






samen sterk voor werk








**JAVA**







**JSP / Servlets**





## Inhoudsopgave



<b>1</b>	<b>INLEIDING</b>	<b>1</b>
1.1	Doelstelling	1
1.2	Vereiste voorkennis	1
1.3	Nodige software	1
<b>2</b>	<b>BROWSERS, WEBSERVERS, WEBSITES, URL'S, WEBFARM, TOMCAT</b>	<b>2</b>
2.1	Port number	2
2.2	Websites en URL's	2
2.3	Webfarm	3
2.4	Tomcat	3
2.4.1	Downloaden	4
2.4.2	Installeren	4
2.4.3	Starten	4
2.4.4	Stoppen	4
2.4.5	Website installeren	4
<b>3</b>	<b>REQUESTS EN RESPONSES</b>	<b>6</b>
3.1	Request onderdelen	6
3.1.1	Method	6
3.1.2	Headers	7
3.1.3	Body	7
3.1.4	Query string	7
3.2	Response onderdelen	7
3.2.1	Status code	7
3.2.2	Headers	8
3.2.3	Body	8
3.3	Samenvatting	8
<b>4</b>	<b>TOMCAT INTEGREREN IN DE IDE</b>	<b>9</b>
4.1	Eclipse 	9
4.2	NetBeans 	9
<b>5</b>	<b>EERSTE WEBSITE MET STATIC CONTENT</b>	<b>10</b>
5.1	Statische pagina	10
5.2	Welkompagina	10
5.3	Eclipse 	10
5.3.1	Project	10
5.3.2	Een welkompagina als statische HTML pagina	11
5.3.3	Afbeeldingen	11







5.3.4	CSS	11
5.3.5	GIT	11
5.3.6	Uittesten	11
5.3.7	WAR	12
<b>5.4</b>	<b>NetBeans</b> 	<b>12</b>
5.4.1	Project	12
5.4.2	Een statische HTML pagina als welkompagina	12
5.4.3	Afbeeldingen	13
5.4.4	CSS	13
5.4.5	GIT	13
5.4.6	Uittesten	13
5.4.7	WAR	13
<b>6</b>	<b>EEN DYNAMISCHE PAGINA MET EEN SERVLET</b>	<b>14</b>
6.1	Dynamische pagina	14
6.2	Servlet	14
6.3	Eclipse 	15
6.4	NetBeans 	15
6.5	Parameters van de methods doGet en doPost	15
6.6	Response	16
6.7	Wijziging servlet	16
6.8	Nadelen van HTML naar de browser sturen in een servlet	16
<b>7</b>	<b>EEN DYNAMISCHE PAGINA MET EEN JSP</b>	<b>17</b>
7.1	De webserver maakt op basis van een JSP een servlet	17
7.2	URL van een JSP	17
7.3	JSP onderdelen	17
7.4	Eclipse 	17
7.5	NetBeans 	17
7.6	index.jsp aanpassen	18
7.7	JSP template	18
7.7.1	Eclipse 	18
7.7.2	NetBeans 	18
<b>8</b>	<b>SERVLET ÉN JSP COMBINEREN</b>	<b>20</b>
8.1	Request doorgeven van een servlet naar een JSP	20
8.2	Request attribuut	20
8.3	Request attribuut lezen in een JSP	20
8.3.1	IndexServlet	21
8.3.2	index.jsp	21

<b>8.4</b>	<b>JSP's in de folder WEB-INF</b>	<b>21</b>
<b>9</b>	<b>`\${EXPRESSION LANGUAGE}`</b>	<b>22</b>
<b>9.1</b>	<b>Request attribuut met een primitief datatype</b>	<b>22</b>
<b>9.2</b>	<b>Request attribuut met een object</b>	<b>22</b>
<b>9.3</b>	<b>Verwijzing naar een onbestaand request attribuut</b>	<b>22</b>
<b>9.4</b>	<b>Hard gecodeerde waarden</b>	<b>22</b>
<b>9.5</b>	<b>Wiskundige operatoren</b>	<b>22</b>
<b>9.6</b>	<b>Vergelijkingsoperatoren</b>	<b>22</b>
<b>9.7</b>	<b>Logische operatoren</b>	<b>22</b>
<b>9.8</b>	<b>Conditionele operator ? :</b>	<b>22</b>
<b>9.9</b>	<b>Operator empty</b>	<b>23</b>
<b>9.10</b>	<b>Één element uit een verzameling lezen</b>	<b>23</b>
9.10.1	Array	23
9.10.2	List	23
9.10.3	Map	23
<b>9.11</b>	<b>Resultaat van een method oproep</b>	<b>23</b>
<b>10</b>	<b>JAVABEAN</b>	<b>24</b>
<b>10.1</b>	<b>Getters en setters</b>	<b>24</b>
<b>10.2</b>	<b>ReadOnly attributen</b>	<b>24</b>
<b>10.3</b>	<b>Getters en Setters maken met je IDE</b>	<b>25</b>
10.3.1	Eclipse 	25
10.3.2	NetBeans 	25
<b>10.4</b>	<b>Constructors</b>	<b>25</b>
<b>10.5</b>	<b>Constructors maken met je IDE</b>	<b>25</b>
10.5.1	Eclipse 	25
10.5.2	NetBeans 	25
<b>10.6</b>	<b>EL en JavaBeans</b>	<b>26</b>
<b>10.7</b>	<b>Geneste attributen</b>	<b>26</b>
<b>11</b>	<b>JSTL (JSP STANDARD TAG LIBRARY)</b>	<b>28</b>
<b>11.1</b>	<b>JSTL JAR bestand toevoegen aan het project</b>	<b>28</b>
11.1.1	Eclipse 	28
11.1.2	NetBeans 	28
<b>11.2</b>	<b>Tag names en tag URI's</b>	<b>28</b>
<b>11.3</b>	<b>&lt;c:forEach&gt;</b>	<b>29</b>
11.3.1	Itereren over een verzameling JavaBeans	30
11.3.2	Itereren over een Map	31
11.3.3	begin, step en end attributen	31

11.3.4	Itereren zonder verzameling	31
11.3.5	varStatus attribuut	31
<b>11.4</b>	<b>&lt;c:if&gt;</b>	<b>32</b>
<b>11.5</b>	<b>&lt;c:choose&gt;</b>	<b>32</b>
<b>11.6</b>	<b>&lt;c:out&gt;</b>	<b>33</b>
<b>11.7</b>	<b>&lt;c:url&gt;</b>	<b>33</b>
<b>11.8</b>	<b>&lt;c:import&gt;</b>	<b>33</b>
11.8.1	Parameters	34
<b>11.9</b>	<b>&lt;c:set&gt;</b>	<b>34</b>
<b>12</b>	<b>RELATIEVE URL'S</b>	<b>35</b>
<b>12.1</b>	<b>Probleem</b>	<b>35</b>
<b>12.2</b>	<b>Oplossing</b>	<b>35</b>
<b>13</b>	<b>SERVLETS INSTANCE</b>	<b>36</b>
<b>13.1</b>	<b>Servlet instance aanmaak</b>	<b>36</b>
<b>13.2</b>	<b>Servlet instance einde</b>	<b>36</b>
<b>13.3</b>	<b>Servlet instance levenscyclus</b>	<b>36</b>
<b>14</b>	<b>MULTITHREADING</b>	<b>37</b>
<b>14.1</b>	<b>AtomicInteger voorbeeld</b>	<b>38</b>
<b>15</b>	<b>SERVLET INITIALISATIEPARAMETERS</b>	<b>39</b>
15.1.1	Eclipse 	39
15.1.2	NetBeans 	39
<b>15.2</b>	<b>@WebServlet</b>	<b>39</b>
<b>15.3</b>	<b>Initialisatieparameters definiëren</b>	<b>39</b>
<b>15.4</b>	<b>Initialisatieparameters lezen in je Servlet</b>	<b>40</b>
<b>16</b>	<b>SERVLET CONTEXT</b>	<b>41</b>
<b>16.1</b>	<b>Initialisatieparameters</b>	<b>41</b>
16.1.1	Servlet context aanspreken	41
16.1.2	Initialisatieparameters lezen	41
16.1.3	Initialisatieparameters lezen in een JSP	42
<b>16.2</b>	<b>Attributen</b>	<b>42</b>
16.2.1	Voorbeeld	42
<b>17</b>	<b>MODEL-VIEW-CONTROLLER</b>	<b>44</b>
<b>17.1</b>	<b>Repository classes</b>	<b>44</b>
<b>18</b>	<b>GET REQUEST VERWERKEN</b>	<b>46</b>

18.1	Wanneer is een request een GET request?	46
18.2	Idempotent requests	46
18.3	StringUtils	47
18.4	Request parameters in de query string	47
18.5	HTML form met method='get'	49
18.6	Client sided validatie	51
18.7	Request parameter lezen in een JSP	51
18.8	Vereenvoudigde form tag	51
18.9	Request method getParameterValues	52
18.10	Request parameters – request attributen	53
19	POST REQUEST VERWERKEN	54
19.1	Het refresh probleem en POST-REDIRECT-GET	55
19.2	Overzicht GET en POST requests	57
19.3	Speciale tekens in de body van een POST request	57
19.4	Bestand uploaden	58
19.5	Dubbele submit vermijden	59
20	REQUEST HEADERS	60
20.1	Één header lezen	60
20.2	Alle headers lezen	62
21	COOKIES	63
21.1	Cookie	63
21.2	Cookie toevoegen	63
21.3	Cookies lezen	63
21.4	Cookie wijzigen	63
21.5	Permanente cookie verwijderen	64
21.6	Voorbeeld	64
21.7	Cookies lezen in een JSP	65
21.8	Cookies inzien die je browser bijhoudt	65
22	SESSIONS	66
22.1	Stateless	66
22.2	Session	66
22.3	Serializable	67
22.3.1	Session persistence	67
22.3.2	Session replication	67

<b>22.4</b>	<b>Session identificatie</b>	<b>67</b>
22.4.1	Session identificatie met een tijdelijke cookie	67
22.4.2	Session identificatie met URL rewriting	68
<b>22.5</b>	<b>Request method getSession</b>	<b>68</b>
<b>22.6</b>	<b>Session attribuut toevoegen</b>	<b>69</b>
<b>22.7</b>	<b>Session attribuut lezen</b>	<b>69</b>
<b>22.8</b>	<b>Session attribuut wijzigen</b>	<b>69</b>
<b>22.9</b>	<b>Session attribuut verwijderen</b>	<b>69</b>
<b>22.10</b>	<b>Volledige session verwijderen</b>	<b>69</b>
<b>22.11</b>	<b>De webserver verwijdert een session</b>	<b>69</b>
<b>22.12</b>	<b>Voorbeeld 1</b>	<b>69</b>
<b>22.13</b>	<b>Voorbeeld 2</b>	<b>70</b>
<b>22.14</b>	<b>Session attribuut lezen in een JSP</b>	<b>72</b>
<b>22.15</b>	<b>URL rewriting afzetten</b>	<b>72</b>
<b>23</b>	<b>LISTENERS</b>	<b>74</b>
23.1	De website wordt gestart of gestopt	74
23.2	Een servlet context attribuut wordt gemaakt, gewijzigd of verwijderd	74
23.3	Een session wordt gemaakt, verwijderd of vervalt	74
23.4	Een session attribuut wordt gemaakt, gewijzigd of verwijderd	75
23.5	Een browser request komt binnen of is helemaal verwerkt	75
23.6	Een servlet attribuut wordt gemaakt, gewijzigd of verwijderd	75
23.7	Meerdere classes die eenzelfde interface implementeren	75
23.8	Listener instances	75
23.9	Configuratie	75
23.10	Listeners en de servlet context	76
23.11	Samenvatting	76
<b>23.12</b>	<b>Voorbeeld</b>	<b>76</b>
23.12.1	Eclipse 	76
23.12.2	NetBeans 	76
23.12.3	Voorbeeldlistener	76
<b>24</b>	<b>FILTERS</b>	<b>78</b>
24.1	Filter eigenschappen	78
24.2	URL patronen	79
24.3	Filter instances	79
24.4	De interface Filter	80
24.5	Filters en de servlet context	80

<b>24.6</b>	<b>Eclipse</b> 	<b>80</b>
<b>24.7</b>	<b>NetBeans</b> 	<b>81</b>
<b>24.8</b>	<b>Voorbeeldfilter</b>	<b>81</b>
<b>25</b>	<b>JDBC</b>	<b>82</b>
<b>25.1</b>	<b>Database</b>	<b>82</b>
<b>25.2</b>	<b>Exceptions</b>	<b>82</b>
<b>25.3</b>	<b>DataSource</b>	<b>82</b>
25.3.1	JDBC driver	83
25.3.2	DataSource definiëren	83
<b>25.4</b>	<b>DataSource in je code</b>	<b>84</b>
25.4.1	AbstractRepository	84
25.4.2	De servlets	85
25.4.3	PizzaRepository	85
<b>26</b>	<b>FOUTOPVANG</b>	<b>88</b>
<b>26.1</b>	<b>Foutpagina's koppelen aan HTTP status codes</b>	<b>88</b>
<b>26.2</b>	<b>Foutpagina's koppelen aan exceptions</b>	<b>88</b>
<b>26.3</b>	<b>Fouten loggen</b>	<b>89</b>
<b>27</b>	<b>INTERNATIONALIZATION</b>	<b>91</b>
<b>27.1</b>	<b>Locale</b>	<b>91</b>
<b>27.2</b>	<b>Request header Accept-Language</b>	<b>91</b>
<b>27.3</b>	<b>Datumopmaak</b>	<b>91</b>
<b>27.4</b>	<b>Getalopmaak</b>	<b>93</b>
<b>27.5</b>	<b>Resource bundles</b>	<b>93</b>
27.5.1	Eclipse 	94
27.5.2	NetBeans 	94
27.5.3	Inhoud van de resource bundles	94
27.5.4	Resource bundles gebruiken in een JSP	94
<b>27.6</b>	<b>Taal en land vragen met hyperlinks</b>	<b>95</b>
<b>28</b>	<b>CUSTOM TAGS</b>	<b>97</b>
<b>28.1</b>	<b>TLD bestand</b>	<b>97</b>
<b>28.2</b>	<b>Eclipse</b> 	<b>97</b>
<b>28.3</b>	<b>NetBeans</b> 	<b>97</b>
<b>28.4</b>	<b>vdab.tld</b>	<b>98</b>
<b>28.5</b>	<b>menu.tag</b>	<b>98</b>
<b>28.6</b>	<b>Custom tag gebruiken in een JSP</b>	<b>98</b>



28.7	Custom tag attributen	99
29	OUDERE WEBSERVERS EN OUDERE WEBSITES	100
29.1	Een servlet registreren in web.xml	100
29.2	Een listener registreren in web.xml	100
29.3	Een filter registreren in web.xml	100
30	HERHALINGSOEFENINGEN	101
31	COLOFON	102

# 1 INLEIDING

## 1.1 Doelstelling

Je leert websites maken met servlets en JSP's (Java Server Pages).

Eerst maakte men Java websites met servlets. Ervaring leerde dat je in een servlet:

- ➕ goed Java code kan schrijven
- ➖ moeilijk HTML kan schrijven

Daarna maakte men Java websites met JSP's. Ervaring leerde dat je in een JSP:

- ➕ goed HTML kan schrijven
- ➖ moeilijk Java code kan schrijven

Uiteindelijk bleek dat je best servlets en JSP's combineert:

- ➕ Je schrijft Java code in servlets.
- ➕ Je schrijft HTML in JSP's.

## 1.2 Vereiste voorkennis

- HTML
- Java PF
- Lambda
- JDBC
- GIT

## 1.3 Nodige software

- Een JDK (Java Developer Kit) met versie 8 of hoger.
- Tomcat webserver.
  - Je leert in de cursus hoe je Tomcat installeert en beheert.
  - Je hebt minstens Tomcat 8 nodig.
- MySQL database server.
- Eclipse of NetBeans.
  - Je neemt cursus onderdelen met 🇳🇱 enkel door als je Eclipse gebruikt.  
Je hebt minstens Eclipse versie Neon.1 nodig.
  - Je neemt cursus onderdelen met 🇧🇪 enkel door als je NetBeans gebruikt.  
Je hebt minstens NetBeans versie 8 nodig.

Als je na deze cursus nog een of meerdere van volgende VDAB cursussen doorneemt

- JPA met Hibernate
- Spring
- Test Driven Development met JUnit
- XML verwerken
- Jenkins

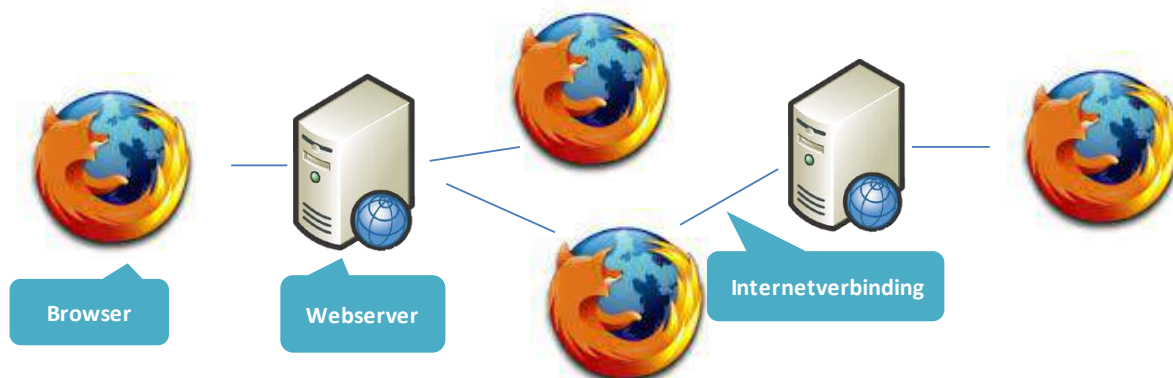
raden we aan in deze cursus Eclipse te gebruiken, gezien de vervolgcursussen enkel nog Eclipse gebruiken, niet NetBeans.

## 2 BROWSERS, WEBSERVERS, WEBSITES, URL'S, WEBFARM, TOMCAT

De computers waarop de browsers draaien en de computers waarop webserver draaien zijn via het internet of intranet (bedrijfsnetwerk) met mekaar verbonden.

Browsers en webserver wisselen data uit met HTTP (HyperText Transfer Protocol).

HTTP gebruikt zelf TCP/IP: Transmission Control Protocol / Internet Protocol.



Ook andere diensten gebruiken TCP/IP. Voorbeelden van zo'n diensten:

- SMTP Simple Mail Transfer Protocol (om mails te versturen).
- FTP File Transfer Protocol (om bestanden uit te wisselen).

### 2.1 Port number

Op één computer kunnen meerdere programma's TCP/IP gebruiken.

Elk programma krijgt op die computer een uniek identificatiegetal: het port number:

- 80 Webserver
- 21 FTP (File Transfer Protocol) server
- 25 Mail server

### 2.2 Websites en URL's

Een webserver bevat één of meerdere websites. Een website bevat pagina's. Elke pagina heeft een unieke identificatie: de URL (Uniform Resource Locator). Een URL heeft volgende opbouw:

<http://pizzaluigi.be/pizzas>



- Alle pagina's van een website delen dezelfde domeinnaam.
- Een domeinnaam is niet hoofdlettergevoelig, een pad wel.
- Je kunt surfen naar een URL en geen pad meegeven (pizzaluigi.be).  
Je ziet dan de 'welkompagina' van die website.
- Als een webserver afwijkt van het standaard TCP/IP port number (80), vermeld je, bij het surfen, in de URL ook het port number: pizzaluigi.be:8080/pizzas.



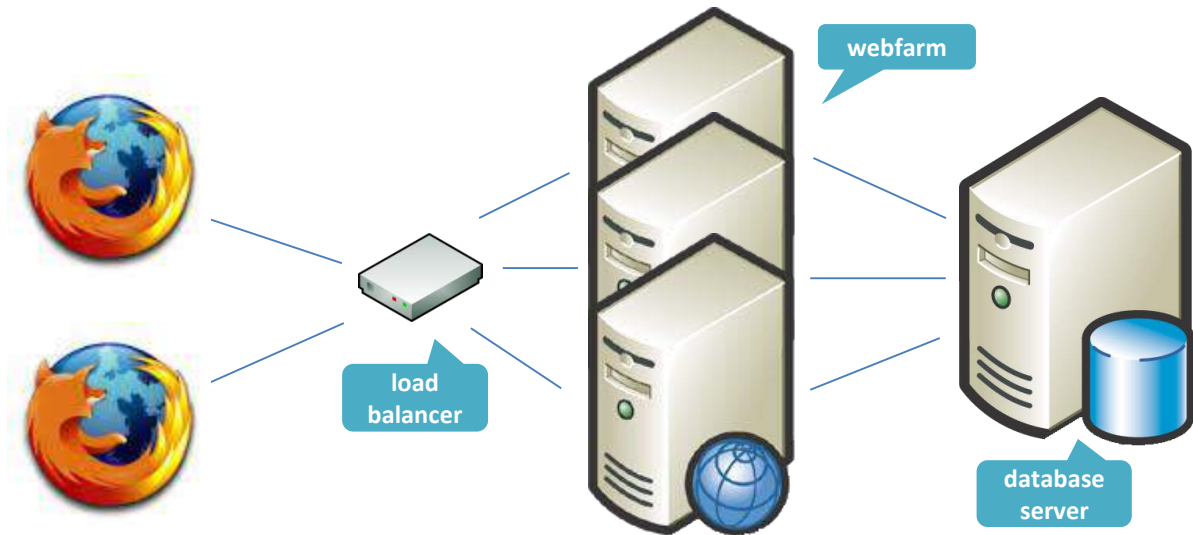
Opmerking: URI (Universal Resource Identifier) is een identifier voor een stukje data. Elke URL is een URI, maar niet elke URI is een URL.

Een URI is een URL als de URI een protocol (zoals http) gebruikt om de plaats van de data aan te geven.

## 2.3 Webfarm

Een website draait meestal niet op één, maar op *meerdere* webserver. Zo'n groep webserver heet een webfarm of een cluster.

Een load balancer is een apparaat dat ook behoort tot de webfarm. De load balancer verdeelt het werk (de pagina's leveren die de browsers bezoeken) over die webserver.



Voordelen

- ⊕ Als één webserver uitvalt, verdeelt de load balancer het werk over de andere webserver in de webfarm. De website blijft zo ter beschikking.
- ⊕ De load balancer verdeelt het werk (de browser vragen) over de webserver in de webfarm. Zo raakt één webserver niet overladen met werk en blijft de website performant.

## 2.4 Tomcat

Veel webserver ondersteunen Java. Veel gebruikte webserver zijn:

Webserver	Van de firma
Tomcat	Apache
Jetty	Eclipse
WildFly	JBoss
WebLogic	Oracle
WebSphere	IBM

Je kan je Java website uitvoeren op al die webserver, zonder de code te wijzigen.

Je gebruikt in de cursus de populairste Java webserver: Tomcat

### 2.4.1 Downloaden

Je kunt Tomcat downloaden op [tomcat.apache.org](http://tomcat.apache.org)

1. Je klikt links in het onderdeel Download op de hyperlink [Tomcat 8](#)
2. Je zoekt de laatste versie die begint met 8.5
3. Je kiest daar bij Binary Distributions voor Core
4. Je kiest daar de hyperlink [zip \(...\)](#)



### 2.4.2 Installeren

Je pakt het ZIP bestand uit in een directory op je computer.

Je maakt in je besturingssysteem een environment variabele met de naam JAVA\_HOME.  
Die variabele bevat het absolute pad van de directory waarin Java geïnstalleerd is.

Je doet dit op Windows als volgt:

1. Je kiest de Windows Start knop.
2. Je klikt met de rechtermuisknop op Computer en je kiest Properties.
3. Je kiest Advanced system settings.
4. Je kiest het tabblad Advanced.
5. Je kiest Environment Variables en je kiest New onder User variables for ...
6. Je tikt JAVA\_HOME bij Variable name.
7. Je tikt het absolute pad naar de directory waarin Java geïnstalleerd is (bijvoorbeeld C:\Program Files\Java\jdk1.8.0\_1) bij Variable value.
8. Je kiest drie keer OK.

### 2.4.3 Starten

Je dubbelklikt startup.bat in de subdirectory bin van de Tomcat Directory.

- Je ziet een Command-Prompt venster waarin Tomcat opstart.
- Je ziet in dit venster enkele diagnostische meldingen tijdens het opstarten.
- Je ziet als laatste melding INFO: Server startup in x ms.

Tomcat is nu gestart. Je laat dit venster openstaan, anders sluit je Tomcat terug af.

Tomcat is in uitvoering op je computer en gebruikt TCP poort 8080.

In een netwerk is localhost een synoniem voor je eigen computer.

Je tikt in de browser adresbalk localhost:8080 en je ziet de Tomcat welkompagina.

### 2.4.4 Stoppen

1. Je drukt Ctrl+C in het Command-prompt venster waarin Tomcat draait of.
2. Je dubbelklikt shutdown.bat in de bin directory van Tomcat.
3. Het Command-Prompt venster met Tomcat verdwijnt na enkele seconden.

### 2.4.5 Website installeren

#### 2.4.5.1 WAR bestanden



Een WAR (web archive) is een bestand met de extensie war, maar is intern een ZIP.  
Het bevat alle ingrediënten van een Java website (code, afbeeldingen, CSS, ...).

#### 2.4.5.2 Tomcat gebruikers

Enkel geregistreerde Tomcat gebruikers kunnen via de browser een website op Tomcat installeren.  
Elke gebruiker heeft één of meerdere rollen (roles).

Enkel gebruikers met de role manager-gui kunnen via de browser een website installeren.

Het bestand tomcat-users.xml in de Tomcat subdirectory conf bevat de gebruikers.

Je voegt in dit bestand een gebruiker met de roles manager-gui en manager-script.

Je beheert met die laatste role Tomcat vanuit NetBeans.

1. Je opent dit bestand met NetBeans of Eclipse: menu File, Open File.
2. Je maakt een blanco regel juist onder de regel `<tomcat-users ...>` en je tikt daarin  
`<user username="cursist" password="cursist" roles="manager-gui,manager-script"/>`
3. Je slaat het bestand op.
4. Je herstart Tomcat.

### 2.4.5.3 Installatie via de browser

1. Je start Tomcat.
2. Je surft met een browser naar localhost:8080.
3. Je kiest in de Tomcat welkompagina de knop Manager App.
4. Je tikt als gebruikersnaam én als paswoord cursist en je logt in.
5. Je kiest de knop naast Select WAR file to upload.
6. Je duidt sterrenbeelden.war (bestand bij cursus) aan op je harde schijf.
7. Je kiest de knop Deploy (to deploy = installeren).
8. Je ziet na enkele seconden /sterrenbeelden in de lijst van websites (Applications) :

<a href="#">/sterrenbeelden</a>	None specified		true	0	Start Stop Reload Undeploy
					Expire sessions with idle ≥ 30 minutes

9. De hyperlink /sterrenbeelden brengt je naar de website. Je kan die testen.
10. Je stopt de website met de knop Stop. Als je daarna surft naar de website, zie je een pagina met status code 404 (Not found): de website is niet actief.
11. Je start de website terug met de knop Start.
12. Je herstart de website met de knop Reload.
13. Je verwijdert de website van Tomcat met de knop Undeploy.
14. Een website kan per gebruiker data bijhouden in het RAM geheugen (zoals een winkelmandje). Je verwijdert die data, als die gedurende 30 minuten niet gelezen of gewijzigd werd, met de knop Expire sessions.

Gelieve de website te verwijderen, want je installeert hem straks via het bestandsbeheer.

### 2.4.5.4 Installatie via het bestandsbeheer

Je kopieert sterrenbeelden.war naar de Tomcat subdirectory webapps.

Tomcat installeert elke WAR in die directory automatisch als een website.

Je surft naar de hoofdlettergevoelige URL van de website: localhost:8080/sterrenbeelden.

Je verwijdert de website door sterrenbeelden.war te verwijderen uit de directory webapps.



Je laat de Tomcat draaien. Je hebt hem nodig in het volgende hoofdstuk.

### 3 REQUESTS EN RESPONSES

Telkens je

- een URL tikt in de browser adresbalk
- of een URL kiest in de browser favorieten
- of een hyperlink aanklikt
- of een knop aanklikt in een formulier van een webpagina



stuurt de browser een request (vraag) naar een webserver en krijgt een response (antwoord) terug. Dit antwoord bevat HTML, CSS, JavaScript en/of afbeelding(en).

De browser hertekent met dit antwoord zijn beeld.

#### 3.1 Request onderdelen

Een request bevat een method, headers, een optionele body en een optionele query string.

##### 3.1.1 Method

De method definieert het *soort* request met één woord: GET of POST.

HTTP schrijft het volgende voor:

GET	GET wordt gebruikt bij elke request waarmee de gebruiker enkel <i>data vraagt</i> . Voorbeeld: een request met de method GET naar <code>pizzaluigi.be/producten</code> vraagt producten op
POST	POST wordt gebruikt bij elke request die meer doet dan enkel data vragen. Voorbeeld: een request met de method POST naar <code>pizzaluigi.be/producten/toevoegen</code> voegt een product toe.

Je ziet hiervan voorbeelden met Firefox.

1. Je kiest rechts boven in Firefox ☰.
2. Je kiest Developer.
3. Je kiest Network.

Je ziet vanaf nu onder in het venster technische informatie over de requests en responses.

4. Je surft naar <http://localhost:8080/sterrenbeelden>

Je ziet onder in het venster een request naar de welkompagina:

✓ Method	File	Domain	Type
● GET	/	localhost:8080	html

De pagina bevat een verwijzing naar `default.css`. Je ziet dus ook een request naar `default.css`:

● GET	default.css	localhost:8080	css
-------	-------------	----------------	-----

Beide requests vragen enkel data. De requests gebruiken daarbij de method GET.

Tweede voorbeeld: je tikt een naam en een bericht en je kiest de knop Toevoegen

De klik op de knop veroorzaakt een request met de method POST.

De request doet meer dan data vragen: de request voegt een item toe aan het gastenboek.

Daarna volgt een request met de method GET die de pagina terug opvraagt:

✓ Method	File	Domain	Type
▲ POST	/	localhost:8080	html
● GET	/	localhost:8080	html

Je leert verder in de cursus hoe je in je website requests verwerkt met de GET en de POST method.

### 3.1.2 Headers

Headers bevatten browserinformatie. Elke header heeft een naam en een waarde.

Voorbeeld: je surft naar <http://localhost:8080/sterrenbeelden>

Je klikt de request onder in het venster aan. Je kiest daarna Headers onder in het venster.

Je ziet onder andere de headers van de request:

▼ Request headers (0,308 KB)	
Host:	"localhost:8080"
User-Agent:	"Mozilla/5.0 (Windows NT 6.3; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0"
Accept:	"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
Accept-Language:	"en-US,nl-BE;q=0.7,en;q=0.3"

- De user-agent header bevat het browsertype.
- De accept-language header bevat de voorkeur talen en landsinstellingen van de gebruiker, bijvoorbeeld: en-US, nl-BE.

### 3.1.3 Body

Een GET request bevat geen body.

Een POST request kan een body hebben. Die bevat data waarmee de website de request verwerkt.

- Elk stukje data heeft een naam en een waarde, gescheiden door =
- Meerdere stukken data worden van mekaar gescheiden door &

Voorbeeld in sterrenbeelden: je tikt een naam en een bericht en je kiest de knop Toevoegen.

De klik op de knop veroorzaakt een request met de method POST. Je klikt deze request onder in het venster aan. Je kiest daarna Params onder in het venster. Je ziet de data van de body in tabelvorm:

▼ Form data	
naam:	"Jean"
bericht:	"prachtig"

### 3.1.4 Query string

Een GET request heeft geen body. Een GET request kan wel data meegeven in de query string. Deze zit op het einde van de URL, begint met ? en bevat één of meerdere parameters.

- Elke parameter heeft een naam en een waarde, gescheiden door =
- Meerdere parameters worden gescheiden door &

Voorbeeld in sterrenbeelden: je tikt een geboortedatum en je kiest de knop Sterrenbeeld.

Je ziet onder in het venster een request met een query string met een parameter datum:

✓	Method	File	Domain	Type
●	GET	/?datum=1-8-1966	localhost:8080	html

## 3.2 Response onderdelen

### 3.2.1 Status code

De status code geeft met een getal aan hoe de website de request verwerkte. Voorbeelden:

- 200 (OK) De request is correct verwerkt.
- 404 (Not Found) De URL bestaat niet in de website.

Voorbeeld in sterrenbeelden: je tikt een geboortedatum en je kiest de knop Sterrenbeeld.

Je klikt de request met de method GET onder in het venster aan.

Je kiest daarna Headers onder in het venster. Je ziet onder andere:

Status code: ● 200 OK



### 3.2.2 Headers

Headers bevatten informatie over de response. Elke header heeft een naam en een waarde. Voorbeeld: de header content-type bevat het MIME type (datatype) van de data in de body:

- text/html HTML
- text/css CSS
- text/javascript JavaScript
- image/png Een afbeelding in PNG formaat

Voorbeeld in sterrenbeelden: je tikt een geboortedatum en je kiest de knop Sterrenbeeld.

Je klikt de request met de method GET onder in het venster aan.

Je kiest daarna Headers onder in het venster. Je ziet onder andere de headers van de response:

▼ Response headers (0,140 KB)	
Content-Length:	"869"
Content-Type:	"text/html; charset=UTF-8"
Date:	"Mon, 08 Jun 2015 08:58:26 GMT"
Server:	"Apache-Coyote/1.1"

### 3.2.3 Body

De body bevat data die de request vraagt. Dit kan HTML zijn, CSS, een afbeelding, ...

Voorbeeld in sterrenbeelden: je tikt een geboortedatum en je kiest de knop Sterrenbeeld.

Je klikt de request met de method GET onder in het venster aan.

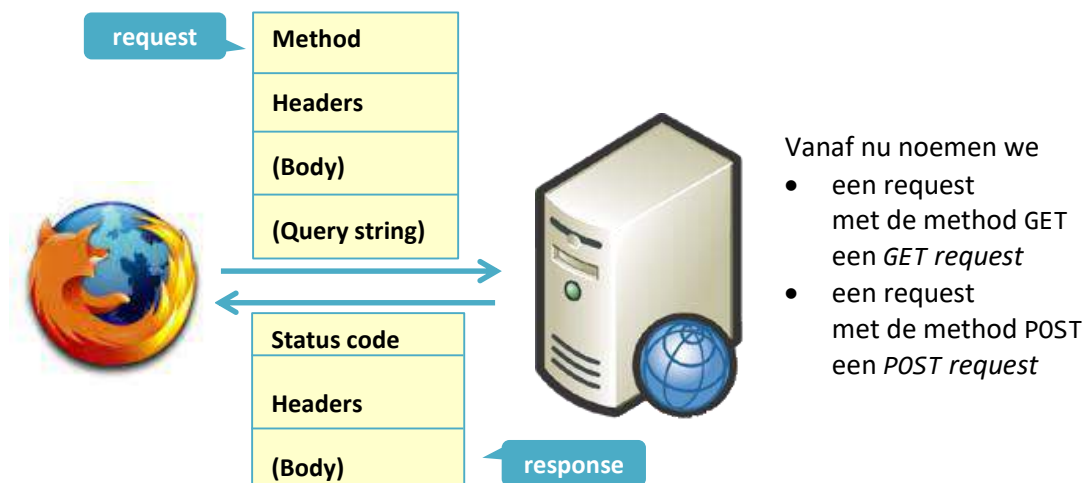
Je kiest daarna Response onder in het venster. Je ziet de HTML in de body van de response.

```

1 <!DOCTYPE html>
2 <html lang="nl">
3 <head>
4 <title>Sterrenbeelden</title>

```

## 3.3 Samenvatting




## 4 TOMCAT INTEGREREN IN DE IDE




Je stopt eerst Tomcat, zodat die niet in conflict komt met de Tomcat die je in de IDE zal integreren.

### 4.1 Eclipse

1. Je kiest rechts boven het Java EE perspective.
2. Je ziet onder een tabblad met de titel Servers.
3. Je doet één van volgende stappen:
  - a. Je klikt op de hyperlink No servers are available. Click this link ... of.
  - b. Je klikt met de rechtermuisknop in de witte oppervlakte van het tabblad Servers en je kiest in het menu New de opdracht Server.
4. Je kiest in het onderdeel Apache voor Tomcat v8.5 Server en je kiest Next.
5. Je kiest Browse en je duidt de directory aan waarin je Tomcat plaatste.
6. Je kiest Finish.
7. Je ziet in het tabblad Servers  Tomcat v8.5 Server at localhost [Stopped, Republish]
8. Je kunt met de rechtermuisknop klikken op die Tomcat
  - a. Je start de webserver met de opdracht Start.
  - b. Je stopt de webserver met de opdracht Stop.

Eclipse maakte een eigen configuratie van Tomcat waarbij geen enkele website gedeployed is.

### 4.2 NetBeans

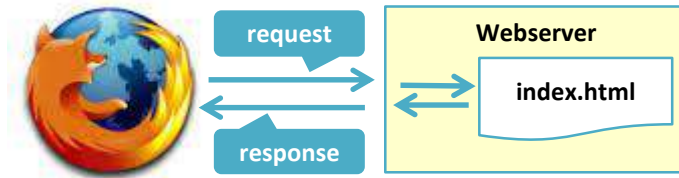
1. Je kiest in het menu Tools de opdracht Servers.
2. Je kiest Add Server.
3. Je kiest Apache Tomcat or TomEE bij Server en je kiest Next.
4. Je kiest Browse bij Server Location en je duidt de Tomcat directory aan.
5. Je tikt bij Username én bij Password cursist.
6. Je verwijdert het vinkje bij Create user if it does not exist.
7. Je kiest Finish en je kiest Close.
8. Je kiest links het tabblad Services.
9. Je opent het onderdeel Servers.
10. Je ziet  Apache Tomcat or TomEE.
11. Je klikt met de rechtermuisknop op die Tomcat
  - a. Je start de webserver met de opdracht Start.
  - b. Je stopt de webserver met de opdracht Stop.

## 5 EERSTE WEBSITE MET STATIC CONTENT

### 5.1 Statische pagina

Een statische pagina (bijvoorbeeld `index.html`) is een bestand op de webserver.

Bij een request naar zo'n pagina, stuurt de webserver de inhoud van dit bestand als response:



Een website `pizzaluigi.be` kan bijvoorbeeld `index.html` bevatten:

```

<!doctype html>
<html lang='nl'>
<head>
<title>Pizza Luigi</title>
</head>
<body>
<h1>Pizza Luigi</h1>
</body>
</html>
  
```

Bij een request naar `pizzaluigi.be/index.html` stuurt de webserver de inhoud van dit bestand als response

### 5.2 Welkompagina

Drie hoofdlettergevoelige URL's in je Java website kunnen de rol spelen van welkompagina:

URL	Te gebruiken als de welkompagina
<code>index.html</code>	een statische HTML pagina is
<code>index.jsp</code>	een JSP is (zie verder in de cursus)
<code>index.htm</code>	een servlet is (zie verder in de cursus)

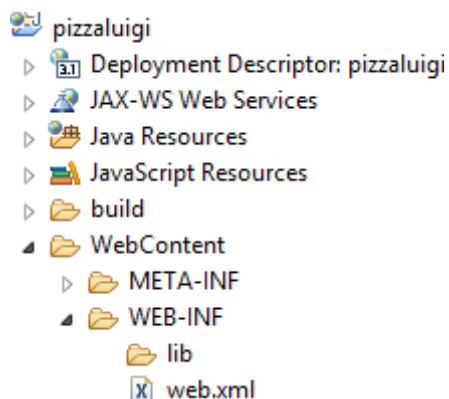
Je ziet van al deze mogelijkheden voorbeelden in de cursus.

### 5.3 Eclipse

#### 5.3.1 Project

1. Je kiest in het menu **File** de opdracht **New, Dynamic Web Project**.
2. Je tikt `pizzaluigi` bij **Project name** en je kiest **Next, Next**.
3. Je vinkt **Generate web.xml deployment descriptor** aan en je kiest **Finish**.

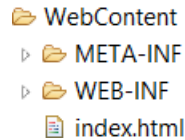
Je ziet de project structuur in de **Project Explorer**:



- **Deployment Descriptor** is een synoniem voor `web.xml` (het configuratiebestand van de website).
- **Java Resources** zal Java sources bevatten.
- **WebContent** zal onderdelen bevatten die geen Java sources zijn: HTML pagina's, afbeeldingen, CSS, JSP's, ...
- **lib** zal JAR bestanden (libraries) bevatten.
- `web.xml` is het configuratiebestand van de website.

### 5.3.2 Een welkompagina als statische HTML pagina

1. Je klikt met de rechtermuisknop op WebContent.
2. Je kiest New, HTML file.
3. Je tikt index bij File name.
4. Je kiest Finish.

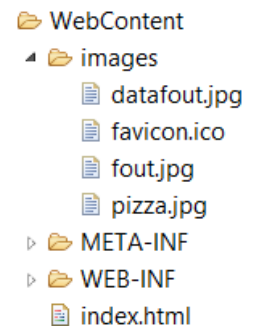


Je wijzigt de pagina:

```
<!doctype html>
<html lang='nl'>
  <head>
    <meta charset='UTF-8'>
    <title>Pizza Luigi</title>
    <link rel='icon' href='images/favicon.ico'>
    <meta name='viewport' content='width=device-width,initial-scale=1'>
    <link rel='stylesheet' href='styles/default.css'>
  </head>
  <body>
    <h1>Pizza Luigi</h1>
    <img src='images/pizza.jpg' alt='pizza' class='fullwidth'>
  </body>
</html>
```

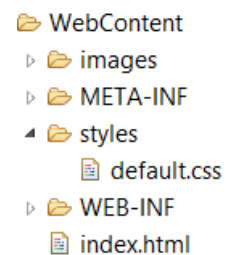
### 5.3.3 Afbeeldingen

1. Je klikt met de rechtermuisknop op WebContent.
2. Je kiest New, Folder.
3. Je tikt images bij Folder name.
4. Je kiest Finish.
5. Je selecteert in de Windows File Explorer de afbeeldingen bij de cursus.
6. Je klikt één afbeelding aan met de rechtermuisknop.
7. Je kiest Copy.
8. Je klikt in Eclipse met de rechtermuisknop op images en je kiest Paste.



### 5.3.4 CSS

1. Je klikt met de rechtermuisknop op WebContent.
2. Je kiest New, Folder.
3. Je tikt styles bij Folder name.
4. Je kiest Finish.
5. Je klikt in de Windows File Explorer met de rechtermuisknop op default.css (bij de cursus) en je kiest Copy.
6. Je klikt in Eclipse met de rechtermuisknop op styles en je kiest Paste.



### 5.3.5 GIT

Je commit de sources en je publiceert op GitHub.

### 5.3.6 Uittesten

Je kunt met Eclipse de website op Tomcat installeren en het resultaat in een browser zien:

1. Je kiest in het menu Window, Web Browser, Default system web browser.
2. Je klikt met de rechtermuisknop op het project in de Project Explorer.
3. Je kiest Run As, Run on Server.
4. Eclipse stelt de Tomcat server voor die je eerder in Eclipse configureerde.
5. Je kunt een vinkje plaatsen bij Always use this server ... en je kiest Finish.

### 5.3.7 WAR

Als de website af is, verpak je de bestanden van die website in een WAR bestand.

Je kan daarmee de website op elke Java webserver installeren.

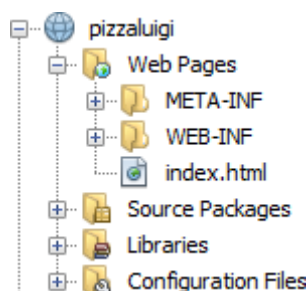
1. Je klikt in de Project Explorer met de rechtermuisknop op het project.
2. Je kiest Export, War file.
3. Je duidt met Browse een directory waarin Eclipse het WAR bestand plaatst en je kiest Finish.

## 5.4 NetBeans

### 5.4.1 Project

1. Je kiest in het menu File de opdracht New Project, Java Web, Web Application.
2. Je kiest Next.
3. Je tikt pizzaluigi bij Project name en je kiest Next.
4. Je controleert of bij Server je Tomcat server geselecteerd is.
5. Je kiest Finish.

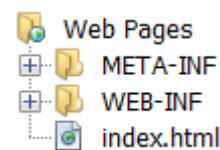
Je ziet de project structuur in het venster Projects :



- Web Pages zal onderdelen bevatten die geen Java sources zijn: HTML pagina's, afbeeldingen, CSS, JSP's, ...  
Je verwijdert de voorbeeldpagina index.html.
- Source packages zal Java sources bevatten.
- Libraries zal JAR bestanden (libraries) bevatten.
- Configuration Files zal configuratiebestanden bevatten.

### 5.4.2 Een statische HTML pagina als welkompagina

1. Je verwijdert in Web Pages de pagina index.html, om die zelf opnieuw te kunnen maken.
2. Je klikt met de rechtermuisknop op Web Pages en je kiest New, Other, Web, HTML.
3. Je kiest Next en je tikt index bij HTML File Name en je kiest Finish.

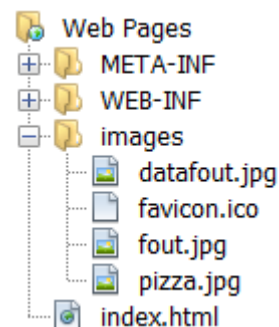


Je wijzigt de pagina:

```
<!doctype html>
<html lang='nl'>
  <head>
    <meta charset='UTF-8'>
    <title>Pizza Luigi</title>
    <link rel='icon' href='images/favicon.ico'>
    <meta name='viewport' content='width=device-width,initial-scale=1'>
    <link rel='stylesheet' href='styles/default.css'>
  </head>
  <body>
    <h1>Pizza Luigi</h1>
    <img src='images/pizza.jpg' alt='pizza' class='fullwidth'>
  </body>
</html>
```

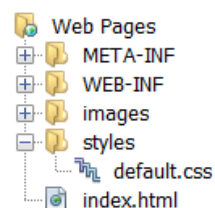
### 5.4.3 Afbeeldingen

1. Je klikt met de rechtermuisknop op Web Pages .
2. Je kiest New, Folder .
3. Je tikt images bij Folder Name en je kiest Finish .
4. Je selecteert in de Windows File Explorer alle afbeeldingen bij de cursus .
5. Je klikt één afbeelding aan met de rechtermuisknop .
6. Je kiest Copy .
7. Je klikt in NetBeans met de rechtermuisknop op images .
8. Je kiest Paste .



### 5.4.4 CSS

1. Je klikt met de rechtermuisknop op Web Page en je kiest New, Folder .
2. Je tikt styles bij Folder Name en je kiest Finish .
3. Je klikt in de Windows File Explorer met de rechtermuisknop op default.css (bij de cursus) en je kiest Copy .
4. Je klikt in NetBeans met de rechtermuisknop op styles .
5. Je kiest Paste .



### 5.4.5 GIT

Je commit de sources en je publiceert op GitHub.

### 5.4.6 Uittesten

Je kan vanuit de IDE de website op Tomcat installeren en het resultaat in een browser zien.

NetBeans gebruikt de default browser van het besturingssysteem.

(Je kunt een andere browser kiezen in het menu Tools, Options, General)

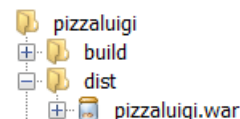
- Je klikt op de groene Run knop  in de toolbar van NetBeans

### 5.4.7 WAR

Als de website af is, verpak je al zijn bestanden in één WAR bestand.

Je kan met dit bestand de website op elke Java webserver installeren.

- Je klikt met de rechtermuisknop op het project.
- Je kiest Clean and Build.
- Je ziet het WAR bestand in het venster Files in de directory dist.

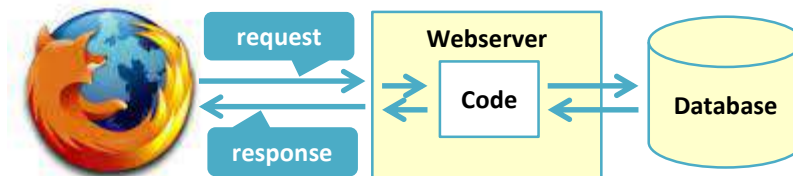


Frituur frida: zie takenbundel

## 6 EEN DYNAMISCHE PAGINA MET EEN SERVLET

### 6.1 Dynamische pagina

Een dynamische pagina is een stukje programmacode op de webserver.  
 Bij een request naar de URL van een dynamische pagina, voert de webserver de code uit.  
 Die code maakt HTML. De webserver stuurt die HTML als response naar de browser.  
 De code kan bijvoorbeeld informatie uit een database lezen en opnemen in de response.



Een website voert code uit bij een request naar `pizzaluigi.be/producten`.

1. De code maakt HTML:

```

<!doctype html>
<html lang='nl'>
  <body>
    <h1>Producten</h1>
    <ul>

```

2. De code leest producten uit de database en maakt per product HTML:

```

  <li>De productnaam</li>

```

3. De code maakt HTML:

```

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

```

### 6.2 Servlet

Een servlet is een Java class die de rol speelt van dynamische pagina.

- Een servlet is een class die erft van de class `javax.servlet.HttpServlet`.
- De class erft van `HttpServlet` het implementeren van de interface `Serializable` en bevat daarom `private static final long serialVersionUID = 1L;`
- Voor de class staat `@WebServlet` met een URL, bijvoorbeeld `@WebServlet("/index.htm")`. De webserver stuurt dan alle browser requests naar die URL naar deze servlet.
  - Dit is geen volledige URL (dus niet `www.pizzaluigi.be/index.htm`),
  - maar een relatieve URL (`/index.htm`) ten opzichte van de base URL van de website (`www.pizzaluigi.be`).

De URL moet beginnen met `/`, tenzij hij het jokerteken `*` bevat. `@WebServlet("*.do")` betekent dat de servlet alle requests naar URL's die eindigen op `.do` verwerkt.

- De method `doGet` van een servlet verwerkt GET request van de browser.
- De method `doPost` van een servlet verwerkt POST requests van de browser.
- Een servlet kan de method `doGet` hebben, de method `doPost`, of beide methods.



### 6.3 Eclipse 🌑

1. Je klikt met de rechtermuisknop op het project en je kiest New, Servlet.
2. Je maakt in de eerste wizard stap de servlet class.
  - a. Je tikt `be.vdab.servlets` bij Java package.
  - b. Je tikt `IndexServlet` bij Class name en je kiest Next.
3. Je associeert in de tweede wizard stap de servlet met een URL.
  - a. Je selecteert `/IndexServlet` in URL mappings en je kiest Edit.
  - b. Je tikt `/index.htm` bij Pattern en je kiest OK, Next.
4. Je voegt in de derde stap methods toe aan de servlet class.
  - a. Je verwijdert vinkjes bij Constructors from superclass en bij `doPost` (de eerste servlet verwerkt enkel GET requests).
  - b. Je kiest Finish.

Het project onderdeel Java Resources, src bevat een package `be.vdab.servlets`. Deze bevat een class `IndexServlet` en daarin de method `doGet`.

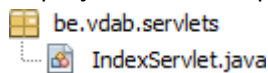


```
be.vdab.servlets
├── IndexServlet.java
│   ├── IndexServlet
│   │   └── serialVersionUID
│   └── doGet(HttpServletRequest request, HttpServletResponse response): void
```

### 6.4 NetBeans 🌑

1. Je klikt met de rechtermuisknop op het project en je kiest New, Servlet.
2. Je maakt in de eerste wizard stap de servlet class.
  - a. Je tikt `IndexServlet` bij Class Name.
  - b. Je tikt `be.vdab.servlets` bij Package en je kiest Next.
3. Je associeert in de tweede wizard stap de servlet met een URL.
  - a. Je wijzigt URL Pattern(s) naar `/index.htm` en je kiest Finish.

Je project bevat een package `be.vdab.servlets` met een class `IndexServlet`.



```
be.vdab.servlets
├── IndexServlet.java
```

Je vereenvoudigt `@WebServlet` bij de class: `@WebServlet("/index.htm")`

De class bevat drie methods: `processRequest`, `doGet` en `doPost`.

1. Je verwijdert de method `processRequest`. Deze bevat voorbeeldcode van NetBeans, maar is geen standaard servlet method.
2. Je verwijdert de method `doPost`: de servlet verwerkt enkel GET requests.
3. Je verwijdert de oproep van `processRequest` in de method `doGet`.
4. Je verwijdert de method `getServletInfo`. Ze is onbelangrijk.

### 6.5 Parameters van de methods `doGet` en `doPost`

De methods `doGet` en `doPost` hebben beide:

- een `HttpServletRequest` parameter die de browser request voorstelt
- een `HttpServletResponse` parameter die de response naar de browser voorstelt



## 6.6 Response

Je eerste servlet stuurt, afhankelijk van het tijdstip, de tekst Goede morgen, Goede middag of Goede avond naar de browser.

- Je roept in de doGet method op de response parameter de method getWriter op. Je krijgt een PrintWriter object.
- Je roept daarop de print of de println method op en je geeft HTML mee. De methods sturen die naar de browser. De println method verstuurt een extra Enter teken.

Gezien je bij het versturen van de welkomboodschap niets wijzigt aan de kant van de webserver, schrijf je de functionaliteit in de method doGet, niet in de method doPost.

Je wijzigt de code in de IndexServlet method doGet:

```
PrintWriter out = response.getWriter();
out.println("<!doctype html>");
out.println("<html lang='nl'><head>");
out.println("<title>Pizza Luigi</title></head>");
out.println("<body><h1>");
int uur = LocalDateTime.now().getHour();
out.print(uur >= 6 && uur < 12 ? "Goede morgen" :
    uur >= 12 && uur < 18 ? "Goede middag": "Goede avond");
out.println("</h1></body></html>");
```

Je commit de sources en je publiceert op GitHub.



Je verwijdert index.html en je start de website. Tomcat zoekt index.html en vindt die niet. Tomcat gebruikt dan index.htm als welkompagina.

De browser kan nog de output van index.html tonen, uit zijn browser-cache.

Je 'refresht' dan de pagina.

## 6.7 Wijziging servlet

Wanneer je de code van een servlet wijzigt, moet je de website niet herstarten.

- Je slaat de wijziging van de servlet op.
- Je wacht bij Eclipse  tot in het venster Console (onder in Eclipse) de boodschap INFO: Reloading Context with name [/pizzaluigi2] is completed verschijnt.
- Je wacht bij NetBeans  tot links onder de boodschap pizzaluigi deployed verschijnt.
- Je ververs de pagina in de browser.

## 6.8 Nadelen van HTML naar de browser sturen in een servlet

- ⊖ De IDE valideert geen HTML tussen " en ".
- ⊖ Als een web designer de HTML verfijnt, is de kans groot dat hij per ongeluk fouten aanbrengt in de Java code.
- ⊖ Je mengt voortdurend Java code en HTML. De HTML en de Java code zijn zo minder leesbaar.
- ⊖ Als je de HTML wil wijzigen, moet je een Java source wijzigen, van de applicatie een WAR bestand maken en dit WAR bestand installeren op de webserver. Dit is omslachtig.

Je pakt in de volgende hoofdstukken die nadelen aan.



Sluitingsdagen: zie takenbundel

## 7 EEN DYNAMISCHE PAGINA MET EEN JSP

Een JSP (Java Server Page) is een HTML source die ook Java code bevat.

### 7.1 De webserver maakt op basis van een JSP een servlet

Bij een eerste request naar een JSP

1. vertaalt de webserver de JSP naar een servlet source
2. compileert die naar Java bytecode
3. maakt een instance van die servlet
4. en stuurt de request naar die instance

De webserver bewaart de gecompileerde servlet op de harde schijf.

Hij slaat zo bij volgende requests het vertalen van JSP naar servlet over.

Als je de JSP source wijzigt, doet de webserver terug de vertaaltappen.

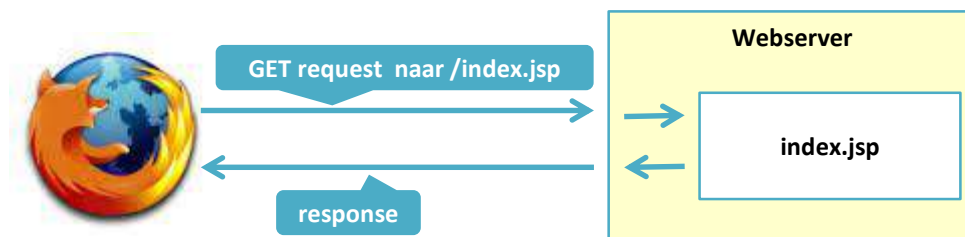
Als je een JSP wijzigt, zie je het resultaat onmiddellijk:

1. Je slaat de JSP op in de IDE.
2. Je 'refresh' de pagina in de browser.

### 7.2 URL van een JSP

De URL van een JSP is gelijk aan de naam van de JSP source.

De URL van `index.jsp` op de website `www.pizzaluigi.be` is `www.pizzaluigi.be/index.jsp`



### 7.3 JSP onderdelen

- HTML De webserver stuurt die HTML als een response naar de browser.
- Scriptlets Een scriptlet bevat Java statements, tussen `<%` en `%>`.
- Page directives Regels die beginnen met `<%@page` en eindigen op `%>`.

Een page directive beschrijft eigenschappen

- van de servlet die de webserver op basis van de JSP maakt
- van de response die je naar de browser stuurt

### 7.4 Eclipse

1. Je klikt met de rechtermuisknop op WebContent en je kiest New, JSP File.
2. Je tikt index bij File name en je kiest Finish.

### 7.5 NetBeans

1. Je klikt met de rechtermuisknop op Web Pages en je kiest New, JSP.
2. Je tikt index bij File Name en je kiest Finish.

## 7.6 index.jsp aanpassen

```
<!-- Een welkom pagina -->
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@page import='java.time.LocalDateTime'%>
<!doctype html>
<html lang='nl'>
  <head><title>Pizza Luigi</title></head>
  <body>
    <h1>
      <%
        int uur = LocalDateTime.now().getHour();
        out.print(uur >= 6 && uur < 12 ? "Goede morgen" :
          uur >= 12 && uur < 18 ? "Goede middag": "Goede avond");
      %>
    </h1>
  </body>
</html>
```

①  
②  
③  
④  
  
⑤  
⑥

- (1) Je tikt JSP commentaar tussen <!-- en -->.
- (2) Je stelt met deze page directive drie eigenschappen in:
  - a. Als je contentType text/html bevat en pageEncoding UTF-8, bevat de response een header Content-type met de waarde text/html; charset=UTF-8.
  - b. session beïnvloedt de servlet die de webserver maakt op basis van de JSP. De webserver maakt standaard in de servlet code om session variabelen bij te houden. Een session variabele is een variabele die de webserver per gebruiker bijhoudt (zoals een winkelmandje op een shopping website). session='false' geeft aan dat de JSP geen session variabelen nodig heeft. Dit bevordert de prestaties.
- (3) Je importeert met deze page directive de class LocalDateTime uit de package java.time. Je importeert een package in een JSP met deze syntax, niet met de Java syntax.
- (4) De webserver stuurt de HTML van de JSP als response body naar de browser.
- (5) Hier begint een scriptlet: een opeenvolging van Java statements.
- (6) In een scriptlet is automatisch een variabele out ter beschikking. Je stuurt met de methods print of println HTML naar de response.

Je plaatst in IndexServlet de regel @WebServlet("index.htm") in commentaar.

De webserver gebuikt nu index.jsp als welkompagina.

Je commit de sources en je publiceert op GitHub. Je kunt de website terug uitproberen.

Een JSP met scriptlets is onhandig (je lost dit op in het volgende hoofdstuk):

- ⊖ De IDE valideert de HTML, die je in scriptlets maakt, niet.
- ⊖ Als een web designer de pagina verfijnt is de kans groot dat hij per ongeluk fouten aanbrengt in de Java code.
- ⊖ Je mengt Java code en HTML code. De HTML en de Java code zijn zo minder leesbaar.

## 7.7 JSP template

Het is vervelend telkens dezelfde kleine wijzigingen te doen in een nieuwe JSP.

Je tikt bijvoorbeeld telkens session="false" in de JSP.

Je lost dit op: je wijzigt de template (sjabloon) waarop elke nieuwe JSP gebaseerd is.

### 7.7.1 Eclipse

1. Je kiest in het menu Window de opdracht Preferences.
2. Je opent het onderdeel Web, JSP Files, Editor, Templates.
3. Je selecteert rechts New JSP File (html) en je kiest Edit.

### 7.7.2 NetBeans

1. Je kiest in het menu Tools de opdracht Templates.
2. Je opent het onderdeel Web en selecteert daarin JSP.
3. Je kiest Open in Editor en je klikt OK bij een eventuele warning.

Je wijzigt bij Eclipse en bij NetBeans de template:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<!doctype html>
<html>
<head>
<title></title>
</head>
<body>
</body>
</html>
```

- Je slaat bij NetBeans de template op en je sluit de template.
- Je kiest bij Eclipse OK en OK.

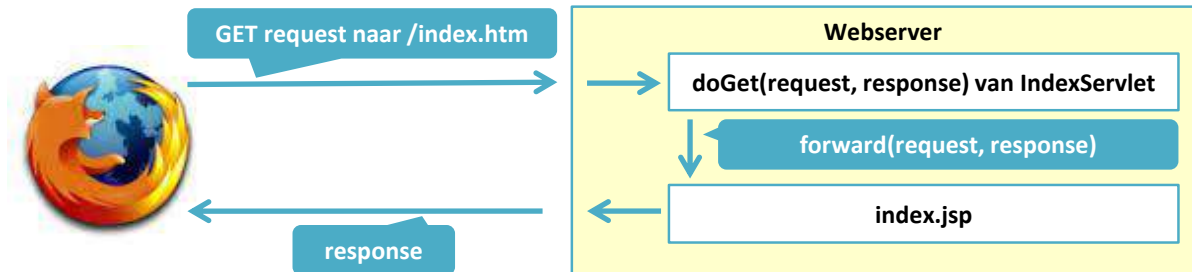


Sluitingsdagen 2: zie takenbundel

## 8 SERVLET ÉN JSP COMBINEREN

Je verwerkt een request in een servlet én in een JSP.

- Je verwerkt de request in de servlet method `doGet` of `doPost`. Die bevat Java code, geen HTML.
- Je geeft de request daarna door aan een JSP. Dit heet 'to forward'.  
De JSP bevat HTML, geen scriptlets, dus ook geen Java code.



De browser weet niet dat de `index.jsp` meewerkt in het verwerken van de request:



- De browser doet een request naar de URL van de servlet: `index.htm`.
- De browser krijgt een response met HTML, maar weet niet welk onderdeel deze HTML aanmaakte.  
In zijn adresbalk staat nog altijd de URL van de servlet, niet die van de JSP.

### 8.1 Request doorgeven van een servlet naar een JSP

Je geeft de request door van een servlet naar een JSP met een `RequestDispatcher` object.

- Je krijgt dit object van de request method `getRequestDispatcher`.
- Je geeft aan die method de naam van JSP mee naar waar je de request doorgeeft:  
`RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp");`
- Je geeft de request en de response door met de `RequestDispatcher` method `forward`:  
`dispatcher.forward(request, response);`

### 8.2 Request attribuut

Je wil regelmatig een waarde (getal, tekst), die je in de servlet aanmaakt, doorgeven aan de JSP. In die JSP neem je die waarde dan op in de HTML die je naar de browser stuurt.

Zo'n waarde heet een `request` attribuut of een `request scope variable`.

Je geeft de waarde door aan de JSP met de request method `setAttribute`. Hij heeft 2 parameters

- een naam voor het request attribuut.  
Je zal in de JSP met die naam het request attribuut aanspreken.
- de waarde van het request attribuut.

Voorbeeld: een request attribuut begroeting met de waarde `Goede morgen` doorgeven:

```
request.setAttribute("begroeting", "Goede morgen");
```

Je kunt meerdere request attributen doorgeven. Je roept telkens de method

```
request.setAttribute(...)
```

op. Je geeft elk request attribuut een unieke naam.

### 8.3 Request attribuut lezen in een JSP

Je leest in een JSP de request attribuut waarde met een miniprogrammeertaal: EL (Expression Language). Elke EL expressie begint met `${` en eindigt op `}`

Je leest de inhoud van het request attribuut begroeting met `${begroeting}`

Nadat de JSP verwerkt is, verdwijnen request attributen automatisch uit het geheugen.

### 8.3.1 IndexServlet

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/index.htm")
public class IndexServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "index.jsp";

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        int uur = LocalDateTime.now().getHour();
        request.setAttribute("begroeting", uur >= 6 && uur < 12 ? "Goede morgen" :
            uur >= 12 && uur < 18 ? "Goede middag": "Goede avond"); ❶
        request.getRequestDispatcher(VIEW).forward(request, response); ❷
    }
}
```

(1) Je maakt een request attribuut begroeting met als inhoud de af te beelden boodschap.

(2) Je maakt een RequestDispatcher object en je geeft de request en response door naar de JSP.

### 8.3.2 index.jsp

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<!doctype html>
<html lang='nl'>
    <head>
        <title>Pizza Luigi</title>
        <link rel='icon' href='images/favicon.ico'>
        <meta name='viewport' content='width=device-width,initial-scale=1'>
        <link rel='stylesheet' href='styles/default.css'>
    </head>
    <body>
        <h1>Pizza Luigi</h1>
        <img src='images/pizza.jpg' alt='pizza' class='fullwidth'>
        <h2>${begroeting}</h2> ❶
    </body>
</html>
```

(1) De JSP vervangt `${begroeting}` door de inhoud van het request attribuut begroeting.

Je kunt de website terug uitproberen. De servlet is nu terug de welkompagina.

## 8.4 JSP's in de folder WEB-INF

Je kunt met de browser `index.jsp` nog opvragen op `localhost:8080/pizzaluigi/index.jsp`

Dit is zinloos: `IndexServlet` heeft die request niet verwerkt en dus is de EL expressie `${begroeting}` leeg.

Je vermijdt een rechtstreekse request naar een JSP, door de JSP te verplaatsen naar de beveiligde project folder `WEB-INF`.

- Een browser kan bestanden in `WEB-INF` (en subfolders) *niet* benaderen.
- Een servlet kan bestanden in `WEB-INF` (en subfolders) *wel* benaderen.

Je doet aanpassingen:

1. Je maakt in `WEB-INF` een folder `JSP` en je sleept `index.jsp` naar die folder.
2. Je wijzigt in `IndexServlet` de constante `VIEW` naar `"/WEB-INF/JSP/index.jsp"`.

Je commit de sources en je publiceert op GitHub. Je kunt de website terug uitproberen.



Sluitingsdagen 3: zie takenbundel

## 9 \${EXPRESSION LANGUAGE}

### 9.1 Request attribuut met een primitief datatype

EL neemt een request attribuut waarde met een primitief datatype op in de response  
 Bij een request attribuut kinderen met de waarde 3, maakt de JSP code  
 Luigi's \${kinderen} kinderen de HTML Luigi's 3 kinderen

### 9.2 Request attribuut met een object

EL voert op een object in een request attribuut de method `toString` uit en neemt het resultaat op in de response.

Je maakt een package `be.vdab.entities` en daarin de class `Begroeting`.  
 (Een entity is een object dat iets uit de werkelijkheid voorstelt.)

```
package be.vdab.entities;
import java.time.LocalDateTime;
public class Begroeting {
    @Override
    public String toString() {
        int uur = LocalDateTime.now().getHour();
        return uur >= 6 && uur < 12 ? "Goede morgen" :
            uur >= 12 && uur < 18 ? "Goede middag" : "Goede avond";
    }
}
```

Je gebruikt de class in de `IndexServlet` method `doGet`. Je vervangt de code in deze method door

```
request.setAttribute("begroeting", new Begroeting());
request.getRequestDispatcher(VIEW).forward(request, response);
```

Je commit de sources en je publiceert op GitHub. Je kunt de website opnieuw uitproberen

### 9.3 Verwijzing naar een onbestaand request attribuut

Als je verwijst naar een onbestaand request attribuut, maakt EL een lege tekst.

### 9.4 Hard gecodeerde waarden

gehele getallen	<code>\${7}</code>	strings (tussen " of ')	<code>\${"james"}</code>
getallen met decimalen	<code>\${40.3399}</code>	booleans (true of false)	<code>\${false}</code>

### 9.5 Wiskundige operatoren

- `+` `-` `*` `/` en `%`
- **div** is synoniem voor `/` (delen), **mod** voor `%` (rest bepalen bij delen)

Als `getal` de waarde 9 bevat geeft `${getal / 2}` de waarde 4.5.

### 9.6 Vergelijkingsoperatoren

- `==` `!=` `>` `<` `>=` `<=`
- **eq** is synoniem voor `==`, **ne** voor `!=`, **gt** voor `>`, **ge** voor `>=`, **lt** voor `<`, **le** voor `<=`

Als `getal` de waarde 9 bevat geeft `${getal == 9}` de waarde `true`.

### 9.7 Logische operatoren

- `!` `&&` `||`
- **not** is synoniem voor `!`, **and** voor `&&`, **or** voor `||`

Als `getal` de waarde 9 bevat geeft `${getal > 8 && getal < 10}` de waarde `true`.

### 9.8 Conditionele operator ? :

Syntax: `voorwaarde ? waardeAlsVoorwaardeTrue : waardeAlsVoorwaardeFalse`

Als `getal` de waarde 7 bevat geeft `${getal == 7 ? "geluk" : "ongeluk"}` de waarde `geluk`.

## 9.9 Operator empty

Je vermeldt na **empty** een expressie. **empty** geeft true terug als de expressie gelijk is aan:

- **null**
- een lege String
- een lege verzameling (array, List, Set, Map)
- de naam van een onbestaand request attribuut

Als klanten een lege List bevat geeft `${empty klanten}` de waarde true.

## 9.10 Één element uit een verzameling lezen

### 9.10.1 Array

Je leest één array element met de syntax `${requestAttribuut[indexVanHetElement]}`.

Als namen een array bevat met de elementen Joe en Averell geeft `${namen[0]}` de waarde Joe.

### 9.10.2 List

Als namen een List bevat met de elementen Joe en Averell geeft `${namen[0]}` de waarde Joe.

### 9.10.3 Map

Als eigenschappen een Map bevat met volgende entries:

key	value
Joe	driftig
Averell	hongerig

geeft `${eigenschappen["Averell"]}` de waarde `hongerig`.

Als de keys String zijn, bestaat een korte syntax: `${eigenschappen.Averell}`.

Opgepast, als de key een geheel getal is, moet dit in je Java code een long zijn, geen int.

## 9.11 Resultaat van een method oproep

Je kan met EL het resultaat van een method oproep lezen, als de website draait op een webserver die minstens servlets 3.0 ondersteunt

Als familienaam de waarde `dalton` bevat, geeft `${familienaam.length()}` de waarde 6.



Sluitingsdagen 4: zie takenbundel



## 10 JAVABEAN

Een JavaBean is een object waarvan de class voldoet aan bepaalde voorwaarden. Je leert de JavaBean standaard kennen en hoe EL die standaard gebruikt.

### 10.1 Getters en setters

Je maakt in de package `be.vdab.entities` de class `Persoon`:

```
package be.vdab.entities;
public class Persoon {
    private String voornaam;
    private String familienaam;
    private int aantalKinderen;
    private boolean gehuwd;
}
```

Persoon
-voornaam: String
-familienaam: String
-aantalKinderen: int
-gehuwd: boolean

De JavaBean standaard definieert dat je per attribuut:

- een public method maakt waarmee je de attribuut waarde kan opvragen.
  - De method naam begint met `get`. Als het attribuut boolean is, begint de naam met `is`.
  - De rest van de methodnaam is gelijk aan de naam van het bijbehorende attribuut, met de eerste letter als hoofdletter.
  - Het method returntype is gelijk aan het attribuut type.
  - De method heeft geen parameters.
  - De method returnt de attribuut waarde. Je vindt die in de bijbehorende private variabele.

Je maakt deze methods (*getters* genoemd) in de class `Persoon`:

```
public String getVoornaam() {
    return voornaam;
}
public String getFamilienaam() {
    return familienaam;
}
public int getAantalKinderen() {
    return aantalKinderen;
}
public boolean isGehuwd() {
    return gehuwd;
}
```

- een public method maakt waarmee je een waarde kan invullen in het attribuut.
  - De method naam begint met `set`.
  - De rest van de methodnaam is gelijk aan de bijbehorende attribuut naam, met de eerste letter als hoofdletter.
  - Het method returntype is `void`.
  - De method heeft één parameter: de waarde die je wilt invullen in het attribuut. Het parameter type is gelijk aan het attribuut type.

Je maakt deze methods (*setters* genoemd) in de class `Persoon`:

```
public void setVoornaam(String voornaam) {
    this.voornaam = voornaam;
}
public void setFamilienaam(String familienaam) {
    this.familienaam = familienaam;
}
public void setAantalKinderen(int aantalKinderen) {
    this.aantalKinderen = aantalKinderen;
}
public void setGehuwd(boolean gehuwd) {
    this.gehuwd = gehuwd;
}
```

### 10.2 ReadOnly attributen

Een attribuut dat enkel een getter (en geen setter) heeft, is een readonly attribuut: een attribuut dat je kan lezen, maar niet kan wijzigen. Voorbeeld:

```
public String getNaam() {
    return voornaam + ' ' + familienaam;
}
```

Je ziet dat een readonly attribuut niet altijd een eigen private variabele nodig heeft.

### 10.3 Getters en Setters maken met je IDE

Je verwijdert de getters en setters in de class Persoon.

#### 10.3.1 Eclipse

1. Je opent de source van de class Persoon.
2. Je kiest in het menu Source de opdracht Generate Getters and Setters.
3. Je kiest de knop Select All en daarna OK.

#### 10.3.2 NetBeans

1. Je opent de source van de class Persoon.
2. Je kiest in het menu Refactor de opdracht Encapsulate Fields.
3. Je kiest de knop Select All en daarna Refactor.

### 10.4 Constructors

Een JavaBean moet een public default constructor hebben.

Dit is momenteel het geval: Je tikte geen constructor, dan maakt de compiler een public default constructor. Je tikt nu een geparametriseerde constructor. De compiler maakt dan geen default constructor meer, dus moet je ook de default constructor zelf tikken.

```
public Persoon(String voornaam, String familienaam, int aantalKinderen,
    boolean gehuwd) {
    this.voornaam = voornaam;
    this.familienaam = familienaam;
    this.aantalKinderen = aantalKinderen;
    this.gehuwd = gehuwd;
}
public Persoon() { // default constructor
}
```

### 10.5 Constructors maken met je IDE

Je verwijdert de constructors in de class Persoon.

#### 10.5.1 Eclipse

1. Je opent de source van de class Persoon.
2. Je kiest in het menu Source de opdracht Generate Constructor using Fields.
3. Je plaatst een vinkje bij Omit call to default constructor super().
4. Je kiest Deselect all en je kiest OK.  
Je krijgt een default constructor.
5. Je herhaalt stappen 1, 2 en 3.
6. Je kiest OK.  
Je krijgt een geparametriseerde constructor.

#### 10.5.2 NetBeans

1. Je opent de source van de class Persoon.
2. Je kiest in het menu Source de opdracht Insert Code, Constructor.
3. Je selecteert geen enkele private variabele en je kiest Generate.  
Je krijgt een default constructor.
4. Je herhaalt stappen 1 en 2.
5. Je selecteert alle private variabele en je kiest Generate.  
Je krijgt een geparametriseerde constructor.

## 10.6 EL en JavaBeans

Als een request attribuut een JavaBean object bevat, lees je met EL een attribuut waarde van dat object met de syntax `${requestAttribuut.attribuut}`.

Als het request attribuut persoon een Persoon object bevat, lees je bijvoorbeeld:

- de naam via `${persoon.naam}`  
EL vertaalt naam naar een method oproep `getNaam()` op het request attribuut persoon
- het aantal kinderen via `${persoon.aantalKinderen}`
- de gehuwde staat via `${persoon.gehuwd}`

Je voegt een opdracht vooraan toe in de method `doGet` van `IndexServlet`:

```
request.setAttribute("zaakvoerder", new Persoon("Luigi", "Peperone", 7, true));
```

Je tikt in `index.jsp` voor `</body>`:

```
<h2>De zaakvoerder</h2>
<dl>
  <dt>Naam</dt><dd>${zaakvoerder.naam}</dd>
  <dt>Aantal kinderen</dt><dd>${zaakvoerder.aantalKinderen}</dd>
  <dt>Gehuwd</dt><dd>${zaakvoerder.gehuwd ? 'Ja' : 'Nee'}</dd>
</dl>
```

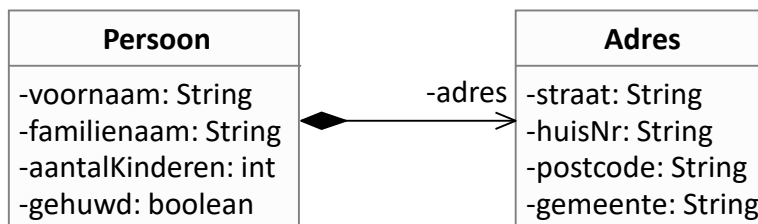
Je kunt de website terug uitproberen.

## 10.7 Geneste attributen

Een attribuut kan een object zijn, dat zelf attributen bevat.

Je maakt als voorbeeld een attribuut adres in de class `Persoon`.

Het type is een class `Adres`, met de attributen `straat`, `huisNr`, `postcode` en `gemeente`.



Je spreekt een genest attribuut aan als `${requestAttribuut.attribuut.genestAttribuut}`.

Als een request attribuut persoon een Persoon object bevat,

spreek je met volgende expressie het genest attribuut straat aan: `${persoon.adres.straat}`.

Je maakt in `be.vdab.entities` een class `Adres`

```
package be.vdab.entities;
import java.io.Serializable;
public class Adres {
    private String straat;
    private String huisNr;
    private int postcode;
    private String gemeente;
    // Je maakt zelf getters en setters voor de private variabelen met je IDE
    // Je maakt zelf een default en een geparametriseerde constructor met je IDE
}
```

Je voegt code toe aan de class `Persoon`:

```
private Adres adres; // Je maakt zelf een bijbehorende getter en setter
```

Je voegt een parameter toe aan de geparametriseerde constructor van `Persoon`:

```
public Persoon(..., Adres adres) {
```

Je voegt een opdracht toe in deze constructor:

```
this.adres = adres;
```

Je wijzigt de eerste opdracht in de method `doGet` van `IndexServlet`:

```
request.setAttribute("zaakvoerder", new Persoon("Luigi", "Peperone", 7, true,  
    new Adres("Grote markt", "3", 9700, "Oudenaarde")));
```

Je tikt in `index.jsp` voor `</dl>`:

```
<dt>Adres</dt>  
<dd>${zaakvoerder.adres.straat} ${zaakvoerder.adres.huisNr}<br>  
    ${zaakvoerder.adres.postcode} ${zaakvoerder.adres.gemeente}</dd>
```

Je commit de sources en je publiceert op GitHub. Je kunt de website terug uitproberen.



Adres: zie takenbundel

## 11 JSTL (JSP STANDARD TAG LIBRARY)

Je kent al HTML tags <head>, <h1>, ... De browser verwerkt die tags.

JSTL tags zijn echter server-sided tags, verwerkt door de webserver.

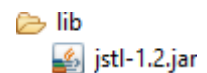
JSTL is één JAR bestand, met 5 libraries. Elke library heeft een 'menselijke' naam.

Menselijke naam	Functionaliteit van de tags in die library
Core	Itereren, condities, URL beheer.
Internationalization	Getalopmaak, datumopmaak, vertaalbare teksten.
Functions	Functies op strings en collections. Je kunt sinds Servlet 3.0 met EL rechtstreeks methods oproepen op Strings en Collections. Je hebt deze library dus niet meer nodig.
SQL	Databasetoegang. Het is een 'best practice' om de database aan te spreken met Java code, niet met de JSP tags uit deze library.
XML	XML produceren en lezen. Deze library valt buiten het bereik van de cursus.

### 11.1 JSTL JAR bestand toevoegen aan het project

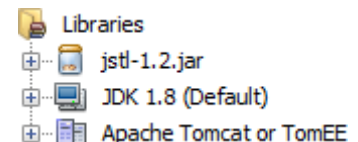
#### 11.1.1 Eclipse

1. Je selecteert jstl-1.2.jar (bij de cursus) in de Windows File Explorer met de rechtermuisknop en je kiest Copy.
2. Je klikt in Eclipse met de rechtermuisknop op de folder lib (in WebContent, WEB-INF) en je kiest Paste.



#### 11.1.2 NetBeans

1. Je klikt met de rechtermuisknop op het project onderdeel Libraries.
2. Je kiest Add JAR/Folder... en je selecteert jstl-1.2.jar (bij de cursus).



### 11.2 Tag names en tag URI's

Elke tag heeft een naam.

- Je itereert met de tag **forEach** over een verzameling.
- Je voert met de tag **if** een JSP onderdeel conditioneel uit.
- ...

Een tag library bevat meerdere tags. Een tag library wordt uniek geïdentificeerd met een URI.

De tags uit de core library hebben de URI <http://java.sun.com/jsp/jstl/core>.



De URI moet geen bestaande URL op het internet te zijn. De URI wordt ook niet als een bestaande URL op het internet opgezocht tijdens het verwerken van een tag library. De conventie is wel dat het begin van een URI een URL is. Dit garandeert dat elke URI uniek is, wat belangrijk is: de URI is de unieke identificatie van de tag library.

Als je een tag library in een JSP gebruikt, moet je boven in de JSP één keer de tag library URI associëren met een eigen gekozen prefix.

Je doet dit met de taglib directive. Die heeft volgende syntax:

```
<%@taglib prefix="gekozenPrefix" uri="URIVanDeTagLibrary"%>
```

Je associeert standaard de prefix c met de URI <http://java.sun.com/jsp/jstl/core>.

```
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
```

Je verwijst in die pagina naar een tag uit de library met de prefix, dus: <c:forEach>, <c:if>

### 11.3 <c:forEach>

Je itereert met `<c:forEach>` over de elementen van een verzameling:

```
<c:forEach var='eenVariabele' items='${eenVerzameling}'>
...
</c:forEach>
```

`forEach` biedt bij elke iteratie het volgend element aan in de variabele `eenVariabele`.

Je probeert dit uit: je maakt een pagina waarin de gebruiker pizza's ziet.

Je maakt in `be.vdab.servlets` een servlet `PizzasServlet`.

Die verwerkt GET requests naar de URL `/pizzas.htm`:

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/pizzas.htm")
public class PizzasServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/pizzas.jsp";
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setAttribute("pizzas",
            Arrays.asList("Prosciutto", "Margherita", "Calzone"));
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

- (1) We gebruiken in de cursus altijd de extensie `htm` in de URL van een Servlet. Dit is belangrijk voor het hoofdstuk Filters verder in de cursus, waarin we het onderscheid willen maken tussen de URL's van servlets (`*.htm`) en de URL's van statische pagina's (`*.html`).

Je maakt `pizzas.jsp` in `WEB-INF/JSP`:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
    <head>
        <title>Pizza's</title>
        <link rel='icon' href='images/favicon.ico'>
        <meta name='viewport' content='width=device-width,initial-scale=1'>
        <link rel='stylesheet' href='styles/default.css'>
    </head>
    <body>
        <h1>Pizza's</h1>
        <ul class='zebra'>
            <c:forEach var='pizza' items='${pizzas}'>
                <li>${pizza}</li>
            </c:forEach>
        </ul>
    </body>
</html>
```

- (1) Je associeert de prefix `c` met de URI van de core library die de tag `forEach` bevat.
- (2) Je itereert over de List in het request attribuut `pizzas`.  
Deze variabele `pizza` wijst bij elke iteratie naar een volgend element.
- (3) Je toont het huidige element.
- (4) Je sluit de `forEach` af.

Je kunt dit uitproberen op `http://localhost:8080/pizzaluigi/pizzas.htm`

### 11.3.1 Itereren over een verzameling JavaBeans

Je maakt in `be.vdab.entities` een class `Pizza`:

```
package be.vdab.entities;
import java.math.BigDecimal;
public class Pizza {
    private long id;
    private String naam;
    private BigDecimal prijs;
    private boolean pikant;
    // je maakt getters voor alle private variabelen
    public Pizza(String naam, BigDecimal prijs, boolean pikant) {
        setNaam(naam);
        setPrijs(prijs);
        setPikant(pikant);
    }
    public Pizza(long id, String naam, BigDecimal prijs, boolean pikant) {
        this(naam, prijs, pikant); // vorige constructor oproepen
        setId(id);
    }
    public void setId(long id) {
        this.id = id;
    }
    public void setPikant(boolean pikant) {
        this.pikant = pikant;
    }
    public static boolean isNaamValid(String naam) { // Deze static function
        // valideert de naam zonder een Pizza instance te moeten maken (zie verder)
        return naam != null && ! naam.trim().isEmpty();
    }
    public void setNaam(String naam) {
        if ( ! isNaamValid(naam)) {
            throw new IllegalArgumentException();
        }
        this.naam = naam;
    }
    public static boolean isPrijsValid(BigDecimal prijs) {
        return prijs != null && prijs.compareTo(BigDecimal.ZERO) >= 0;
    }
    public void setPrijs(BigDecimal prijs) {
        if ( ! isPrijsValid(prijs)) {
            throw new IllegalArgumentException();
        }
        this.prijs = prijs;
    }
}
```

Je vervangt in `PizzasServlet` `Arrays.asList(...)`:

```
Arrays.asList(
    new Pizza(12, "Prosciutto", BigDecimal.valueOf(4), true),
    new Pizza(14, "Margherita", BigDecimal.valueOf(5), false),
    new Pizza(17, "Calzone", BigDecimal.valueOf(4), false))
```

Je vervangt in `pizzas.jsp` de regel `<li>${pizza}</li>`:

```
<li>${pizza.naam} ${pizza.prijs}&euro;</li>
```

①

(1) Je roept met `${pizza.naam}` de method `getNaam` op van het `Pizza` object.

Je roept met `${pizza.prijs}` de method `getPrijs` op van het `Pizza` object.

Je kunt dit terug uitproberen.

### 11.3.2 Itereren over een Map

Je kunt met `forEach` itereren over een Map. De variabele bij `var` bevat per iteratie:

- een eigenschap `key` met de key van de huidige entry `${eenVariabele.key}`
- een eigenschap `value` met de value van de huidige entry `${eenVariabele.value}`

Je vervangt in `PizzasServlet` de opdracht `request.setAttribute(...)`:

```
Map<Long, Pizza> pizzas = new LinkedHashMap<>(); // keys zijn pizza ids
pizzas.put(12L, new Pizza(12, "Prosciutto", BigDecimal.valueOf(4), true));
pizzas.put(14L, new Pizza(14, "Margehrita", BigDecimal.valueOf(5), false));
pizzas.put(17L, new Pizza(17, "Calzone", BigDecimal.valueOf(4), false));
request.setAttribute("pizzas", pizzas);
```

Je vervangt in `pizzas.jsp` de regels `<forEach> ... </forEach>`:

```
<c:forEach var='entry' items='${pizzas}'>
  <li>${entry.key}: ${entry.value.naam} ${entry.value.prijs}&euro;</li>
</c:forEach>
```

Je kunt dit terug uitproberen

### 11.3.3 begin, step en end attributen

Je gebruikt de `forEach` attributen `begin`, `step` en/of `end` om elementen over te slaan:

begin	De iteratie start met het element dat dit getal als index heeft.
end	De iteratie loopt tot en met het element dat dit getal als index heeft.
step	Bij elke iteratie selecteert <code>forEach</code> het element met een index <code>step</code> hoger dan de index van het element uit de vorige iteratie.

### 11.3.4 Itereren zonder verzameling

Je kunt met `forEach` itereren zonder bijbehorende verzameling.

Je geeft met de attributen `begin` en `end` aan hoeveel keer `forEach` moet itereren.

Voorbeeld: vier keer itereren: `<c:forEach var='index' begin='1' end='4'>`

Je wijzigt in `pizzas.jsp` het element `h1`

```
<h1>Pizza's
  <c:forEach begin='1' end='5'>
    &#9733; <%-- de HTML code van een ster --%>
  </c:forEach>
</h1>
```

Je kunt dit terug uitproberen.

### 11.3.5 varStatus attribuut

Je kunt aan `forEach` een attribuut `varStatus` toevoegen, met een variabelenaam

```
<c:forEach var='eenVar' items='${eenVerzameling}' varStatus='status'>
  ...
</c:forEach>
```

`status` bevat in de `forEach` iteratie de iteratiestatus en heeft volgende eigenschappen:

Eigenschap	Betekenis
count	volgnummer van de huidige iteratie, de nummering begint vanaf 1.
index	volgnummer van de huidige iteratie, de nummering begint vanaf 0. Als ook het attribuut <code>begin</code> is meegegeven, wordt de inhoud van <code>begin</code> bijgeteld.
first	true bij de eerste iteratie, false bij de volgende iteraties.
last	true bij de laatste iteratie, false bij de vorige iteraties.



Bij de iteraties van de forEach

```
<c:forEach begin="4" end="6" varStatus="status">
...
</c:forEach>
```

hebben de status eigenschappen deze waarden:

<code>\${status.count}</code>	<code>\${status.index}</code>	<code>\${status.first}</code>	<code>\${status.last}</code>
1	4	true	false
2	5	false	false
3	6	false	True

#### 11.4 <c:if>

Je voert met `<c:if>` een deel van de JSP enkel uit als een voorwaarde true is:

```
<c:if test='voorwaardeUitgedruktAlsELExpressie'>
...
</c:if>
```

Voorbeeld: je vervangt in pizzas.jsp de regel `<li> ... </li>`:

```
<li>
${entry.key}: ${entry.value.naam} ${entry.value.prijs}&euro;
  <c:if test='${entry.value.pikant}'>
    pikant
  </c:if>
</li>
```

Je kunt dit terug uitproberen.

`<c:if>` heeft geen else gedeelte. `<c:choose>` is daartoe een oplossing

#### 11.5 <c:choose>

Deze tag is vergelijkbaar met het switch statement uit Java:

```
<c:choose>
  <c:when test='${voorwaarde1}'>
    Dit deel wordt enkel uitgevoerd als voorwaarde1 true is
  </c:when>
  <c:when test='${voorwaarde2}'>
    Dit deel wordt enkel uitgevoerd als voorwaarde2 true is
  </c:when>
  <c:otherwise>
    Dit deel wordt uitgevoerd als alle voorwaarden bij <c:when> false zijn.
  </c:otherwise>
</c:choose>
```

Voorbeeld: je vervangt in pizzas.jsp de regels `<c:if ...> ... </c:if>`:

```
<c:choose>
  <c:when test='${entry.value.pikant}'>
    pikant
  </c:when>
  <c:otherwise>
    niet pikant
  </c:otherwise>
</c:choose>
```

Je kunt dit terug uitproberen.

Je kunt soms een `if ... else` kort uitdrukken met de EL conditionele operator `?` :

`<c:choose ...> ... </c:choose>` wordt `${entry.value.pikant ? "pikant" : "niet pikant"}`

## 11.6 <c:out>

Je moet sommige tekens als een character entity code in HTML opnemen: & als &amp;

<c:out> doet die vertaalslag: `<c:out value='tekst_of_ELExpressie' />`

Je voegt in de PizzasServlet method doGet een pizza toe, met een naam die & bevat:

```
pizzas.put(23L, new Pizza(23, "Fungi & Olive", BigDecimal.valueOf(5), false));
```

Je vervangt in pizzas.jsp `${entry.value.naam}` door `<c:out value='${entry.value.naam}' />`

Je kunt dit terug uitproberen.

Je klikt met de rechtermuisknop in de pagina in de browser en je kiest View (page) source.

Je ziet dat Fungi & Olive vervangen is door het correcte *Fungi & Olive*

## 11.7 <c:url>

Je maakt met <c:url> een URL.

Je kunt achteraan op de URL parameters meegeven (de query string):

```
<c:url value='url' var='resultaatURL'>
  <c:param name='naamEersteParameter' value='waardeEersteParameter' />
  <c:param name='naamTweedeParameter' value='waardeTweedeParameter' />
</c:url>
```

Je maakt bij elke pizza een hyperlink naar een detailpagina over die pizza.

De URL in de hyperlink bevat een parameter met het pizza id.

Bij pizza 12 is de URL `/pizzaluigi/pizzas/detail?id=12`

Je tikt voor </li>:

```
<c:url value='/pizzas/detail.htm' var='detailURL'>
  <c:param name='id' value='${entry.key}' />
</c:url>
<a href='${detailURL}'>Detail</a>
```

Je kan dit terug uitproberen.

Als je klikt op een Detail hyperlink, krijg je een response met foutcode 404 (Not Found):

er is nu nog geen servlet geassocieerd met de URL `/pizzas/detail.htm`

Als je de muisaanwijzer laat rusten op een Detail hyperlink,

zie je wel al de URL die <c:url> opbouwde, inclusief de parameter id.

## 11.8 <c:import>

Je tikt code, die je nodig hebt in meerdere JSP's, in een kleine JSP,

die je daarna met <c:import> importeert in die andere JSP's.

Syntax: `<c:import url='LocatieEnNaamVanDeTeImporterenJSP' />`

Je maakt menu.jsp in WEB-INF/JSP:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<header>
<nav><ul>
<li><a href="<c:url value='/' />">Welkom</a></li>
<li><a href="<c:url value='/pizzas.htm' />">Pizza's</a></li>
<li><a href="<c:url value='/statistiek.htm' />">Statistiek</a></li>
<li><a href="<c:url value='/pizzas/tussensprijzen.htm' />">Pizza's tussen prijzen
</a></li>
<li><a href="<c:url value='/pizzas/voorkeuren.htm' />">Voorkeurpizza's</a></li>
<li><a href="<c:url value='/pizzas/toevoegen.htm' />">Pizza toevoegen</a></li>
<li><a href="<c:url value='/identificatie.htm' />">Identificatie</a></li>
<li><a href="<c:url value='/headers.htm' />">Headers</a></li>
<li><a href="<c:url value='/pizzas/bestellen.htm' />">Bestellen</a></li>
</ul></nav>
</header>
```

Je importeert menu.jsp in pizzas.jsp, na <body>:

```
<c:import url='/WEB-INF/JSP/menu.jsp'/>
```

De eerste slash verwijst naar het “web onderdeel” van het project.

Je tikt als tweede regel in index.jsp:

```
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
```

Je tikt na <body>:

```
<c:import url='/WEB-INF/JSP/menu.jsp'/>
```

Je kunt de website uitproberen.

### 11.8.1 Parameters

Een import bestand kan ook parameters hebben.

Je maakt head.jsp in WEB-INF/JSP:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<title>${param.title}</title>
<link rel='icon' href='images/favicon.ico'>
<meta name='viewport' content='width=device-width,initial-scale=1'>
<link rel='stylesheet' href='styles/default.css'>
```

❶

(1) Je verwijst naar een parameter title, die je zal meegeven bij het importeren.

Je wijzigt in index.jsp de regels tussen <head> en </head>:

```
<c:import url='/WEB-INF/JSP/head.jsp'>
  <c:param name='title' value='Pizza Luigi'>
</c:import>
```

❶

(1) Je vult de parameter title met de waarde Pizza Luigi.

Je wijzigt in pizzas.jsp de regels tussen <head> en </head>:

```
<c:import url='/WEB-INF/JSP/head.jsp'>
  <c:param name='title' value="Pizza's"/>
</c:import>
```

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

## 11.9 <c:set>

Je vult een variabele in met <c:set>. Je kunt daarna in de rest van de JSP de variabele gebruiken.

Syntax: <c:set var='naamVanDeVariabele' value='inhoudVanDeVariabele'>

- Je mag bij value een EL expressie vermelden.
- Als de variabele nog niet bestaat, maakt <c:set> de variabele aan.
- Als de variabele al bestaat, overschrijft <c:set> de inhoud van de variabele.
- Je leest daarna de inhoud van de variabele met de EL expressie \${naamVanDeVariabele}.

Voorbeeld:

```
<c:set var='geluk' value='${7*70}'>
```



Opmerking: elke oproep van een server-sided tag (<c:forEach>, ...)

veroorzaakt een lege regel in de HTML die je naar de browser stuurt.

Je kan die vermijden door de page directive eigenschap trimDirectiveWhitespaces op true te plaatsen:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'
  trimDirectiveWhitespaces='true'%>
```



Sauzen: zie takenbundel

## 12 RELATIEVE URL'S

Een relatieve URL is een URL die niet begint met /.

head.jsp bevat een relatieve URL: styles/default.css. De browser interpreteert die als volgt:

- Hij neemt huidige request URL.  
Dit is `http://localhost:8080/pizzaluigi/pizzas.htm`.
- Hij verwijdert de tekens na de laatste /.  
Hij behoudt `http://localhost:8080/pizzaluigi/`.
- Hij voegt de relatieve URL toe:  
`http://localhost:8080/pizzaluigi/styles/default.css`

De browser leest op die URL de stylesheet. Dit lukt momenteel.

### 12.1 Probleem

Relatieve URL's kunnen problemen veroorzaken.

Je wijzigt de URL van PizzasServlet naar /pizzas/all.htm om dit te zien

```
@WebServlet("/pizzas/all.htm")
```

Je voert de servlet uit op `http://localhost:8080/pizzaluigi/pizzas/all.htm`

Het element `<h1>` is niet meer gekleurd, omdat de browser `default.css` niet vindt:

- Hij neemt de huidige request URL:  
`http://localhost:8080/pizzaluigi/pizzas/all.htm`
- Hij verwijdert de tekens na de laatste /:  
`http://localhost:8080/pizzaluigi/pizzas/`
- Hij voegt de relatieve URL toe:  
`http://localhost:8080/pizzaluigi/pizzas/styles/default.css`

De browser vindt op die URL de CSS niet: `/pizzas` is overbodig in de URL

### 12.2 Oplossing

Je wijzigt head.jsp

- Je voegt na de eerste regel een regel toe:  
`<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>`
- Je wijzigt "images/favicon.ico" naar:  
`<c:url value='/images/favicon.ico' />`  
De `c:url` tag voegt voor /images het context path van de website toe (het deel /pizzaluigi in de URL).  
Zo wordt de URL een absolute URL : /pizzaluigi/images/favicon.ico.  
Opgelet, de `c:url` tag doet dit enkel als de waarde die je meegeeft bij value begint met /.
- Je wijzigt "styles/default.css" naar:  
`<c:url value='/styles/default.css' />`
- Je wijzigt in index.jsp "images/pizza.jpg" naar:  
`<c:url value='/images/pizza.jpg' />`

Je kan de website uitproberen.

Je wijzigt de URL van PizzasServlet terug naar /pizzas: `@WebServlet("/pizzas.htm")`

Je commit de sources en je publiceert op GitHub.

## 13 SERVLETS INSTANCE

### 13.1 Servlet instance aanmaak

- Sommige webserver maken tijdens de start van de website van elke servlet één instance.
- Andere webserver (zoals Tomcat) maken pas een servlet instance wanneer een eerste request voor die servlet binnenkomt.

Nadat de request verwerkt is, houden de webserver de instance bij in het interne geheugen. Ze sturen alle volgende requests, waarvan de URL hoort bij die servlet, naar die instance.

Onmiddellijk nadat een webserver een servlet instance maakt, roept hij op die instance één keer de method `init` op. De servlet erft die method van zijn base class `HttpServlet` en kan die overriden. Je ziet verder in de cursus dat je bepaalde initialisaties in deze method `init` uitschrijft.

### 13.2 Servlet instance einde

Een webserver kan een servlet instance verwijderen uit het interne geheugen (bijvoorbeeld omdat de vrije geheugenruimte klein is).

- De webserver mag geen servlet instance verwijderen die nog een request verwerkt.
- Voor de webserver de instance verwijdert, roept de webserver op die instance de method `destroy` op. Je kan in die method opkuiswerk doen, zoals het verwijderen van een tijdelijk bestand dat je in de servlet gebruikte. De servlet erft de method `destroy` van zijn base class `HttpServlet` en kan die overriden.

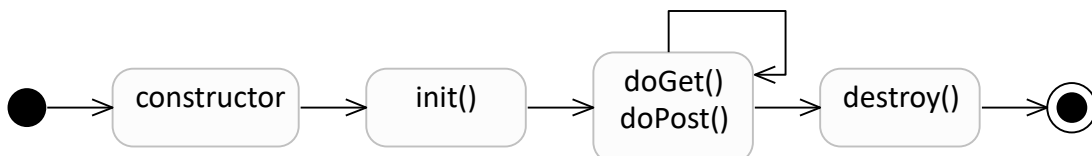
Opmerking: de webserver roept de method `destroy` ook op wanneer je:



- de website stopt
- de webserver stopt
- de website verwijdert

### 13.3 Servlet instance levenscyclus

1. De webserver maakt een servlet instance met de default constructor.
2. De webserver roept de method `init` op.
3. De servlet instance verwerkt GET requests met `doGet` en POST requests met `doPost`.
4. De webserver roept bij het stoppen van de website de method `destroy` op.



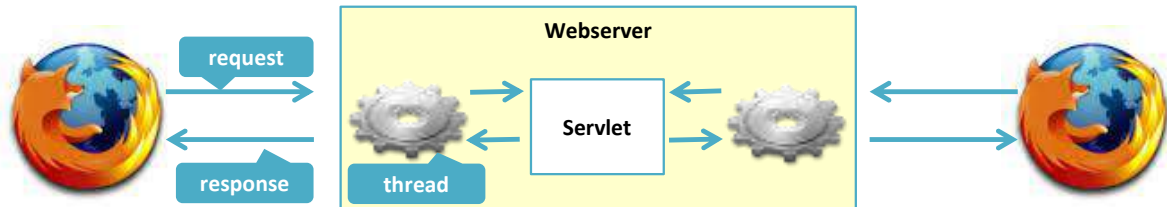
## 14 MULTITHREADING

De webserver verwerkt elke request met een aparte thread. Als de webserver gelijktijdige requests krijgt, gebruikt hij dus gelijktijdige threads. Van een servlet bestaat één instance.

Meerdere threads kunnen dus tegelijk de methods doGet en doPost van die instance uitvoeren.

Voorbeeld: Tomcat krijgt twee gelijktijdige requests binnen naar /menu/pizzas.htm.

Tomcat verwerkt die requests met twee gelijktijdige threads. Deze voeren gelijktijdig de doGet method van dezelfde PizzasServlet instance uit. Servlet code moet dus thread safe zijn.



- De request en response parameters van doGet en doPost zijn thread safe.
- Een variabele die je declareert in de method doGet of doPost is thread safe: gelijktijdige threads krijgen elk hun eigen versie van een lokale variabele.
- Een instance variabele (bijvoorbeeld gedeclareerd met **private**) is soms thread safe.
  - een **final** instance variabele met een primitief datatype is thread safe.  
Je kunt die variabele enkel initialiseren:
    - bij de declaratie (bijvoorbeeld **private final int getal = 666;**)
    - of in de constructor
 Java voert beide initialisaties altijd met één thread uit.  
Daarna kunnen andere threads die variabelen enkel lezen, niet wijzigen.
  - een niet-**final** instance variabele met een primitief datatype is niet thread safe.  
Je vervangt ze door één van volgende thread safe classes:
    - AtomicBoolean een thread safe boolean
    - AtomicInteger een thread safe int
    - AtomicLong een thread safe long
- Een instance variabele kan een reference zijn naar een object. Als de bijbehorende class thread safe is, is het uitvoeren van methods op die instance variabele ook thread safe.  
De class StringBuffer is bijvoorbeeld thread safe.  
Als de class niet thread safe is, is het uitvoeren van haar methods niet thread safe.  
JDBC objecten (Connection, Statement, ResultSet, ...) zijn niet thread safe.  
Je declareert ze daarom niet als private variabelen, maar als lokale variabelen.  
Collection objecten (ArrayList, HashSet, HashMap, ...) zijn niet thread safe.  
De package java.util.concurrent bevat alternatieve classes die wél thread safe (maar wat minder performant) zijn:
  - CopyOnWriteArrayList een thread safe List
  - CopyOnWriteArraySet een thread safe Set
  - ConcurrentHashMap een thread safe Map

## 14.1 AtomicInteger voorbeeld

Je maakt in `IndexServlet` een teller die bijhoudt hoeveel requests de servlet kreeg.

Je tikt in `IndexServlet`

```
private final AtomicInteger aantalKeerBekeken = new AtomicInteger(); ❶
```

- (1) De constructor van `AtomicInteger` initialiseert de getalwaarde in die `AtomicInteger` op 0.

Je tikt in het begin van de method `doGet`

```
request.setAttribute("aantalKeerBekeken",  
    aantalKeerBekeken.incrementAndGet()); ❶
```

- (1) De method `incrementAndGet` van de class `AtomicInteger` verhoogt de teller in de `AtomicInteger` op een thread-safe manier.

Je tikt in `index.jsp` voor `</body>`

```
<div>Deze pagina werd ${aantalKeerBekeken} keer bekeken.</div>
```

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

## 15 SERVLET INITIALISATIEPARAMETERS

Een servlet kan initialisatiewaarden bevatten die configuratiegegevens voorstellen (bijvoorbeeld het pad op de harde schijf waar de servlet een bestand met data moet lezen).

Er is kans dat je zo'n waarde moet wijzigen nadat de website af is.

Als je die waarde hard codeert in je Java source, is die wijziging een omslachtig stappenproces:

1. De websitebeheerder contacteert de programmeur.
2. De programmeur wijzigt de Java source.
3. De programmeur maakt een WAR bestand van de website.
4. De programmeur geeft dit WAR bestand aan de websitebeheerder.
5. De websitebeheerder deployt dit WAR bestand op de webserver.

Je plaatst zo'n waarden beter in `web.xml` (het configuratiebestand van je website).

Je leert hoe je die waarden kan lezen in de Java code van je servlet.

Als zo'n waarde achteraf wijzigt, is het stappenproces eenvoudiger:

1. De websitebeheerder wijzigt de waarde in `web.xml` in het WAR bestand.
2. De websitebeheerder deployt dit WAR bestand.

Je zal als voorbeeld het e-mailadres van de webmaster in `web.xml` plaatsen.

### 15.1.1 Eclipse

Je ziet `web.xml` met een dubbelklik op Deployment descriptor: `pizzaluigi`.

Je mag de regels `<display-name>` tot en met `</welcome-file-list>` verwijderen.

### 15.1.2 NetBeans

Je voegt `web.xml` toe:

1. Je klikt met de rechtermuisknop op het project.
2. Je kiest New, Other, Web, Standard Deployment Descriptor (`web.xml`).
3. Je kiest Next en je kiest Finish.

Je mag de regels `<session-config>` tot en met `</session-config>` verwijderen.

## 15.2 @WebServlet

Je moet in `@WebServlet` naast een URL ook een servlet naam kiezen. Dit is in `IndexServlet`

```
@WebServlet(urlPatterns = "/index.htm", name = "indexservlet")
```

## 15.3 Initialisatieparameters definiëren

Je definieert de servlet en zijn parameters in `web.xml`, tussen `<web-app ...>` en `</web-app>`:

```
<servlet>
  <servlet-name>indexservlet</servlet-name>
  <servlet-class>be.vdab.servlets.IndexServlet</servlet-class>
  <init-param>
    <param-name>emailAdresWebMaster</param-name>
    <param-value>joske.vermeulen@vdab.be</param-value>
  </init-param>
</servlet>
```

①  
②  
③  
④  
⑤

- (1) Je verwijst bij `servlet-name` naar de name bij `@WebServlet`.
- (2) Je tikt bij `servlet-class` de package én naam van de servlet class.
- (3) Je maakt per initialisatieparameter een element `init-param`.
- (4) Je tikt de parameter naam bij `param-name`.  
Iedere parameter moet een unieke parameter naam hebben.
- (5) Je tikt de parameter waarde bij `param-value`.



## 15.4 Initialisatieparameters lezen in je Servlet

Je leest met de servlet method `getInitParameter` de waarde van één initialisatieparameter.

- Je geeft de parameternaam mee aan de method.
- Je krijgt de parameterwaarde als een `String`, of `null` als de parameter niet bestaat.

Als je de method oproept in de servlet constructor krijg je een `NullPointerException`.

Je mag de method `getInitParameter` wel oproepen in de methods `init`, `doGet` en `doPost`.

Je voegt een opdracht toe als eerste in de method `doGet` van `IndexServlet`:

```
request.setAttribute("emailAdresWebMaster",  
    this.getInitParameter("emailAdresWebMaster"));
```

❶  
❷

- (1) Je geeft het emailadres door aan de JSP als het request attribuut `emailAdresWebMaster`
- (2) Je leest dit emailadres uit de initialisatieparameter `emailAdresWebMaster` in `web.xml`

Je voegt in `index.jsp` code toe voor `</body>`:

```
<div>WebMaster:  
<a href='mailto:${emailAdresWebMaster}'>${emailAdresWebMaster}</a></div>
```

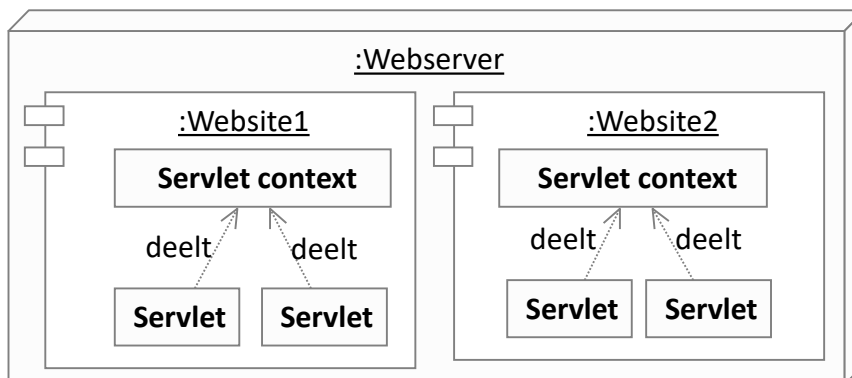
Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.



Initialisatieparameters: zie takenbundel

## 16 SERVLET CONTEXT

Elke website heeft één servlet context. Alle servlets van die website delen deze servlet context. De servlet context is dus interessant om data te onthouden die je in elke servlet nodig hebt.



De servlet context bevat twee soorten data:

- initialisatieparameters
- attributen

### 16.1 Initialisatieparameters

Je leerde vroeger servlet initialisatieparameters kennen. Je kan die gebruiken in *één* servlet. Als je initialisatieparameters hebt die je in *meerdere* servlets wil gebruiken, beschrijf je die als servlet context initialisatieparameters. Je doet dit voor het emailadres van de webmaster.

Je verwijdert in `web.xml` de regels van `<servlet>` tot en met `</servlet>`.

Je tikt tussen `<web-app ...>` en `</web-app>`:

```

<context-param>
  <param-name>emailAdresWebMaster</param-name>
  <param-value>joske.vermeulen@vdab.be</param-value>
</context-param>
  
```

①  
②  
③

- (1) Je maakt per initialisatieparameter een element `context-param`.
- (2) Je tikt per parameter een unieke parameternaam bij `param-name`.
- (3) Je tikt de parameterwaarde bij `param-value`.

#### 16.1.1 Servlet context aanspreken

Je spreekt vanuit een servlet de servlet context aan

- met de servlet method `getServletContext` die je oproept in de methods `init`, `doGet` of `doPost`.
- of met de request method `getServletContext`.

Beide methods geven je de servlet context onder de gedaante van de interface `ServletContext`.

#### 16.1.2 Initialisatieparameters lezen

Je leest met de `ServletContext` method `getInitParameter` een initialisatieparameter waarde:

- Je geeft de parameternaam mee.
- Je krijgt de parameterwaarde als een `String`, of `null` als de parameter niet bestaat.

Je vervangt in de method `doGet` van `IndexServlet`

```
this.getInitParameter("emailAdresWebMaster");
```

door

```
this.getServletContext().getInitParameter("emailAdresWebMaster");
```

Je kunt de website terug uitproberen.

### 16.1.3 Initialisatieparameters lezen in een JSP

Je leest in een JSP de waarde van een servlet context initialisatieparameter als  
`${initParam.naamVanDeInitialisatieParameter}`

Voorbeeld: `${initParam.emailAdresWebMaster}`

## 16.2 Attributen

Een servlet context attribuut is een veranderlijke waarde in de servlet context.

Een synoniem voor een servlet context attribuut is een application scope variable.

Gezien alle servlets de servlet context delen, delen ze ook dit servlet context attribuut.

Elke servlet kan een servlet context attribuut toevoegen, wijzigen en verwijderen.

Je moet de toegang tot servlet context attributen thread safe maken

gezien gelijktijdige threads servlet code kunnen uitvoeren en daarin een servlet context attribuut kunnen aanspreken.

Je beheert servlet context attributen met `ServletContext` methods:

- `setAttribute(attribuutNaam, attribuutWaarde)`  
 Je voegt een attribuut toe, als de servlet context het attribuut nog niet bevatte, of je plaatst een ander object of primitieve waarde (int, float, ...) in het attribuut.
- `getAttribute(naamVanHetTeLezenAttribuut)`  
 Je leest de attribuut inhoud onder de gedaante van `Object`.  
 Je krijgt **null** als het attribuut niet bestaat.
- `removeAttribute(naamVanHetTeVerwijderenAttribuut)`  
 Je verwijdert het attribuut waarvan je de naam meegeeft.

### 16.2.1 Voorbeeld

Je houdt in de servlet context het aantal browser requests per servlet bij.

Je voegt een constante toe aan `IndexServlet`:

```
private static final String INDEX_REQUESTS = "indexRequests";
```

Je voegt de method `init` toe:

```
@Override
public void init() throws ServletException {
    this.getServletContext().setAttribute(
        INDEX_REQUESTS, new AtomicInteger();
}
```

- (1) De webserver roept de method `init` 1 keer op, na het aanmaken van de servlet instance.
- (2) Je voegt aan de servlet context een attribuut toe met de naam `indexRequests`. De waarde van het attribuut is een `AtomicInteger`. De default constructor van `AtomicInteger` initialiseert de waarde in de `AtomicInteger` op 0.

Je voegt een opdracht toe aan de method `doGet`:

```
((AtomicInteger) this.getServletContext().getAttribute(INDEX_REQUESTS))
    .incrementAndGet();
```

- (1) Je leest de inhoud van het servlet context attribuut met de naam `indexRequests`. Je krijgt deze inhoud onder de gedaante van `Object`. Je cast deze inhoud naar zijn ware gedaante: `AtomicInteger`.
- (2) Je verhoogt de waarde in de `AtomicInteger` met 1. Dit werkt ook als meerdere threads dit tegelijk doen (bij gelijktijdige browser requests): `AtomicInteger` is thread safe.

Je voegt een constante toe aan `PizzasServlet`:

```
private static final String PIZZAS_REQUESTS = "pizzasRequests";
```

Je voegt de method `init` toe:

```
@Override
public void init() throws ServletException {
    this.getServletContext().setAttribute(PIZZAS_REQUESTS, new AtomicInteger());
}
```

Je voegt een opdracht toe aan de method `doGet`:

```
((AtomicInteger) this.getServletContext().getAttribute(PIZZAS_REQUESTS))
    .incrementAndGet();
```

Je maakt in `be.vdab.servlets` een servlet `StatistiekServlet`.

Die verwerkt GET requests naar de URL `/statistiek.htm`:

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/statistiek.htm")
public class StatistiekServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/statistiek.jsp";
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

Je maakt `statistiek.jsp` in `WEB-INF/JSP`:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
    <head>
        <c:import url='/WEB-INF/JSP/head.jsp'>
            <c:param name='title' value='Statistieken'/>
        </c:import>
    </head>
    <body>
        <c:import url='/WEB-INF/JSP/menu.jsp'/>
        <h1>Statistiek</h1>
        <dl>
            <dt>Welkom</dt>
            <dd>${indexRequests}</dd>
            <dt>Pizzas</dt>
            <dd>${pizzasRequests}</dd>
        </dl>
    </body>
</html>
```

(1) Je leest een servlet context attribuut met EL als `${naamVanHetServletContextAttribuut}`.




Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.



Servlet context initialisatieparameters: zie takenbundel

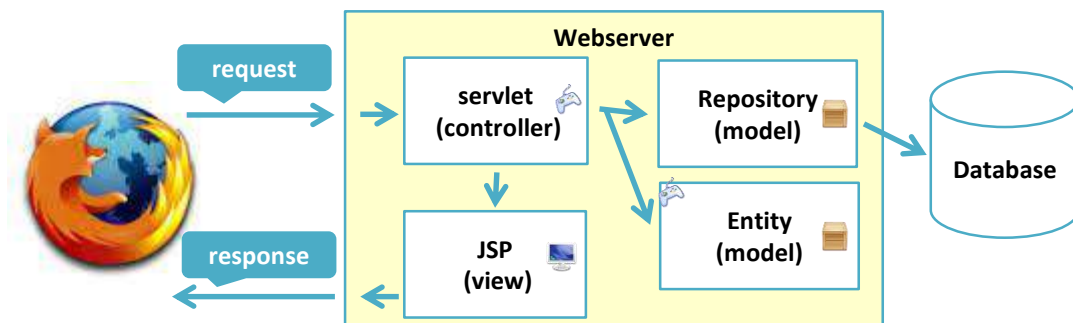
## 17 MODEL-VIEW-CONTROLLER

Bij MVC (Model-View-Controller) bevat je website drie soorten onderdelen:

	Onderdeel	Taak van dit onderdeel
	Model	De werkelijkheid voorstellen en databasebewerkingen doen. <ul style="list-style-type: none"> <li>• Entities stellen de werkelijkheid voor.</li> <li>• Repository classes doen databasebewerkingen.</li> </ul>
	View	Data mooi opgemaakt tonen aan de gebruiker. Bij ons is dit onderdeel een JSP.
	Controller	Requests binnenkrijgen. Bij ons is dit onderdeel een servlet.

Model, view en controller werken samen bij het verwerken van een request:

1. De controller  (ArtikelServlet) krijgt een browser request om artikels te zien.
2. Hij spreekt classes aan uit het model : hij roept de class ArtikelRepository op. Die leest artikels uit de database en geeft ze aan ArtikelServlet als een `List<Artikel>`. Artikel behoort ook tot het model .
3. De controller  geeft de request en de artikels door aan de view  (JSP). Die maakt HTML en stuurt die als response naar de browser.



### 17.1 Repository classes

Een synoniem voor repository is DAO: Data Access Object.

Je maakt per entity class één bijbehorende repository class.

- PizzaRepository methods doen databasebewerkingen met Pizza objecten.
- KlantRepository methods doen databasebewerkingen met Klant objecten.

PizzaRepository houdt voorlopig de pizza's bij in een static Map variabele in het interne geheugen, als vervanging van een relationele database.

Je maakt een package `be.vdab.repositories` en daarin een class `PizzaRepository`:

```

package be.vdab.repositories;
// enkele imports ...
public class PizzaRepository {
    private static final Map<Long, Pizza> PIZZAS = new ConcurrentHashMap<>(); ❶
    static {
        PIZZAS.put(12L, new Pizza(12, "Prosciutto", BigDecimal.valueOf(4), true)); ❷
        PIZZAS.put(14L, new Pizza(14, "Margehrita", BigDecimal.valueOf(5), false));
        PIZZAS.put(17L, new Pizza(17, "Calzone", BigDecimal.valueOf(4), false));
        PIZZAS.put(23L, new Pizza(23, "Fungi & Olive", BigDecimal.valueOf(5),
            false));
    }
    public List<Pizza> findAll() { ❸
        return new ArrayList<>(PIZZAS.values());
    }
}
  
```

```

public Optional<Pizza> read(long id) {
    Pizza pizza = PIZZAS.get(id);
    return optional.ofNullable(pizza);
}

public List<Pizza> findByPrijsBetween(BigDecimal van, BigDecimal tot) {
    return PIZZAS.values().stream()
        .filter(
            pizza -> pizza.getPrijs().compareTo(van) >= 0 &&
                pizza.getPrijs().compareTo(tot) <= 0
        )
        .collect(Collectors.toList());
}

public void create(Pizza pizza) { // pizza toevoegen
    pizza.setId(Collections.max(PIZZAS.keySet()) + 1);
    PIZZAS.put(pizza.getId(), pizza);
}
}

```

④

- (1) Je gebruikt ConcurrentHashMap als de implementatie van Map, want als meerdere browsers tegelijk requests sturen naar de website, spreken meerdere threads tegelijk de Map aan.
- (2) Dit is een static constructor: een method zonder naam, met het keyword static. Java voert een static constructor 1 keer uit, als je de 1<sup>o</sup> keer de class PizzaRepository aanspreekt. Je kunt in een static constructor enkel static members aanspreken.
- (3) Je leest straks met deze method vanuit PizzasServlet alle pizza's.
- (4) Je simuleert autonummering door bij de hoogste id van de bestaande pizza's één op te tellen en dit te gebruiken als de id van de nieuwe pizza.

Je maakt in PizzasServlet een private variabele PizzaRepository.

Je kan daarmee de diensten van PizzaRepository oproepen.

```
private final PizzaRepository pizzaRepository = new PizzaRepository();
```

①

- (1) Je maakt de variabele final. Dit is een 'good practice'.

Als je de variabele na initialisatie per ongeluk op null plaatst, krijg je een dan compilerfout.

Je roept de diensten van PizzaRepository op in de method doGet:

```

((AtomicInteger) this.getServletContext()
    .getAttribute(PIZZAS_REQUESTS)).incrementAndGet();
request.setAttribute("pizzas", pizzaRepository.findAll());
request.getRequestDispatcher(VIEW).forward(request, response);

```

Je iterateert in pizzas.jsp met de forEach over de pizza's:

```

<c:forEach var='pizza' items='${pizzas}'>
    <li>${pizza.id}:
        <c:out value='${pizza.naam}'/> ${pizza.prijs} &euro;
        ${pizza.pikant ? "pikant" : "niet pikant"}
        <c:url value='/pizzas/detail.htm' var='detailURL'>
            <c:param name='id' value='${pizza.id}'/>
        </c:url>
        <a href="<c:out value='${detailURL}'/>">Detail</a>
    </li>
</c:forEach>

```

Je commit de sources en je publiceert op GitHub. Je kan de website uitproberen.



Opmerking: ConcurrentHashMap houdt de pizza's in een willekeurige volgorde bij. Je lost dit probleem later op, wanneer je een echte database gebruikt.



MVC: zie takenbundel

## 18 GET REQUEST VERWERKEN

### 18.1 Wanneer is een request een GET request?

Een request is een GET request als de gebruiker:

- een URL tikt in de browser adresbalk
- een URL kiest in de browser favorieten
- een hyperlink aanklikt
- een form submit waarvan het method attribuut in de form tag `get` bevat  
`<form method='get' ...>`

### 18.2 Idempotent requests

Je gebruikt GET voor idempotent requests: requests die de toestand van de website niet wijzigen. Je kunt een idempotent request daarom zonder zorgen meerdere keren uitvoeren en telkens hetzelfde resultaat zien. Idempotent request zijn in praktijk requests waarmee je data leest.

Je gebruikt POST voor niet-idempotent requests.

- Een request waarmee je een product toevoegt aan de database. Het is onveilig die request meerdere keren te herhalen: je zou hetzelfde product meerdere keren toevoegen.
- Een request waarmee je een mail verstuurt. Het is onveilig die request meerdere keren te herhalen: je zou dezelfde mail meerdere keren versturen.

Browsers weten dat ze GET requests mogen herhalen en POST requests niet.

- Als je in een browser een 'refresh' vraagt van een GET request, voert de browser de request opnieuw uit en toont het resultaat.
- Als je een 'refresh' vraagt van een POST request, toont de browser een waarschuwing:  
 To display this page, Firefox must send information that will repeat action (such as a search or order confirmation) that was performed earlier

Als je de method POST misbruikt voor idempotent requests, zijn er nog nadelen:

- ➖ De gebruiker wil het resultaat van een POST request later nog eens zien. Hij voegt de URL toe aan zijn favorieten. Als hij de URL aanklikt in de favorieten, doet de browser een GET request en voert dus de `doGet` method uit in plaats van de `doPost` method.
- ➖ De gebruiker wil het resultaat van een POST request aan een andere persoon tonen. Hij stuurt die persoon een mail die de URL bevat. Als die persoon de URL volgt, doet zijn browser een GET request en voert dus de `doGet` method uit in plaats van de `doPost` method.
- ➖ Zoekrobots (Google, Bing, ...) voegen enkel GET requests toe aan hun database. Als de website enkel POST requests verwerkt, scoort hij dus slecht bij zoekrobots

Opmerkingen



- Je vraagt een paswoord in een HTML form met `<input type='password' .../>`. Je verhindert zo dat iemand het paswoord kan lezen terwijl je het tikt. Je submit de bijbehorende form met `method='post'`. Bij `method='get'` zie je bij het submitten van de form de waarden van alle invoervakken in de URL. Iemand zou het paswoord kunnen lezen in de URL!
- Als de gebruiker een request doet waarbij hij een bestand oplaadt, submit je de bijbehorende form met `method='post'`. Je kunt bestanden enkel opladen met een POST request, niet met een GET request.

### 18.3 StringUtils

Je moet in de rest van de cursus regelmatig nagaan of een String naar een Long kan geconverteerd worden. Je moet ook soms nagaan of een String naar een BigDecimal kan geconverteerd worden.

Je maakt daartoe een package `be.vdab.util` en daarin de class `StringUtils`:

```
package be.vdab.util;
import java.math.BigDecimal;
public class StringUtils {
    public static boolean isLong(String string) {
        try {
            Long.parseLong(string);
            return true;
        } catch (NumberFormatException ex) {
            return false;
        }
    }
    public static boolean isBigDecimal(String string) {
        try {
            new BigDecimal(string);
            return true;
        } catch (NullPointerException | NumberFormatException ex) {
            return false;
        }
    }
}
```

### 18.4 Request parameters in de query string

`pizzas.jsp` bevat per pizza een hyperlink [Detail](#). De bijbehorende URL bevat in de query string een parameter `id` (vb.: `/pizzaluigi/pizzas/detail.htm?id=12`).

Bij een klik op de hyperlink doet de browser een GET request naar de servlet die bij de URL hoort.

Je leest in de method `doGet` van de servlet zo'n parameter met de request method `getParameter`

- Je geeft de naam van de te lezen parameter mee.
- Je krijgt de parameterwaarde als een String terug, of `null` als de parameter niet voorkomt in de query string.

Je maakt in `be.vdab.servlets` een servlet `PizzaDetailServlet`.

Die verwerkt GET requests naar `/pizzas/detail.htm` (de URL van [Detail](#)):

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/pizzas/detail.htm")
public class PizzaDetailServlet extends HttpServlet {
    private static final long serialVersionUID= 1L;
    private static final String VIEW = "/WEB-INF/JSP/pizzadetail.jsp";
    private final PizzaRepository pizzaRepository = new PizzaRepository();
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String id = request.getParameter("id");
        if (StringUtils.isLong(id)) {
            pizzaRepository.read(Long.parseLong(id))
                .ifPresent(pizza -> request.setAttribute("pizza", pizza));
        } else {
            request.setAttribute("fout", "id niet correct");
        }
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

❶  
❷  
❸



- (1) Je leest de waarde van de parameter `id` in de query string.  
 Je krijgt die waarde als een `String` en converteert hem naar een `long`.  
 Je leest in de database de pizza met dit `id`.  
 Je geeft deze pizza door aan de JSP als een request attribuut met de naam `pizza`.  
 Als de pizza niet gevonden is in de database, zal dit request attribuut ontbreken.
- (2) Als de query string geen parameter `id` bevat, of een parameter `id` bevat die geen getal bevat, werpt de conversie naar `long` bij (1) een exception. Deze exception kan optreden als een hacker volgende URL tikt in de browser: `/pizzaluigi/pizzas/detail.htm?id=zorro`
- (3) Je geeft dan een request attribuut met de naam `fout` door aan de JSP.  
 Dit request attribuut bevat een foutmelding die je wil tonen aan de gebruiker.

Je maakt `pizzadetail.jsp` in `WEB-INF/JSP`:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
  <head>
    <c:import url='/WEB-INF/JSP/head.jsp'>
      <c:param name='title' value='${pizza.naam}'/>
    </c:import>
  </head>
  <body>
    <c:import url='/WEB-INF/JSP/menu.jsp'/>
    <c:choose>
      <c:when test='${not empty fout}'>
        <div class='fout'>${fout}</div>
      </c:when>
      <c:when test='${empty pizza}'>
        <div class='fout'>Pizza niet gevonden</div>
      </c:when>
      <c:otherwise>
        <h1>${pizza.naam}</h1>
        <dl><dt>Nummer</dt><dd>${pizza.id}</dd>
          <dt>Naam</dt><dd>${pizza.naam}</dd>
          <dt>Prijs</dt><dd>${pizza.prijs}</dd>
          <dt>Pikant</dt><dd>${pizza.pikant ? 'ja' : 'nee'}</dd>
        </dl>
      </c:otherwise>
    </c:choose>
  </body>
</html>
```

①  
②  
③

- (1) Als de JSP een request attribuut `fout` heeft doorgegeven
- (2) toon je de foutmelding in dit request attribuut.
- (3) Als de servlet de pizza niet vond in de database, bevat het request attribuut `pizza` de waarde `null`. De expressie `empty pizza` geeft dan `true` terug.

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

## 18.5 HTML form met method='get'

Als de gebruiker Pizza's tussen prijzen kiest in het menu, doet de browser een GET request naar `/pizzas/tussenprijzen.htm`. Je toont dan de pagina met lege invoervakken:

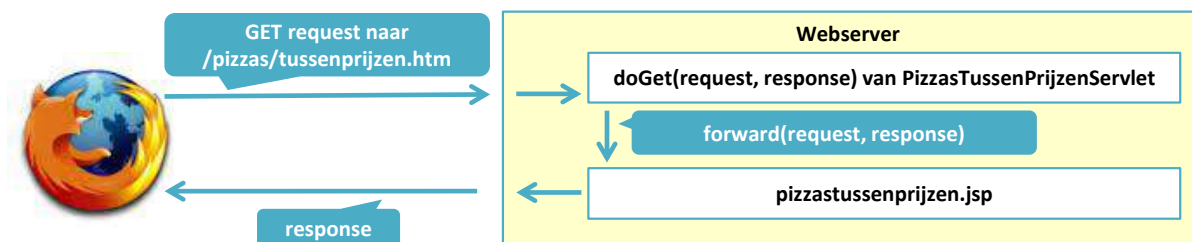
**Pizza's tussen prijzen**

Van prijs

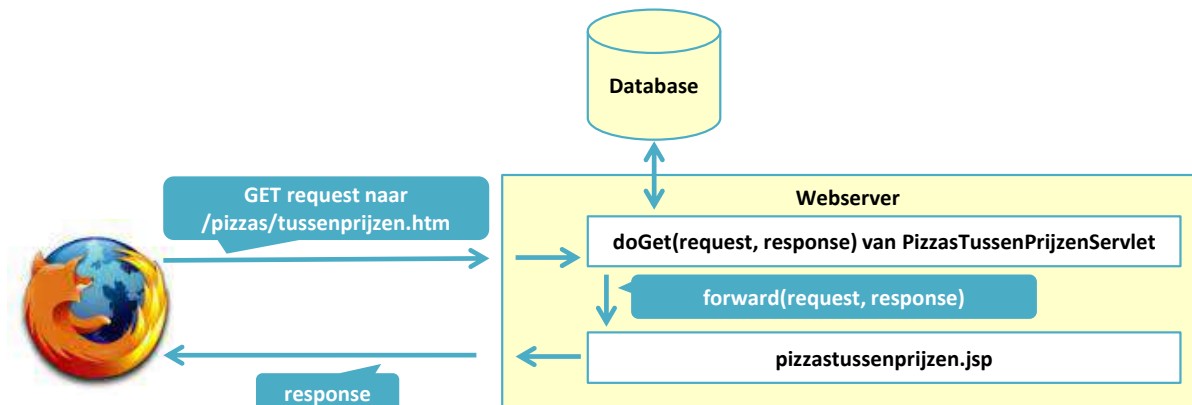
Tot prijs

Zoeken

Je moet bij het verwerken van deze request nog geen pizza's zoeken in de database: de gebruiker tikte nog niets in de invoervakken.



Nadat de gebruiker de invoervakken intikt en de knop Zoeken kiest doet de browser een GET request naar de URL `/pizzas/tussenprijzen.htm?van=1&tot=10`. De query string (in het vetjes) bevat dan de inhoud van de invoervakken. Je zoekt met de informatie in die query string de pizza's in de database en je toont de pagina met daarin ook deze pizza's.



Bemerk in de diagrammen dat je in beide gevallen dezelfde servlet en dezelfde JSP gebruikt omdat het in beide gevallen over dezelfde pagina gaat.

Je maakt in `be.vdab.servlets` een servlet `PizzasTussenPrijzenServlet`.

Die verwerkt GET requests naar de URL `/pizzas/tussenprijzen.htm`.

```

package be.vdab.servlets;
// enkele imports ...
@WebServlet("/pizzas/tussenprijzen.htm")
public class PizzasTussenPrijzenServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/pizzastussenprijzen.jsp";
    private final PizzaRepository pizzaRepository = new PizzaRepository();
  
```

```

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    if (request.getQueryString() != null) {
        Map<String, String> fouten = new HashMap<>();
        String van = request.getParameter("van");
        if ( ! StringUtils.isBigDecimal(van)) {
            fouten.put("van", "tik een getal");
        }
        String tot = request.getParameter("tot");
        if ( ! StringUtils.isBigDecimal(tot)) {
            fouten.put("tot", "tik een getal");
        }
        if (fouten.isEmpty()) {
            request.setAttribute("pizzas", pizzaRepository.findByPrijsBetween(
                new BigDecimal(van), new BigDecimal(tot)));
        } else {
            request.setAttribute("fouten", fouten);
        }
    }
    request.getRequestDispatcher(VIEW).forward(request, response);
}
}

```

- (1) De method `getQueryString` geeft je het onderdeel query string van de huidige request URL, of null als de query string leeg is. Bij null doet de gebruiker een request (vanuit het menu) om de pagina met de lege form te zien. Anders heeft de gebruiker de form gesubmit, moet je de invoervakken valideren en als die correct zijn de pizza's zoeken in de database.
- (2) Als de gebruiker de form submitte, controleer je of de invoervakken getallen bevatten. Als van geen getal bevat, zal je in de JSP een foutmelding tonen bij dit vak. Als tot geen getal bevat, zal je in de JSP een foutmelding tonen bij dit ander vak. Je bouwt een Map met deze foutmeldingen en geeft hem straks aan de JSP. De entry key is de naam van het foutieve invoervak. De value is een bijbehorende foutmelding.
- (3) Je voegt een entry toe aan de Map fouten. Je leest deze Map in de JSP.

Je maakt `pizzastussenprijzen.jsp` in WEB-INF/JSP:

```

<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
    <head>
        <c:import url='/WEB-INF/JSP/head.jsp'>
            <c:param name='title' value='Pizzas tussen prijzen'/>
        </c:import>
    </head>
    <body>
        <c:import url='/WEB-INF/JSP/menu.jsp'>
        <h1>Pizza's tussen prijzen</h1>
        <form method='get' action='<c:url value="/pizzas/tussenprijzen.htm"/>'>
            <label>Van prijs<span>${fouten.van}</span>
            <input name='van' autofocus></label>
            <label>Tot prijs<span>${fouten.tot}</span>
            <input name='tot'></label>
            <input type='submit' value='Zoeken'>
        </form>
        <c:if test='${not empty pizzas}'>
            <ul class='zebra'>
                <c:forEach var='pizza' items='${pizzas}'>
                    <li><c:out value='${pizza.naam}'> ${pizza.prijs}&euro;</li>
                </c:forEach>
            </ul>
        </c:if>
    </body>
</html>

```

```

    </ul>
  </c:if>
  <c:if test='${not empty param and empty fouten and empty pizzas}'> ❸
    <div class='fout'>Geen pizza's gevonden</div>
  </c:if>
</body>
</html>

```

- (1) Je tikt bij action een URL. De browser stuurt naar die URL een request bij de form submit.
- (2) Je leest in de Map<String,String> fouten de entry met de key van. Deze key bestaat als de gebruiker een fout tikte in het invoervak van. De value is dan de bijbehorende foutmelding.
- (3) Deze expressie zal true teruggeven als de gebruiker de form submitte (dan is de query string niet leeg en geeft not empty param true terug), de gebruiker de invoervakken correct tikte (empty fouten) en het request attribuut pizzas niet bestaat of een lege List bevat.

Je kunt de website uitproberen.

## 18.6 Client sided validatie

Je valideert de invoervakken nu server sided: met Java code, nadat de invoervakken gesubmit zijn. Deze validatie gebeurt door de webserver, nadat de browser de request verstuurd.

Je kan bij moderne browsers de invoervakken ook client sided (op de browser zelf) valideren.

Deze validatie gebeurt voor de browser de request verstuurt.

- ⊕ De validatie gebeurt daardoor zeer snel.
- ⊕ Als de validatie fouten geeft, verstuurt de browser zelfs geen request.

Je voegt client sided validatie toe aan de invoervakken in tussenprijzen.jsp

```

<input name='van' autofocus type='number' min='0' required>
<input name='tot' type='number' min='0' required>

```

Een bijkomend voordeel van <input ...type='number'> is dat de browser van een smartphone of tablet een handig virtueel toetsenbord toont om cijfers te tikken:



Je kunt de website uitproberen.

Naast client sided validatie blijft server sided validatie essentieel

- Oudere browsers doen geen client sided validatie: ze verwerken de validatie attributen min, required, ... niet.
- Hackers verwijderen de client sided validatie:
  - Ze klikken in de browser met de rechtermuisknop op het invoervak van en kiezen Inspect element.
  - Ze dubbelklikken type en drukken de Delete toets.
  - Ze doen hetzelfde met min en required en submitten de form.



## 18.7 Request parameter lezen in een JSP

Je leest in een JSP de inhoud van een request parameter met volgende syntax

```

${param.naamVanDeRequestParameter}

```

De variabele param is in een JSP altijd aanwezig. Zo'n variabele heet een 'implicit object'.

Als de gebruiker na het submitten de pagina terug ziet, zijn de invoervakken leeg, wat vervelend is.

Je vult nu de invoervakken met de waarde die de gebruiker intikte voor het submitten

```

<input name='van' value='${param.van}' type='number' min='0' autofocus required>
<input name='tot' value='${param.tot}' type='number' min='0' required>

```

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

## 18.8 Vereenvoudigde form tag

Je vereenvoudigt in pizzastussenprijzen.jsp de <form ...> tag naar <form>.

- Als je geen method meegeeft, is de default method get.
- Als je geen action meegeeft, is de action de huidige URL (/pizzas/tussenprijzen.htm).

## 18.9 Request method `getParameterValues`

Een request kan meerdere parameters hebben met dezelfde naam.

Dit is het geval als je een form submit die meerdere invoervakken met dezelfde naam bevat.

Je toont een pagina met de pizza's.

De gebruiker vinkt zijn voorkeurspizza's aan en klikt op de knop.

Je toont de pagina dan opnieuw

en je toont de geselecteerde pizza's onder de knop.

De vinkjes hebben allen hetzelfde *name* attribuut: *id*,

maar een verschillend *value* attribuut:

het pizza id waar ze bij staan.

Als de gebruiker de eerste én de laatste pizza aanvinkt en de form submit,

verstuurt hij volgende GET request: `/pizzaluigi/pizzas/voorkeuren?id=12&id=14`

Je leest in de servlet method `doGet` de parameterwaarden met de request method `getParameterValues`: `String[] ids = request.getParameterValues("id");`

Je krijgt een array met de parameter waarden. In het voorbeeld "12" en "14".

Je maakt in `be.vdab.servlets` een servlet `VoorkeurPizzasServlet`.

Die verwerkt GET requests naar de URL `/pizzas/voorkeuren.htm`.

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/pizzas/voorkeuren.htm")
public class VoorkeurPizzasServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/voorkeuropizzas.jsp";
    private final PizzaRepository pizzaRepository = new PizzaRepository();
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setAttribute("pizzas", pizzaRepository.findAll());
        if (request.getParameterValues("id") != null) {
            request.setAttribute("voorkeurPizzas",
                Arrays.stream(request.getParameterValues("id"))
                    .map(id -> pizzaRepository.read(Long.parseLong(id)))
                    .filter(optionalPizza -> optionalPizza.isPresent())
                    .map(optionalPizza -> optionalPizza.get())
                    .collect(Collectors.toList()));
        }
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

Je maakt `voorkeuropizzas.jsp` in `WEB-INF/JSP`:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
    <head>
        <c:import url='/WEB-INF/JSP/head.jsp'>
            <c:param name='title' value="Voorkeuropizza's"/>
        </c:import>
    </head>
```

```

<body>
  <c:import url='/WEB-INF/JSP/menu.jsp'/>
  <h1>Voorkeurpizza's</h1>
  <form>
    <ul class='zonderbolletjes'>
      <c:forEach var='pizza' items='${pizzas}'>
        <li>
          <label><input type='checkbox' name='id' value='${pizza.id}'>
            <c:out value='${pizza.naam}'/></label>
          </li>
        </c:forEach>
      </ul>
      <input type='submit' value='Toon mijn keuzes'>
    </form>
    <c:if test='${not empty voorkeurPizzas}'>
      <h1>Je voorkeurpizza's</h1>
      <ul class='zebra'>
        <c:forEach var='pizza' items='${voorkeurPizzas}'>
          <li><c:out value='${pizza.naam}'/></li>
        </c:forEach>
      </ul>
    </c:if>
  </body>
</html>

```

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

Opmerking: je gebruikt soms radio buttons in een HTML form:

☐ Man

☐ Vrouw



```

<label><input type='radio' name='geslacht' value='M'>Man</label>
<label><input type='radio' name='geslacht' value='V'>Vrouw</label>

```

In de servlet, die de gesubmitte form verwerkt,

lees je de keuze met de opdracht `request.getParameter("geslacht");`

Die geeft "M", "V" of null terug (als de gebruiker geen keuze maakte).

## 18.10 Request parameters – request attributen

Een request parameter krijg je binnen van de browser en kan je in je servlet en in je JSP lezen.

Een request attribuut maak je in je servlet en kan je in je JSP lezen.



Ingrediënten: zie takenbundel

## 19 POST REQUEST VERWERKEN

Een request is enkel een POST request als de gebruiker

- een form submit met als method attribuut post (`<form method='post' ...>`)

Je maakt een pagina met een form waarmee je een pizza toevoegt.

Je maakt in `be.vdab.servlets` een servlet `PizzaToevoegenServlet`. Die verwerkt:

- GET requests de lege HTML form tonen aan de gebruiker.
- POST requests het ingevulde HTML form verwerken na de submit.

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/pizzas/toevoegen.htm")
public class PizzaToevoegenServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/pizzatoevoegen.jsp";
    private static final String SUCCESS_VIEW = "/WEB-INF/JSP/pizzas.jsp";
    private final PizzaRepository pizzaRepository = new PizzaRepository();

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.getRequestDispatcher(VIEW).forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        Map<String, String> fouten = new HashMap<>();
        String naam = request.getParameter("naam");
        if (! Pizza.isNaamValid(naam)) {
            fouten.put("naam", "verplicht");
        }
        String prijsString = request.getParameter("prijs");
        BigDecimal prijs = null;
        if (StringUtils.isBigDecimal(prijsString)) {
            prijs = new BigDecimal(prijsString);
            if (! Pizza.isPrijsValid(prijs)) {
                fouten.put("prijs", "tik een positief getal");
            }
        } else {
            fouten.put("prijs", "tik een getal");
        }
        if (fouten.isEmpty()) {
            boolean pikant = "pikant".equals(request.getParameter("pikant"));
            pizzaRepository.create(new Pizza(naam, prijs, pikant));
            request.setAttribute("pizzas", pizzaRepository.findAll());
            request.getRequestDispatcher(SUCCESS_VIEW).forward(request, response);
        } else {
            request.setAttribute("fouten", fouten);
            request.getRequestDispatcher(VIEW).forward(request, response);
        }
    }
}
```

- (1) De gebruiker doet een GET request naar de servlet om de lege HTML form te zien.
- (2) De gebruiker tikt in die form de gegevens van de nieuwe pizza en submit de form. De browser doet op dat moment een POST request naar de servlet.
- (3) De nieuwe pizza is toegevoegd aan de database. Je leest alle pizza's uit de database (ook de nieuwe pizza) en toont die aan de gebruiker.



Je maakt pizzatoevoegen.jsp in WEB-INF/JSP:

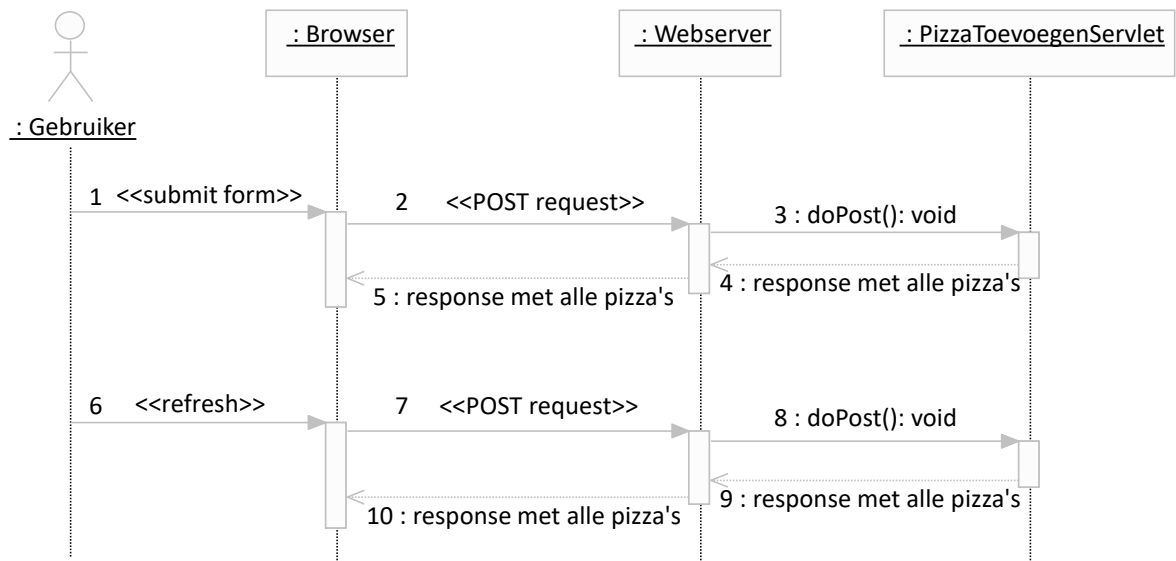
```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
  <head>
    <c:import url='/WEB-INF/JSP/head.jsp'>
      <c:param name='title' value='Pizza toevoegen'>
    </c:import>
  </head>
  <body>
    <c:import url='/WEB-INF/JSP/menu.jsp'>
    </c:import>
    <h1>Pizza toevoegen</h1>
    <form method='post' id='toevoegform'>
      <label>Naam<span>${fouten.naam}</span></label>
      <input name='naam' value='${param.naam}' autofocus required></label>
      <label>Prijs<span>${fouten.prijs}</span></label>
      <input name='prijs' value='${param.prijs}' type='number' min='0' required
        step='0.01'>
      </label>
      <div><label>
        <input type='checkbox' name='pikant' value='pikant'> Pikant</label></div>
      <input type='submit' value='Toevoegen' id='toevoegknop'>
    </form>
  </body>
</html>
```

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

## 19.1 Het refresh probleem en POST-REDIRECT-GET


Je ziet het refresh probleem met volgende stappen:

1. Je voegt een pizza toe in de website. Je ziet daarna een lijst met alle pizza's.  
De browser adresbalk bevat nog de URL van de POST request:  
http://localhost:8080/pizzaluigi/pizzas/toevoegen.htm.
2. Je doet onmiddellijk een pagina 'refresh' in de browser.
3. De browser toont een waarschuwing. Je vraagt de request toch uit te voeren.
4. De browser voert de laatste request opnieuw uit en voegt de pizza nog eens toe:  
de laatste request was de POST request waarmee je een pizza toevoegde.  
Dit had de gebruiker niet verwacht: hij zag een pagina met alle pizza's  
en dacht bij een 'refresh' alle pizza's opnieuw op te vragen.





Je kan de volgorde van requests ook zien met Firefox:

1. Je surft met Firefox naar `http://localhost:8080/pizzaluigi/pizzas/toevoegen.htm`.
2. Je kiest rechts boven in Firefox .
3. Je kiest Developer.
4. Je kiest Network.
5. Je vult de naam en de prijs van een pizza in en je kiest Toevoegen.
6. Je ziet onder in het venster dat je met die keuze een POST request verstuurd.

De `doPost` method die deze request verwerkt, stuurt HTML naar de browser (kolom Type). Deze HTML is aangemaakt door `pizzatoevoegen.jsp`.

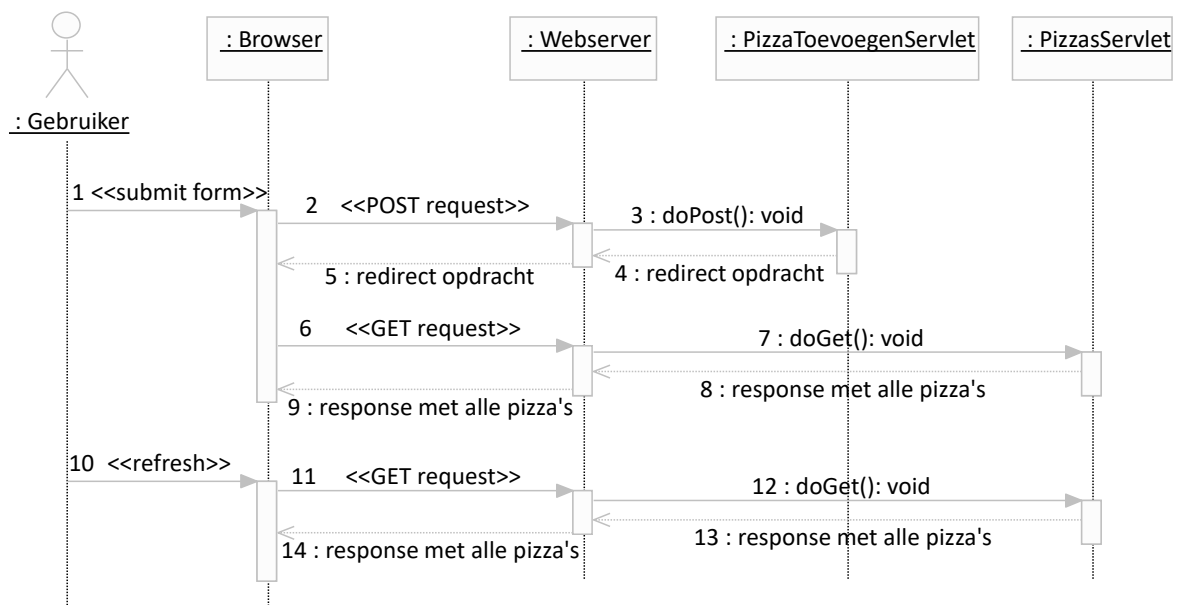
✓	Method	File	Domain	Type	Size
● 200	POST	toevoegen.htm	localhost:8080	html	1,19 KB

7. Je drukt F5 om de pagina te verversen en je kiest Resend bij de waarschuwing.
8. Je ziet onder in het venster dat je met die keuze dezelfde POST request terug verstuurd (en zo de pizza nog eens toevoegt).

✓	Method	File	Domain	Type	Size
● 200	POST	toevoegen.htm	localhost:8080	html	1,19 KB

Je lost dit probleem als volgt op:

1. Als je in `doPost` de request correct verwerkte, stuur je een redirect response naar de browser. Zo'n response bevat de status code 302 (Found), een lege body en een Location header met een URL. In ons voorbeeld is dit de URL waarop je alle pizza's ziet: `http://localhost:8080/pizzaluigi/pizzas.htm`.
2. Wanneer de browser een redirect response ontvangt, doet de browser onmiddellijk een GET request naar de URL vermeld in de response header Location. Bij ons is dit een GET request naar de servlet die alle pizza's toont. Als de gebruiker een 'refresh' doet, herhaalt de browser die idempotent GET request.



Je stuurt een redirect opdracht naar de browser met de response method `sendRedirect`.

Je geeft een String mee die een concatenatie is van

- het context path van de website, dat je ophaalt als `request.getContextPath()`
- de URL binnen de website, die je in de Location header invult (zoals `/pizzas.htm`)

Je vervangt in de class `PizzaToevoegenServlet` de constante `SUCCESS_VIEW` door:

```
private static final String REDIRECT_URL = "/pizzas.htm";
```

Je vervangt in de method `doPost` de laatste 2 opdrachten binnen `if (fouten.isEmpty())`:

```
response.sendRedirect(request.getContextPath() + REDIRECT_URL);
```

Je kan de volgorde van de requests ook zien met Firefox:

1. Je surft met Firefox naar `http://localhost:8080/pizzaluigi/pizzas/toevoegen.htm`.
  2. Je vult de naam en de prijs van een pizza in en je kiest Toevoegen.
  3. Je ziet onder in het venster dat je met die keuze een POST request verstuurd.
- De `doPost` method die deze request verwerkt, stuurt nu geen HTML naar de browser (kolom Type), maar een response met status code 302 en een response header Location. Je ziet de response headers als je de regel 302 POST `toevoegen.htm` aanklikt:

Response headers (0,156 KB)	
Content-Length:	"0"
Date:	"Thu, 15 Jan 2015 12:45:28 GMT"
Location:	"http://localhost:8080/pizzaluigi/pizzas.htm"
Server:	"Apache-Coyote/1.1"

Je ziet dat de browser, bij de ontvangst van de response, direct een GET request doet naar de URL in die response header Location

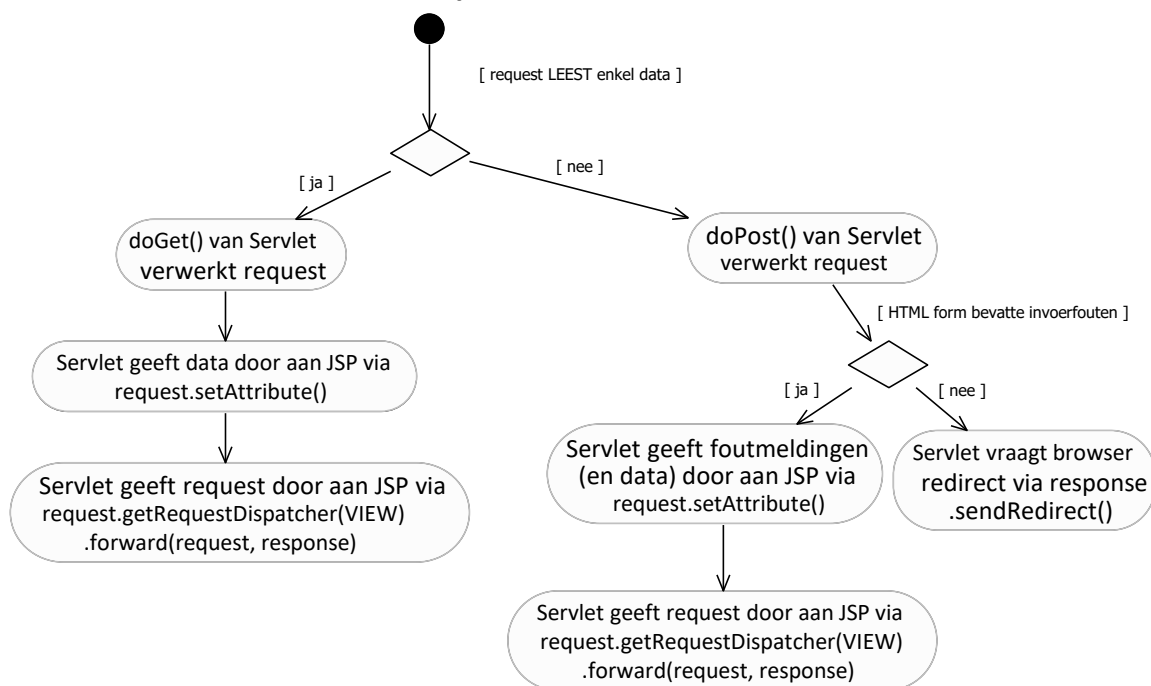
De `doGet` method van de bijbehorende servlet (`PizzasServlet`) stuurt in samenwerking met de bijbehorende JSP (`pizzas.jsp`) HTML naar de browser

✓	Method	File	Domain	Type	Size
▲ 302	POST	toevoegen.htm	localhost:8080	html	0 KB
● 200	GET	pizzas.htm	localhost:8080	html	3,70 KB

4. Je drukt F5 om de pagina te verversen.
  5. Je ziet onder in het venster dat je met die keuze de GET request terug verstuurd.
- Deze GET request toont terug alle pizza's (en voegt geen pizza toe).

● 200	GET	pizzas.htm	localhost:8080	html	3,70 KB
-------	-----	------------	----------------	------	---------

## 19.2 Overzicht GET en POST requests



## 19.3 Speciale tekens in de body van een POST request

Je geeft in een JSP aan dat je de tekens in die HTML voorstelt met de UTF-8 standaard, met het page directive onderdeel `pageEncoding="UTF-8"`.

Je kan met UTF-8 alle tekens uit alle menselijke talen voorstellen.

De gebruiker kan in invoervakken van een form ook vreemde tekens intikken.

Wanneer de gebruiker de form submit, stuurt de browser de inhoud van de invoervakken naar de webserver als request parameters met UTF-8.

Default interpreteert een servlet de tekens in de request parameters van een POST request met de oudere ISO 8859-1 standaard, die niet alle tekens uit alle landen ondersteunt.

Als je een pizza met de naam Hawaï maakt. Is het resultaat een pizza met de naam HawaÃ

Je lost dit op. Je geeft in het begin van de method doPost aan dat je de request parameters interpreteert met UTF-8: `request.setCharacterEncoding("UTF-8");`

Je commit de sources en je publiceert op GitHub. Je kan nu wel een pizza Hawaï toevoegen.

Ook de request parameters van een GET request kunnen 'speciale tekens' bevatten.

Tomcat interpreteert de tekens default met de verkeerde ISO 8859-1 standaard.

Je lost dit op in `server.xml` in de Tomcat directory conf

1. Je zoekt in dit bestand het element `<Connector port="8080" ...`
2. Je tikt bij dit element volgend attribuut: `URIEncoding="UTF-8"`.
3. Je herstart de Tomcat webserver.

## 19.4 Bestand uploaden

De gebruiker kan bij het toevoegen van een pizza ook een pizzafoto uploaden.

Je voegt daartoe in de JSP een attribuut `enctype="multipart/form-data"` aan de form tag:

```
<form method='post' id='toevoegform' enctype='multipart/form-data'>
```

Je tikt voor `<input type='submit' .../>`, een input tag met `type='file'`:

```
<label>Foto<span>${fouten.foto}</span>
<input type='file' name='foto'></label>
```

Je tikt voor de class `PizzaToevoegenServlet` `@MultipartConfig`.

Je leest in de method doPost het opgeladen bestand met de request method `getPart`.

- Je geeft de naam van de request parameter, met het opgeladen bestand, mee aan de method.
- Je krijgt een `Part` object terug, of `null` als de request parameter niet voorkomt.

De belangrijkste `Part` methods:

- `getSize` een long met het aantal bytes van het opgeladen bestand.
- `getContentType` een String met het MIME-type van het opgeladen bestand.
- `write` je bewaart hiermee het bestand op de harde schijf van de webserver.

Je maakt een folder `pizzafotos` op hetzelfde niveau als de folder `images`.

Je vervangt de laatste `if ... else` in de method doPost van `PizzaToevoegenServlet`:

```
Part fotoPart = request.getPart("foto");
boolean fotoIsOpgeladen = fotoPart != null && fotoPart.getSize() != 0;
if (fotoIsOpgeladen && ! fotoPart.getContentType().contains("jpeg")) {
    fouten.put("foto", "geen JPEG foto");
}
if (fouten.isEmpty()) {
    boolean pikant = "pikant".equals(request.getParameter("pikant"));
    Pizza pizza = new Pizza(naam, prijs, pikant);
    pizzaRepository.create(pizza);
    if (fotoIsOpgeladen) {
        String pizzaFotosPad =
            this.getServletContext().getRealPath("/pizzafotos");
        fotoPart.write(pizzaFotosPad + '/' + pizza.getId() + ".jpg");
    }
    response.sendRedirect(request.getContextPath() + REDIRECT_URL);
} else {
    request.setAttribute("fouten", fouten);
    request.getRequestDispatcher(VIEW).forward(request, response);
}
```

(1) Je vraagt de request parameter `foto` die de opgeladen foto bevat.

(2) Je controleert of de gebruiker een foto heeft opgeladen.

(3) Je geeft aan de servlet context method `getRealPath` een directory binnen de website mee.

Je krijgt een String terug met het absolute pad van de directory.

Als de website bijvoorbeeld draait op `c:\mijnsite` krijg je `c:\mijnsite\pizzafotos`

(4) Je bewaart de opgeladen foto weg in deze directory.

De bestandsnaam is het pizza id, gevolgd door .jpg

Je toont in de pagina met alle pizza's naast elke pizza, die een foto heeft, een hyperlink Foto. Als de gebruiker de hyperlink aanklikt, toon je de foto.

Je voegt een variabele toe aan PizzasServlet:

```
private String pizzaFotosPad;
```

Je voegt een opdracht toe aan de method init:

```
pizzaFotosPad = this.getServletContext().getRealPath("/pizzafotos");
```

Je wijzigt de method doGet. Deze biedt aan de JSP een request attribuut pizzaIdsMetFoto aan. Dit attribuut bevat de id's van pizzas waar een foto bij hoort.

```
((AtomicInteger) this.getServletContext().getAttribute(PIZZAS_REQUESTS))
    .incrementAndGet();
List<Pizza> pizzas = pizzaRepository.findAll();
request.setAttribute("pizzas", pizzas);
request.setAttribute("pizzaIdsMetFoto",
    pizzas.stream()
        .filter(pizza -> Files.exists(Paths.get(pizzaFotosPad,
            pizza.getId() + ".jpg")))
        .map(pizza -> pizza.getId())
        .collect(Collectors.toList()));
request.getRequestDispatcher(VIEW).forward(request, response);
```

Je tikt in pizzas.jsp onder `<a href='${detailURL}'>Detail</a>`

```
<c:if test='${pizzaIdsMetFoto.contains(pizza.id)}'>
    <c:url value='/pizzafotos/${pizza.id}.jpg' var='fotoURL' />
    <a href='${fotoURL}'>Foto</a>
</c:if>
```

Je kunt de website terug uitproberen.

Opmerking: telkens je de website deployt, overschrijf je de directory pizzafotos met de inhoud van die directory op de ontwikkelmachine.



Ook als je de website uitprobeert vanuit Eclipse (tijdens het ontwikkelen) gebeurt dit: de directory waar je website draait verschilt van de directory waar je de website maakt.

Je kunt dus beter vóór het deployen een backup nemen van deze directory en na het deployen deze backup terugplaatsen.

## 19.5 Dubbele submit vermijden

Als de gebruiker de submit knop twee keer snel na mekaar aanklikt, submit hij dezelfde HTML form twee keer. Hij zou dezelfde pizza twee keer toevoegen.

Je vermijdt dit door de submit knop te disablen bij de eerste submit.

Je tikt JavaScript code voor `</body>`:

```
<script>
    document.getElementById('toevoegform').onsubmit = function() {
        document.getElementById('toevoegknop').disabled = true;
    };
</script>
```

Je commit de sources en je publiceert op GitHub




Sauzen verwijderen: zie takenbundel

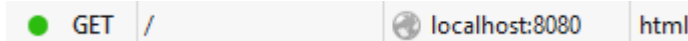
## 20 REQUEST HEADERS

De browser stuurt in elke request enkele headers mee.

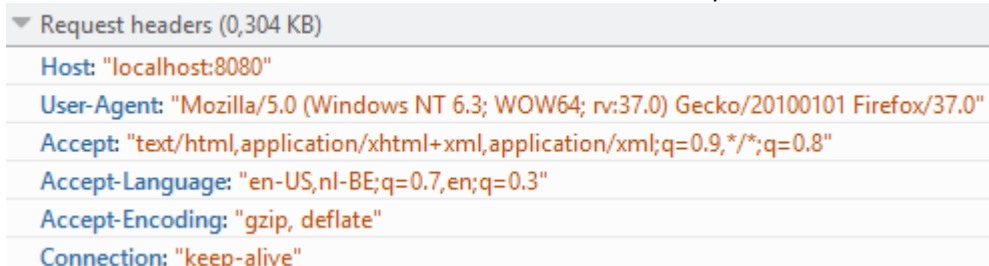
Deze headers bevatten informatie over de browser.

Je kan de volgorde van requests ook zien met Firefox:

1. Je kiest rechts boven in Firefox het icoon .
2. Je kiest Developer.
3. Je kiest Network.
4. Je surft naar `http://localhost:8080/pizzaluigi`.
5. Je klikt op de regel die deze request voorstelt:



6. Je ziet in een venster de headers die de browser in deze request meestuurde:



De header User-Agent geeft bijvoorbeeld het browser type aan.

### 20.1 Één header lezen

Je leest één request header op volgende manier:

- Je roept de request method `getHeader` op en je geeft de header naam mee.  
Voorbeeld: `request.getHeader("user-agent")`
- De method geeft een `String` terug met de request header inhoud.
- De method geeft `null` terug als de request die header niet bevat.

Voorbeeld: je vraagt de request header `user-agent`, die het browser type bevat

- Firefox  
`Mozilla/5.0 (Windows NT 6.1; WOW64; rv:11.0) Gecko/20100101 Firefox/11.0`
- Oude Internet Explorer  
`Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)`
- Recente Internet Explorer  
`Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko`
- ...

Je converteert de header naar kleine letters.

- Als de header het woord **firefox** bevat, is de browser Firefox.
- Als de header het woord **msie** of het woord **Trident** bevat, is de browser Internet Explorer.
- Als de header het woord **chrome** bevat, is de browser Chrome.
- ...

Je maakt in `be.vdab.servlets` een servlet `HeadersServlet`

Die verwerkt GET requests naar de URL `/headers.htm`:

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/headers.htm")
public class HeadersServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/headers.jsp";
    private final Map<String, String> browsers = new HashMap<>();
    public HeadersServlet() {
        browsers.put("firefox", "Firefox");
        browsers.put("chrome", "Chrome");
        browsers.put("msie", "Internet Explorer");
        browsers.put("trident", "Internet Explorer");
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String userAgent = request.getHeader("user-agent").toLowerCase();
        browsers.entrySet().stream()
            .filter(entry -> userAgent.contains(entry.getKey()))
            .findFirst()
            .ifPresent(browser -> request.setAttribute("browser", browser));
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

- (1) De entry value is een browsersoort.  
De key is het woord dat enkel bij die browsersoort voorkomt in de header user-agent.
- (2) Je vraagt de request header user-agent en je converteert hem naar kleine letters.
- (3) Je overloopt de gekende browsers.
- (4) Als de header user-agent het woord bevat dat enkel bij de huidige browser past, heb je de browser gevonden.

Je maakt `headers.jsp` in `WEB-INF/JSP`:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core' %>
<!doctype html>
<html lang='nl'>
    <head>
        <c:import url='/WEB-INF/JSP/head.jsp'>
            <c:param name='title' value="Pizza's"/>
        </c:import>
    </head>
    <body>
        <c:import url='/WEB-INF/JSP/menu.jsp'>
            Je browser: ${empty browser ? "onbekend" : browser}
        </c:import>
    </body>
</html>
```

Je kunt dit uitproberen.

## 20.2 Alle headers lezen

Je leest de inhoud van álle headers in twee stappen:

1. Je voert in `doGet` of `doPost` op de request parameter de method `getHeaderNames` uit.  
Je krijgt de namen van alle request headers als een `Enumeration<String>`.
2. Je itereert hierover en je leest de header waarde met de request method `getHeader`.

```
@WebServlet("/headers.htm")
public class HeadersServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/headers.jsp";
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        Map<String, String> headers = new LinkedHashMap<>();
        for (Enumeration<String> headerNames = request.getHeaderNames();
            headerNames.hasMoreElements(); ) {
            String headerName = headerNames.nextElement();
            headers.put(headerName, request.getHeader(headerName));
        }
        request.setAttribute("headers", headers);
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

Je vervangt Je browser: `${browser}` in de body van `headers.jsp` door:

```
<dl>
  <c:forEach var='h' items='${headers}'>
    <dt>${h.key}</dt><dd>${h.value}</dd>
  </c:forEach>
</dl>
```

Je commit de sources en je publiceert op GitHub. Je kunt dit uitproberen.

## 21 COOKIES

Een cookie is data die de browser bijhoudt ten dienste van een website.

- Een browser kan maximaal 20 cookies bijhouden per website.
- Één cookie is maximaal 4KB groot.
- Voorbeelden van cookies: de gebruikersnaam, voorkeur achtergrondkleur.
- Elke cookie heeft een naam en een String waarde. Die kan geen 'vreemde' tekens bevatten (é, ñ, ...). Als je toch een tekst met 'vreemde' tekens bewaart in een cookie, converteer je die tekst naar zijn URL encoded vorm met de class URLEncoder. Er is ook een class URLDecoder. Je converteert daarmee een tekst in URL encoded terug naar zijn oorspronkelijke vorm.
- Er bestaan twee soorten cookies:
  - Tijdelijke cookies.  
De browser onthoudt tijdelijke cookies in het interne geheugen, tot je de browser sluit.
  - Permanente cookies.  
De browser onthoudt permanente cookies op de harde schijf.  
Als je de browser sluit, blijven permanente cookies dus bestaan.  
Een permanente cookie heeft een vervaltijdstip, die de website programmeur bepaalt.  
De browser verwijdert de permanente cookie na dit vervaltijdstip.
- De browser stuurt bij elke request alle website cookies mee in de request header Cookie.
- Als een website een cookie wil toevoegen, wijzigen of verwijderen, geeft hij dit aan in de response header Set-Cookie.
- Bewaar in cookies geen confidentiële data (paswoorden, betaalkaartnummers, ...).  
Je kunt in de meeste browsers gemakkelijk de inhoud van cookies zien.
- De gebruiker kan in de browser cookies uitschakelen.  
Als de website cookies gebruikt, zal hij bij die gebruiker niet goed functioneren.

### 21.1 Cookie

De class Cookie stelt een cookie voor:

Cookie
-name: String -value: String -maxAge: int
+Cookie(name: String, value: String)

Alle attributen (name, value, maxAge) hebben getters en setters.  
Een Cookie met een maxAge -1 is een tijdelijke cookie.  
Een cookie met een positieve waarde in maxAge is een permanente cookie.  
maxAge geeft dan aan na hoeveel seconden de cookie verdwijnt.

### 21.2 Cookie toevoegen

Je voegt een cookie op volgende manier toe:

1. Je maakt een Cookie object met de Cookie constructor.  
Je geeft de cookienaam en -waarde mee als parameters.
2. Je roept op het Cookie object de method setMaxAge op bij een permanente cookie.  
Je geeft het aantal seconden mee dat de browser de cookie moet bijhouden.
3. De methods doGet en doPost hebben een response parameter.  
Je voert daarop de method addCookie op en je geeft het Cookie object mee.

### 21.3 Cookies lezen

Je kunt niet één bepaalde cookie lezen, maar wel álle cookies, op volgende manier:

1. De methods doGet en doPost hebben een request parameter.  
Je voert op die request parameter de method getCookies uit.  
De method geeft je alle cookies als een array van Cookie objecten.

### 21.4 Cookie wijzigen

Je wijzigt een cookie met volgende stappen:

1. Je maakt een Cookie object met de naam en de waarde van de te wijzigen cookie.
2. Je roept de response method addCookie op en je geeft dit Cookie object mee.



## 21.5 Permanente cookie verwijderen

Je verwijdert op volgende manier een permanente cookie:

1. Je maakt een Cookie object met de naam van de te verwijderen cookie.
2. Je roept op dit Cookie object de method `setMaxAge` op met een waarde -1.
3. Je roept de response method `addCookie` op en je geeft het Cookie mee.

## 21.6 Voorbeeld

De gebruiker tikt een gebruikersnaam. Je onthoudt die in een permanente cookie.

Als de gebruiker de pagina terug opvraagt, toon je de inhoud van die cookie in het tekstvak.

Je maakt in `be.vdab.servlets` een servlet `IdentificatieServlet`

Die verwerkt GET requests en POST requests naar de URL `/identificatie.htm`

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/identificatie.htm")
public class IdentificatieServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/identificatie.jsp";
    private static final int COOKIE_MAXIMUM_LEEFTIJD =
        60 /* seconden */ * 30 /* minuten */;
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        Cookie cookie = new Cookie("gebruikersnaam",
            URLEncoder.encode(request.getParameter("gebruikersnaam"), "UTF-8")); ❶
        cookie.setMaxAge(COOKIE_MAXIMUM_LEEFTIJD);
        response.addCookie(cookie);
        response.sendRedirect(request.getRequestURI()); ❷
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        if (request.getCookies() != null) {
            for (Cookie cookie : request.getCookies()) {
                if ("gebruikersnaam".equals(cookie.getName())) {
                    request.setAttribute("gebruikersnaam", ❸
                        URLDecoder.decode(cookie.getValue(), "UTF-8")); ❹
                    break;
                }
            }
        }
        request.getRequestDispatcher(VIEW).forward(request, response);
    }
}
```

- (1) De request parameter naam bevat tekst in Unicode UTF-8 formaat.  
Je converteert de tekst naar URL encoded formaat met de static `URLEncoder` method `encode`.  
De 1° parameter bevat de te converteren tekst.  
De 2° parameter bevat het formaat van de tekst.
- (2) Je doet een redirect naar de URL van de request. Dit is hier de URL van de huidige servlet.
- (3) Je geeft de gebruikersnaam door aan de JSP.
- (4) De cookie naam bevat tekst in URL encoded formaat.  
Je converteert die tekst naar Unicode UTF-8 met de static `URLDecoder` method `decode`.  
De 1° parameter bevat de tekst.  
De 2° parameter bevat het formaat van de geconverteerde tekst.

Je maakt identificatie.jsp in WEB-INF/JSP:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core' %>
<!doctype html>
<html lang='nl'>
  <head>
    <c:import url='/WEB-INF/JSP/head.jsp'>
      <c:param name='title' value='Identificatie' />
    </c:import>
  </head>
  <body>
    <c:import url='/WEB-INF/JSP/menu.jsp' />
    <h1>Identificatie</h1>
    <form method='post'>
      <label>Naam<input name='gebruikersnaam' value='${gebruikersnaam}'
        autofocus></label>
      <input type='submit' value='Onthoud me'>
    </form>
  </body>
</html>
```

Je commit de sources en je publiceert op GitHub.

Je kunt dit uitproberen. Nadat je je naam intikt blijft die onthouden, ook als je de browser herstart.

## 21.7 Cookies lezen in een JSP

In een JSP is er altijd een EL variabele cookie aanwezig.

Zo'n variabele die aanwezig is, zonder dat jij ze hoeft te maken, heet een 'implicit object'.

- De variabele cookie is een map waarin elke entry één cookie voorstelt.
- De entry key is de cookie naam.
- De entry value is de cookie zelf.

Je leest dus de waarde van de cookie naam met volgende EL expressie

```
${cookie.gebruikersnaam.value}
```

Die EL expressie converteert de cookie waarde echter niet vanuit UTF-8 !

## 21.8 Cookies inzien die je browser bijhoudt

1. Je kiest rechts boven in Firefox ☰.
2. Je kiest Options.
3. Je kiest Privacy.
4. Je kiest remove individual cookies.
5. Je dubbelklikt localhost in de lijst.
6. Je ziet op de regel daar onder een cookie met de name naam.
7. Je ziet in de onderste helft detail informatie over die cookie.



Meisjes jongens: zie takenbundel

## 22 SESSIONS

### 22.1 Stateless

HTTP is een stateless protocol: elke request is een zelfstandige handeling. Zijn verwerking hangt niet af van de verwerking van vorige requests. Variabelen die je aanmaakt gedurende het verwerken van een request, verdwijnen uit het interne geheugen van de webserver nadat de request verwerkt is. Ze zijn niet meer beschikbaar bij een volgende request.

Een stateless website heeft volgende voordelen:

- ⊕ Bij veel gelijktijdige gebruikers heeft de webserver geen interne geheugen tekorten.
- ⊕ Je kan de webserver herstarten, zonder dat de gebruikers daar last van hebben: je onthoudt niets in het RAM geheugen over de gebruikers.

### 22.2 Session

Sommige data moet je toch onthouden over requests heen, zoals gebruikersnaam of winkelmandje



Één servlet instance verwerkt de requests van *alle* gebruikers.

Je onthoudt een winkelmandje niet in een *private* variabele van die servlet instance.

Alle gebruikers zouden hetzelfde winkelmandje delen!

Je gebruikt voor zulke data een session: een stuk webserver geheugen waarin je data voor één gebruiker onthoudt. Het is aan te raden de hoeveelheid data in session te minimaliseren:

- ⊕ Bij veel gelijktijdige gebruikers bevat het interne geheugen van de webserver veel sessions. Als elk van die sessions weinig data bevat, kom je geen geheugen te kort.
- ⊕ Als de website draait op meerdere webserver in een webfarm, moeten die webserver de sessions niet enkel onthouden in hun eigen interne geheugen. Ze moeten de sessions ook uitwisselen met de andere webserver wanneer de data in de sessions wijzigt. Dit uitwisselen gebeurt over het interne netwerk tussen de webserver. Dit uitwisselen is essentieel zodat alle webserver dezelfde data per gebruiker bevatten. Als één van de webserver uitvalt, kennen de overgebleven webserver nog de data per gebruiker. Als de sessions weinig data bevat, gebeurt dit uitwisselen performant. Dit uitwisselen heet *session replication*.

Voorbeelden van weinig data in session:

- Als de gebruiker per artikel maar één stuk kan bestellen, volstaat in het winkelmandje artikelnummers bij te houden, niet volledige artikel objecten.
- Als de gebruiker per artikel meerdere stuks kan bestellen, volstaat het in het winkelmandje artikelnummers en -aantallen bij te houden.

Één session kan meerdere stukken data bevatten. Elk stukje data heet een session attribuut. Een session kan bijvoorbeeld twee attributen bevatten: gebruikersnaam en winkelmandje.



## 22.3 Serializable

Als de waarde in een session attriboot een object is (geen primitief data type), moet de bijbehorende class `Serializable` implementeren. De redenen hiervoor zijn:

- Session persistence
- Session replication

### 22.3.1 Session persistence

De webserver onthoud sessions in zijn RAM geheugen. Om deze sessions niet te verliezen bij een herstart van de webserver, doet de webserver volgende stappen bij een herstart:

1. De webserver schrijft de sessions via serialization naar een tijdelijk bestand. Dit serialization proces werkt enkel als de objecten `Serializable` zijn.
2. De webserver herstart.
3. De webserver leest de sessions terug uit het tijdelijk bestand.

### 22.3.2 Session replication

In een web cluster(webfam) stuurt een webserver elke session wijziging via het netwerk naar de andere webserver uit de cluster. Dit heet session replication.

Alle webserver moeten dezelfde sessions bevatten: webserver A verwerkt een request, maar een andere webserver B verwerkt een volgende request van dezelfde gebruiker:

- Mieke legt in haar mandje appels.
- Webserver A verwerkt die request en onthoudt het mandje als een session attriboot.
- Webserver A stuurt die session (inclusief de session ID) naar webserver B.
- Webserver B onthoudt die session in zijn geheugen.
- Mieke vraagt haar mandje te zien.
- Webserver B verwerkt die request: hij haalt het mandje uit de session van Mieke.

Je kan objecten enkel over het netwerk versturen als ze `Serializable` zijn.

## 22.4 Session identificatie

Als je bij het verwerken van een request een nieuwe session maakt, geeft de webserver die session een unieke identificatie: de session ID.

Als dezelfde browser later nog een request doet, is het essentieel dat de webserver de session van die browser opzoekt, en niet de session van een andere browser.

Er bestaan hiervoor twee strategieën: een tijdelijke cookie of URL rewriting.

### 22.4.1 Session identificatie met een tijdelijke cookie

Als tijdelijke cookies ingeschakeld zijn in de browser, doet de webserver volgende stappen:

1. na het aanmaken van een session, stuurt de webserver een tijdelijke cookie naar de browser met de naam `JSESSIONID` en als inhoud de session ID van de session.
2. De browser stuurt bij volgende request de cookies door, ook de cookie `JSESSIONID`. De webserver kan daarmee de session opzoeken die bij die browser hoort.

Voorbeeld:

1. Een request komt binnen om een winkelmandje te maken.
2. Je maakt daarbij een session, die de session ID `0AAB9C8DE666` krijgt.
3. De webserver stuurt een tijdelijke cookie `JSESSIONID=0AAB9C8DE666` naar de browser.
4. De browser stuurt bij alle volgende requests die cookie naar de webserver.
5. De webserver vindt aan de hand van het session ID in die cookie tussen de sessions van alle gebruikers de session van die ene browser.

### 22.4.2 Session identificatie met URL rewriting

Als tijdelijke cookies uitgeschakeld zijn op de browser doet de webserver volgende stappen:

1. na het aanmaken van een session, voegt de webserver aan alle URL's die je naar die browser stuurt de session ID toe. Hij voegt aan de URL's het volgende toe: `;jsessionid=` en de session ID. Dit heet URL rewriting.
2. De browser volgt de URL's bij volgende requests.  
De webserver zoekt de session die bij de browser hoort via `jsessionid` in de request URL.

Voorbeeld:

1. Een request naar `/maakwinkelmandje` komt binnen om een winkelmandje te maken.
2. Je maakt daarbij een session, met de session ID `0AAB9C8DE666`.
3. Je doet op het einde van de request een redirect naar `/toonwinkelmandje`.
4. De webserver voegt aan de URL, en alle volgende URL's die hij naar de browser stuurt, de session ID toe. De gewijzigde URL is `/toonwinkelmandje;jsessionid=0AAB9C8DE666`.
5. De browser doet dus vanaf dan requests naar URL's die `jsessionid` bevatten.
6. De webserver vindt met `sessionid` in de URL de session van die browser.

Direct na het aanmaken van een session weet de webserver niet of in de browser tijdelijke cookies ingeschakeld zijn. Hij gebruikt daarom in de eerste response beide strategieën: een tijdelijke cookie én URL rewriting.

Als de volgende request een cookie `JSESSIONID` bevat, zijn cookies ingeschakeld, en past de webserver geen URL rewriting meer toe.

Jij moet de webserver helpen om URL rewriting toe te passen

- Je maakt een URL in een JSP met `<c:url ...>`.  
`<c:url ...>` voegt de session ID toe bij URL rewriting.
- Als je `sendRedirect` gebruikt in de servlet method `doPost`, doe je op de URL, waarnaar de browser een redirect doet, een extra handeling.  
Je voert de method `response.encodeRedirectURL` uit, waarbij je de URL meegeeft.  
De method voegt de session ID toe aan de URL bij URL rewriting.  
Je past dit toe in de method `doPost` van `PizzaToevoegenServlet`  
`response.sendRedirect(response.encodeRedirectURL(request.getContextPath() + REDIRECT_URL));`



Opmerking: de webserver stuurt enkel de session ID naar de browser, nooit de session attributen. Ze mogen dus confidentiële informatie bevatten.

### 22.5 Request method getSession

Je kan in de servlet methods `doGet` en `doPost` de request method `getSession` oproepen:

- Als de gebruiker nog geen session heeft maakt de method een session aan en geeft die terug als een `HttpSession` object.
- Als de gebruiker wel al een session heeft, (aangemaakt bij een vorige request), geeft de method die session.

Er bestaat ook een versie van `getSession` waarbij je `false` meegeeft.

Je controleert met zo'n oproep of de gebruiker al een session heeft:

- Als de gebruiker nog geen session heeft, krijg je `null` terug.
- Als de gebruiker al een session heeft, krijg je die session.

Je mag de method `getSession` niet meer oproepen nadat je:

- een forward gedaan hebt naar een JSP.
- of een `sendRedirect` gedaan hebt naar een URL.

Als je dit wel doet, krijg je een `IllegalStateException`.

## 22.6 Session attribuut toevoegen

Je voegt een attribuut toe aan een session met de session method `setAttribute`.  
Je geeft een attribuutnaam en een attribuutwaarde mee.

## 22.7 Session attribuut lezen

Je leest een session attribuut met de session method `getAttribute`.

- Je geeft de attribuut naam mee als parameter van `getAttribute`.
- Je krijgt de attribuut waarde terug, onder de vorm van `Object`.
- Je krijgt `null` terug als het session attribuut niet bestaat.

## 22.8 Session attribuut wijzigen

Je wijzigt een session attribuut met de session method `setAttribute`.  
Je geeft de naam van het te wijzigen attribuut mee en de nieuwe waarde voor dit attribuut.

## 22.9 Session attribuut verwijderen

Je verwijdert een session attribuut met de session method `removeAttribute`.  
Je geeft de naam van het te verwijderen attribuut mee als parameter.

## 22.10 Volledige session verwijderen

Je verwijdert de session (en zijn session attributen) met de session method `invalidate`.



Opmerking: je voegt session attributen toe, wijzigt session attributen en verwijdert session (attributen) met een POST request.  
Als je enkel session attributen leest, doe je dit met een GET request.

## 22.11 De webserver verwijdert een session

Als een session gedurende een aantal minuten niet aangesproken werd, verwijdert de webserver die session uit zijn geheugen. Men zegt dat de session 'vervalt'. Anders zou na een tijd het webserver geheugen vollopen met ongebruikte sessions. Dit gebeurt als de gebruiker een winkelmandje heeft, en de browser sluit of naar een andere website gaat. De website krijgt hiervan geen signaal en je kunt dus zijn session niet verwijderen. Elk webserver merk heeft een eigen waarde voor het aantal minuten waarna hij session vervalt. Bij Tomcat is dit 30 minuten. Je kan die waarde wijzigen in `web.xml`:

```
<web-app ...>
...
<session-config>
  <session-timeout>20</session-timeout>  <!-- 20 minuten -->
</session-config>
```

## 22.12 Voorbeeld 1

Je onthoudt de gebruikersnaam niet meer in een cookie, maar in een session attribuut.

Je wijzigt `IdentificatieServlet`:

```
package be.vdab.servlets;
// enkele imports ...
@WebServlet("/identificatie.htm")
public class IdentificatieServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/identificatie.jsp";
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession(false);
        if (session != null) {
            String gebruikersnaam =
                (String) session.getAttribute("gebruikersnaam");
```

❶  
❷  
❸

```

        if (gebruikersnaam != null) {
            request.setAttribute("gebruikersnaam", gebruikersnaam);
        }
    }
    request.getRequestDispatcher(VIEW).forward(request, response);
}
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    HttpSession session = request.getSession();
    session.setAttribute("gebruikersnaam",
        request.getParameter("gebruikersnaam"));
    // redirect naar huidige URL:
    response.sendRedirect(response.encodeRedirectURL(request.getRequestURI()));
}
}

```

- (1) Je haalt de session van de gebruiker op (als hij al een session heeft).
- (2) Als de gebruiker al een session heeft
- (3) probeer je het session attribuut gebruikersnaam te lezen.
- (4) Als dit session attribuut bestaat
- (5) geef je het als een request attribuut door aan de JSP.
- (6) Je haalt de session van de gebruiker op (als hij er al een heeft), of je maakt een session
- (7) Je onthoudt de ingetikte gebruikersnaam in een session attribuut gebruikersnaam.

Je kan dit uitproberen. Je gebruikersnaam wordt onthouden zolang je in de browser blijft.

## 22.13 Voorbeeld 2

Je maakt een pagina waarin de gebruiker pizza's toevoegt aan een winkelmandje.

Je onthoudt dit winkelmandje in een Session attribuut.

Assortiment	Uw mandje
<input type="checkbox"/> Calzone	<ul style="list-style-type: none"><li>• Fungi &amp; Olive</li><li>• Prosciutto</li></ul>
<input checked="" type="checkbox"/> Fungi & Olive	
<input type="checkbox"/> Margherita	
<input checked="" type="checkbox"/> Prosciutto	
<input type="button" value="Toevoegen aan mandje"/>	

Je maakt in `be.vdab.servlets` een servlet `PizzaBestellenServlet`.

Die verwerkt GET en POST requests naar de URL `/pizzas/bestellen.htm`:

```

package be.vdab.servlets;
// enkele imports ...
@WebServlet("/pizzas/bestellen.htm")
public class PizzaBestellenServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String VIEW = "/WEB-INF/JSP/pizzabestellen.jsp";
    private static final String MANDJE = "mandje";
    private final PizzaRepository pizzaRepository = new PizzaRepository();
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setAttribute("allePizzas", pizzaRepository.findAll());
        HttpSession session = request.getSession(false);
        if (session != null) {
            @SuppressWarnings("unchecked")
            Set<Long> mandje = (Set<Long>) session.getAttribute(MANDJE);

```



```

        if (mandje != null) {
            request.setAttribute("pizzasInMandje",
                mandje.stream()
                    .map(id -> pizzaRepository.read(id))
                    .filter(optionalPizza -> optionalPizza.isPresent())
                    .map(optionalPizza -> optionalPizza.get())
                    .collect(Collectors.toList()));
        }
    }
    request.getRequestDispatcher(VIEW).forward(request, response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    if (request.getParameterValues("id") != null) {
        HttpSession session = request.getSession();
        @SuppressWarnings("unchecked")
        Set<Long> mandje = (Set<Long>) session.getAttribute(MANDJE);
        if (mandje == null) {
            mandje = new LinkedHashSet<>();
        }
        mandje.addAll(
            Arrays.stream(request.getParameterValues("id"))
                .map(id -> Long.parseLong(id))
                .collect(Collectors.toSet()));
        session.setAttribute(MANDJE, mandje);
    }
    response.sendRedirect(response.encodeRedirectURL(request.getRequestURI()));
}
}

```

- (1) De gebruiker ziet met een GET request de invoerpagina van het mandje.
- (2) Je plaatst alle pizza's op request scope, om ze te tonen in de JSP.
- (3) Je haalt de session van de gebruiker op (als hij al een session heeft).
- (4) Je cast bij (5) een object onder de gedaante van Object (het resultaat van session.getAttribute) naar een Set<Long>. Je onderdrukt hier de compiler warning die de cast veroorzaakt.
- (5) Je leest de inhoud van het session attribuut mandje.
- (6) De gebruiker heeft in zijn session al een attribuut mandje.
- (7) Het mandje bevat enkel pizza ids. Jij wil de gebruiker de pizzanamen tonen. Je leest daartoe per id de bijbehorende pizza (met onder andere hun namen).
- (8) Je haalt de session van de gebruiker op, of je maakt een session als hij er nog geen had.
- (9) Je leest de inhoud van het session attribuut mandje.
- (10) De gebruiker heeft in zijn session nog geen attribuut mandje.
- (11) Je maakt een leeg mandje voor de gebruiker.
- (12) Je voegt de ids van de geselecteerde pizza's toe aan het mandje.
- (13) Je wijzigt het session attribuut mandje met het bijgewerkte mandje. Als je website uitgevoerd wordt op meerdere webserver, stuurt de webserver dit session attribuut naar de andere webserver.

Je maakt pizzabestellen.jsp in WEB-INF/JSP:

```

<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
    <head>
        <c:import url='/WEB-INF/JSP/head.jsp'>
            <c:param name='title' value='Bestellen'/>
        </c:import>
    </head>

```



```


</head>
<body>
  <c:import url='/WEB-INF/JSP/menu.jsp'/>
  <h1>Pizza's bestellen</h1>
  <c:if test='${not empty allePizzas}'>
    <h2>Assortiment</h2>
    <form method='post' id='toevoegform'>
      <ul class='zonderbolletjes'>
        <c:forEach var='pizza' items='${allePizzas}'>
          <li><label><input type='checkbox' name='id' value='${pizza.id}'>
            <c:out value='${pizza.naam}'/></label></li>
        </c:forEach>
      </ul>
      <input type='submit' value='Toevoegen aan mandje' id='toevoegknop'>
    </form>
  </c:if>
  <c:if test='${not empty pizzasInMandje}'>
    <h2>Uw mandje</h2>
    <ul><c:forEach var='pizza' items='${pizzasInMandje}'>
      <li><c:out value='${pizza.naam}'/></li>
    </c:forEach></ul>
  </c:if>
  <script>
    document.getElementById('toevoegform').onsubmit = function() {
      document.getElementById('toevoegknop').disabled=true;
    };
  </script>
</body>
</html>

```

Je kunt dit uitproberen. Je voegt pizza's toe aan het mandje. Je bezoekt daarna andere pagina's. Als je daarna Bestellen kiest, bevat het mandje nog altijd dezelfde pizza's.

Je ziet dat elke gebruiker zijn eigen session heeft door de website te bezoeken met meerdere browsertypes (Firefox, Chrome, ...). Elk type browser simuleert een andere gebruiker.

Je ziet URL rewriting door in de browser de cookies af te zetten. Je doet dit bij Firefox als volgt.

1. Je kiest rechts boven in Firefox .
2. Je kiest Options.
3. Je kiest Privacy.
4. Je kiest Use custom settings for history naast Firefox will.
5. Je verwijdert het vogeltje bij Accept cookies from sites.
6. Je kiest OK.

## 22.14 Session attribuut lezen in een JSP

Je leest in een JSP een session attribuut met EL : `${naamVanHetSessionAttribuut}`

Dit werkt enkel als die JSP sessions ondersteunt.

Je laat daartoe het onderdeel `session='false'` weg in de regel `<%@page ...%>` (de default is `session='true'`)

## 22.15 URL rewriting afzetten

Als tijdelijke cookies afstaan, gebruikt de webserver URL rewriting.

Dit is gevaarlijk: een gebruiker kan een URL (mét session ID),

die hij ziet in de adresbalk van de gebruiker, doormailen naar een andere persoon.

Als die andere persoon surft naar de URL, neemt hij de identiteit over van de eerste persoon.

Je kan URL rewriting afzetten met volgende regel voor `</session-config>`:

```
<tracking-mode>COOKIE</tracking-mode>
```

Nu wisselen de browser en de website de session ID enkel uit als tijdelijke cookie. Als de cookies zijn uitgeschakeld, werkt de website niet. Je kan dan een foutmelding tonen. Je voegt daartoe code toe aan de JavaScript in `pizzabestellen.jsp`:

```
<script>
document.getElementById('toevoegform').onsubmit = function() {
    if ( ! navigator.cookieEnabled) {
        alert("Dit werkt enkel als cookies aanstaan");
        return false;
    }
    document.getElementById('toevoegknop').disabled = true;
};
</script>
```

①  
②  
③

- (1) Het object `navigator` stelt je browser voor. De eigenschap `cookieEnabled` bevat `true` als cookies aanstaan en bevat `false` als cookies afstaan.
- (2) Je toont met de functie `alert` een popup venster met een boodschap.
- (3) Je geeft `false` terug in de huidige functie die zelf gekoppeld is aan het `onsubmit` event. Je verhindert met die `false` returnwaarde het submitten van de form.

Je commit de sources en je publiceert op GitHub.



Zoek de friet: zie takenbundel



Sauzen raden: zie takenbundel

## 23 LISTENERS

De servlet specificatie definieert enkele gebeurtenissen (events) die kunnen optreden in je website.

Bij elk van die gebeurtenissen hoort een interface.

Als je wil reageren op zo'n gebeurtenis (eigen code uitvoeren wanneer de gebeurtenis optreedt), implementeer je de interface die bij de gebeurtenis hoort in een eigen class.

Als de gebeurtenis optreedt, roept de webserver de code in die class op.

Zo'n class heet een listener.

Je leert hier onder de belangrijkste gebeurtenissen en de bijbehorende interfaces.

Voor de methods in de interface staat tussen `<<` en `>>` de gebeurtenis die deze method oproept.

Het overzicht bevat ook enkele classes.

Na de getters in de class staat **in groen** de betekenis van de returnwaarde van de getter.

### 23.1 De website wordt gestart of gestopt

<pre>&lt;&lt;interface&gt;&gt; ServletContextListener</pre>
<pre>&lt;&lt;de website start&gt;&gt; +contextInitialized(event: ServletContextEvent) &lt;&lt;de website stopt&gt;&gt; +contextDestroyed(event: ServletContextEvent)</pre>

Beide methods hebben een `ServletContextEvent` parameter

<pre>ServletContextEvent</pre>
<pre>+getServletContext(): ServletContext <b>de servlet context</b></pre>

### 23.2 Een servlet context attribuut wordt gemaakt, gewijzigd of verwijderd

<pre>&lt;&lt;interface&gt;&gt; ServletContextAttributeListener</pre>
<pre>&lt;&lt;een servlet context attribuut wordt toegevoegd&gt;&gt; +attributeAdded(event: ServletContextAttributeEvent) &lt;&lt;een servlet context attribuut wordt verwijderd&gt;&gt; +attributeRemoved(event: ServletContextAttributeEvent) &lt;&lt;een servlet context attribuut wordt gewijzigd&gt;&gt; +attributeReplaced(event: ServletContextAttributeEvent)</pre>

Alle methods hebben een `ServletContextAttributeEvent` parameter

<pre>ServletContextAttributeEvent</pre>
<pre>+getName(): String <b>de naam van het attribuut</b> +getValue(): Object <b>de waarde van het attribuut</b></pre>

### 23.3 Een session wordt gemaakt, verwijderd of vervalt

<pre>&lt;&lt;interface&gt;&gt; HttpSessionListener</pre>
<pre>&lt;&lt;een session wordt gemaakt&gt;&gt; +sessionCreated(event: HttpSessionEvent) &lt;&lt;een session wordt verwijderd of vervalt&gt;&gt; +sessionDestroyed(event: HttpSessionEvent)</pre>

De methods hebben een `HttpSessionEvent` parameter

<pre>HttpSessionEvent</pre>
<pre>+getSession(): HttpSession <b>de session</b></pre>

## 23.4 Een session attribuut wordt gemaakt, gewijzigd of verwijderd

<<interface>> HttpSessionAttributeListener	
<<een session attribuut wordt toegevoegd>>	+attributeAdded(event: HttpSessionBindingEvent)
<<een session attribuut wordt verwijderd>>	+attributeRemoved(event: HttpSessionBindingEvent)
<<een session attribuut wordt gewijzigd>>	+attributeReplaced(event: HttpSessionBindingEvent)

De methods hebben een HttpSessionBindingEvent parameter

HttpSessionBindingEvent	
+getName(): String	de naam van het attribuut
+getValue(): Object	de waarde van het attribuut
+getSession(): HttpSession	de sessie

## 23.5 Een browser request komt binnen of is helemaal verwerkt

<<interface>> ServletRequestListener	
<<een request komt binnen>>	+requestInitialized(event: ServletRequestEvent)
<<een request is verwerkt>>	+requestDestroyed(event: ServletRequestEvent)

De methods hebben een ServletRequestEvent parameter

ServletRequestEvent	
+getServletRequest(): ServletRequest	de request
+getServletContext(): ServletContext	de servlet context

## 23.6 Een servlet attribuut wordt gemaakt, gewijzigd of verwijderd

<<interface>> ServletRequestAttributeListener	
<<een servlet attribuut wordt toegevoegd>>	+attributeAdded(event: ServletRequestAttributeEvent)
<<een servlet attribuut wordt verwijderd>>	+attributeRemoved(event: ServletRequestAttributeEvent)
<<een servlet attribuut wordt gewijzigd>>	+attributeReplaced(event: ServletRequestAttributeEvent)

De methods hebben een ServletRequestAttributeEvent parameter

ServletRequestAttributeEvent	
+getName(): String	de naam van het attribuut
+getValue(): Object	de waarde van het attribuut

## 23.7 Meerdere classes die eenzelfde interface implementeren

Je kunt in meerdere classes eenzelfde interface implementeren.

Als de bijbehorende gebeurtenis optreedt, roept de webserver code in al die classes op.

Twee classes kunnen bijvoorbeeld de interface HttpSessionListener implementeren.

Bij het maken of verwijderen van een session, roept de webserver beide classes op.

## 23.8 Listener instances

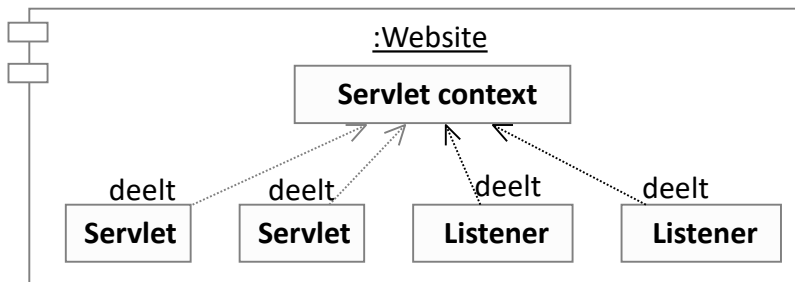
De webserver maakt listener instances wanneer dat de website start. De webserver kan daarna met meerdere threads listener methods uitvoeren, ze moeten dus thread safe zijn.

## 23.9 Configuratie

Je moet een listener class kenbaar maken aan de webserver, door @WebListener vóór de listener class te tikken.

## 23.10 Listeners en de servlet context

Listeners delen de servlet context van de website met de servlets:



## 23.11 Samenvatting



Als een event optreedt roept de webserver de bijbehorende listeners op.



## 23.12 Voorbeeld

Een listener houdt in een servlet context attribuut `aantalMandjes` het aantal mandjes bij.

Je listener implementeert `ServletContextListener`.

- Je initialiseert de teller in de method `contextInitialized`. De webserver roept deze method één keer op, als de website start.

Je listener implementeert ook `HttpSessionAttributeListener`.

- Je verhoogt de teller in de method `attributeAdded`, als de attribuut naam mandje is. De webserver roept de method `attributeAdded` op als je een session attribuut toevoegt.
- Je verlaagt de teller in de method `attributeRemoved`, als attribuut naam mandje is. De webserver roept de method `attributeRemoved` op als je een session attribuut verwijdert.

### 23.12.1 Eclipse

1. Je klikt met de rechtermuisknop op het project en je kiest New, Listener.
2. Je tikt `be.vdab.listeners` bij Java package.
3. Je tikt `MandjeListener` bij Class name en je kiest Next.
4. Je vinkt de interface(s) aan die de listener implementeert: `ServletContextListener` en `HttpSessionAttributeListener` en je kiest Finish.

### 23.12.2 NetBeans

1. Je klikt met de rechtermuisknop op het project en je kiest New, Other, Web, Web Application Listener en je kiest Next.
2. Je tikt `MandjeListener` bij Class Name.
3. Je tikt `be.vdab.listeners` bij Package en je kiest Next.
4. Je vinkt de interface(s) aan die de listener implementeert: `Context Listener` en `HTTP Session Attribute Listener` en je kiest Finish.

### 23.12.3 Voorbeeldlistener

```

package be.vdab.listeners;
// enkele imports ...
@WebListener
public class MandjeListener
    implements ServletContextListener, HttpSessionAttributeListener {
    private static final String MANDJE = "mandje";
    private static final String AANTAL_MANDJES = "aantalMandjes";
    @Override
    public void contextInitialized(ServletContextEvent event) {
        event.getServletContext().setAttribute(AANTAL_MANDJES, new AtomicInteger());
    }
  
```

```

@Override
public void contextDestroyed(ServletContextEvent event) {
}

@Override
public void attributeAdded(HttpSessionBindingEvent event) {           ④
    if (MANDJE.equals(event.getName())) {                             ⑤
        ((AtomicInteger)
            event.getSession().getServletContext().getAttribute(AANTAL_MANDJES))
            .incrementAndGet();                                         ⑥
    }
}

@Override
public void attributeRemoved(HttpSessionBindingEvent event) {         ⑦
    if (MANDJE.equals(event.getName())) {
        ((AtomicInteger)
            event.getSession().getServletContext().getAttribute(AANTAL_MANDJES))
            .decrementAndGet();                                         ⑧
    }
}

@Override
public void attributeReplaced(HttpSessionBindingEvent event) {
}
}

```

- (1) Je maakt met `@WebListener` een class als listener kenbaar aan de webserver.
- (2) Je implementeert `ServletContextListener` en reageert zo op starten en stoppen van de website. Je implementeert ook `HttpSessionAttributeListener` en reageert zo op het toevoegen, verwijderen en wijzigen van session attributen.
- (3) De website start, je maakt een servlet context attribuut `aantalMandjes` die een `AtomicInteger` bevat. Deze bevat dan de wiskundige waarde nul.
- (4) Telkens een attribuut toegevoegd wordt aan een sessie, roept de webserver de method `attributeAdded` op.
- (5) Je controleert of het toegevoegde attribuut de naam `mandje` heeft.
- (6) Je verhoogt de teller in het servlet context attribuut `aantalMandjes`.
- (7) Telkens een attribuut verwijderd wordt uit een sessie, roept de webserver de method `attributeRemoved` op.
- (8) Je verlaagt de teller in het servlet context attribuut `aantalMandjes`.

Je toont in `statistiek.jsp` het aantal mandjes, na `<h1>`

```
<div>${aantalMandjes} mandje(s)</div>
```

Je commit de sources en je publiceert op GitHub. Je kunt dit uitproberen.



Statistiek: zie takenbundel

## 24 FILTERS

Een filter onderschept een browser request op het moment dat die binnenkomt in de webserver, vóór die request in een servlet binnenkomt.

Handelingen die je in alle servlets zou moeten coderen, schrijf je beter één keer in een filter die de requests voor die servlets onderschept.

Voorbeeld: je hebt in de `doPost` method van `PizzaToevoegenServlet` en `IdentificatieServlet` de request parameters correct geïnterpreteerd met `request.setCharacterEncoding("UTF-8");`

Het is vervelend die regel te moeten herhalen in de `doPost` method van alle servlets.

Als je in één van de servlets de regel vergeet, bevat de database verkeerde data.

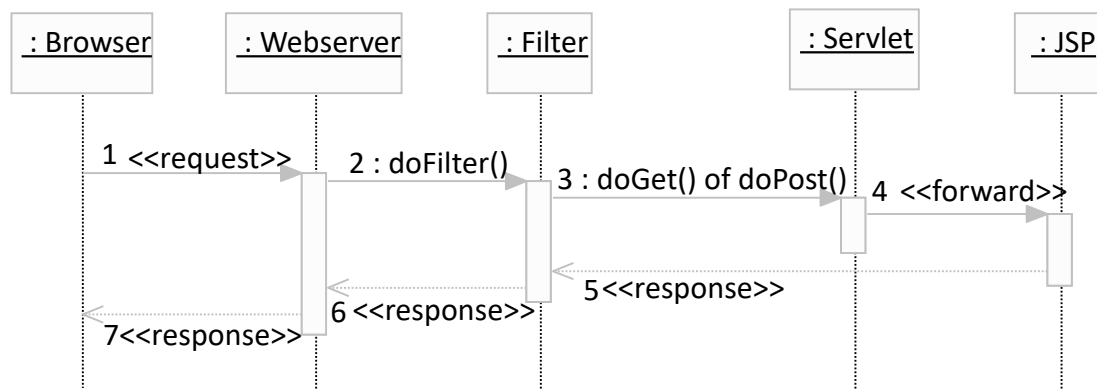
Je zal de opdracht één keer tikken in een filter die alle servlet requests onderschept.

### 24.1 Filter eigenschappen

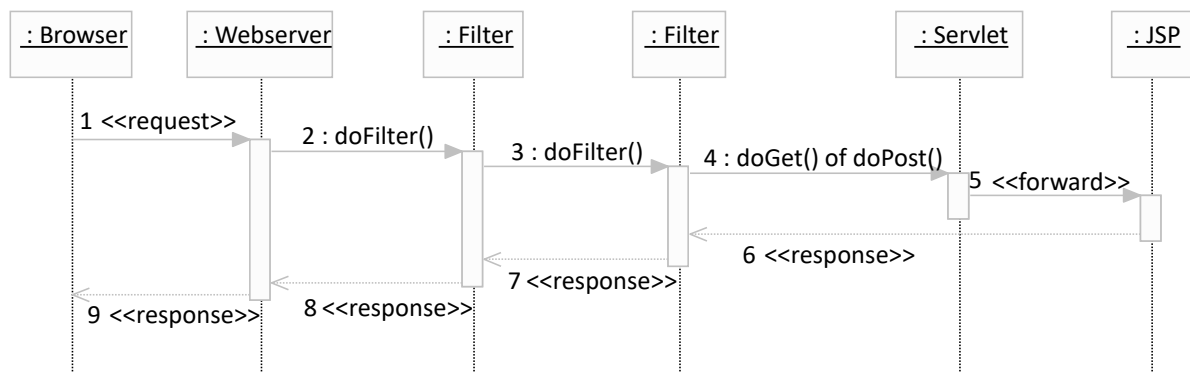
- Een filter is een class die de interface `Filter` implementeert.
- Je definieert in een regel `@WebFilter` voor de filter class een URL patroon (bvb `*.htm`). De filter onderschept enkel requests waarvan de URL past bij dit URL patroon. Bij het URL patroon `*.htm` onderschept de filter alle servlets uit deze cursus.
- De webserver stuurt een browser request naar de method `doFilter` in de filter. Je kunt in die method de request lezen en wijzigen Voorbeeld: `request.setCharacterEncoding("UTF-8");` Je geeft daarna de request door aan de servlet waarvoor de request bedoeld is.
- De webserver stuurt de response die de JSP genereert naar dezelfde filter method `doFilter`. Je kunt daarin de response (en bijbehorende request) lezen en wijzigen. Daarna gaat de response naar de browser.

Een filter lijkt op een listener die `ServletRequestListener` implementeert, maar verwerkt enkel requests met een URL die past bij zijn URL patroon.

Een sequence diagram van een filter die een request naar een servlet onderschept:



Meerdere filters kunnen eenzelfde request onderscheppen:



## 24.2 URL patronen

Je associeert een filter met een URL patroon. Dit patroon heeft één van volgende vormen:

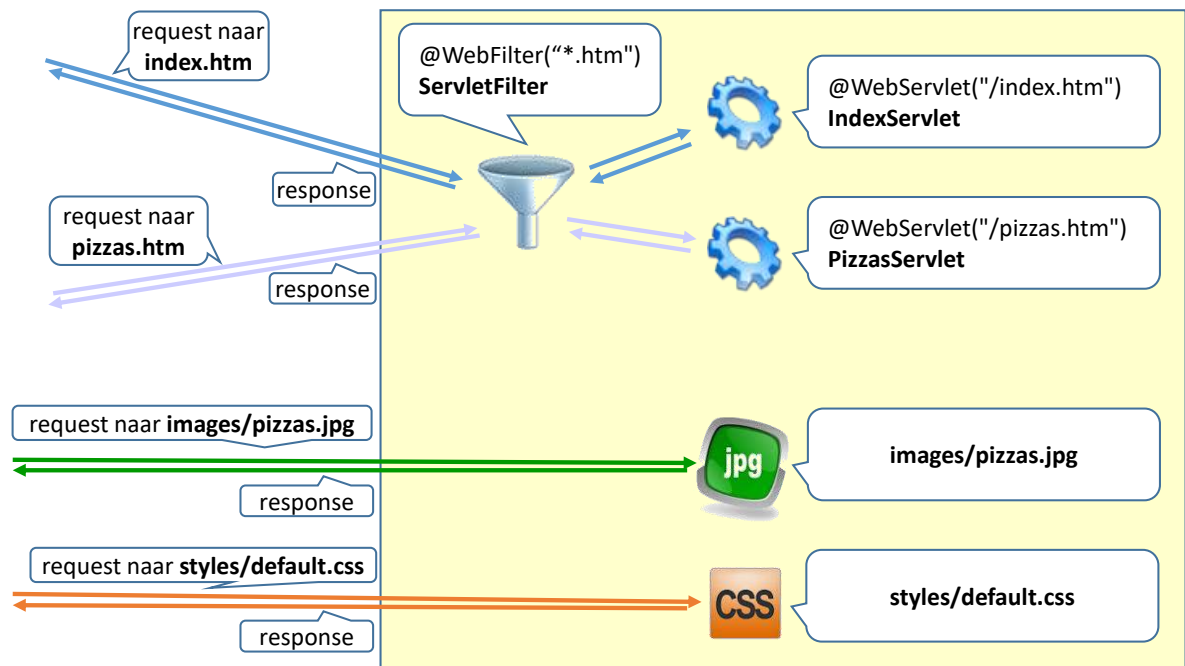
- beginnen met / en eindigen op /\*
- beginnen met \*.
- /\* (dit patroon betekent alle URL's).
- beginnen met / en het teken \* niet bevatten

Als een URL patroon niet één van die vormen heeft, start de website niet.

Tomcat werpt een `InvalidArgumentException` met als message `Invalid <url-pattern>`.

URL patroon	URL's die bij dit URL patroon passen
/menu/pizzas.htm	/menu/pizzas.htm
/pizzas/*	/pizzas/bestellen /pizzas/detail /pizzas/tussenprijzen /pizzas/toevoegen /pizzas/voorkeuren
*.htm	De URL's van de servlets uit deze cursus.
/*	Alle URL's.

Voorbeeld: een filter met het patroon `*.htm` onderschept enkel de requests naar servlets in deze cursus, niet requests naar CSS bestanden en afbeeldingen:



Je kunt een filter ook associëren met meerdere URL patronen.

De filter onderschept dan requests waarvan de URL past bij één van de patronen.

## 24.3 Filter instances

De webserver maakt één instance per filter class.

Afhankelijk van het merk van de webserver doet hij dit wanneer:

- de website start (Tomcat doet dit).
- of een eerste request binnenkomt die de filter moet onderscheppen.

De webserver verwerkt gelijktijdige requests met gelijktijdige threads die gelijktijdig de code van één filter instance uitvoeren. De filter class moet daarom thread safe zijn.



## 24.4 De interface Filter

Een filter is een class die de interface Filter implementeert. De interface bevat 3 methods:

- **init**

De webserver roept de method 1 keer op, na het maken van een filter instance.

Je doet in de method éénmalige initialisaties in de filter.

De method krijgt een FilterConfig parameter binnen:

<pre>&lt;&lt;interface&gt;&gt; FilterConfig  +getServletContext(): ServletContext +getInitParameter(name: String)</pre>	<pre>getServletContext geeft toegang tot de servlet context.  getInitParameter leest filter initialisatieparameters in web.xml.</pre>
---	---

- **destroy**

De webserver roept de method op bij het stoppen van de website.

Je kunt 'opkuiswerk' doen, zoals een tijdelijk bestand verwijderen dat je in de filter aanmaakte.

- **doFilter**

Als de webserver een request binnenkrijgt die de filter moet onderscheppen, roept de webserver deze method op. De parameters zijn de request en de response

- Je krijgt de request onder de gedaante van de interface ServletRequest.

De interface HttpServletRequest (waarmee je een request behandelt in een servlet) erft van ServletRequest.

Je krijgt de response onder de gedaante van de interface ServletResponse.

De interface HttpServletResponse (waarmee je een response behandelt in een servlet) erft van ServletResponse.

- De method bevat ook een derde FilterChain parameter. Je roept op die parameter de method doFilter op en je geeft de request en response mee.

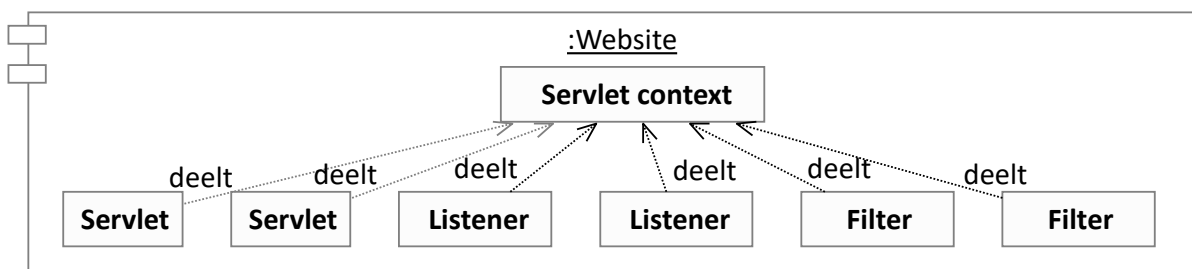
De webserver geeft dan de request en response door aan

- de volgende filter die de request verwerkt of
- de servlet die uiteindelijk de request verwerkt.

Als je de method doFilter niet oproept, krijgen de volgende filters en de servlet (waarvoor de request bedoeld was) de request niet binnen.

## 24.5 Filters en de servlet context

Filters delen de servlet context van de website met de servlets en de listeners:



## 24.6 Eclipse

1. Je klikt met de rechtermuisknop op het project en je kiest New, Filter.
2. Je tikt be.vdab.filters bij Java package.
3. Je tikt ServletFilter bij Class name en je kiest Next.
4. Je wijzigt bij Filter mappings het URL patroon van requests die de filter onderschept
  - a. Je selecteert bij Filter mappings de regel /ServletFilter
  - b. Je kiest Edit
  - c. Je wijzigt Pattern naar \*.htm en je kiest OK.
5. Je kiest Finish.

## 24.7 NetBeans

1. Je klikt met de rechtermuisknop op het project en je kiest New, Other, Web, Filter en je kiest Next.
2. Je tikt ServletFilter bij Class Name.
3. Je tikt be.vdab.filters bij Package en je kiest Next.
4. Je ziet bij Filter Mappings in de rechterkolom het URL patroon van de filter.  
Je wijzigt met Edit dit patroon naar \*.htm  
Je kunt met New, Edit en Delete meerdere URL patronen associëren met de filter.
5. Je kiest Finish.

Je wijzigt @WebFilter voor de class naar zijn eenvoudigste vorm @WebFilter("\*.htm")

De class bevat voorbeeldcode in meerdere methods. Je verwijdert alle methods, behalve de methods doFilter, init en destroy. Je maakt de binnenkant van de methods leeg.

## 24.8 Voorbeeldfilter

Je wijzigt de method doFilter

```
request.setCharacterEncoding("UTF-8");  
chain.doFilter(request, response);
```

❶  
❷

- (1) Je geeft aan dat de request parameters uitgedrukt zijn in Unicode UTF-8.
- (2) Je geeft de request en de response door aan de servlet waarvoor de request bedoeld was.  
Nadat de servlet (en de bijbehorende JSP) de request en response verwerken gaat de code verder in eventuele opdrachten die je na (2) zou schrijven.

Je verwijdert in IdentificatieServlet en PizzaToevoegenServlet de regel

```
request.setCharacterEncoding("UTF-8");
```

Je commit de sources en je publiceert op GitHub.

Je kunt de website uitproberen.



Statistiek 2: zie takenbundel

## 25 JDBC

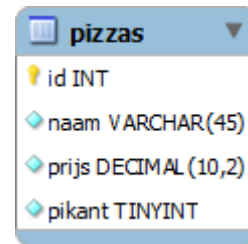
### 25.1 Database

Je maakt in MySQL een database pizzaluigi en daarin een table pizzas.

- id is een auto increment kolom.
- tinyint is een synoniem voor boolean.

Je voegt aan de table enkele records toe.

Je geeft de gebruiker cursist select en insert rechten in deze table.



### 25.2 Exceptions

Als je databasebewerkingen uitvoert, kan een SQLException optreden in repository class methods. Je roept deze methods op vanuit servlets. Als een exception optreedt, toon je een foutmelding in de browser. Je moet daartoe de exception opvangen in de servlet.

Als je in de servlet een SQLException opvangt, heb je daar hard gecodeerd dat je JDBC gebruikt. Er bestaan echter andere libraries om databasebewerkingen uit te voeren, zoals Hibernate of JPA. Die werpen andere types exceptions als fouten optreden. Als je in de repository classes JDBC vervangt door zo'n library, moet je ook in de servlets andere types exceptions opvangen. Dit is vervelend.

Je lost dit op door SQLExceptions op te vangen in de repository classes.

Je werpt in de catch blokken RepositoryExceptions (een eigen exception class) naar de servlets. De servlet vangt geen SQLExceptions meer, maar RepositoryExceptions en is zo niet meer hard gecodeerd gekoppeld aan JDBC.

Je maakt in be.vdab.repositories een class RepositoryException

```
package be.vdab.repositories;
```

```
public class RepositoryException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    public RepositoryException(Throwable cause) {
        super(cause);
    }
}
```

①  
②

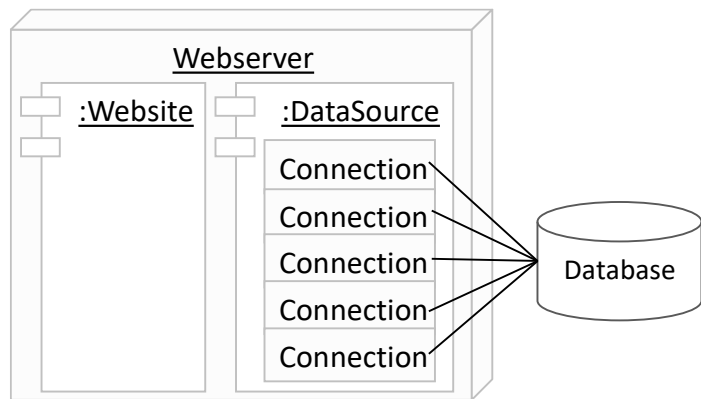
- (1) Je maakt een constructor die een andere exception als parameter binnenkrijgt (Throwable is een interface die alle exceptions implementeren).  
Die andere exception zal de oorspronkelijke SQLException zijn die JDBC werpt.
- (2) Je geeft die andere exception door aan de constructor van je base class (RuntimeException)  
Die constructor onthoudt de informatie over die andere exception.  
Als een RepositoryException optreedt zie je in de browser en in het Eclipse venster Console niet enkel informatie over die RepositoryException, maar ook over de onthouden SQLException. Deze extra informatie helpt je om de fout in te schatten en te corrigeren.

### 25.3 DataSource

Je hebt een Connection nodig om een databasebewerking uit te voeren.

Een Connection openen vraagt tijd. Bij eenvoudige SQL statements kan het openen van een Connection meer tijd vragen dan het uitvoeren van het SQL statement zelf. Het is dus interessant om niet bij elke browser request nieuwe Connection te maken, maar een bestaande Connection te hergebruiken.

Java webserver bieden hiertoe een DataSource aan. Dit is een verzameling Connection objecten die de webserver continu openhoudt. Als je in je code een Connection nodig hebt, vraag je de DataSource één van zijn Connections.

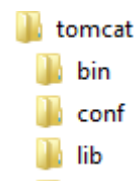


De werking van de DataSource :

1. De webserver maakt de DataSource aan wanneer de website start. Hij opent enkele Connections naar de database en onthoudt ze in de DataSource.
  2. Een browser request komt binnen. De webserver voert met een thread je code uit. Je vraagt in je Java code een Connection aan de DataSource. Je krijgt de Connection zeer snel: ze stond al open naar de database. De DataSource onthoudt dat die Connection in gebruik is.
  3. Terwijl de webserver die request verwerkt, komt een andere request binnen. De webserver verwerkt die request met een andere thread. Je vraagt in de Java code een Connection aan de DataSource. Die zoekt een Connection die niet in gebruik is en geeft die. Hij geeft je dus niet de Connection die de eerste thread nog in gebruik heeft. Hij onthoudt ook nu dat de Connection, die hij je geeft, in gebruik is.
  4. Je sluit in de code, uitgevoerd door de eerste thread, de Connection. De DataSource onderschept dit sluiten echter en laat de Connection naar de database open. Hij onthoudt wel dat die Connection niet meer in gebruik is. Het is essentieel dat je een Connection sluit na gebruik. Anders zijn binnen de kortste keren alle Connection objecten van de DataSource in gebruik en blokkeert de website.
- Een DataSource is een object dat de interface DataSource implementeert
  - Een synoniem van een DataSource is een connection pool.
  - Als je meerdere databases gebruikt, maak je één DataSource per database.
  - Een DataSource object is thread safe.

### 25.3.1 JDBC driver

Je kopieert de JDBC driver (mysql-connector-java-x.y.z-bin) naar de directory lib in de Tomcat directory:  
Tomcat zoekt daar JDBC drivers en gebruikt die bij het openen van de Connections in de DataSource.



### 25.3.2 DataSource definiëren

Je definieert een DataSource in een XML bestand. De naam, plaats en inhoud van dit bestand verschilt per webserver merk. Bij Tomcat is dit context.xml in de folder META-INF.

#### 25.3.2.1 Eclipse

Een Eclipse website project heeft al een folder META-INF (in het gedeelte WebContent). Je maakt in die folder context.xml:

1. Je klikt met de rechtermuisknop op META-INF.
2. Je kiest New, Other.
3. Je kiest XML, XML File en Next.
4. Je tikt context.xml bij File name en je kiest Finish.

### 25.3.2.2 NetBeans

Een NetBeans website project heeft al een folder META-INF (in het gedeelte Web Pages) met daarin het bestand context.xml.

#### 25.3.2.3 context.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<Context path='/pizzaluigi'>
  <Resource name='jdbc/pizzaluigi'
    type='javax.sql.DataSource'
    driverClassName='com.mysql.jdbc.Driver'
    url='jdbc:mysql://localhost/pizzaluigi?useSSL=false'
    username='cursist'
    password='cursist'
    closeMethod='close' />
</Context>
```

- (1) Eclipse toont een warning omdat dit bestand niet geassocieerd is met een schema.
- (2) Je maakt per DataSource een child element Resource binnen het element Context. Je geeft elke DataSource een unieke naam met het attribuut name. Zo'n naam noemt een JNDI (Java Naming and Directory Interface) name. Er is een conventie dat een DataSource naam begint met jdbc/
- (3) type bevat de Java interface die een DataSource voorstelt: javax.sql.DataSource.
- (4) driverClassName bevat de JDBC driver class waarmee Tomcat de database opent.
- (5) url bevat de JDBC URL waarmee Tomcat de database opent.
- (6) username bevat de gebruikersnaam waarmee Tomcat de database opent.
- (7) password bevat het bijbehorend paswoord.
- (8) Als de website stopt, sluit Tomcat standaard de Connections in de DataSource pas wanneer de garbage collector het geheugen opruimt. Dit kan een eindje duren. Met deze regel voert Tomcat op de DataSource de method close uit als de website stopt. Deze method sluit direct alle connecties in de DataSource.



context.xml verwijst in het **url** attribuut naar de database.



## 25.4 DataSource in je code

### 25.4.1 AbstractRepository

Je maakt in be.vdab.repositories een base class voor alle repository classes:

```
package be.vdab.repositories;
import javax.sql.DataSource;
abstract class AbstractRepository {
  public final static String JNDI_NAME = "jdbc/pizzaluigi";
  protected DataSource dataSource;
  public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
  }
}
```

- (1) De class is abstract: het is niet de bedoeling is van de class een instance te maken. De class heeft package (geen public) visibility: de class is enkel bereikbaar in repository classes.
- (2) Je hebt de JNDI naam van de datasource (jdbc/pizzaluigi) in veel stukken code nodig. In de plaats van hem telkens te herhalen, vermeld je hem hier één keer. Op de andere plaatsen verwijst je dan naar deze constante JNDI\_NAME
- (3) Je maakt de DataSource protected, zo kunnen de derived repository classes hem aanspreken.
- (4) Je zal deze setter oproepen vanuit de servlets.

### 25.4.2 De servlets

Je tikt in `PizzaBestellenServlet`, `PizzaDetailServlet`, `PizzasServlet`, `PizzasTussenPrijzenServlet`, `PizzaToevoegenServlet` en `VoorkeurPizzasServlet`

```
@Resource(name = PizzaRepository.JNDI_NAME) ❶
void setDataSource(DataSource dataSource) {
    pizzaRepository.setDataSource(dataSource); ❷
}
```

Je wijzigt in elk van die servlets

```
private final PizzaRepository pizzaRepository = new PizzaRepository();
naar
private final transient PizzaRepository pizzaRepository = new PizzaRepository(); ❸
```

- (1) Je tikt `@Resource` voor een `DataSource` setter (op de volgende regel) in een servlet.  
Je vult de `name` parameter met de JNDI name van de `DataSource` in `context.xml`.  
Juist nadat Tomcat de servlet instance maakt, roept Tomcat zo'n setter één keer *automatisch* op en vult de parameter van de setter met een verwijzing naar die `DataSource`.
- (2) Je geeft de `DataSource` door aan de repository class, waar je hem later zal gebruiken.  
Als je servlet meerdere repository classes heeft, kan je die ook hier een `DataSource` geven.
- (3) Als Java de servlet via serialization wegschrijft, moet hij het repository object niet meeschrijven.



@Resource

verwijst in  
de name  
parameter  
naar de



DataSource



in



context.xml

Die verwijst  
in het url  
attribuut  
naar de



Database

### 25.4.3 PizzaRepository

```
package be.vdab.repositories;
// enkele imports ...
```

```
public class PizzaRepository extends AbstractRepository {
    private static final String BEGIN_SELECT =
        "select id, naam, prijs, pikant from pizzas";
    private static final String FIND_ALL = BEGIN_SELECT + "order by naam";
    private static final String READ = BEGIN_SELECT + "where id=?";
    private static final String FIND_BY_PRIJS_BETWEEN = BEGIN_SELECT +
        "where prijs between ? and ? order by prijs";
    private static final String CREATE =
        "insert into pizzas(naam, prijs, pikant) values (?, ?, ?)";
    public List<Pizza> findAll() {
        try (Connection connection = dataSource.getConnection();
            Statement statement = connection.createStatement()) {
            List<Pizza> pizzas = new ArrayList<>();
            connection.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
            connection.setAutoCommit(false);
            try (ResultSet resultSet = statement.executeQuery(FIND_ALL)) {
                while (resultSet.next()) {
                    pizzas.add(resultSetRijNaarPizza(resultSet));
                }
            }
            connection.commit();
            return pizzas;
        } catch (SQLException ex) {
            throw new RepositoryException(ex); ❶
        } ❷
    }
    private Pizza resultSetRijNaarPizza(ResultSet resultSet) ❸
        throws SQLException {
        return new Pizza(resultSet.getLong("id"), resultSet.getString("naam"),
            resultSet.getBigDecimal("prijs"), resultSet.getBoolean("pikant"));
    } ❹
}
```

```

public Optional<Pizza> read(long id) {
    try (Connection connection = dataSource.getConnection();
        PreparedStatement statement = connection.prepareStatement(READ)) {
        Optional<Pizza> pizza;
        connection.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
        connection.setAutoCommit(false);
        statement.setLong(1, id);
        try (ResultSet resultSet = statement.executeQuery()) {
            if (resultSet.next()) {
                pizza = Optional.of(resultSetRijNaarPizza(resultSet));
            }
            else {
                pizza = Optional.empty();
            }
        }
        connection.commit();
        return pizza;
    } catch (SQLException ex) {
        throw new RepositoryException(ex);
    }
}

public List<Pizza> findByPrijsBetween(BigDecimal van, BigDecimal tot) {
    try (Connection connection = dataSource.getConnection();
        PreparedStatement statement =
            connection.prepareStatement(FIND_BY_PRIJS_BETWEEN)) {
        List<Pizza> pizzas = new ArrayList<>();
        connection.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
        connection.setAutoCommit(false);
        statement.setBigDecimal(1, van);
        statement.setBigDecimal(2, tot);
        try (ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                pizzas.add(resultSetRijNaarPizza(resultSet));
            }
        }
        connection.commit();
        return pizzas;
    } catch (SQLException ex) {
        throw new RepositoryException(ex);
    }
}

public void create(Pizza pizza) {
    try (Connection connection = dataSource.getConnection();
        PreparedStatement statement = connection.prepareStatement(
            CREATE, Statement.RETURN_GENERATED_KEYS)) {
        connection.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
        connection.setAutoCommit(false);
        statement.setString(1, pizza.getNaam());
        statement.setBigDecimal(2, pizza.getPrijs());
        statement.setBoolean(3, pizza.isPikant());
        statement.executeUpdate();
        try (ResultSet resultSet = statement.getGeneratedKeys()) {
            resultSet.next();
            pizza.setId(resultSet.getLong(1));
        }
        connection.commit();
    }
    catch (SQLException ex) {
        throw new RepositoryException(ex);
    }
}
}

```

5

- (1) Als een `SQLException` optreedt,
- (2) vang je die op en je werpt een `RepositoryException` ter vervanging.  
Je geeft de oorspronkelijke `SQLException` mee ter informatie.
- (3) Je moet in meerdere methods (`findAll`, `findByPrijsBetween`, `read`) één rij in een `ResultSet` omzetten naar een `Pizza` object. Je herhaalt de bijbehorende code niet in elk van die methods.  
Je tikt de code één keer in deze method, die je oproept vanuit de andere methods.
- (4) Als in deze method een `SQLException` optreedt, werp je deze door  
naar de method die deze method heeft opgeroepen.
- (5) Je vult het id, dat de database plaatste in de autonumber kolom, in het `Pizza` object.  
De servlet die de method `create` oproept, vindt het id van de nieuwe pizza in dit `Pizza` object  
en kan hiermee bijvoorbeeld een redirect doen naar een URL die dit id bevat.

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.



Database: zie takenbundel



## 26 FOUTOPVANG

### 26.1 Foutpagina's koppelen aan HTTP status codes

Een browser request kan een fout veroorzaken waarvoor jouw code niet verantwoordelijk is. Het bekendste voorbeeld is een request naar een URL die niet bestaat in de website. De webserver stuurt in dat geval een response met een status code 404 (Not Found) en een response body (HTML) die de webserver zelf aanmaakt.

Je ziet de lelijke response als je surft naar `http://localhost:8080/pizzaluigi/xxx`

Je kunt met een status code een eigen mooie foutpagina associëren. Als een fout met die status code optreedt, stuurt de webserver een response met in de body je eigen foutpagina.

Je associeert in `web.xml` een pagina met een status code, tussen `<web-app>` en `</web-app>`:

```
<error-page>
  <error-code>404</error-code>
  <location>/WEB-INF/JSP/404.jsp</location>
</error-page>
```

①  
②

(1) Je tikt bij `error-code` de status code waarmee je de eigen foutpagina associeert.

(2) Je tikt bij `location` de plaats en de naam van de eigen foutpagina.

Je maakt `404.jsp` in `WEB-INF/JSP`:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
  <head>
    <c:import url='/WEB-INF/JSP/head.jsp'>
      <c:param name='title' value='Pagina niet gevonden' />
    </c:import>
  </head>
  <body>
    <c:import url='/WEB-INF/JSP/menu.jsp' />
    <h1>Pagina niet gevonden</h1>
    <img src='<c:url value="/images/fout.jpg"/>' alt='fout'>
    <p>De pagina die u zoekt bestaat niet op onze website.</p>
  </body>
</html>
```

Je kunt de website uitproberen.

### 26.2 Foutpagina's koppelen aan exceptions

Als een exception optreedt in de website, stuurt de webserver een response met status code 500 (Internal Server Error) en een response body met informatie over die exception.

Je probeert dit uit met volgende stappen:

- Je start de website.
- Je stopt MySQL.
  - Je klikt op de Start knop van Windows.
  - Je tikt `local services`.
  - Je kiest `View local services`.
  - Je kiest `MySQL` in de lijst midden in het venster (na `MySQL` kan nog een versienummer komen).
  - Je kiest `Stop the Service`.
- Je probeert in de browser de pizza's te zien.

De foutpagina heeft problemen

- ➔ Een gewone eindgebruiker is overdonderd door de informatie.
- ➔ Een hacker leert via de informatie de binnenkant van de website kennen. (Hij weet bijvoorbeeld dat de database een MySQL database is).

Een oplossing is een foutpagina te associëren met de status code 500, zoals je een foutpagina associeerde met de status code 404.

Een andere oplossing is een pagina te associëren met een type exception. Als een exception van dat type optreedt, stuurt de webserver een response met in de body die eigen foutpagina.

Je associeert repositoryexception.jsp met het exception type RepositoryException in web.xml, tussen <web-app> en </web-app>:

```
<error-page>
  <exception-type>be.vdab.repositories.RepositoryException</exception-type> ❶
  <location>/WEB-INF/JSP/repositoryexception.jsp</location> ❷
</error-page>
```

- (1) Je tikt bij exception-type het type exception waarmee je een eigen pagina associeert.
- (2) Je tikt bij location de plaats en de naam van de eigen pagina.

Je maakt repositoryexception.jsp in WEB-INF/JSP:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
  <head>
    <c:import url='/WEB-INF/JSP/head.jsp'>
      <c:param name='title' value='Problemen bij ophalen data'/>
    </c:import>
  </head>
  <body>
    <c:import url='/WEB-INF/JSP/menu.jsp'>
    <h1>Problemen bij het ophalen van data</h1>
    <img src='<c:url value="/images/datafout.jpg"/>' alt='data fout'>
    <p>We kunnen de gevraagde data niet ophalen
      wegens een technische storing.<br>
      Gelieve de helpdesk te contacteren.</p>
    </body>
</html>
```

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.

## 26.3 Fouten loggen



Als een exception optreedt, is het belangrijk de exception informatie weg te schrijven in een logbestand op de webserver. De websitebeheerder en/of ontwikkelaar vindt in dit bestand informatie om de fout te onderzoeken en corrigeren.

Je schrijft met de class Logger (uit de package java.util.logging) data naar een logbestand.

Je maakt in de class PizzaRepository een Logger instance

```
private final static Logger LOGGER =
  Logger.getLogger(PizzaRepository.class.getName());
```

De static method getLogger maakt een Logger object. Je geeft de naam van de huidige class mee.

Je tikt voor elke throw new RepositoryException(); opdracht volgende opdracht:

```
LOGGER.log(Level.SEVERE, "Probleem met database pizzaluigi", ex); ❶
```

- (1) Je schrijft met de method log informatie naar een logbestand.
  - Je geeft bij een exception als eerste parameter SEVERE mee.
  - Je geeft als tweede parameter een omschrijving van de fout mee.
  - Je geeft als derde parameter de opgetreden exception mee.

Je commit de sources en je publiceert op GitHub. Je kan de website uitproberen.

Je ziet de informatie in het logbestand bij Eclipse in het venster Console en bij NetBeans in het venster Output.

## 27 INTERNATIONALIZATION



Je houdt bij internationalization rekening met de taal en het land van de gebruiker.

- |          |   |
|----------|---|
| Teksten  | <ul style="list-style-type: none"> <li>• Je toont aan een Nederlandstalige gebruiker de tekst Goede morgen.</li> <li>• Je toont aan een Engelstalige gebruiker de tekst Good morning.</li> </ul>  |
| Datums   | <ul style="list-style-type: none"> <li>• Je toont datums aan Belgische gebruikers als dag/maand/jaar (31/1/2017).</li> <li>• Je toont datums aan Amerikaanse gebruikers als maand/dag/jaar (1/31/2017).</li> </ul>  |
| Getallen | <ul style="list-style-type: none"> <li>• Je toont getallen aan een Belgische gebruiker met een komma tussen eenheden en decimalen, en een punt tussen duizendtallen (1.000,23).</li> <li>• Je toont getallen aan een gebruiker uit de USA met een punt tussen eenheden en decimalen, en een komma tussen duizendtallen (1,000.23).</li> </ul> |

Elke taal heeft een ISO code, bestaande uit twee letters: nl (Nederlands), fr (Frans), ...

Elk land heeft een ISO code, bestaande uit twee letters: BE (België), NL (Nederland), ...

### 27.1 Locale

De class `Locale` (uit de package `java.util`) stelt een geografische regio voor.

Dit is de combinatie van de taal (bijvoorbeeld nl) en het land (bijvoorbeeld BE)

### 27.2 Request header Accept-Language

Een browser stuurt in elke request een header `Accept-Language` mee, met de taal en het land van de gebruiker. Voorbeeld: `Accept-Language: nl-be`.

De header kan ook meerdere taal-land combinaties bevatten.

De meest geprefereerde taal-land combinatie is als eerste vermeld.

Voorbeeld: `Accept-Language: nl-be, en-us`.

De gebruiker bepaalt in de browser instellingen de inhoud van de header.

Je doet dit bij Firefox in de Options, op het tabblad Content:

1. Je kiest bij Languages voor Choose.
2. Je kunt talen en taal-land combinaties toevoegen en verwijderen met Add en Remove.
3. Je kunt de volgorde instellen met Move up en Move down.

De webserver maakt op basis van de `Accept-Language` request header één of meerdere `Locale` objecten.

Je leest die `Locale` objecten in `doGet` of `doPost` met twee request methods:

- `getLocale` Geeft één `Locale` object terug met enkel de meest geprefereerde taal-land combinatie.
- `getLocales` Geeft een Enumeration van `Locale` objecten terug met alle taal-land combinaties.

Je kunt dus in je servlet code de voorkeurtal en -land van de gebruiker weten.

Je doet datumopmaak, getalopmaak en het tonen van teksten in meerdere talen in JSP's met JSTL internationalization tags.

### 27.3 Datumopmaak

De datumopmaak werkt standaard met het verouderde type `Date` dat een datum en/of tijd voorstelt.

Je toont hiermee de opgemaakte systeemdatum.

Je tikt in de `IndexServlet` method `doGet`:

```
Date nu = Calendar.getInstance().getTime();
request.setAttribute("nu", nu);
```

Je toont de datum in `index.jsp`, opgemaakt volgens het land van de gebruiker.

Je doet dit met de JSTL tag `formatDate` uit de JSTL library `internationalization`.

Die library heeft de URI `http://java.sun.com/jsp/jstl/fmt`

Je associeert boven in de pagina de URI met zijn conventie prefix `fmt`.

```
<%@taglib prefix='fmt' uri='http://java.sun.com/jsp/jstl/fmt'%>
```

Je tikt voor `</body>`

```
<div>Vandaag:<fmt:formatDate value='${nu}' /></div>
```

Je kunt de website uitproberen en via de browser instellingen

een gebruiker uit België simuleren en daarna een gebruiker uit de USA.

Als je de instelling wijzigt, zie je de gewijzigde datumopmaak met een pagina 'refresh'.

Je verfijnt de opmaak met volgende `formatDate` attributen:

- `type` Je bepaalt of je de datum en/of de tijd toont:
  - `date` Enkel het datum deel tonen.
  - `time` Enkel het tijd deel tonen.
  - `both` Het datum én het tijd deel tonen.
- `dateStyle` Je bepaalt hoe uitgebreid het datum deel is:
  - `short` Dag, maand en jaar als getallen tonen.
  - `medium` Dag en jaar als getallen tonen, maand als afkorting
  - `long` Dag en jaar als getallen tonen, maand voluit geschreven
  - `full` Zelfde als `long`, maar nu ook met dag van de week
- `timeStyle` Je bepaalt hoe uitgebreid het tijd deel is:
  - `short` Uren en minuten
  - `medium` Uren, minuten en seconden
  - `long` Uren, minuten, seconden en tijdzone
  - `full` Kleine variant van `long`, afhankelijk van taal gebruiker.

Je wijzigt : `<fmt:formatDate value='${nu}' type='date' dateStyle='Long' />`

Vanaf Java 8 bestaan modernere types om een dag en/of tijd voor te stellen.

- `LocalDate` stelt een datum voor
- `LocalTime` stelt een tijd voor
- `LocalDateTime` stelt een datum én tijd voor.

Je wijzigt in de `IndexServlet` method `doGet` de regel

```
Date nu = Calendar.getInstance().getTime();
```

naar

```
LocalDateTime nu = LocalDateTime.now();
```

Je tikt in `index.jsp` voor de regel

```
<div>Vandaag: ...
```

volgende regel:

```
<fmt:parseDate value="${nu}" pattern="yyyy-MM-dd" var="nuAlsDate"
type="date"/>
```

1

- (1) `parseDate` ontleedt de `LocalDateTime` variabele als een String bestaande uit een jaar, een maand en een dag en maakt op basis hiervan een `Date` variabele.

Je wijzigt : `<fmt:formatDate value='${nu}' />`

naar `<fmt:formatDate value='${nuAlsDate}' />`

## 27.4 Getalopmaak

Je toont het aantal pizza's dat Luigi maakte sinds hij zijn zaak startte: 23000

Je tikt in de `IndexServlet` method `doGet`:

```
request.setAttribute("aantalPizzasVerkocht", 23000);
```

Je toont dit getal in `index.jsp`, opge maakt volgens het land van de gebruiker met de JSTL tag `formatNumber`.

Je voegt opdrachten toe binnen de `<dl> ... </dl>` structuur:

```
<dt>Aantal pizza's verkocht</dt>
<dd><fmt:formatNumber value='${aantalPizzasVerkocht}'/></dd>
```

Je kunt de website uitproberen en via de browser instellingen een gebruiker uit België simuleren en daarna een gebruiker uit de USA.

Als je de instelling wijzigt, zie je de nieuwe getalopmaak met een pagina 'refresh'.

Je verfijnt de opmaak met de volgende `formatNumber` attributen:

- `groupingUsed`
  - `true` Scheidingsteken gebruiken tussen duizendtallen.
  - `false` Geen scheidingsteken gebruiken tussen duizendtallen.
- `minFractionDigits`  
Het minimum aantal cijfers na de komma.  
Als het getal minder cijfers na de komma heeft, voegt `formatNumber` nullen toe.  
Voorbeeld: `<fmt:formatNumber value='666.7' minFractionDigits='2' />`  
toont aan een gebruiker uit België 666,70
- `maxFractionDigits`  
Het maximum aantal cijfers na de komma.  
Als het getal meer cijfers na de komma heeft, rondt `formatNumber` het getal af.  
Voorbeeld: `<fmt:formatNumber value='666.749' maxFractionDigits='2' />`  
toont aan een gebruiker uit België 666,75

## 27.5 Resource bundles

Je toont teksten in een JSP in meerdere talen.

Je leest die teksten uit resource bundles.

Één resource bundle bevat teksten in één taal. Één regel is één tekst.

De regel bestaat uit een unieke sleutel, gevolgd door `=`, gevolgd door de tekst zelf.

De tekst kan parameters bevatten. Je tikt een parameter als `{volgNrVanDeParameter}`.

De volgnummers beginnen per tekst vanaf 0. Voorbeeld:

```
jeNaamBestaatUitLetters=Je naam bestaat uit {0} letters
```

(1) De tekst bevat één parameter, aangegeven met `{0}`.

①

Je maakt per taal in de website een resource bundle.

Je geeft eenzelfde tekst dezelfde sleutel over de resource bundles heen.

Je geeft de taal die bij de resource bundle hoort aan in de bestandsnaam:

Je eindigt de bestandsnaam met een `_`, gevolgd door de taalcode (teksten\_n1)

Men spreekt sommige talen in meerdere landen. Je kan de tekst per land vertalen in een resource bundle specifiek voor dat land. Je geeft dit land aan in de bestandsnaam (teksten\_n1\_BE)

Als je een tekst ophaalt in JSP, doorzoekt JSP de resource bundles met deze volgorde:

- De resource bundle met de gebruikerstaal- én landcode (teksten\_n1\_BE).
- Als de tekst daarin niet voorkomt, de resource bundle met enkel de taalcode (teksten\_n1).
- Als de tekst daarin niet voorkomt, de resource bundle zonder taalcode (teksten).



Je kan meerdere taal-land combinaties instellen in je browser. JSP gebruikt in dit zoekproces *alle* taal-land combinaties in de request header Accept-Language. Als deze header bijvoorbeeld en én nl bevat, zoekt JSP een tekst in teksten\_en, daarna in teksten\_nl en daarna in teksten.

### 27.5.1 Eclipse

Je plaatst de resource bundles in een folder resourceBundles:

1. Je klikt met de rechtermuisknop op src (onderdeel van Java Resources).
2. Je kiest New, Other, General, Folder en Next.
3. Je tikt resourceBundles bij Folder name en je kiest Finish.

Je maakt in resourceBundles een resource bundle teksten.properties:

1. Je klikt met de rechtermuisknop op resourceBundles.
2. Je kiest New, Other, General, File en Next.
3. Je tikt teksten.properties bij File Name en je kiest Finish.

Je maakt in resourceBundles ook een resource bundle teksten\_en.properties:

1. Je klikt met de rechtermuisknop op resourceBundles.
2. Je kiest New, Other, General, File en Next.
3. Je tikt teksten\_en.properties bij File Name en je kiest Finish.

### 27.5.2 NetBeans

Je plaatst de resource bundles in een folder resourceBundles:

1. Je klikt met de rechtermuisknop op Source Packages.
2. Je kiest New, Other, Other, Folder en Next.
3. Je tikt resourceBundles bij Folder Name en je kiest Finish.

Je maakt in resourceBundles een resource bundle teksten.properties:

1. Je klikt met de rechtermuisknop op resourceBundles.
2. Je kiest New, Other, Other, Properties File en Next.
3. Je tikt teksten bij File Name en je kiest Finish.

Je maakt in resourceBundles ook een resource bundle teksten\_en.properties:

1. Je klikt met de rechtermuisknop op teksten.properties.
2. Je kiest Add, Locale.
3. Je tikt en bij Language Code en je kiest OK.

### 27.5.3 Inhoud van de resource bundles

teksten.properties:	identificatie=Identificatie naam=Naam onthoudMe=Onthoud me naamLetters=Je naam bestaat uit {0} letters
teksten_en.properties:	identificatie=Identification naam=Name onthoudMe=Remember me naamLetters=Your name consists of {0} letters

### 27.5.4 Resource bundles gebruiken in een JSP

Je wijzigt identificatie.jsp:

```
<%@page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<%@taglib prefix='fmt' uri='http://java.sun.com/jsp/jstl/fmt'%>
<fmt:setBundle basename='resourceBundles.teksten'/>
<!doctype html>
<html lang='nl'>
  <head>
    <c:import url='/WEB-INF/JSP/head.jsp'>
      <c:param name='title' value='Identificatie'/>
    </c:import>
```

①  
②

```

</head>
<body>
  <c:import url='/WEB-INF/JSP/menu.jsp' />
  <h1><fmt:message key='identificatie' /></h1>
  <form method='post'>
    <label><fmt:message key='naam' />
    <input name='gebruikersnaam' value='${gebruikersnaam}'
      autofocus required></label>
    <input type='submit' value="<fmt:message key='onthoudMe' />" />
  </form>
  <c:if test='${not empty naam}'>
    <div>
      <fmt:message key='naamLetters'>
        <fmt:param value='${naam.length()}' />
      </fmt:message>
    </div>
  </c:if>
</body>
</html>

```

- (1) Je associeert fmt met de JSTL internationalization library URI.
- (2) Je definieert met setBundle de directory en de base name van de resource bundles die je in de JSP gebruikt. Je gebruikt een punt als scheidingsteken tussen directory en bestand. Je laat de extensie .properties weg.  
De base name is het deel van de bestandsnaam zonder land- of taalcode.
- (3) Je leest de tekst met de sleutel identificatie uit de resource bundles.
- (4) Je geeft de parameter in de tekst met de sleutel naamLetters een waarde met param. Je gebruikt meerdere param tags bij een tekst met meerdere parameters.

Je kunt de website uitproberen en via de browser instellingen een Engelstalige gebruiker simuleren en daarna een Nederlandstalige gebruiker.

Als je de instelling wijzigt, zie je de nieuwe teksten met een pagina 'refresh'

## 27.6 Taal en land vragen met hyperlinks

Je bepaalt tot nu de gebruikerstaal en -land met de Accept-Language request header.

Je leert hier een andere manier: de gebruiker kiest met hyperlinks zijn taal en land.

Je tikt in identificatie.jsp voor </body>

```

<div>
  <c:url value='' var='nlBEURL'>
    <c:param name='locale' value='nl-BE' />
  </c:url>
  <c:url value='' var='enUSURL'>
    <c:param name='locale' value='en-US' />
  </c:url>
  <a href='${nlBEURL}'>Ik spreek Nederlands en woon in België</a>
  <a href='${enUSURL}'>I speak English and live in the USA</a>
</div>

```

- (1) Je geeft een lege value mee.  
<c:url ...> maakt dan een URL die gelijk is aan de URL van de huidige request.

Je onthoudt de keuze in een session attribuut in de IdentificatieServlet method doGet:

```

String locale = request.getParameter("locale");
if (locale != null) {
  request.getSession().setAttribute("locale", locale);
}

```

Je verwijdert in identificatie.jsp in de regel <%@page ... /> het attribuut session="false", omdat je het session attribuut locale leest.



Je tikt in dezelfde JSP voor `<fmt:setBundle .../>`

```
<c:if test='${not empty sessionScope.locale}'>  
  <fmt:setLocale value='${sessionScope.locale}'/>  
</c:if>
```

①

(1) Je stelt met `setLocale` de locale in voor `formatDate`, `formatNumber` en `message`

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.



Vertalen: zie takenbundel

## 28 CUSTOM TAGS

Je gebruikt al tags uit de JSTL tag libraries. Je maakt in dit hoofdstuk eigen (custom) tags. Je hebt in de cursus al herbruikbare code gecentraliseerd met de JSTL tag import. Een custom tag kan hetzelfde, maar vereist minder tikwerk bij het oproepen en heeft meer mogelijkheden naar parameters toe.

Je kunt een custom tag op twee manieren maken:

- Als een bestand met de extensie tag.  
Een tag bestand heeft dezelfde syntax en mogelijkheden als in een JSP.  
Je plaatst een tag bestand in de folder tags in WEB-INF, of in een subfolder van tags.  
Het is een goede gewoonte per tag library een subfolder te maken in de folder tags en alle tags van die library in die folder te plaatsen.
- Als een Java class.  
Je doet dit enkel als je custom tag vooral code bevat en weinig HTML naar de browser stuurt.  
Dit soort custom tags valt buiten het bereik van de cursus.

### 28.1 TLD bestand

Je verzamelt custom tags die een samenhangend geheel vormen in een tag library.

Je maakt per tag library één TLD (Tag Library Descriptor) bestand.

Dit XML bestand bevat volgende informatie:

- De naam en de locatie van elke tag uit de tag library.
- De URI waarmee je de tag library associeert  
De conventie is dat de URI van je firma een onderdeel is van de URI.  
Op die manier heeft elke tag library op de wereld een unieke URI.
- De voorkeur prefix (bv vdab) die je gebruikt als je in een JSP naar de tag library verwijst.

Je plaatst een TLD in de folder WEB-INF, of in een subfolder van WEB-INF.

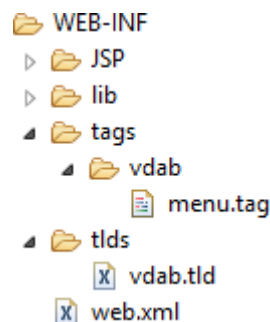
### 28.2 Eclipse

Je maakt een custom tag:

1. Je maakt in WEB-INF een folder tags.
2. Je maakt in tags een folder vdab.
3. Je klikt met de rechtermuisknop op vdab.
4. Je kiest New, Other, Web, JSP Tag en Next.
5. Je tikt menu bij File Name en je kiest Finish.

Je maakt een TLD:

1. Je maakt in WEB-INF een folder tlds.
2. Je klikt met de rechtermuisknop op tlds.
3. Je kiest New, Other, XML, XML File en Next.
4. Je tikt vdab.tld bij File Name en je kiest Finish.



### 28.3 NetBeans

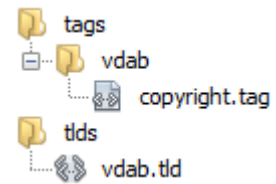
Je maakt een TLD:

1. Je klikt met de rechtermuisknop op het project.
2. Je kiest New, Other, Web, Tag Library Descriptor en Next.
3. Je tikt vdab bij TLD Name.
4. Je tikt http://vdab.be/tags bij URI.
5. Je tikt vdab bij Prefix en je kiest Finish.

Je maakt een custom tag.

NetBeans helpt je deze custom tag te registreren in `vdab.tld`.

1. Je klikt met de rechtermuisknop op het project.
2. Je kiest New, Other, Web, Tag File en Next.
3. Je tikt copyright bij Tag File Name.
4. Je wijzigt Folder naar tags/vdab.
5. Je vinkt Add Tag File to Tag Library Descriptor aan.
6. Je duidt met de Browse knop naast TLD File `vdab.tld` aan.
7. Je kiest Finish.



## 28.4 vdab.tld

```
<?xml version='1.0' encoding='UTF-8'?>
<taglib version='2.1' xmlns='http://java.sun.com/xml/ns/javaee'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd'>
  <tlib-version>1.0</tlib-version>
  <short-name>vdab</short-name>
  <uri>http://vdab.be/tags</uri>
  <tag-file>
    <name>menu</name>
    <path>/WEB-INF/tags/vdab/menu.tag</path>
  </tag-file>
</taglib>
```

①  
②  
③  
④  
⑤  
⑥

- (1) Je definieert met `tlib-version` een versienummer van de tag library.
- (2) Je definieert met `short-name` de voorkeur prefix voor de tag library.
- (3) Je definieert met `uri` de URI van de tag library.
- (4) Je definieert per tag die behoort tot de tag library een element `tag-file`.
- (5) Je definieert met `name` de tag naam.
- (6) Je definieert met `path` de plaats en de naam van het tag bestand.

## 28.5 menu.tag

Je wijzigt `menu.tag`:

```
<%@tag description='menu' pageEncoding='UTF-8'%>
```

①

- (1) Een custom tag bevat een page directive tag.  
Je tikt bij `description` een optionele omschrijving van wat de tag doet.  
De IDE toont die omschrijving als je de tag gebruikt in een JSP.

Je kopieert onder deze regel alle regels uit `menu.jsp`, behalve de eerste regel.

Je verwijdert `menu.jsp`.

## 28.6 Custom tag gebruiken in een JSP

Je tikt in elke JSP, behalve `head.jsp`, onder `<%@page .../>`

```
<%@taglib uri='http://vdab.be/tags' prefix='vdab'%>
```

Je vervangt in elke JSP, behalve `head.jsp`, `<c:import url='/WEB-INF/JSP/menu.jsp' />` door `<vdab:menu/>`

Je kunt de website uitproberen.

## 28.7 Custom tag attributen

Je kunt een custom tag voorzien van één of meerdere attributen (parameters).  
Als je de custom tag oproept in een JSP, geef je waarden mee voor de attributen.

Je kunt per attribuut volgende eigenschappen definiëren

- name. Deze eigenschap is verplicht.
- description. Deze eigenschap is optioneel.
- required. Is het verplicht dit attribuut mee te geven als je de tag oproept in een JSP. Default is een attribuut optioneel.
- type. Het datatype van het attribuut. Deze eigenschap is optioneel. Als het attribuut een waarde van een verkeerd type binnenkrijgt, werpt Java een Exception bij het uitvoeren van de website.

Je maakt een custom tag head ter vervanging van head.jsp.

Je registreert de tag in WEB-INF/tlds/vdab.tld:

- Bij NetBeans helpt de wizard, waarmee je de Tag File toevoegt, om de Tag File te registreren (vinkje bij Add Tag File to Tag Library Descriptor)

- Bij Eclipse registreer je de tag in vdab.tld.

Je tikt onder </tag-file>:

```
<tag-file>
  <name>head</name>
  <path>
    /WEB-INF/tags/vdab/head.tag
  </path>
</tag-file>
```

Je maakt head.tag:

```
<%@tag description='head onderdeel van pagina' pageEncoding='UTF-8'%>
<%@attribute name='title' required='true' type='java.lang.String'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<title>${title}</title>
<link rel='icon' href='<c:url value="/images/favicon.ico"/>'>
<meta name='viewport' content='width=device-width,initial-scale=1'>
<link rel='stylesheet' href='<c:url value="/styles/default.css"/>'>
```

1

(1) Je definieert één attribuut met een page directive attribute.

Je tikt bij name de attribuut naam.

Je geeft bij required aan dat dit attribuut verplicht in te vullen is.

Je tikt bij type het attribuut type.

Je verwijdert head.jsp.

Je vervangt in elke JSP <c:import url='/WEB-INF/JSP/head.jsp'>...</c:import>  
door <vdab:head title='Tik hier wat bij value van c:param stond'/>

Je commit de sources en je publiceert op GitHub. Je kunt de website uitproberen.



Menu: zie takenbundel

## 29 OUDERE WEBSERVERS EN OUDERE WEBSITES

Het is pas sedert versie 3 van de servlet specificatie dat je:

- een servlet kan registreren met `@WebServlet`
- een listener kan registreren met `@WebListener`
- een filter kan registreren met `@WebFilter`

Als je

- een website maakt die moet draaien op een oudere webserver die de versie 3 van de servlet specificatie niet implementeert, zoals Tomcat 6
- een oudere website moet onderhouden die gemaakt werd toen versie 3 van de servlet specificatie nog niet bestond

moet je servlets, listeners en filters registreren in `web.xml`, in plaats van met annotations.

Je leert hier hoe je dit kan doen (voor als je het ooit nodig zou hebben).

Je mag servlets, listeners en filters maar op één manier registreren: met annotations of in `web.xml`

### 29.1 Een servlet registreren in web.xml

Je plaatst in `IndexServlet` de regel `@WebServlet` in commentaar.

Je registreert de servlet in `web.xml` in twee stappen, tussen `<web-app>` en `</web-app>`

1. Je associeert een vrij te kiezen servlet-name met de servlet class:

```
<servlet>
  <servlet-name>IndexServlet</servlet-name>
  <servlet-class>be.vdab.servlets.IndexServlet</servlet-class>
</servlet>
```

2. Je associeert die servlet-name met de URL die bij de servlet hoort:

```
<servlet-mapping>
  <servlet-name>IndexServlet</servlet-name>
  <url-pattern>/index.htm</url-pattern>
</servlet-mapping>
```

### 29.2 Een listener registreren in web.xml

Je plaatst in `MandjeListener` de regel `@WebListener` in commentaar.

Je registreert de listener in `web.xml`, tussen `<web-app>` en `</web-app>`:

```
<listener>
  <listener-class>be.vdab.listeners.MandjeListener</listener-class>
</listener>
```

### 29.3 Een filter registreren in web.xml

Je plaatst in `ServletFilter` de regel `@WebFilter` in commentaar.

Je registreert een filter in `web.xml` in twee stappen, tussen `<web-app>` en `</web-app>`

1. Je associeert een vrij te kiezen filter-name met de filter class:

```
<filter>
  <filter-name>ServletFilter</filter-name>
  <filter-class>be.vdab.filters.ServletFilter</filter-class>
</filter>
```

2. Je associeert die filter-name met het URL patroon dat bij de filter hoort:

```
<filter-mapping>
  <filter-name>ServletFilter</filter-name>
  <url-pattern>*.htm</url-pattern>
</filter-mapping>
```

Je commit de sources en je publiceert op GitHub. Je kunt de website terug uitproberen.

## 30 HERHALINGSOEFENINGEN



Gastenboek: zie takenbundel



Gastenboekbeheer: zie takenbundel

## 31 COLOFON

<b>Domeinexpertisemanager:</b>	Jean Smits
<b>Moduleverantwoordelijke:</b>	Hans Desmet
<b>Medewerkers:</b>	Hans Desmet
<b>Versie:</b>	16/8/2017