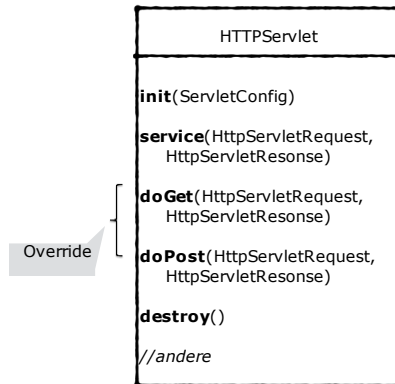


```
public class DiceServlet extends HttpServlet {
```

```
...
```

```
}
```

# HTTPServlet



De methodes `init()`, `service()`, `destroy()`, ... zijn al uitgewerkt in de klasse `HTTPServlet`.

'**extends `HttpServlet`**' zorgt ervoor dat jouw java-klasse ze meekrijgt. Jij hoeft ze niet zelf te schrijven.

De methodes `doGet()`, `doPost()` zijn leeg in de klasse `HTTPServlet`.

Jij moet ze nog uitwerken in jouw servlet. Je 'overschrijft' ze dus in je servlet.

# DOGET

VOOR ALS ER EEN GET REQUEST BINNENKOMT

De web container steekt alle gegevens van de request in een Request-object ...

```
@WebServlet("/Dice")  
public class DiceServlet extends HttpServlet {
```

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

```
        ...
```

```
    }  
}
```

... en zet ook alvast een Response-object klaar

... en exceptions voor het geval er iets fout gaat  
(zie OOP)

In deze methode komt de code die zal uitgevoerd worden wanneer een GET-request binnenkomt.

# PRINTWRITER

OM WEG TE SCHRIJVEN NAAR HET RESPONSE-OBJECT

```
@WebServlet("/Dice")
public class DiceServlet extends HttpServlet {

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

        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        ...
        out.println("</html>");
    }
}
```

... en begin te schrijven

Vraag een  
PrintWriter-object aan  
het Response-object...

We willen een dynamische pagina genereren, dus we willen html wegschrijven naar de response.

Hiervoor kunnen we de Java klasse PrintWriter gebruiken.

out.println() —> zie System.out.println(), maar nu schrijf je niet weg naar de console, maar naar de response



1. ONTWIKKELAAR SCHRIJFT  
JAVA **SERVLET**

**2-6. WEB CONTAINER**  
**DOET AL DE REST**

# HTTPServlet

**Eénmaal** opgeroepen:

- door web container
- na creatie van de servlet

**Telkens** opgeroepen:

- door webcontainer
- bij iedere aanroep van servlet
- roept doGet() of doPost() op
- telkens in afzonderlijke thread

Override

**Eénmaal** opgeroepen:

- door web container
- voor vernietigen van de servlet

```
HTTPServlet

init(ServletConfig)

service(HttpServletRequest,
HttpServletResponse)

doGet(HttpServletRequest,
HttpServletResponse)

doPost(HttpServletRequest,
HttpServletResponse)

destroy()

//andere
```

De methodes `init()`, `service()`, `destroy()`, ... zijn al uitgewerkt in de klasse `HTTPServlet`.

'**extends `HttpServlet`**' zorgt ervoor dat jouw java-klasse ze meekrijgt. Jij hoeft ze niet zelf te schrijven.

De methodes `doGet()`, `doPost()` zijn leeg in de klasse `HTTPServlet`.

Jij moet ze nog uitwerken in jouw servlet. Je 'overschrijft' ze dus in je servlet.

# AGENDA

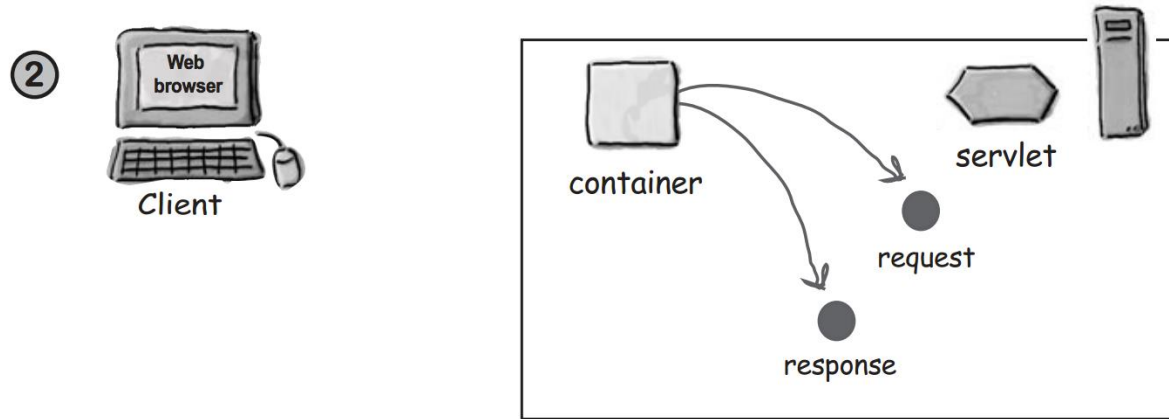
- ✓ HTTP Server vs. Web Container
- ✓ Servlet schrijven
- Request handling
- JSP
- Foutopsporing





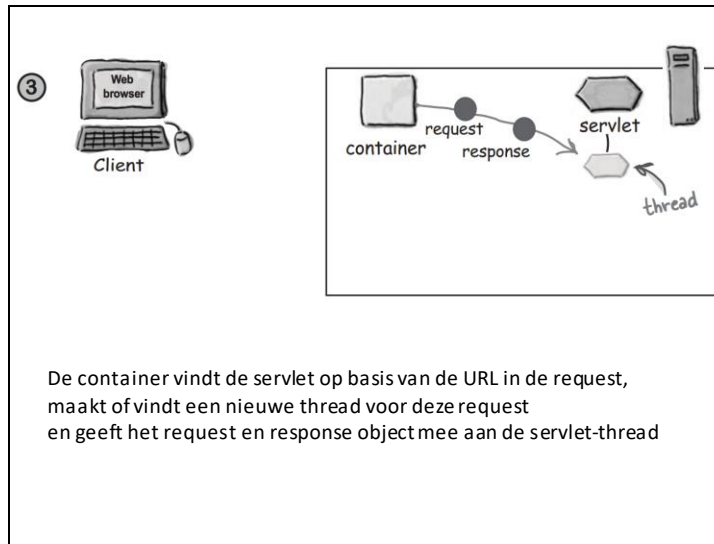
Web container heeft bij het opstarten ervan een servlet object gemaakt van de servlet (door de default constructor van de server klasse op de roepen) en heeft de `init()` methode hierop al opgeroepen.

Deze tekeningen en slides gaan enkel over wat de `service()` methode doet.

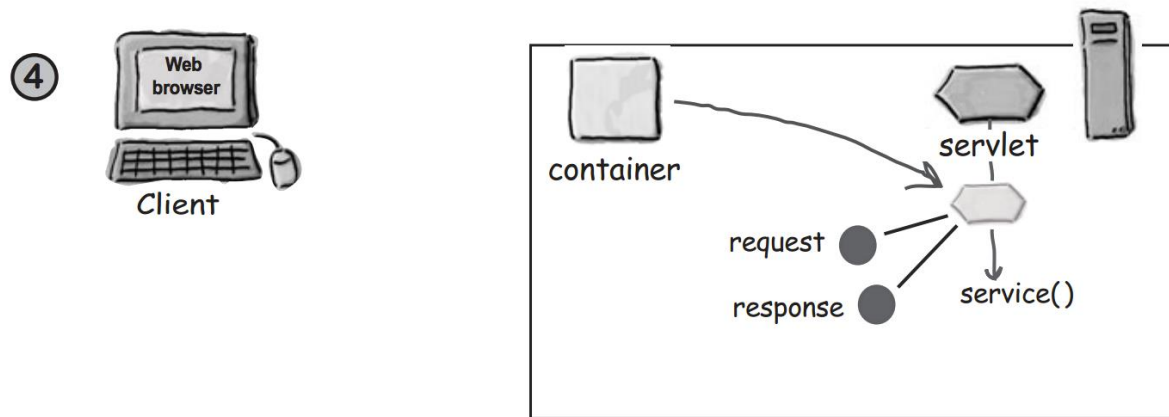


De container ziet dat het een request voor een servlet is en maakt 2 objecten:

1. **HttpServletResponse**
2. **HttpServletRequest**

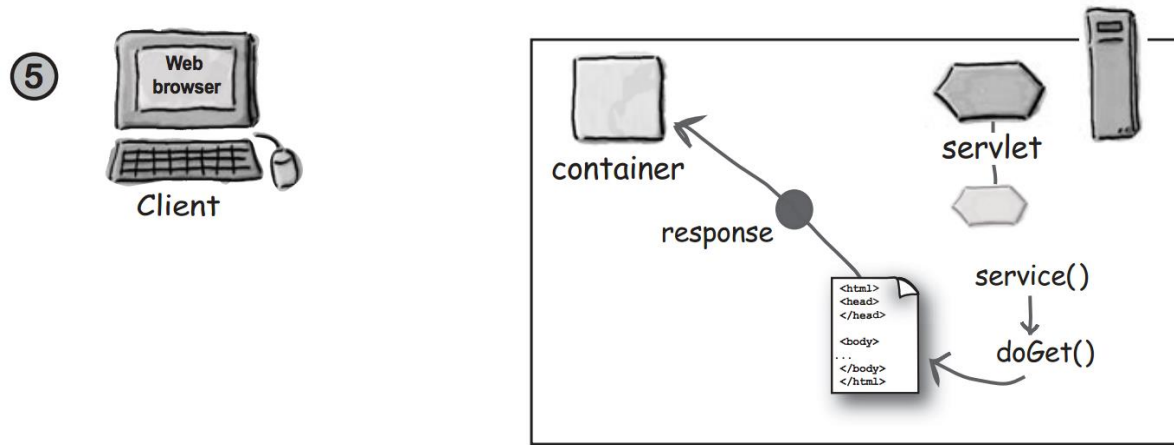


Opmerking: als er nog geen servlet-object is aangemaakt zal de webcontainer er zelf eentje aanmaken. Van dan af zal hij telkens dit object hergebruiken.



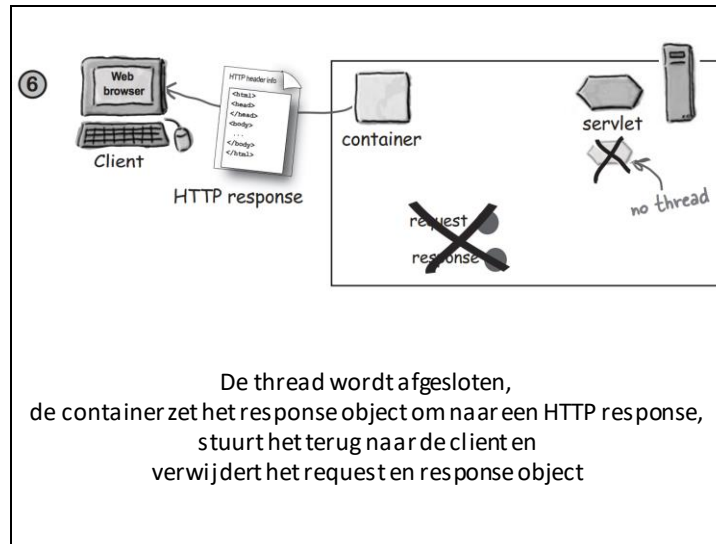
De container roept de *service()*\* methode van de servlet op.  
In de *service()* methode wordt de *doGet()* methode opgeroepen.

\* De *service()* methode bestaat al, jouw servlet heeft die overgeërfd van de bestaande klasse `HttpServlet` (zie OOP)



De `doGet()` methode genereert de dynamische pagina en schrijft deze weg naar het response object.





Ook hier alleen service() methode die getekend staat op de figuren.

Wanneer de web container geshut down wordt, dan pas wordt de destroy() methode opgeroepen en zullen de instanties van de servlet verwijderd worden.

# AGENDA

- ✓ HTTP Server vs. Web Container
- ✓ Servlet schrijven
- ✓ Request handling
- JSP
- Foutopsporing



```

@WebServlet("/")
public class DiceServlet extends HttpServlet {

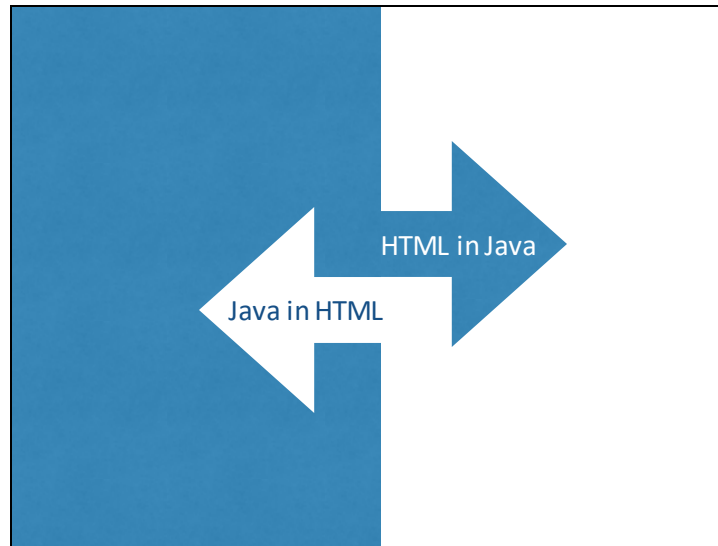
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Dice</title>");
        out.println("<link rel='stylesheet'");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Play");
        out.println("<main");
        out.println("<h2>Play");
        int eyes = new Random().nextInt(6) + 1;
        out.println("<p>You rolled " + eyes);
        out.println("<p><a href='\"#\"'>Roll again");
        out.println("</main>");
        out.println("</body>");
        out.println("</html>");
    }
}

```

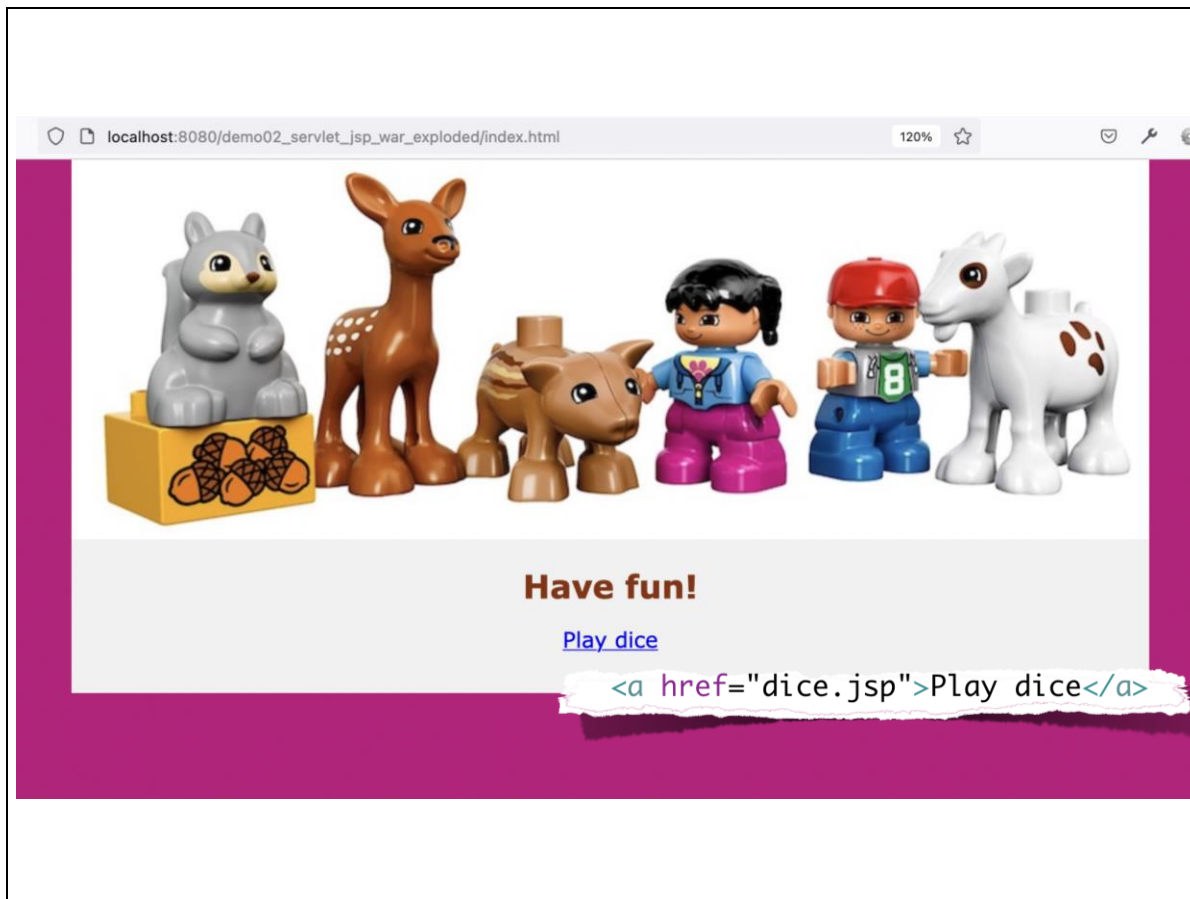
**Nadeel:**

- veel `out.println()` → onleesbare code
- geen ondersteuning voor validatie HTML
- ...

Nadeel van html schrijven in servlet?



Andere aanpak: in plaats van HTML in Java —> Java in HTML



```

<%@ page import="java.util.Random" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
  <title>Dice</title>
  <link rel="stylesheet" href="style/style.css" type="text/css">
</head>
<body>
<div id="container">
  <h1>Playing dice with Web 2</h1>
  <main style="text-align: center">
    <h2>Playing with the dice</h2>
    <% int eyes = new Random().nextInt(6) + 1; %>
    <p>You threw a <% out.println(eyes); %> </p>
    <p><a href="index.html">Home</a></p>
  </main>
</div>
</body>
</html>

```

dice.jsp

Deze pagina is geen html pagina, maar een **JSP** (Java Server Pages) pagina:

- lijkt op html pagina
- heeft bovenaan, een aantal JSP **directives**: info voor de compiler
- bevat JSP **scriptlets**: stukjes Java-code

All JSP scriptlets have access to a number of global variables that are made available free of charge. One of these is out, an object of type JspWriter. This object has a method called println() which outputs text into the HTML stream.

[http://www.tutorialspoint.com/jsp/jsp\\_implicit\\_objects.htm](http://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm)

# JSP SCRIPTLETS

```
<html>  
<body>  
<%  
  // This is a scriptlet. Notice that the "date"  
  // variable we declare here is available in the  
  // embedded expression later on.  
  System.out.println( "Evaluating date now" );  
  java.util.Date date = new java.util.Date();  
  out.println("The time is now " + date);  
%>  
</body>  
</html>
```

in HTML

tussen <% %>

Java code

A JSP scriptlet is used to contain any code fragment that is valid for the scripting language used in a page.

JSP Scriptlets= stukjes Java code die tussen de html staan

In een scriptlet kan je alle Java code schrijven, ook for-lussen, if-statements, veranderlijken declareren en later aanroepen.

```

<%@ page import="java.util.List" %>
<%@ page import="java.util.ArrayList" %>
<%@ page contentType="text/html;char set=UTF-8" language="java" %>
<!DOCTYPE html>
<%
    List<String> persons = new ArrayList<>();
    persons.add("Anne");
    persons.add("Brecht");
%>
<html>
<body>
<ul>
    <%
        for (String person : persons) {
    %>

    <li><%= person%>
    </li>

    <%
        }
    %>

</ul>
</body>
</html>

```

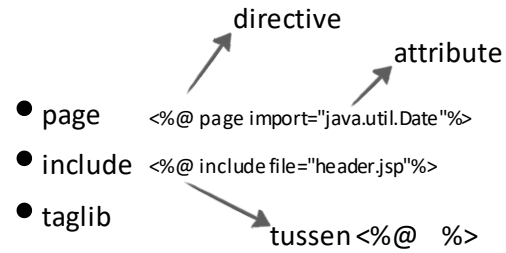
→ Declare List  
 → Add objects to list  
 → Open For Loop  
 → Write person to list item  
 → Close For Loop

Code in een scriptlet hoeft niet compleet te zijn: accolade die for-loop opent staat in andere scriptlet dan de accolade die de for-loop sluit.

Demo01\_Dice, bestand webapp/showingItems.jsp



# JSP DIRECTIVES





Directives are elements that relay messages to the JSP container and affect how it compiles the JSP page.

JSP directives: elementen die extra info voor de compiler bevatten

taglib: is voor later

# JSP DIRECTIVES

- `<%@ page %>`  gebruikt bovenaan de pagina
  - import statements
  - character encoding
  - ...
- `<%@ include %>`  gebruikt waar je de andere pagina wilt tonen
  - html of jsp bestanden herbruiken

# OPMERKING?

```
<html>
<body>
<%
  java.util.Date date = new java.util.Date();
  out.println("<p>The time is now <strong>" + date +
    "</strong></p>");
%>
</body>
</html>
```

toch weer  
HTML in Java!



# JSP EXPRESSION

```
<html>
<body>
  <% java.util.Date date = new java.util.Date(); %>
  <p>The time is now <strong><%= date %></strong></p>
</body>
</html>
```

tussen <%= %>

A JSP expression is used to insert the value of a scripting language expression, converted into a string, into the data stream returned to the client.

JSP Expressions zorgen ervoor dat je `out.println()` niet meer hoeft te gebruiken: het toont de waarde van een uitdrukking als string.

# JAVA SERVER PAGES

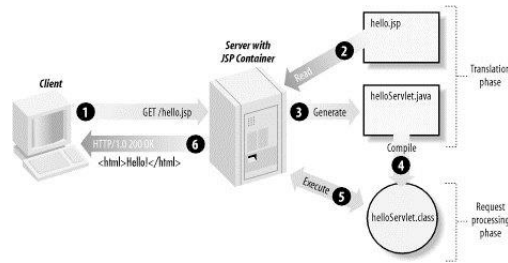
- extensie `.jsp`
- Java code in HTML pagina:
  - scriptlets: `<% ... %>`
  - directives: `<%@ ... %>`
  - expressions: `<%= ... %>`
- → servlets!

JSP pagina wordt door de Web Container gebruikt om een servlet te genereren. Dus achter de schermen werkt alles nog steeds zoals gezien in vorige les:

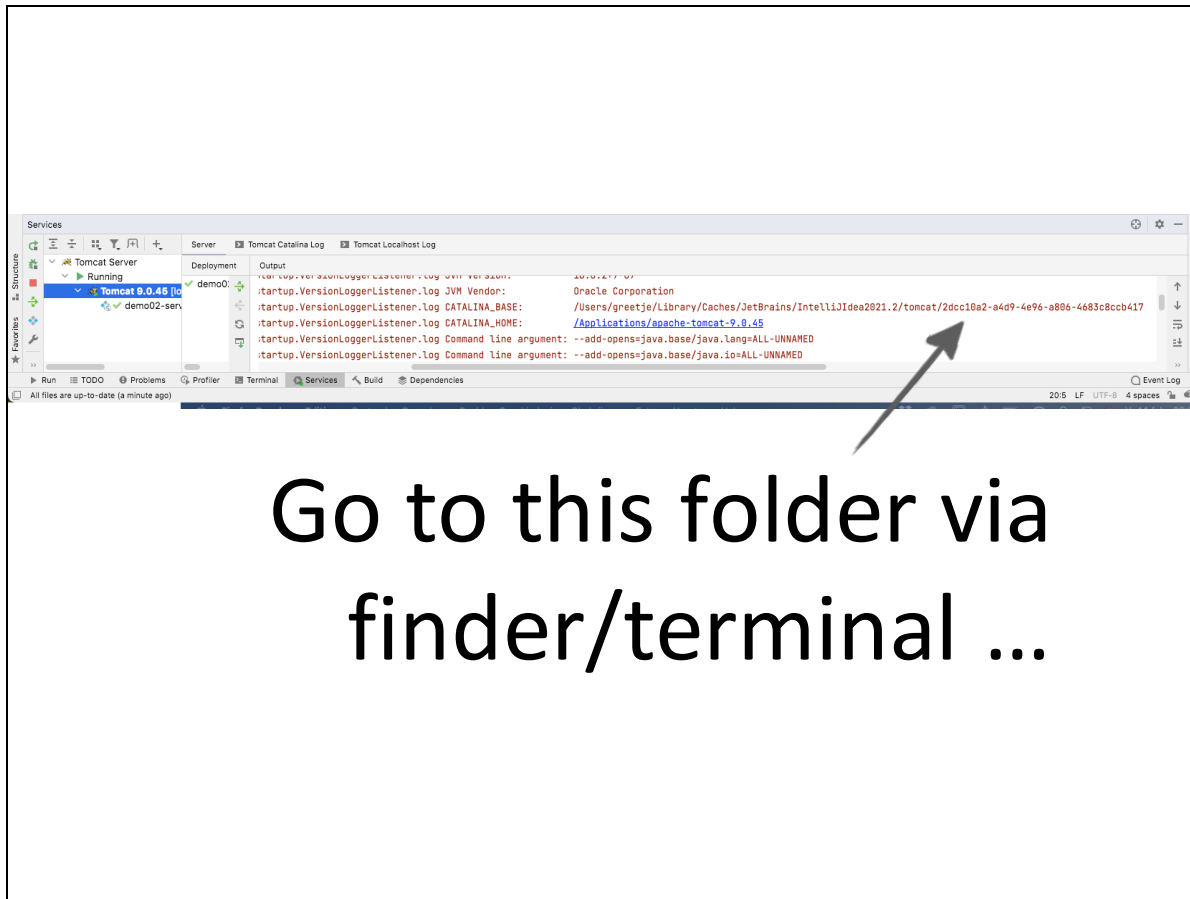
- request komt binnen,
- servlet wordt opgezocht (als ze nog niet bestaat wordt ze eerst gegenereerd),
- code uit servlet wordt uitgevoerd,
- ...

zie ook [http://www.tutorialspoint.com/jsp/jsp\\_architecture.htm](http://www.tutorialspoint.com/jsp/jsp_architecture.htm) en [http://www.tutorialspoint.com/jsp/jsp\\_life\\_cycle.htm](http://www.tutorialspoint.com/jsp/jsp_life_cycle.htm)

# JSP @ WEB CONTAINER



De Web container genereert met de jsp-pagina een servlet. Een servlet is een java-klasse. Alle code in scriptlets wordt overgenomen; HTML wordt omgezet naar `out.println("...")`





Je kan deze gegenereerde servlet zelf bekijken

bv. `work/Catalina/localhost/Web_JSP/org/apache/jsp/dice_jsp.java`

specifiek op Mac:

`/Users/.../Library/Caches/IntelliJ IDEA 2018.2/tomcat/Unnamed_Demo4_JSPExample/work/Catalina/localhost/ROOT/org/apache/jsp`



# AGENDA

- ☑ HTTP Server vs. Web Container
- ☑ Servlet schrijven
- ☑ Request handling
- ☐ Foutopsporing



# FOUTOPSPORING



*Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.*

Rich Cook

## HTTP Status 500 – Internal Server Error

Type Exception Report

**Message** An exception occurred processing [/index.jsp] at line [10]

**Description** The server encountered an unexpected condition that prevented it from fulfilling the request.

### Exception

```
org.apache.jasper.JasperException: An exception occurred processing [/index.jsp] at line [10]
```

```

7:    </head>
8:    <body>
9:    <% User user = null; %>
10:    <p>De gebruiker is <%= user.getFirstName() %></p>
11:    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
12:    </body>
13:    </html>

```

Stacktrace:

```
org.apache.jasper.servlet.JspServletWrapper.handl
org.apache.jasper.servlet.JspServletWrapper.servi
org.apache.jasper.servlet.JspServlet.serviceJspFi
org.apache.jasper.servlet.JspServlet.service(JspS
javax.servlet.http.HttpServlet.service(HttpServlet
org.apache.tomcat.websocket.server.WsFilter.doFil
```

### Root Cause

```
java.lang.NullPointerException
org.apache.jsp.index.jsp._jspService(index.jsp:ja
org.apache.jasper.runtime.HttpServletService(Http
javax.servlet.http.HttpServletService(HttpServlet
org.apache.jasper.servlet.JspServletWrapper.servi
org.apache.jsp.index.jsp._jspService(index.jsp:ja
org.apache.jasper.servlet.JspServletService(JspS
javax.servlet.http.HttpServletService(HttpServlet
org.apache.tomcat.websocket.server.WsFilter.doFil
```

**Note** The full stack trace of the root cause is available in the server logs.

Run:  Tomcat 9.0.53 

```
23-Sep-2019 19:47:21.103 SEVERE [http-nio-8080-exec-7] org.
context with path [/web_war_exploded] threw exception [An
```

```

7:  </head>
8:  <body>
9:    <% User user = null; %>
10:    <p>De gebruiker is <%= user.getFirstName() %></p>
11:    <p>Lorem ipsum dolor sit amet, consectetur adipisicing
        nostrum quos, repellat sapiente vero! Eos et ipsam sapient!
12:  </body>
13: </html>

```

Stacktrace:] with root cause

```
java.lang.NullPointerException
```

```
at org.apache.jsp.index_jsp._jspService(index_jsp.java:
at org.apache.jasper.runtime.Http1sBase.service(Http1s
```

# LOG FILES

## WIE ZOEKT DIE VINDT

# INSTRUCTIES

- ☑ Analyseer de **HTTP error code**
- ☑ Ontcijfer de **logfile**
- ☑ **Debug**
- ☑ Bekijk de code in **View Page Source**
- ☑ Bekijk de request/response in **Dev. Tools**