



CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

SERVLET

Docente: prof. Romano Simone
a.a. 2024-2025

Cos'è una Servlet?

- È una classe Java particolare
- In particolare, è una classe che fornisce un servizio comunicando con il client mediante protocolli di tipo ***request/response***: tra questi protocolli il più noto e diffuso è HTTP
 - Le Servlet estendono le funzionalità di un Web Server generando contenuti dinamici
 - Sono eseguite direttamente in un Web Container (***Application Server o più precisamente Servlet Container***)
- In termini pratici sono classi che derivano dalla classe **HttpServlet**
 - HttpServlet implementa vari metodi che possiamo ridefinire

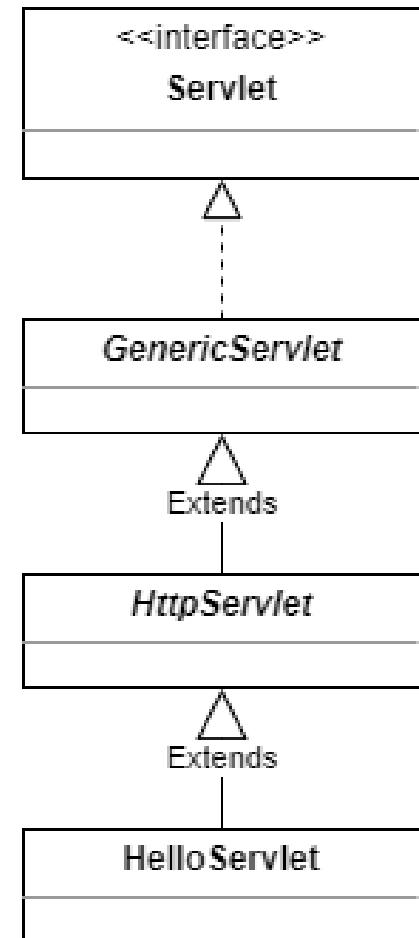
Esempio di Servlet: Hello world!

- Ridefiniamo **doGet()** e implementiamo la logica di risposta a HTTP GET
- Produciamo in output un testo HTML che costituisce la pagina restituita dal server HTTP:

```
@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<title>Hello World!</title>");
    }
    ...
}
```

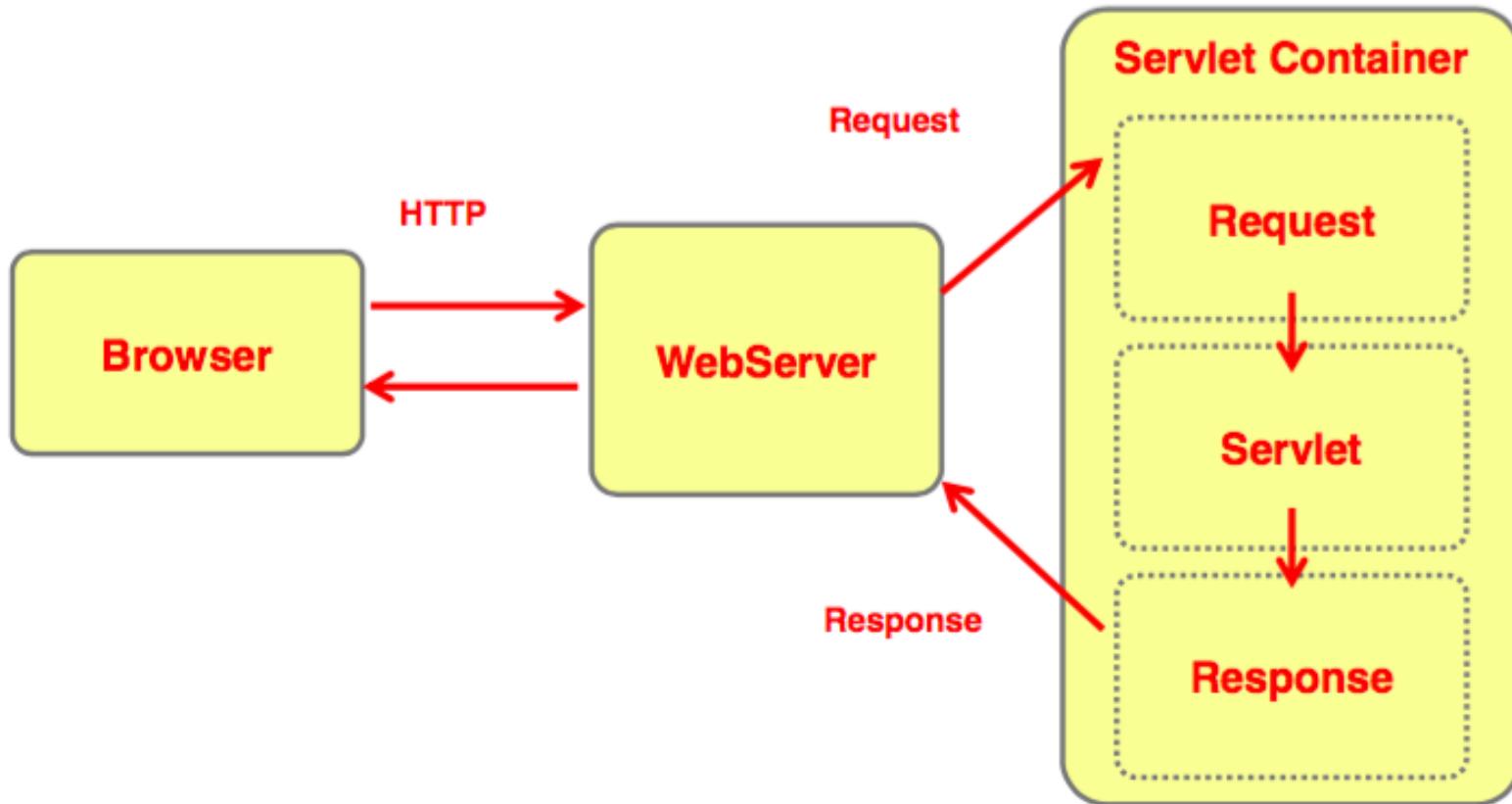
Gerarchie delle Servlet

- Le Servlet sono classi Java che elaborano richieste seguendo un protocollo condiviso
- Le Servlet HTTP sono il tipo più comune di Servlet e possono processare request HTTP, producendo response HTTP
- Abbiamo quindi la catena di ereditarietà mostrata a lato
- Nel seguito ragioneremo sempre e solo su Servlet HTTP
- Le classi che ci interessano sono contenute nel package
jakarta.servlet.http.*



The request-response model

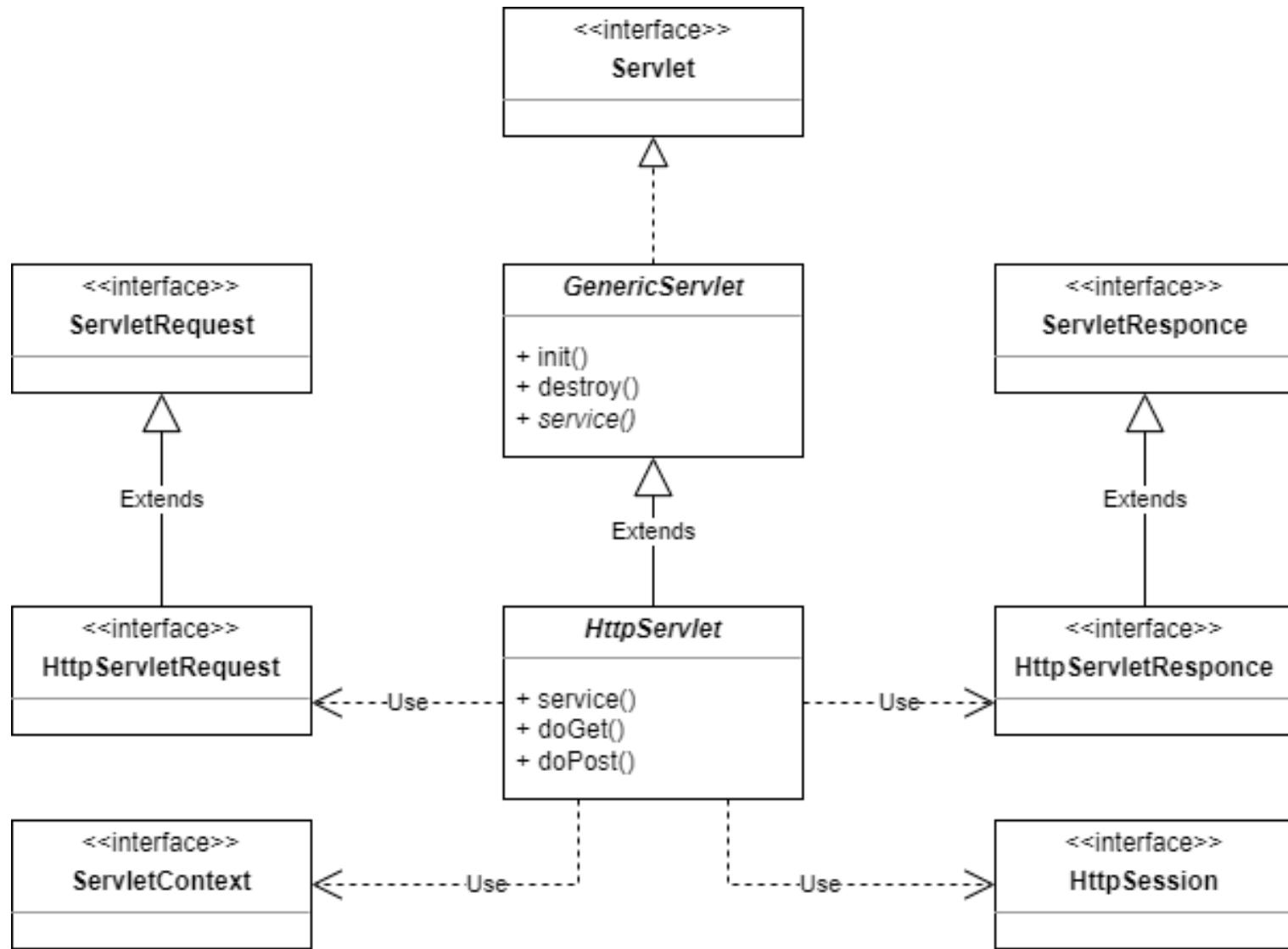
- All'arrivo di una richiesta HTTP il Servlet Container (Web Container) crea un oggetto **request** e un oggetto **response** e li passa alla Servlet:



Request e Response

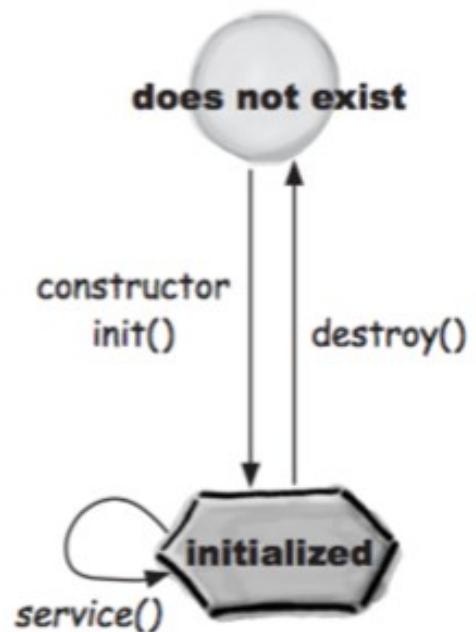
- Gli oggetti di tipo **Request rappresentano la chiamata al Server effettuata dal Client**
- Sono caratterizzati da varie informazioni
 - Chi ha effettuato la Request
 - Quali parametri sono stati passati nella Request
 - Quali header sono stati passati
- Gli oggetti di tipo **Response rappresentano le informazioni restituite al client in risposta ad una Request**
 - Dati in forma testuale (es. html, text) o binaria (es. immagini)
 - HTTP header, cookie, ...

Classi e interfacce per Servlet



Ciclo di vita delle Servlet

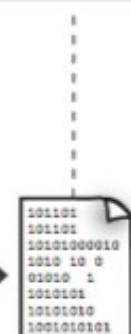
- Il Servlet Container controlla e supporta automaticamente il ciclo di vita di una Servlet
- Esiste un solo stato principale: **inizialized**
- Se la Servlet non è inizialized, può essere:
 - being initialized** (eseguendo il suo costruttore e il suo metodo **init()**)
 - being destroyed** (eseguendo il suo metodo **destroy()**)
 - o semplicemente non esiste



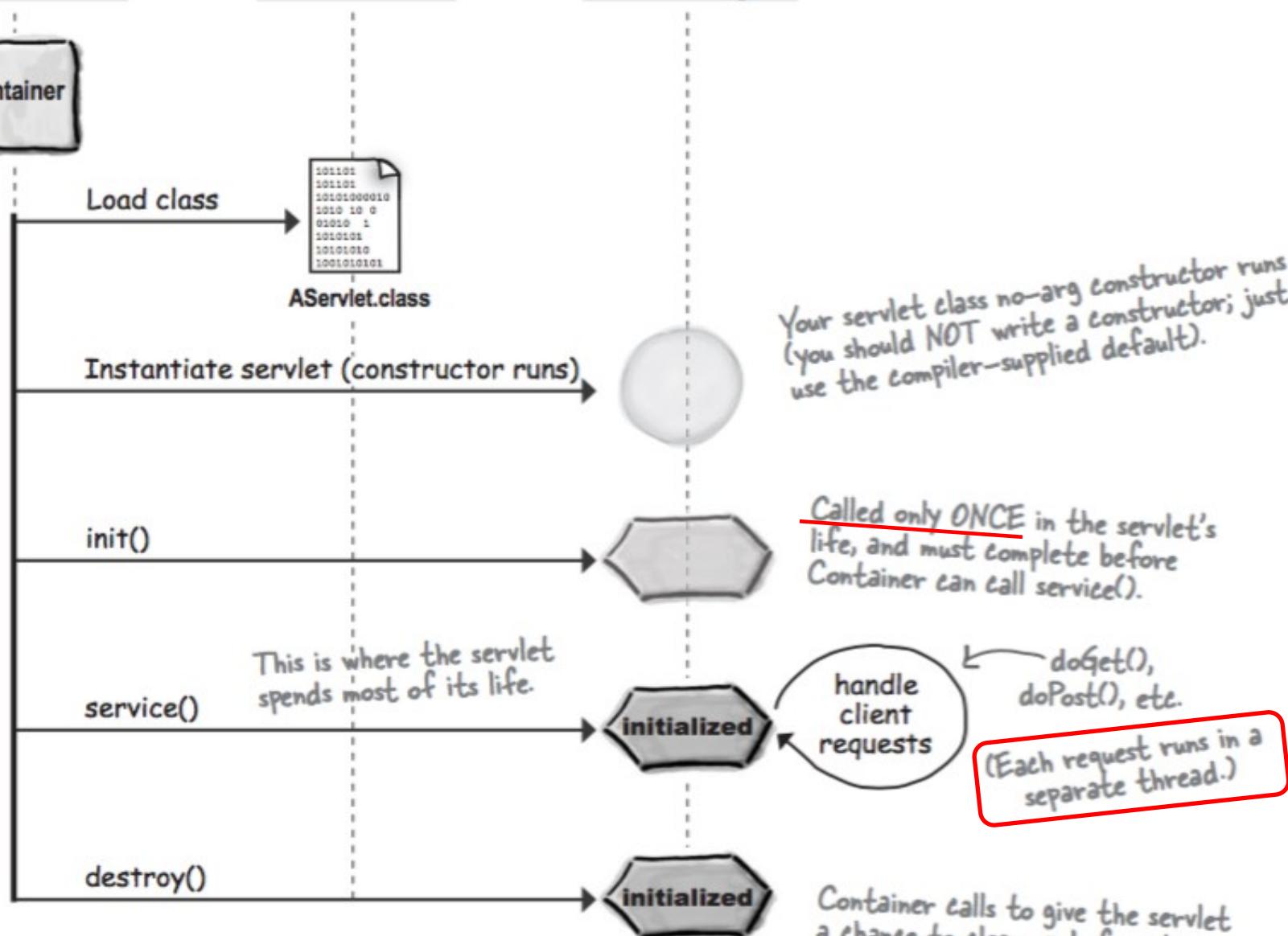
Web Container



Servlet Class



Servlet Object



Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

Three Big lifecycle moments

1

init()

When it's called

The Container calls init() on the servlet instance *after* the servlet instance is created but *before* the servlet can service any client requests.

What it's for

Gives you a chance to initialize your servlet before handling any client requests.

Do you override it?

Possibly.

If you have initialization code (like getting a database connection or registering yourself with other objects), then you'll override the init() method in your servlet class.

2

service()

When it's called

When the first client request comes in, the Container starts a new thread or allocates a thread from the pool, and causes the servlet's service() method to be invoked.

What it's for

This method looks at the request, determines the HTTP method (GET, POST, etc.) and invokes the matching doGet(), doPost(), etc. on the servlet.

Do you override it?

No. Very unlikely.

You should NOT override the service() method. Your job is to override the doGet() and/or doPost() methods and let the service() implementation from HttpServlet worry about calling the right one.

Three Big lifecycle Moments (2)

3
doGet()
and/or
doPost()

When it's called

The service() method invokes doGet() or doPost() based on the HTTP method (GET, POST, etc.) from the request.

(We're including only doGet() and doPost() here, because those two are probably the only ones you'll ever use.)

What it's for

This is where *your* code begins! This is the method that's responsible for whatever the heck your web app is supposed to be DOING.

You can call other methods on other objects, of course, but it all starts from here.

Do you override it?

ALWAYS at least ONE of them! (doGet() or doPost())

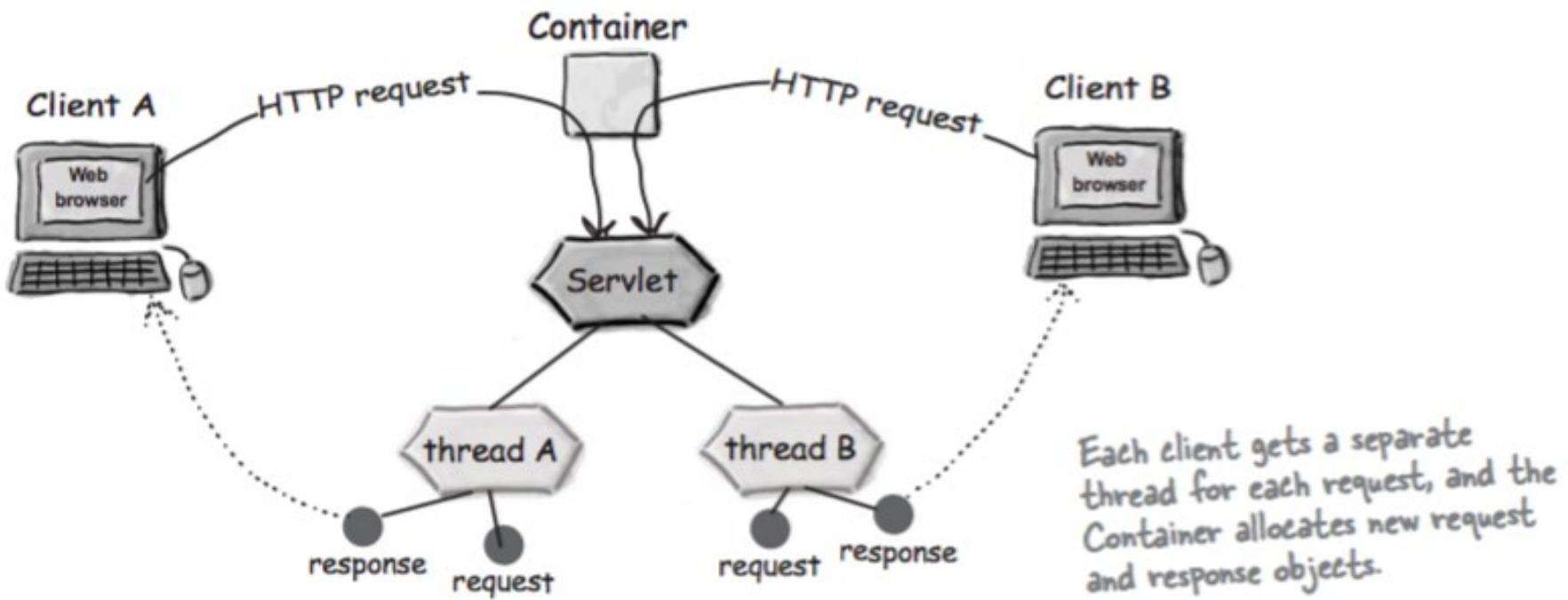
Whichever one(s) you override tells the Container what you support. If you don't override doPost(), for example, then you're telling the Container that this servlet does not support HTTP POST requests.

Servlet & Multithreading

- **Modello “normale”:** una sola istanza di Servlet e un thread assegnato ad ogni richiesta HTTP per la Servlet
- Nella modalità normale **più thread condividono** la **stessa istanza** di una Servlet e quindi si crea una situazione di concorrenza:
 - Il metodo **init()** della Servlet viene chiamato una sola volta quando la Servlet è caricata dal Web Container
 - I metodi **service()** e **destroy()** possono essere chiamati solo dopo il completamento dell'esecuzione di init()
 - Il metodo **service()** (e quindi **doGet()** e **doPost()**) può essere invocato (indirettamente) da numerosi client in modo concorrente ed è quindi necessario gestire le sezioni critiche (a completo carico del programmatore dell'applicazione Web):
 - Ad esempio, usando i blocchi synchronized

Ogni richiesta in un thread separato

- The Container runs multiple *threads* to process multiple requests to a single Servlet
 - and every client request generates a new pair of request and response objects



Modello single-threaded (*deprecated*)

- Alternativamente si può indicare al Container di creare un'istanza della Servlet per ogni richiesta concorrente
 - Questa modalità prende il nome di Single-Threaded Model
 - È onerosa in termine di risorse ed è deprecata nelle specifiche 2.4 delle Servlet
 - Se una Servlet vuole operare in modo single-threaded deve implementare l'interfaccia

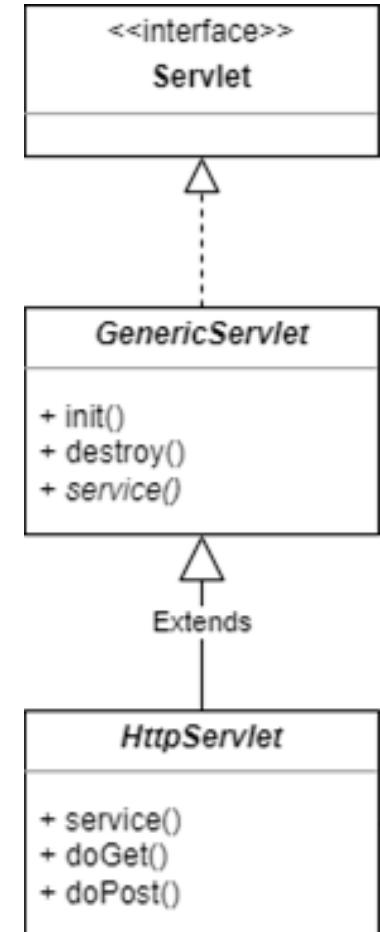
SingleThreadModel

Ricapitolando: Metodi per il controllo del ciclo di vita

- **init()**: viene chiamato una sola volta al caricamento della Servlet
 - In questo metodo si può inizializzare l'istanza: ad esempio si crea la connessione con un database
- **service()**: viene chiamato ad ogni HTTP Request
 - Chiama **doGet()** o **doPost()** a seconda del tipo di HTTP Request ricevuta
- **destroy()**: viene chiamato una sola volta quando la Servlet deve essere disattivata (es. quando è rimossa)
 - Tipicamente serve per rilasciare le risorse acquisite (es. connessione a db, eliminazione di variabili di stato per l'intera applicazione, ...)

Metodi per il controllo del ciclo di vita (2)

- I metodi **init()**, **destroy()** e **service()** sono definiti nella classe astratta **GenericServlet**
- **service()** è un metodo astratto
- **HttpServlet** fornisce una implementazione di **service()** che delega l'elaborazione della richiesta ai metodi:
 - **doGet()**
 - **doPost()**
 - **doPut()**
 - **doDelete()**



Anatomia di Hello World basata su tecnologia Servlet

- Usiamo l'esempio “Hello World” per affrontare i vari aspetti della realizzazione di una Servlet:
 1. Importiamo i package necessari
 2. Definiamo la classe **HelloServlet** che discende da **HttpServlet**
 3. Ridefiniamo il metodo **doGet()**

```
public class HelloServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    {
        ...
    }
}
```

Hello World: definizione di `doGet`

- Dobbiamo tener conto che in `doGet()` possono essere sollevate **eccezioni** di due tipi:
 - quelle specifiche delle Servlet
 - quelle legate all'input/output
- Decidiamo di non gestirle per semplicità e quindi ricorriamo alla clausola **throws**
- Se non ci servono informazioni sulla richiesta allora non usiamo il parametro **request**
- Dobbiamo semplicemente costruire la risposta e quindi usiamo il solo parametro **response**

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

ServletResponse interface

(javax.servlet.ServletResponse)

response

- The response is what goes back to the client
- The thing the browser gets, parses, and renders for the user
- Typically, you use the response object to get an output stream (usually a Writer) and you use that stream to write the HTML (or some other type of content) that goes back to the client
- The response object has other methods besides just the I/O output

<<interface>>
ServletResponse

getBufferSize()
setContentType()
getOutputStream()
getWriter()
setContentLength()
// MANY more methods...

HttpServletResponse interface

(javax.servlet.http.HttpServletResponse)

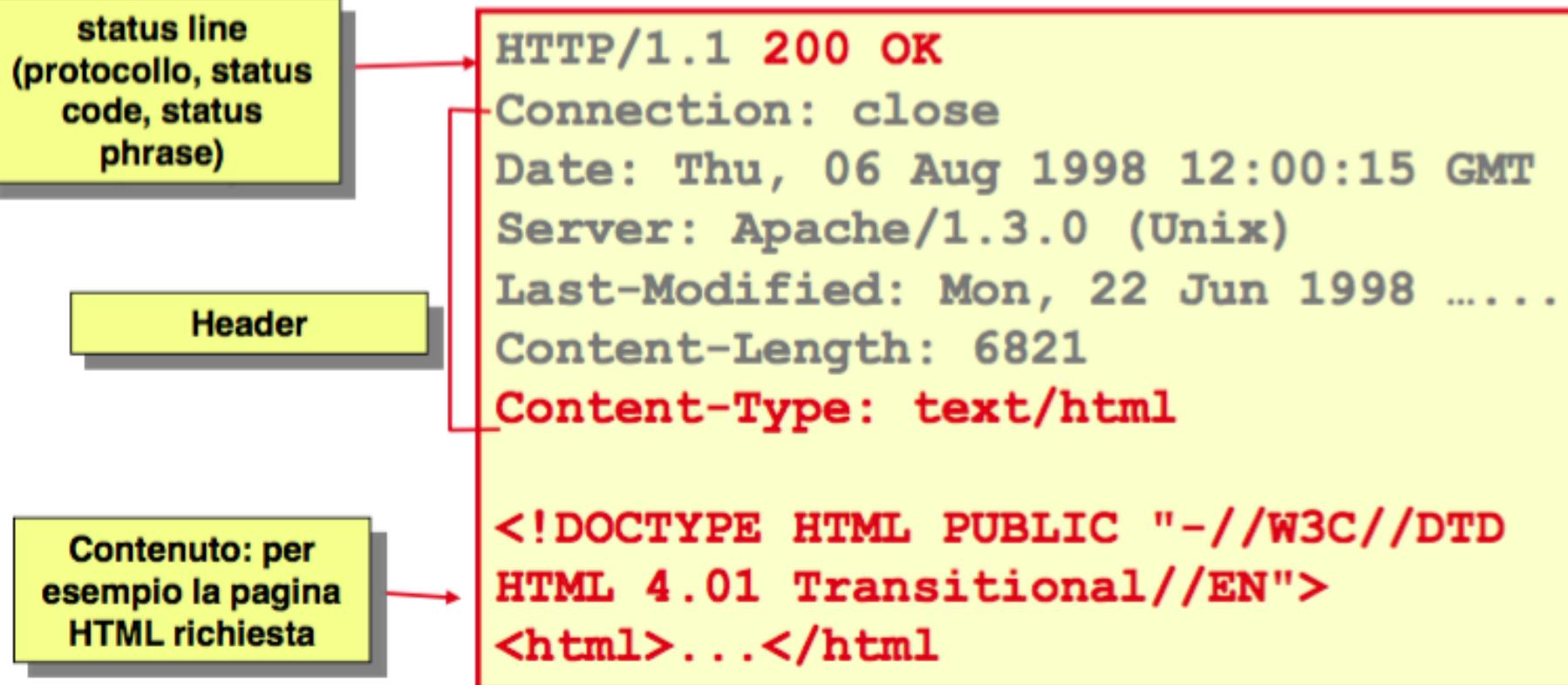
<<interface>>
HttpServletResponse

addCookie()
addHeader()
encodeURL()
sendError()
setStatus()
sendRedirect()
// MANY more methods...

L'oggetto response

- Contiene i dati restituiti dalla Servlet al Client:
 - **Status line** (status code, status phrase)
 - **Header** della risposta HTTP
 - **Response body**: il contenuto (ad es. pagina HTML)
- Ha come tipo l'interfaccia **HttpServletResponse** che espone metodi per:
 - Specificare lo status code della risposta HTTP
 - Indicare il **content type** (tipicamente **text/html**)
 - Ottenerne un **output stream** in cui scrivere il contenuto da restituire
 - ...

Formato della risposta HTTP



In rosso ciò che può/deve essere specificato a livello di codice della servlet

Gestione dello status code

- Per definire lo status code HttpServletResponse fornisce il metodo

public void setStatus(int statusCode)

- This method is used to set the return status code when there is no error (e.g., 200 OK)

- Esempi di status Code

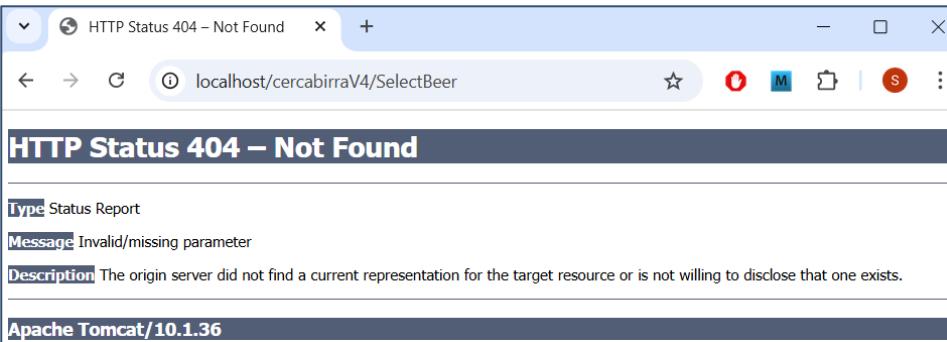
- **200 OK**
- **404 File not found**
- **...**

- Per inviare errori possiamo anche usare:

public void sendError (int statusCode)

public void sendError(int statusCode, String message)

Example (cercabirraV4.zip)



SelectBeer.java

```
@WebServlet("/SelectBeer")
public class SelectBeer extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String color = request.getParameter("color");
        BeerExpert expert = new BeerExpert();
        if (expert.isValid(color)) {
            List<String> beers = expert.getBrands(color);
            request.setAttribute("beers", beers);
            RequestDispatcher dispatcher = request.getRequestDispatcher("result.jsp");
            dispatcher.forward(request, response);
        } else {
            response.sendError(HttpServletResponse.SC_NOT_FOUND, "Invalid/missing parameter");
        }
    }
}
```

```
public class BeerExpert {

    private List<String> colors = new ArrayList<String>();

    public BeerExpert() {
        colors.add("light");
        colors.add("brown");
        colors.add("amber");
        colors.add("dark");
    }

    public List<String> getBrands(String color) {
        List<String> brands = new ArrayList<String>();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        } else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
        return (brands);
    }

    public List<String> getColors() {
        return colors;
    }

    public boolean isValid(String color) {
        return colors.contains(color);
    }
}
```

Gestione degli header HTTP

- **public void setHeader(String headerName, String headerValue)** imposta un header arbitrario
- **public void setDateHeader(String name, long millis)** imposta la data
- **public void setIntHeader(String name, int headerValue)** imposta un header con un valore intero (evita la conversione intero-stringa)
- **addHeader, addDateHeader, addIntHeader** aggiungono una nuova occorrenza di un dato header

```
response.addDateHeader("Date", System.currentTimeMillis());
response.setHeader("Pragma", "no-cache");
response.addIntHeader("Expires", 0);
```

- **setContentType** configura il content-type (*si usa quasi sempre*)
- **addCookie** consente di gestire i cookie nella risposta
- **sendRedirect** imposta location header e cambia lo status code in modo da forzare una ridirezione

Gestione del contenuto

- Per definire il response body possiamo operare in due modi utilizzando due metodi di **response**:
 - **public PrintWriter getWriter()**: mette a disposizione uno stream di caratteri (un'istanza di **PrintWriter**)
 - utile per restituire un testo nella risposta (tipicamente HTML)
- **public ServletOutputStream getOuputStream()**: mette a disposizione uno stream di byte (un'istanza di **ServletOutputStream**)
 - più utile per una risposta con contenuto binario (per esempio un'immagine)

Output choices (*when you do not use JSP*)

► PrintWriter

Example:

```
PrintWriter writer = response.getWriter();  
  
writer.println("some text and HTML");
```

Use it for:

Printing text data to a character stream. Although you *can* still write character data to an OutputStream, this is the stream that's designed to handle character data.

► OutputStream

Example

```
ServletOutputStream out = response.getOutputStream();  
  
out.write(aByteArray);
```

byte[] aByteArray;

Use it for:

Writing *anything else!*

Hello World: implementazione di `doGet`

- Adesso abbiamo tutti gli elementi per implementare correttamente il metodo `doGet()` di `HelloServlet`:

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

Risposta generata

HTTP/1.1 200 OK
Content-Type: text/html
<html>
<head><title>Hello</title></head>
<body>Hello World!</body>
</html>

Hello World: definizione e implementazione di *doPost*

- Servlet can implement a **doPost** method that simply calls doGet
 - It handles both GET and POST requests
 - This approach is a good standard practice if you want HTML interfaces to have some flexibility in how they send data to the Servlet
 - *Attention!!! These methods are inherently different*

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

request

- **request** contiene i dati inviati dal client HTTP al server
- Viene creata dal Servlet container e passata alla Servlet come parametro ai metodi `doGet()` e `doPost()`
- È un'istanza di una classe che implementa l'interfaccia
HttpServletRequest
- Fornisce metodi per accedere a varie informazioni HTTP Request
 - URL
 - HTTP Request header
 - Tipo di autenticazione e informazioni su utente
 - Cookie
 - Session (lo vedremo nel dettaglio in seguito)
 - ...

Struttura di una request HTTP

Request line
contiene i comandi
(**GET, POST...**),
l'URL e la versione
di protocollo

**Header
lines**

```
GET /search?q=Introduction+to+XML HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept: text/html, image/gif
Accept-Language: en-us, en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com/
```

Get from the request

- The client's platform and browser info

```
String client = request.getHeader("User-Agent");
```

- The cookies associated with this request

```
Cookie[] cookies = request.getCookies();
```

- The session associated with this client

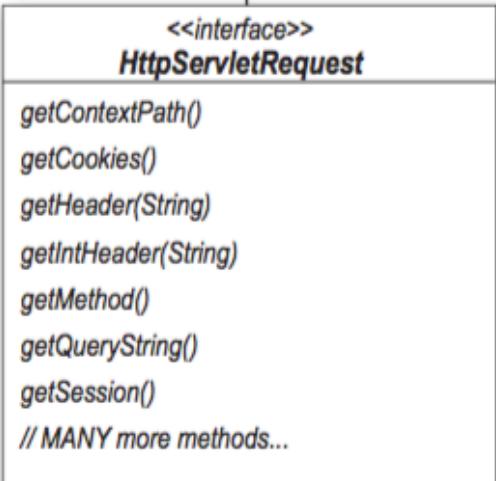
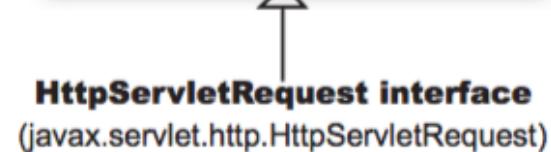
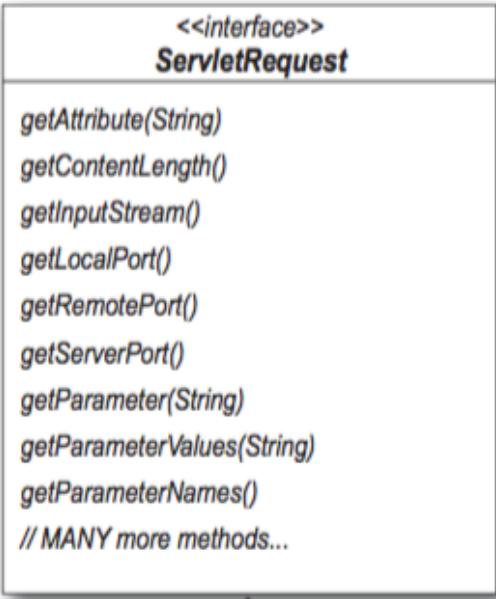
```
HttpSession session = request.getSession();
```

- The HTTP Method of the request

```
String theMethod = request.getMethod();
```

- An input stream from the request

```
InputStream input = request.getInputStream();
```



Request URL

- Una URL HTTP ha la sintassi

http://[host]:[port]/[request path]?[query string]

- La **request path** è il percorso alla risorsa richiesta
- La **query string** è composta da un insieme di parametri che sono forniti dall'utente
- Non solo da compilazione form; può apparire in una pagina Web in un anchor:

Add To Cart

- Il metodo **getParameter()** di **request** ci permette di accedere ai vari parametri
 - Es: **String bookId = request.getParameter("add");** -> **bookID** varrà "101"

Metodi per accedere all'URL

- **String getParameter(String parName)**
 - restituisce il (primo) valore di un parametro individuato per nome
- **Enumeration getParameterNames()**
 - restituisce tutti i nomi dei parametri sotto forma di enumerazione di stringhe
- **String[] getParameterValues(String parName)**
 - restituisce tutti i valori che un determinato parametro ha assegnato
- **String getContextPath()**
 - restituisce informazioni sulla parte dell'URL che indica il contesto della Web application
- **String getQueryString()**
 - restituisce la query string
- **String getRequestURI()**
 - per ottenere l'URI della richiesta fino alla query string (non inclusa)

Metodi per accedere all'Header

- **String getHeader(String name)**
 - restituisce il (primo) valore di un header individuato per nome sotto forma di stringa
- **Enumeration getHeaders(String name)**
 - restituisce tutti i valori dell'header individuato da name sotto forma di enumerazione di stringhe (utile ad esempio per Accept che ammette n valori)
- **Enumeration getHeaderNames()**
 - elenco dei nomi di tutti gli header presenti nella richiesta
- **int getIntHeader(name)**
 - valore di un header convertito in intero
- **long getDateHeader(name)**
 - valore della parte Date di header, convertito in long

Headers (HeaderServlet.zip)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    Enumeration<String> headers = request.getHeaderNames();
    while (headers.hasMoreElements()) {
        String name = headers.nextElement();
        System.out.print(name + ": ");
        Enumeration<String> values = request.getHeaders(name);
        while (values.hasMoreElements()) {
            String value = values.nextElement();
            System.out.println(value);
        }
    }
}
```



A screenshot of an Eclipse IDE terminal window. The title bar shows "Problems", "Servers", "Terminal", "Data Source Explorer", "Properties", "Console", "Git Staging", and "History". Below the title bar, the status bar displays "Tomcat v10.1 Server at localhost [Apache Tomcat] C:\Program Files\Eclipse_2024-12\plugins\org.eclipse.justj.openjdkhotspot\jre\full\win32\x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe (25 mar 2025, 15:16:15 elapsed: 0:03:29) [pid: 16948]". The main terminal area contains the following text:

```
host: localhost
connection: keep-alive
sec-ch-ua: "Chromium";v="134", "Not:A-Brand";v="24", "Google Chrome";v="134"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
sec-fetch-site: none
sec-fetch-mode: navigate
sec-fetch-user: ?1
sec-fetch-dest: document
accept-encoding: gzip, deflate, br, zstd
accept-language: it-IT,it;q=0.9,en;q=0.8
```

Rendiamo Hello World più dinamica...

- Proviamo a complicare leggermente il nostro esempio, avvicinandoci a un esempio di utilità realistica
 - La Servlet non restituisce più un testo fisso ma una pagina in cui un elemento è variabile
 - *Anziché scrivere Hello World scriverà Hello + un nome passato come parametro*
- Ricordiamo che in un URL (e quindi in una GET) possiamo inserire una query string che ci permette di passare parametri con la sintassi:
<path>?<nome1>=<valore1>&<nome2>=<valore2>&...
- Per ricavare il parametro utilizzeremo il parametro **request** passato a **doGet(...)**
- Analizziamo quindi le caratteristiche di **HttpServletRequest**

```
http://.../HelloServlet?to=Mario
```

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

```
HTTP/1.1 200 OK  
Content-Type: text/html  
<html>  
<head><title>Hello</title></head>"  
<body>Hello to Mario!</body>"  
</html>"
```

Hello World: inseriamo un form con metodo post

- Proviamo adesso a creare una pagina html con un form che invii il nome proprio dell’utente a *HelloServlet* tramite una richiesta di tipo post
- Specifichiamo che l’agent per l’elaborazione della form è *HelloServlet* nell’attributo
- Diamo il nome *to* alla text field per l’inserimento del nome
- Ricorda che:
 - Nelle richieste GET, i parametri sono passati nella query string
 - Nella richieste POST i parametri sono inseriti nel body del messaggio HTTP
- Nota che la gestione dei parametri nelle Servlet non cambia tra richieste GET e POST

```
<form action="HelloServlet" method="post">
    Name: <input type="text" name="to"> <input type="submit" value="Submit">
</form>
```

Deployment Descriptor

- Un'applicazione Web deve essere installata, configurata e messa in esercizio su un macchina server, questo processo prende il nome di **deployment**

Il **Deployment Descriptor (DD)** fornisce le informazioni per il deployment dell'applicazione Web, quali:

- La mappatura degli URL sulle Servlet
- La definizione delle pagine di benvenuto e delle pagine di errore
- La configurazione delle caratteristiche di sicurezza dell'applicazione Web
- ...

web.xml

- Il Deployment Descriptor consiste in un file di configurazione in formato XML **web.xml** in **WEB-INF**)
- Può contenere, tra le altre cose, l'elenco delle Servlet e per ognuna di loro permette di definire una serie di parametri come coppie nome-valore
 - Nome
 - Classe Java corrispondente
 - Una serie di parametri di configurazione (coppie nome-valore, valori di inizializzazione)
 - Contiene mappatura fra URL e Servlet che compongono l'applicazione (**IMPORTANTE!**)
- **Dalla versione 3.0 è possibile utilizzare le annotazioni (al posto dei tag XML):**
 - **@WebServlet**
 - **@ServletFilter**
 - **@WebListener**
 - **@WebInitParam**

Mappatura Servlet-URL

- Esempio di descrittore con mappatura:

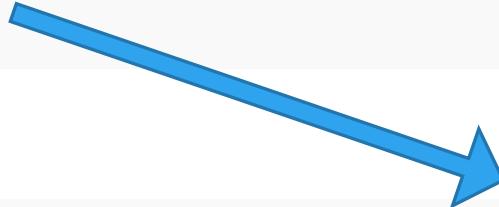
```
<web-app>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>myPackage.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myURL</url-pattern>
  </servlet-mapping>
</web-app>
```

- Esempio di URL che viene mappato su myServlet:

```
http://MyHost:8080/MyWebApplication/myURL
```

Mappatura Servlet-URL (2)

```
<servlet>
    <servlet-name>testServlet</servlet-name>
    <servlet-class>it.unit.servlet.MyTestServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>testServlet</servlet-name>
    <url-pattern>/admin/testServlet</url-pattern>
</servlet-mapping>
```



`http://localhost:8080/MyWebApplication/admin/testServlet`

Pagine di benvenuto e di errore (cercabirraV.zip)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  id="WebApp_ID" version="4.0">
  <display-name>cercabirraV2</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <error-page>
    <error-code>404</error-code>
    <location>/common/error.html</location>
  </error-page>
  <error-page>
    <error-code>500</error-code>
    <location>/common/error.html</location>
  </error-page>
  <error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/error.jsp</location>
  </error-page> Si possono gestire anche le eccezioni!
  <error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/error.jsp</location>
  </error-page>
</web-app>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Errore</title>
  </head>
  <body>
    <h3>Errore!</h3>
  </body>
</html>
```

Pagina di errore generale:

```
<error-page>
  <location>/general-error.html</location>
</error-page>
```

Servlet configuration

- Una Servlet accede ai propri parametri di configurazione mediante l'interfaccia **ServletConfig**
- Ci sono 2 modi per accedere a oggetti di questo tipo:
 1. Il parametro di tipo **ServletConfig** passato al metodo **init()**
 2. il metodo **getServletConfig()** della Servlet, che può essere invocato in qualunque momento
- **ServletConfig** espone un metodo per ottenere il valore di un parametro in base al nome:
 - **String getInitParameter(String parName)**

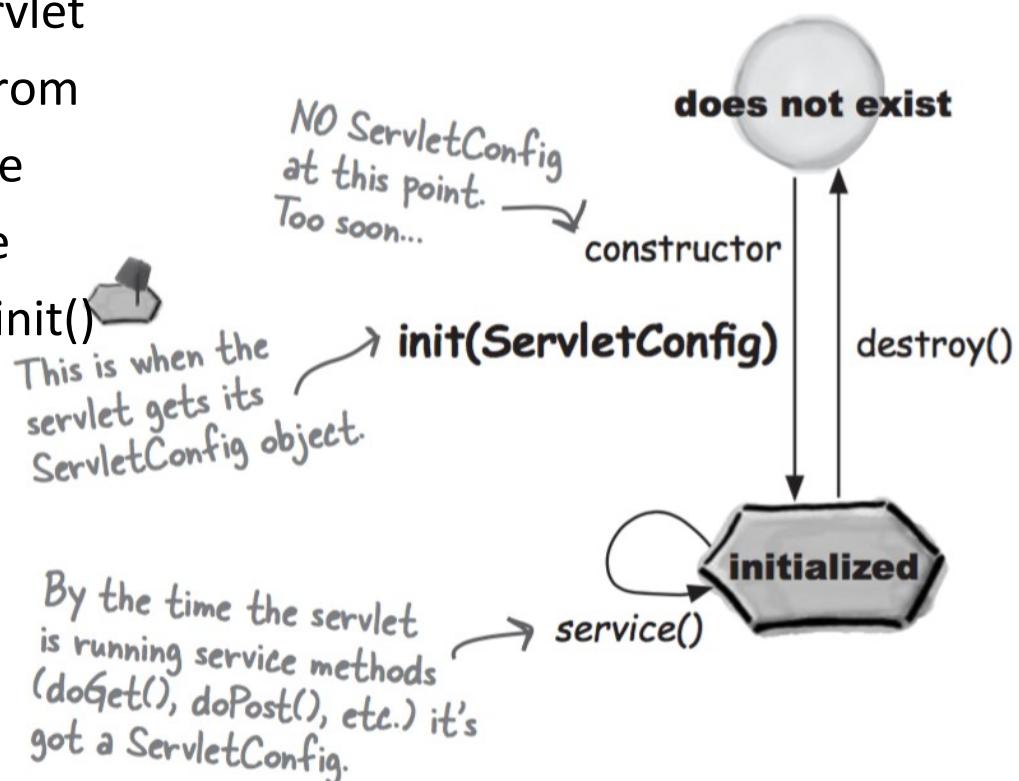
Esempio di parametro
di configurazione

```
<init-param>
    <param-name>parName</param-name>
    <param-value>parValue</param-value>
</init-param>
```

- Es: **getServletConfig().getInitParameter("parName")**

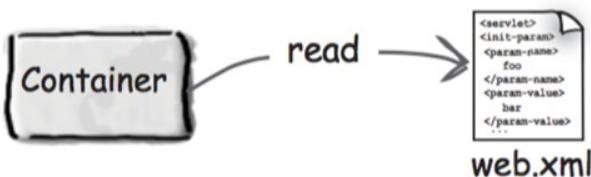
You can't use Servlet init parameters until the Servlet is initialized

- When the Container initializes a Servlet, it makes a unique ServletConfig for the Servlet
- The Container “reads” the Servlet init parameters (**init-param**) from web.xml and gives them to the ServletConfig, then passes the ServletConfig to the servlet’s init() method

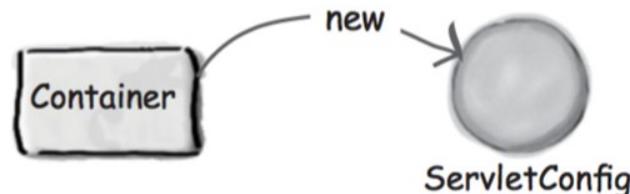


The Servlet init parameters are read only ONCE

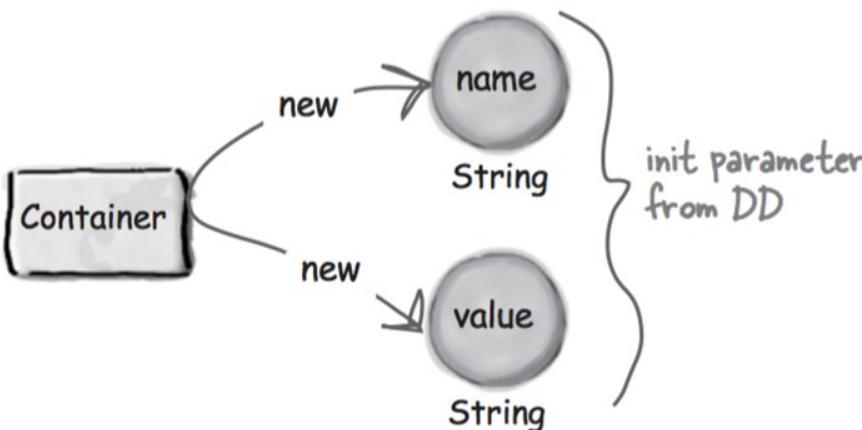
- ① Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



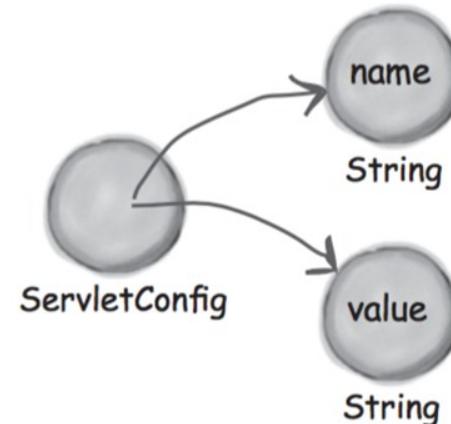
- ② Container creates a new ServletConfig instance for this servlet.



- ③ Container creates a name/value pair of Strings for each servlet init parameter. Assume we have only one.

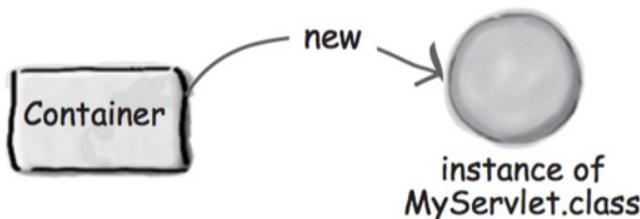


- ④ Container gives the ServletConfig references to the name/value init parameters.

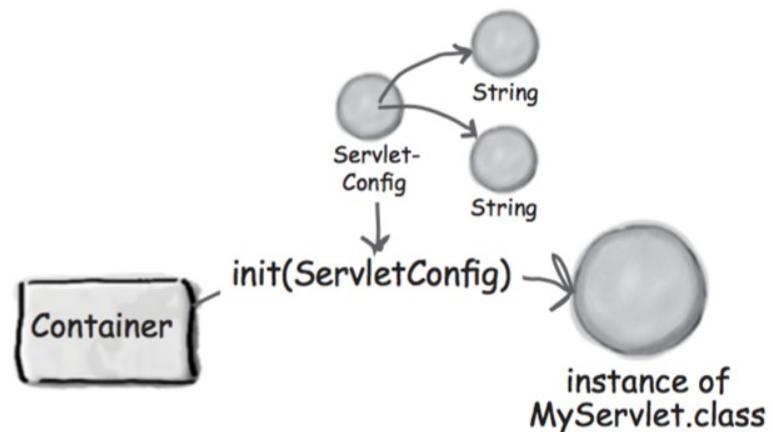


The Servlet init parameters are read only ONCE (2)

- ⑤ Container creates a new instance of the servlet class.



- ⑥ Container calls the servlet's init() method, passing in the reference to the ServletConfig.



Esempio di parametri di configurazione specificati tramite annotazioni

- Estendiamo il nostro esempio *Hello World* rendendo parametrico il titolo della pagina HTML e la frase di saluto
- Usiamo le annotazioni per specificare i parametri di inizializzazione
- Nota che occorre specificare gli attributi **name** e **urlPatterns** dell'annotazione **@WebServlet** (vedi HelloServlet.zip)

```
@WebServlet(name = "HelloServlet", urlPatterns = { "/HelloServlet" },  
    initParams = {@WebInitParam(name = "title", value = "Hello Page"),  
                 @WebInitParam(name = "greeting", value = "Ciao") })  
public class HelloServlet extends HttpServlet {
```

Esempio di parametri di configurazione specificati in web.xml

- Alternativamente, avremmo potuto definire sia il mapping fra URL e Servlet sia i parametri di configurazione nel deployment descriptor (vedi HelloServletDD.zip)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
  version="6.0">
  <display-name>HelloServletDD</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>it.unisa.HelloServlet</servlet-class>
    <init-param>
      <param-name>title</param-name>
      <param-value>Hello Page</param-value>
    </init-param>
    <init-param>
      <param-name>greeting</param-name>
      <param-value>Ciao</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/HelloServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

web.xml

Nota che *HelloServlet* non contiene né annotazione per il mapping fra URL e Servlet, né annotazioni per la definizione dei parametri di inizializzazione -- questi sono definiti in web.xml

HelloServlet parametrico

- Ridefiniamo quindi anche il metodo init(): memorizziamo i valori dei parametri in due proprietà
 - Nota che, qualsiasi sia la definizione dei parametri di inizializzazione (cioè tramite annotazioni o in web.xml), la loro gestione nella Servlet non cambia

```
public class HelloServlet extends HttpServlet
{
    private String title, greeting;

    public void init(ServletConfig config)
        throws ServletException
    {
        super.init(config);
        title = config.getInitParameter("title");
        greeting = config.getInitParameter("greeting");
    }
    ...
}
```

Il metodo doGet() con parametri di inizializzazione

```
http://.../hello?to=Mario
```

```
public void doGet(HttpServletRequest request,  
HttpServletResponse response)  
throws ServletException, IOException  
{  
    String toName = request.getParameter("to");  
    response.setContentType("text/html");  
    PrintWriter out = response.getWriter();  
    out.println("<html>");  
    out.println("<head><title>Hello page</title></head>");  
    out.println("<body>" + "Ciao " + toName + "!</body>");  
    out.println("</html>");  
}
```

```
HTTP/1.1 200 OK  
Content-Type: text/html  
<html>  
<head><title>Hello page</title></head>  
<body>Ciao Mario!</body>  
</html>
```

Servlet context

- Ogni Web application viene eseguita in un **contesto**:
 - corrispondenza 1:1 tra una Web application e suo contesto
- Si può ottenere un'istanza di tipo **ServletContext** (è un'interfaccia) all'interno della Servlet utilizzando il metodo **getServletContext()**
 - Consente di accedere ai parametri di inizializzazione e agli attributi del contesto
 - Consente di accedere alle risorse statiche della Web application (es. immagini) mediante il metodo **InputStream getResourceAsStream(String path)**
- **IMPORTANTE: Servlet context viene condiviso tra tutti gli utenti, le richieste e le Servlet facenti parte della stessa Web application**

Parametri di inizializzazione del contesto

- Parametri di inizializzazione del contesto definiti all'interno di elementi di tipo **context-param** in web.xml

```
<web-app>
  <context-param>
    <param-name>feedback</param-name>
    <param-value>feedback@deis.unibo.it</param-value>
  </context-param>
  ...
</ web-app >
```

- Sono accessibili a tutte le Servlet della Web application

```
...
ServletContext ctx = getServletContext();
String feedback =
ctx.getInitParameter("feedback");
...
```

Attributi di contesto

- Gli attributi di contesto sono accessibili a tutte le Servlet e funzionano come variabili “*globali*”
- Vengono gestiti a runtime:
 - possono essere creati, scritti e letti dalle Servlet
 - Possono contenere oggetti anche complessi

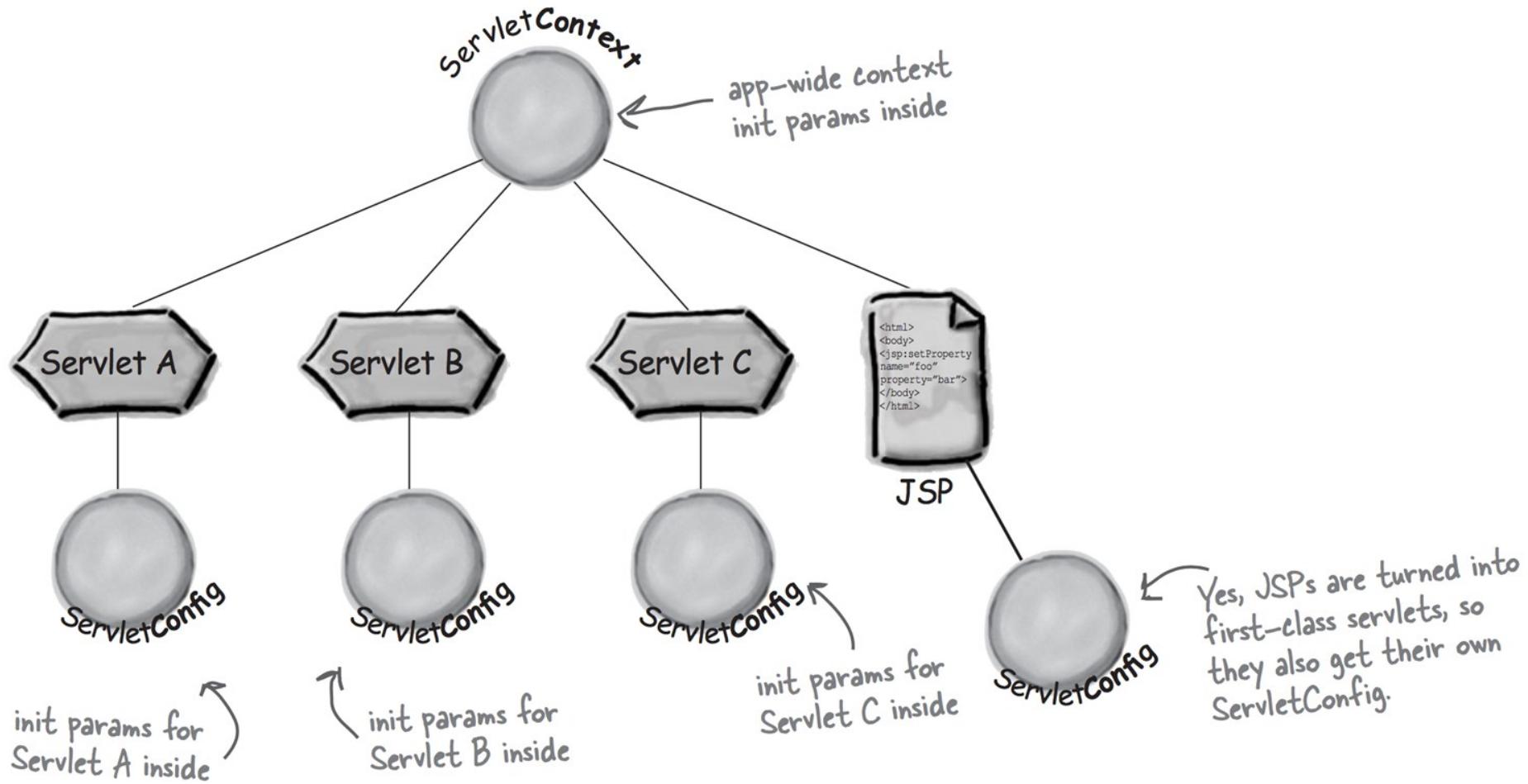
scrittura

```
ServletContext ctx = getServletContext();
ctx.setAttribute("utente1", new User("Giorgio Bianchi"));
ctx.setAttribute("utente2", new User("Paolo Rossi"));
```

lettura

```
ServletContext ctx = getServletContext();
Enumeration aNames = ctx.getAttributeNames();
while (aNames.hasMoreElements()) {
    String aName = (String)aNames.nextElement();
    User user = (User) ctx.getAttribute(aName);
    ctx.removeAttribute(aName);
}
```

ServletContext and ServletConfig



Gestione dello stato (di sessione)

- **HTTP è un protocollo stateless:** *non fornisce in modo nativo meccanismi per il mantenimento dello stato tra diverse richieste provenienti dallo stesso client*
- Le applicazioni Web hanno spesso bisogno di stato. Sono state definite due tecniche per mantenere traccia delle informazioni di stato:
 1. **uso dei cookie: meccanismo di basso livello**
 2. **uso della sessione (session tracking): meccanismo di alto livello**
- La sessione rappresenta un'utile astrazione ed essa stessa può far ricorso a due meccanismi base di implementazione:
 1. Cookie
 2. URL rewriting

Session Tracking and e-Commerce

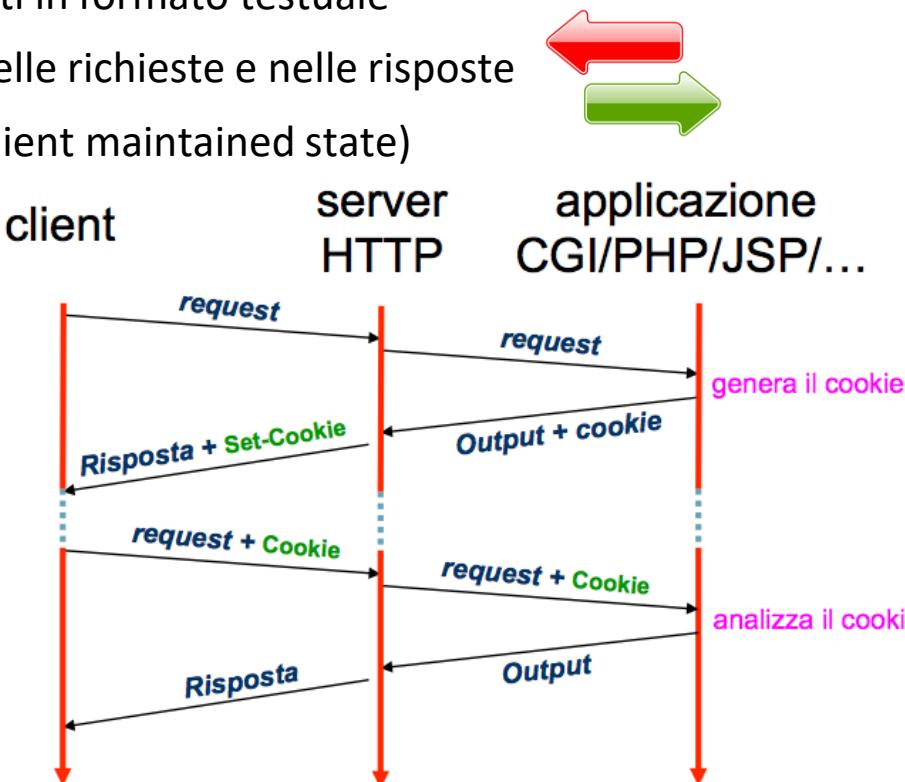
- **Why session tracking?**

- When clients at on-line store add item to their shopping cart, how does server know what's already in cart?
- When clients decide to proceed to checkout, how can server determine which previously created cart is theirs?
- ...

Shopping Cart		
Home	4:21 PM	100%
Kestrel 4000 LTD Ultegra Di2 2013 Chain Reaction Cycles, the world's largest online bicycle store.	\$370.00	
LEGO CITY POLICE STATION Agile and confident on XC race courses, the Vertex provides incredible climbing rollover...	\$398.20	
iPad Mini with Retina Display 32GB Agile and confident on XC race courses, the Vertex provides incredible climbing rollover...	\$490.75	
Nike Air Max 2014 iD Men's Running Shoe Agile and confident on XC race courses, the Vertex provides incredible climbing rollover...	\$560.40	
Samsung UN46F5500 46-Inch Full HD 1080p... Agile and confident on XC race	\$849.99	

Cookies

- Il **cookie** è un'unità di informazione che il Web server deposita sul Web browser lato cliente
 - Può contenere valori che sono propri del dominio funzionale dell'applicazione (in genere informazioni associate all'utente)
 - Sono parte dell'header HTTP, trasferiti in formato testuale
 - Vengono mandati avanti e indietro nelle richieste e nelle risposte
 - Vengono memorizzati dal browser (client maintained state)
- Attenzione però: possono essere rifiutati dal browser (tipicamente perché disabilitati)



The potential of Cookies

- Idea
 - Servlet sends a simple name and value to client
 - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings)
- Typical uses of Cookies
 - Identifying a user during an e-commerce session
 - Servlets have a higher-level API for this task
 - Avoiding username and password
 - Customizing a site
 - Focusing advertising

Cookies and Focused Advertising

AltaVista - Welcome - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://www.altavista.com/

alta vista POINT, CLICK ... EARN FREE STUFF! alta vista: REWARDS go!

Search Home Comparison Shop Channels Rewards Email & Tools Free Internet Access

Welcome ICAST TRAIL Search for: Today In: Shopping

Super Searches Shopping Entertainment News

Web Search Power Search Advanced Search Shopping Search

AltaVista - Web Page Results for: servlets jsp book - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://www.altavista.com/cgi-bin/query?q=servlets+jsn+book&kl=XXX&pg=0&Ttranslate=on

alta JavaServer Pages & Java Servlets HASSLE-FREE

Search Home Comparison Shop Channels Rewards Email & Tools Free Internet Access

Sign Up! | AltaVista Members Sign In | Member Center | My AltaVista

Home > Web Page Results for: servlets jsp book

Search for: Help | Customize Settings | Family Filter is off

servlets jsp book any language Search

Shop now... Books Music

AltaVista Recommends

- Find editor-selected sites: [servlets jsp book](#)
- Browse, Shop, Earn: [AltaVista Rewards](#)

Amazon.com

ne wh All You

La classe cookie

- Un cookie contiene un certo numero di informazioni, tra cui:
 - una coppia nome/valore
 - Caratteri non utilizzabili: []()=, " / ? @ : ;
 - il dominio Internet dell'applicazione che ne fa uso
 - path dell'applicazione
 - una expiration date espressa in secondi (-1 indica che il cookie non sarà memorizzato su file associato, 60*60*24 indica 24 ore)
 - un valore booleano per definirne il livello di sicurezza
- La **classe Cookie** modella il cookie HTTP
 - Si recuperano i cookie dalla **request** utilizzando il metodo **getCookies()**
 - Si aggiungono cookie alla **response** utilizzando il metodo **addCookie()**

Esempi di uso di cookie

- Con il metodo ***setSecure(true)*** il client viene forzato a inviare il cookie solo su protocollo sicuro (HTTPS)

creazione

```
Cookie c = new Cookie("MyCookie", "test");
c.setSecure(true);
c.setMaxAge(-1); // -1 represents that the given cookie exists until the browser is shutdown
c.setPath("/");
response.addCookie(c);
```

lettura

```
Cookie[] cookies = request.getCookies();
if(cookies != null)
{
    for(int j=0; j<cookies.length(); j++)
    {
        Cookie c = cookies[j];
        out.println("Un cookie: " +
                    c.getName() + "=" + c.getValue());
    }
}
```

Esempi di uso di cookie (2)

- To **delete a cookie** then you simply need to follow up following three steps:
 1. Read an already existing cookie and store it in Cookie object
 2. Set cookie age as zero using **setMaxAge()** method to delete an existing cookie
 3. Add this cookie back into response header
- Es:

```
Cookie[] cookies = null;  
cookies = request.getCookies();  
Cookie cookie = cookies[i]; //i-esimo Cookie  
cookie.setMaxAge(0);  
response.addCookie(cookie);
```

Access Count with Cookies (AccessCount.zip)

```
@WebServlet("/AccessCountCookie")
public class AccessCountCookie extends HttpServlet {
    private static final long serialVersionUID = 1L;

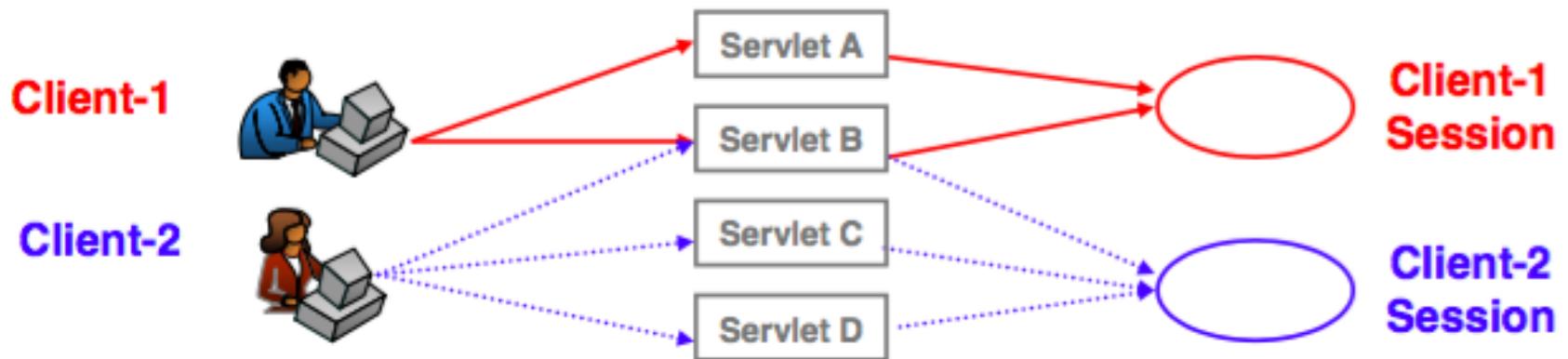
    public ShowSessionCookie() {
        super();
    }

    private String getCookieValue(HttpServletRequest request, String cookieName, String defaultValue) {
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookieName.equals(cookie.getName())) {
                    return (cookie.getValue());
                }
            }
        }
        return (defaultValue);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String countString = getCookieValue(request, "accessCount", "1");
        int count = 1;
        try {
            count = Integer.parseInt(countString);
        } catch (NumberFormatException nfe) {
            throw nfe;
        }
        Cookie c = new Cookie("accessCount", String.valueOf(count + 1));
        c.setMaxAge(60 * 60 * 24 * 365); // it expires after one year
        response.addCookie(c);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML>" + "<HTML>" + "<HEAD><TITLE>Access Count Servlet</TITLE></HEAD>" + "<BODY>"
            + "<H1>Access Count Servlet</H1>" + "<H2>This is visit number " + count + " by this browser.</H2>\n"
            + "</BODY></HTML>");
    }
}
```

Uso della sessione: Session

- La sessione Web è un'entità gestita dal Web Container
- *È condivisa fra tutte le richieste provenienti dallo stesso client: consente di mantenere, quindi, informazioni di stato (di sessione)*
- Può contenere dati di varia natura ed è identificata in modo univoco da un **session ID**
- Viene usata dai componenti di una Web application per mantenere lo stato del client durante le molteplici interazioni dell'utente con la Web application (conversazione)

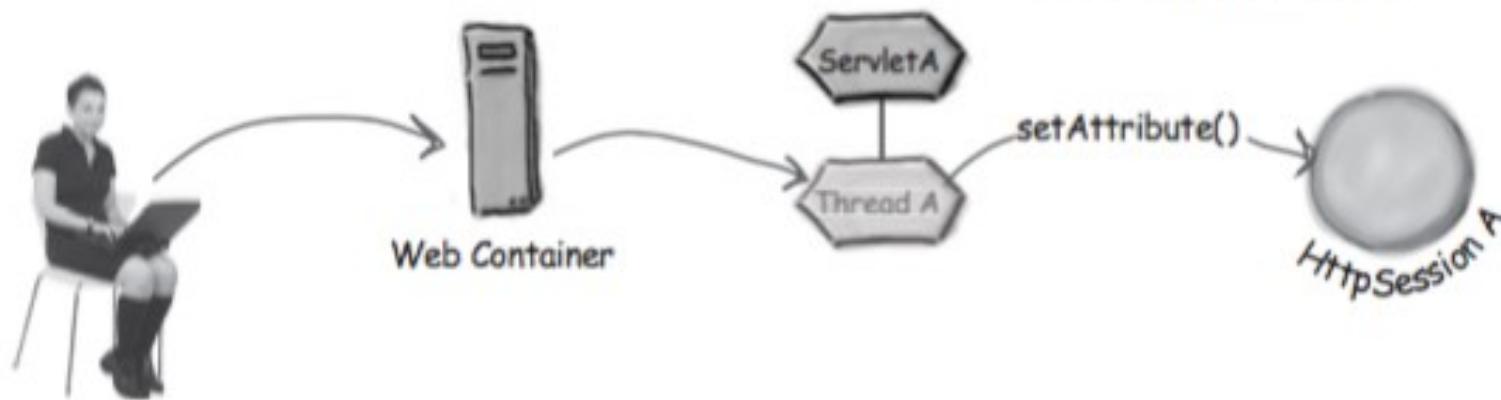


①

Diane selects "Dark" and hits the submit button.

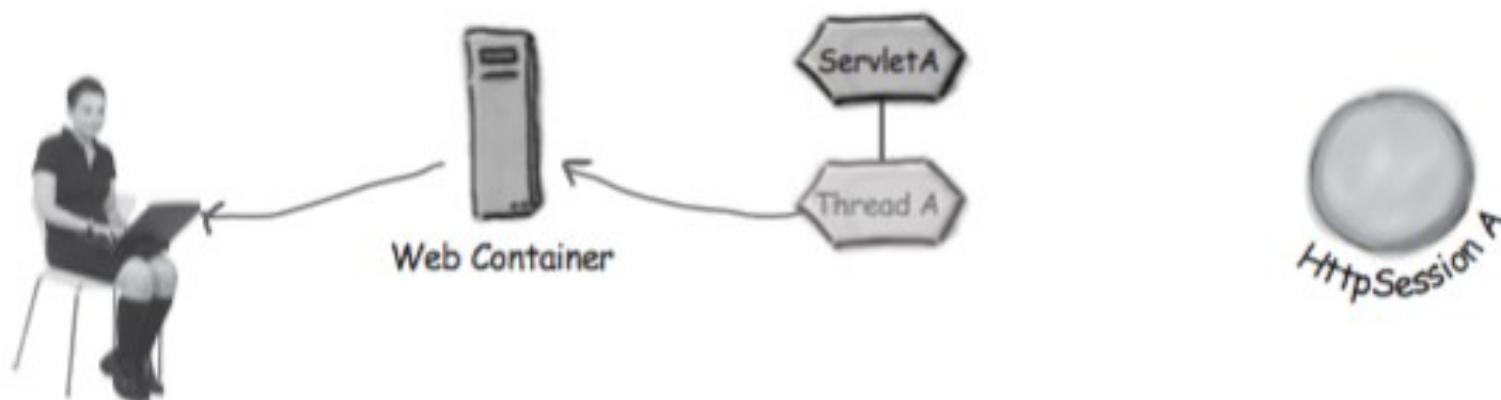
The Container sends the request to a new thread of the BeerApp servlet.

The BeerApp thread finds the session associated with Diane, and stores her choice ("Dark") in the session as an attribute.



②

The servlet runs its business logic (including calls to the model) and returns a response... in this case another question, "What price range?"

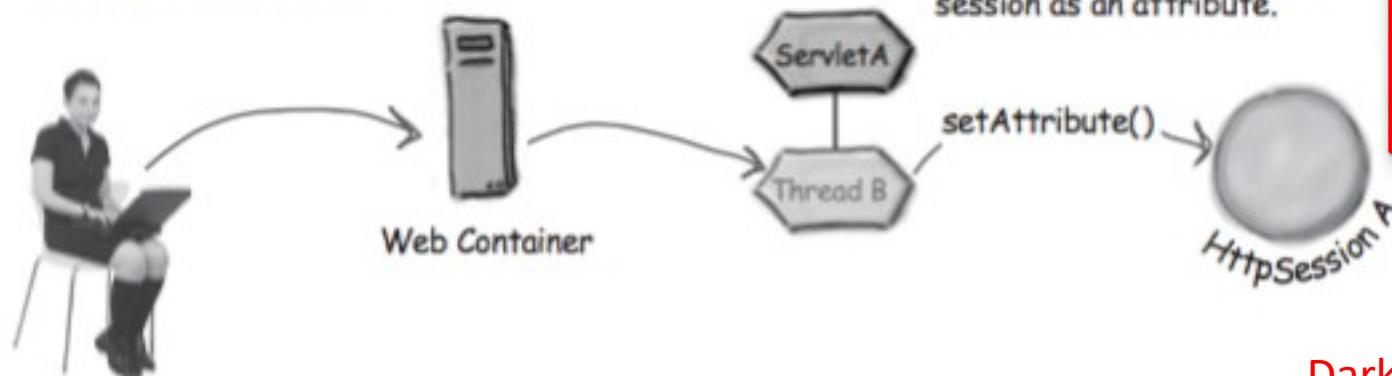


③

Diane considers the new question on the page, selects "Expensive" and hits the submit button.

The Container sends the request to a new thread of the BeerApp servlet.

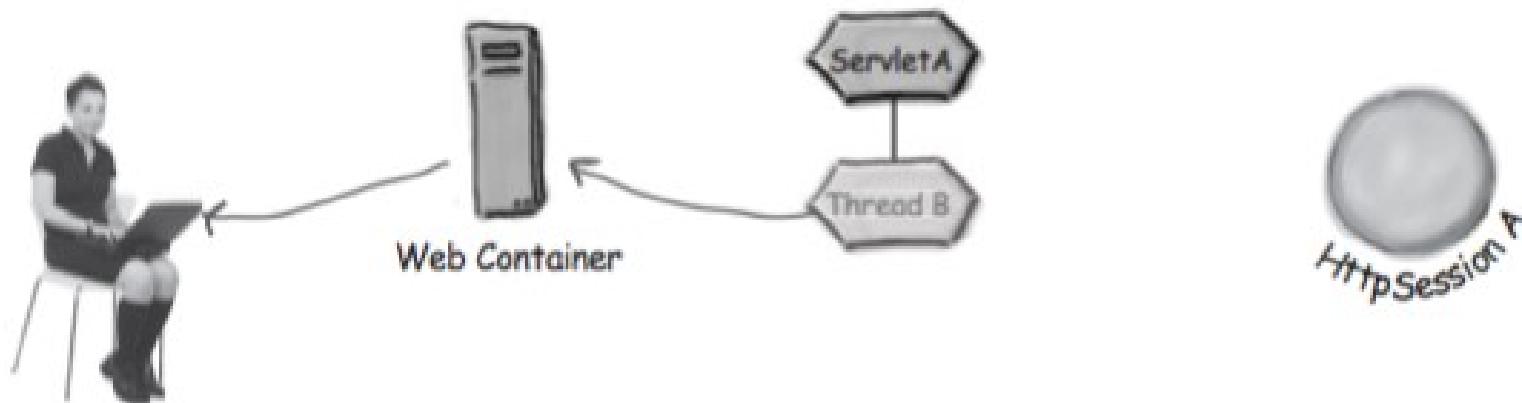
The BeerApp thread finds the session associated with Diane, and stores her new choice ("Expensive") in the session as an attribute.



Dark and Expensive

④

The servlet runs its business logic (including calls to the model) and returns a response... in this case another question.

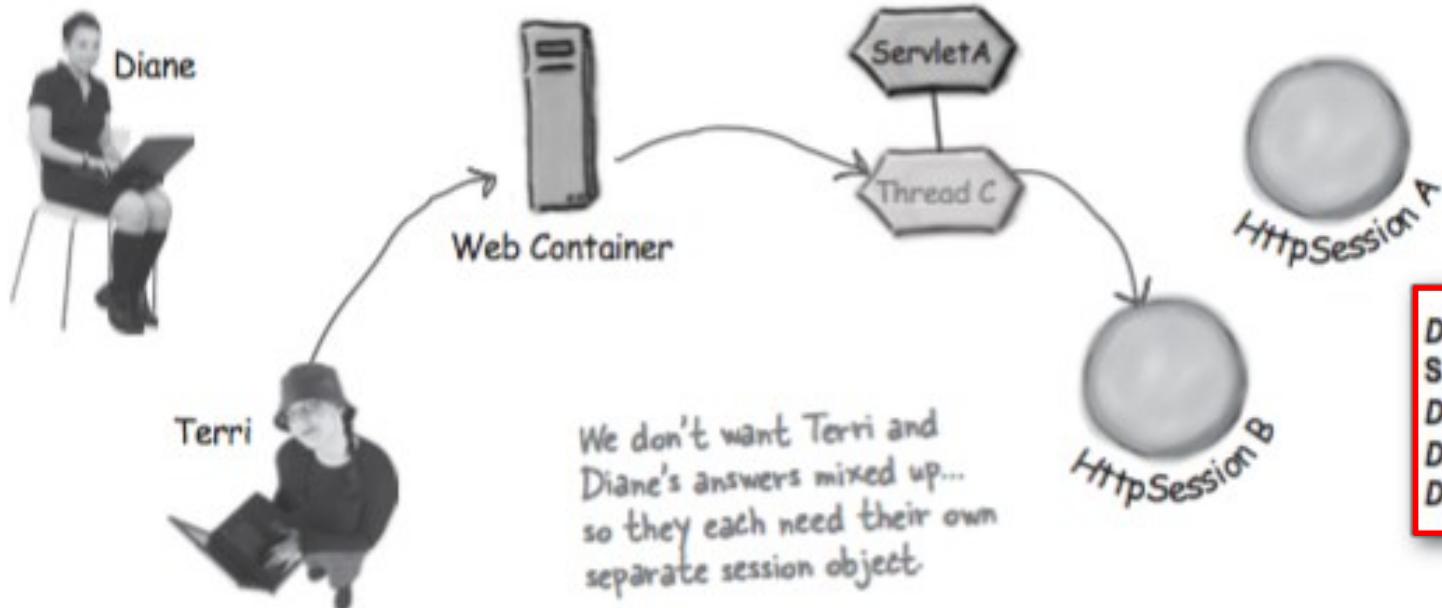


⑤

Diane's session is still active, but meanwhile Terri selects "Pale" and hits the submit button.

The Container sends Terri's request to a new thread of the BeerApp servlet.

The BeerApp thread starts a new Session for Terri, and calls `setAttribute()` to store her choice ("Pale").



Accesso alla sessione

- L'accesso avviene mediante l'interfaccia **HttpSession**
- Per ottenere un riferimento ad un oggetto di tipo HttpSession si usa il metodo **getSession(boolean createNew)** o **getSession()** dell'interfaccia **HttpServletRequest**

public HttpSession getSession(boolean createNew)

- Valori di createNew:
 - **true**: ritorna la sessione esistente o, se non esiste, ne crea una nuova
 - **false**: ritorna, se possibile, la sessione esistente, altrimenti ritorna **null**

public HttpSession getSession()

- ritorna la sessione esistente o, se non esiste, ne crea una nuova

Gestione del contenuto di una sessione

- Si possono memorizzare **dati specifici dell'utente negli attributi della sessione** (coppie nome/valore)
- Sono simili agli attributi di contesto, *ma con scope fortemente diverso!*, e consentono di memorizzare e recuperare oggetti

```
Cart sc = (Cart) session.getAttribute("shoppingCart");
sc.addItem(item);
...
session.removeAttribute("shoppingCart");
...
session.setAttribute("shoppingCart", new Cart());
...
Enumeration e = session.getAttributeNames();
while(e.hasMoreElements())
    out.println("Key; " + (String)e.nextElement());
```

Altre operazioni con le sessioni

- **String getId()** restituisce l'ID di una sessione
- **boolean isNew()** dice se la sessione è nuova
- **void invalidate()** permette di invalidare (*distruggere*) una sessione
- **long getCreationTime()** dice da quanto tempo è attiva la sessione (in millisecondi)
- **long getLastAccessedTime()** fornisce informazioni su quando è stata utilizzata l'ultima volta

```
String sessionID = session.getId();
if(session.isNew())
    out.println("La sessione e' nuova");
session.invalidate();
out.println("Millisec:" + session.getCreationTime());
out.println(session.getLastAccessedTime());
```

Session ID, Cookie e URL Rewriting

- *Il session ID è usato per identificare le richieste provenienti dallo stesso utente e mapparle sulla corrispondente sessione*

JSESSIONID=FE8FC061C396D7BA00D1C18EFDE8146B

1. Una tecnica per trasmettere l'ID è quella di includerlo in un cookie (**session cookie**): *sappiamo però che non sempre i cookie sono attivati nel browser*
2. Un'alternativa è rappresentata dall'inclusione del session ID nell'URL: si parla di **URL rewriting**

URL-Rewriting

- Idea
 - Client appends some extra data on the end of each URL that identifies the session
 - Server associates that identifier with data it has stored about that session
 - Es: **http://host/path/file.html;jsessionid=FE8FC1234567890**
- Advantage
 - *Works even if cookies are disabled or unsupported*
- Disadvantages
 - Must encode all URLs that refer to your own site
 - All pages must be dynamically generated
 - **The session ID is visible in the URL, this might pose a security risk**

URL-Rewriting (2)

- In passato era buona prassi codificare sempre gli URL generati dalle Servlet usando il metodo **encodeURL()** di **HttpServletResponse**
 - Usato per garantire una gestione corretta della sessione
 - Se il server sta usando i cookie, ritorna l'URL non modificato
 - Se il server sta usando l'URL rewriting, prende in input un URL, e se l'utente ha i cookie disattivati, codifica l'id di sessione nell'URL
- Il metodo encodeURL() dovrebbe essere usato per:
 - hyperlink ()
 - form (<form action="...">)
- Es:

```
String url = "order-page.html";
url = response.encodeURL(url);
```

Cookie attivati: order-page.html

Cookie disattivati: order-page.html;jsessionid=7199A66051A35953113E8D134B02034F

URL-Rewriting: to use or not to use?

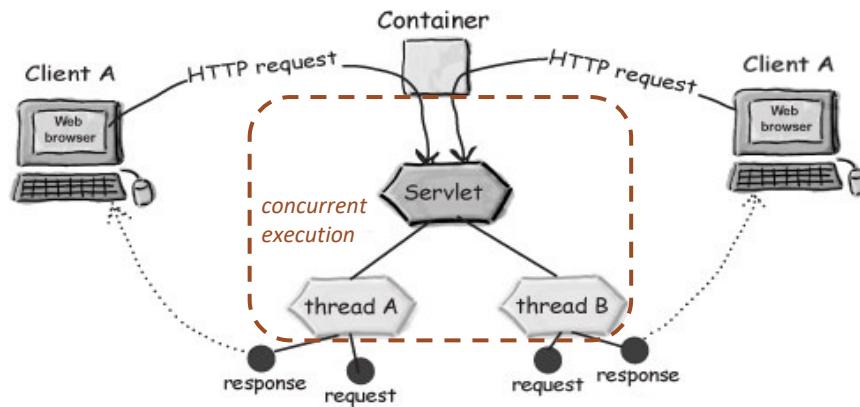
- Since the session ID is visible in the URL, if you copy that URL and then share it with others you are exposing your session ID so posing a security risk
- What to do?
 - Use cookies and HTTPS!!!

Session tracking basics

```
...  
  
HttpSession session = request.getSession();  
  
SomeClass value = (SomeClass) session.getAttribute("someID");  
if (value == null) {  
    value = new SomeClass();  
}  
doSomethingWith(value);  
session.setAttribute("someID", value);
```

To Synchronize or not to Synchronize?

- There are no race conditions when multiple different users access the page simultaneously
- It seems practically impossible for the same user to access the session concurrently
- The rise of Ajax makes synchronization important
 - With Ajax calls, it is actually quite likely that two requests from the same user could arrive concurrently



Use a synchronized block

...

```
HttpSession session = request.getSession();
synchronized (session) {
    @SuppressWarnings("unchecked")
    List<String> previousItems = (List<String>) session.getAttribute("previousItems");
    if (previousItems == null) {
        previousItems = new ArrayList<String>();
    }
    String newItem = request.getParameter("newItem");
    if (newItem != null) {
        previousItems.add(newItem);
    }
    session.setAttribute("previousItems", previousItems);
}
...
...
```

Access Count with Session (see AccessCount.zip)

```
@WebServlet("/AccessCountSession")
public class AccessCountSession extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public AccessCountSession() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        HttpSession session = request.getSession();
        synchronized (session) {
            String heading;
            Integer accessCount = (Integer) session.getAttribute("accessCount");
            if (accessCount == null) {
                accessCount = 0;
                heading = "Welcome, Newcomer";
            } else {
                heading = "Welcome Back";
                accessCount = accessCount + 1;
            }
            session.setAttribute("accessCount", accessCount);
            PrintWriter out = response.getWriter();
            out.println("<!DOCTYPE html> <html>" + "<head><title>Session Tracking Example</title></head>"
                    + "<body><h1>" + heading + "</h1>"
                    + "<h2>Information on Your Session:</h2>"
                    + "<p>ID: " + session.getId() + "</p>"
                    + "<p>Creation Time: " + new Date(session.getCreationTime())
                    + "</p>" + "<p>Time of Last Access: " + new Date(session.getLastAccessedTime())
                    + "</p>" + "<p>Number of Previous Accesses: "
                    + accessCount + "</p>" +
                    "</body></html>");
        }
    }
}
```

Attenzione: scope DIFFERENZIATI (scoped objects)

- Gli oggetti di tipo `HttpServletRequest` , `HttpSession` , `ServletContext` forniscono metodi per immagazzinare e ritrovare oggetti nei loro rispettivi ambiti (*scope*)
- Lo scope è definito dal **tempo di vita (lifespan)** e dall'**accessibilità** da parte delle Servlet

Ambito	Interfaccia	Tempo di vita	Accessibilità
Request	<code>HttpServletRequest</code>	Fino all'invio della risposta	Servlet corrente e ogni altra pagina inclusa o in forward
Session	<code>HttpSession</code>	Lo stesso della sessione utente	Ogni richiesta dello stesso client
Application	<code>ServletContext</code>	Lo stesso dell'applicazione	Ogni richiesta alla stessa Web app anche da clienti diversi e per servlet diverse

Funzionalità degli scoped object

- Gli oggetti scoped forniscono i seguenti metodi per immagazzinare e ritrovare oggetti nei rispettivi ambiti (scope):
 - **void *setAttribute(String name, Object o);***
 - **Object *getAttribute(String name);***
 - **void *removeAttribute(String name);***
 - **Enumeration *getAttributeNames();***

Ridirezione del browser

- È possibile inviare al browser una risposta che lo forza ad accedere ad un'altra pagina/risorsa (*ridirezione*) -- può risiedere sullo stesso server o su un server diverso
- Possiamo ottenere una ridirezione agendo sull'oggetto **response** invocando il metodo
 - **public void *sendRedirect(String url)*;**
 - Nota che *sendRedirect* imposta lo status code della risposta a **302 (Found)** e imposta il **location** header con l'URL passato come argomento al metodo

SendRedirect example (Redirect.zip)

index.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4<html>
5   <body>
6     <h2>Servlet SendRedirect Example</h2>
7     <form id="LoginFormId" method="post" action="LoginServlet">
8       Username:<input id="userInput" type="text" name="username" required/>
9       <br>
10      Password:<input id="passInput" type="password" name="password" required/>
11      <br>
12      <input id="btn" type="submit" value="Login" />
13      <%if(request.getAttribute("error") != null) {%
14        <p>You are not an authorized user!</p>
15      %}>
16    </form>
17 </html>
```

```

@WebServlet("/loginServlet")
public class Login extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");

        String login = request.getParameter("username");
        String pwd = request.getParameter("password");

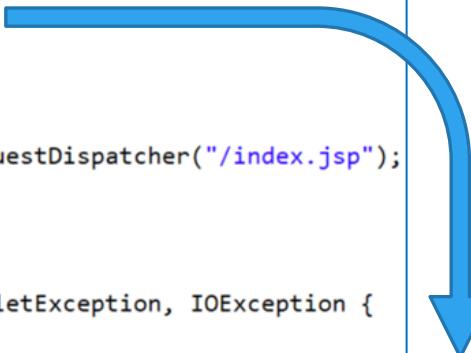
        if(login.equalsIgnoreCase("root") && pwd.equals("admin")) {
            request.getSession().setAttribute("uname", login);
            response.sendRedirect("welcomeServlet");
            return;
        }

        request.setAttribute("error", Boolean.TRUE);
        RequestDispatcher dispatcher = request.getRequestDispatcher("/index.jsp");
        dispatcher.forward(request, response);
    }

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

```

SendRedirect example (2)



```

@WebServlet("/welcomeServlet")
public class Welcome extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");

        String param1 = (String) request.getSession().getAttribute("uname");

        PrintWriter out = response.getWriter();
        out.println("<!doctype html>");
        out.println("<html><body>");
        out.println("<h2>Servlet SendRedirect Example</h2>");
        out.println("<p>Congratulations! " + param1 + ", You are an authorised login!</p>");
        out.println("</body></html>");
    }

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

```

Includere una risorsa (include)

- Per includere una risorsa (Servlet, JSP o file HTML) si ricorre a un oggetto di tipo **RequestDispatcher** che può essere ottenuto a partire da un oggetto **request** (o da un oggetto **ServletContext**)
 - In entrambi i casi, occorre invocare il metodo **getRequestDispatcher** al quale va passato l'URL della risorsa da includere
- Si invoca poi il metodo **include** passando come parametri **request** e **response** che vengono così condivisi con la risorsa inclusa

può essere anche una pagina JSP o file HTML

```
RequestDispatcher dispatcher =  
    request.getRequestDispatcher("/inServlet");  
dispatcher.include(request, response);
```

Inoltro (forward)

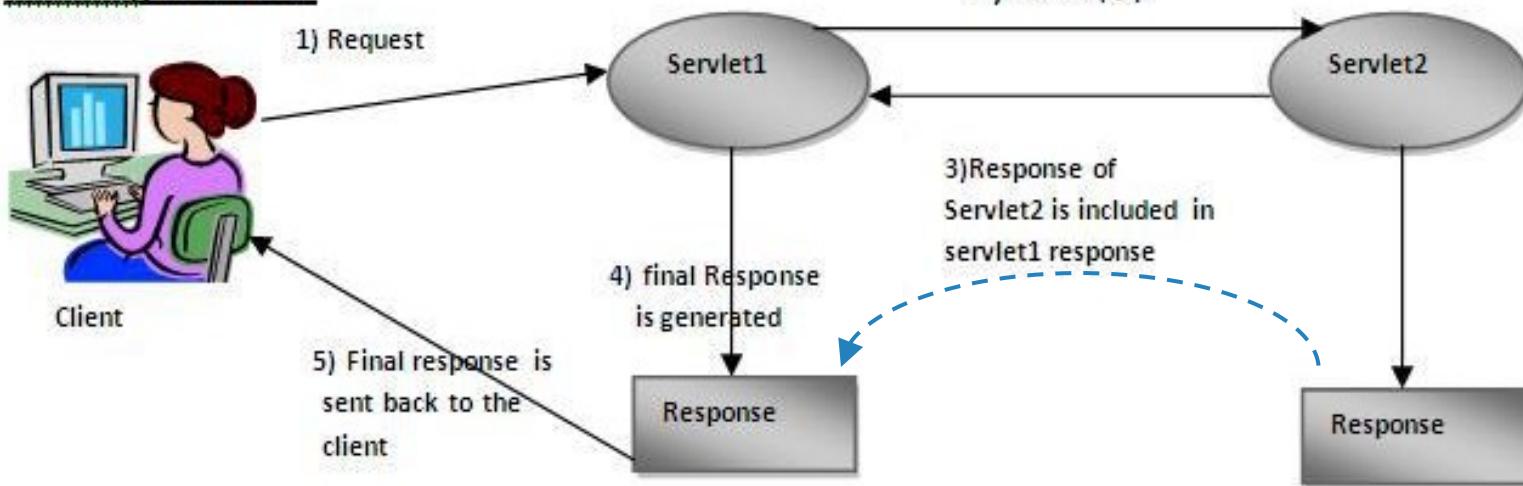
- Si usa in situazioni in cui una Servlet si occupa di parte dell'elaborazione della richiesta e delega a qualcun altro la gestione della risposta
 - **Attenzione: la gestione della risposta è di competenza esclusiva della risorsa che riceve l'inoltro**
 - **Se nella prima Servlet è stato fatto un accesso a ServletOutputStream o PrintWriter si ottiene una IllegalStateException**
- Si deve ottenere un oggetto di tipo **RequestDispatcher** a partire da un oggetto **request** (o da un oggetto **ServletContext**) passando come parametro l'URL della risorsa
- Si invoca poi il metodo **forward** passando anche in questo caso **request** e **response**

può essere anche una pagina JSP o file HTML

```
RequestDispatcher dispatcher =  
    request.getRequestDispatcher("/inServlet");  
dispatcher.forward(request, response);
```

Include e forward

include() method



forward() method:

