



CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

SERVLET

a.a. 2020-2021

Cos'è una Servlet?

- È una classe Java che fornisce risposte a richieste HTTP
- In termini più generali 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

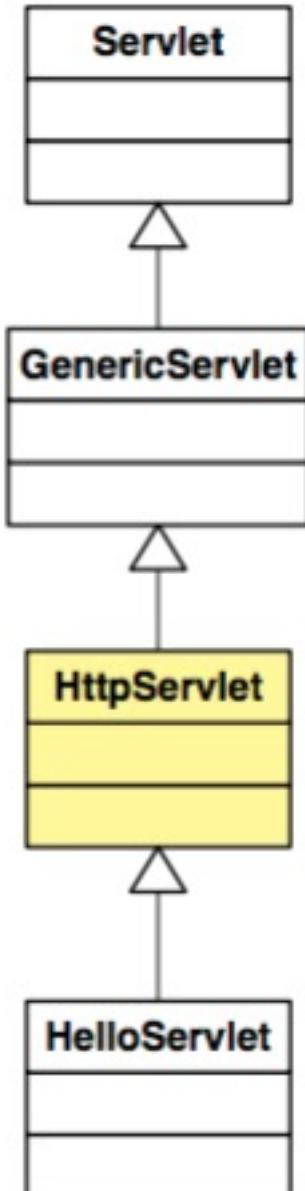
Esempio di Servlet: Hello world!

- 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:

```
...
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>");
    }
    ...
}
```

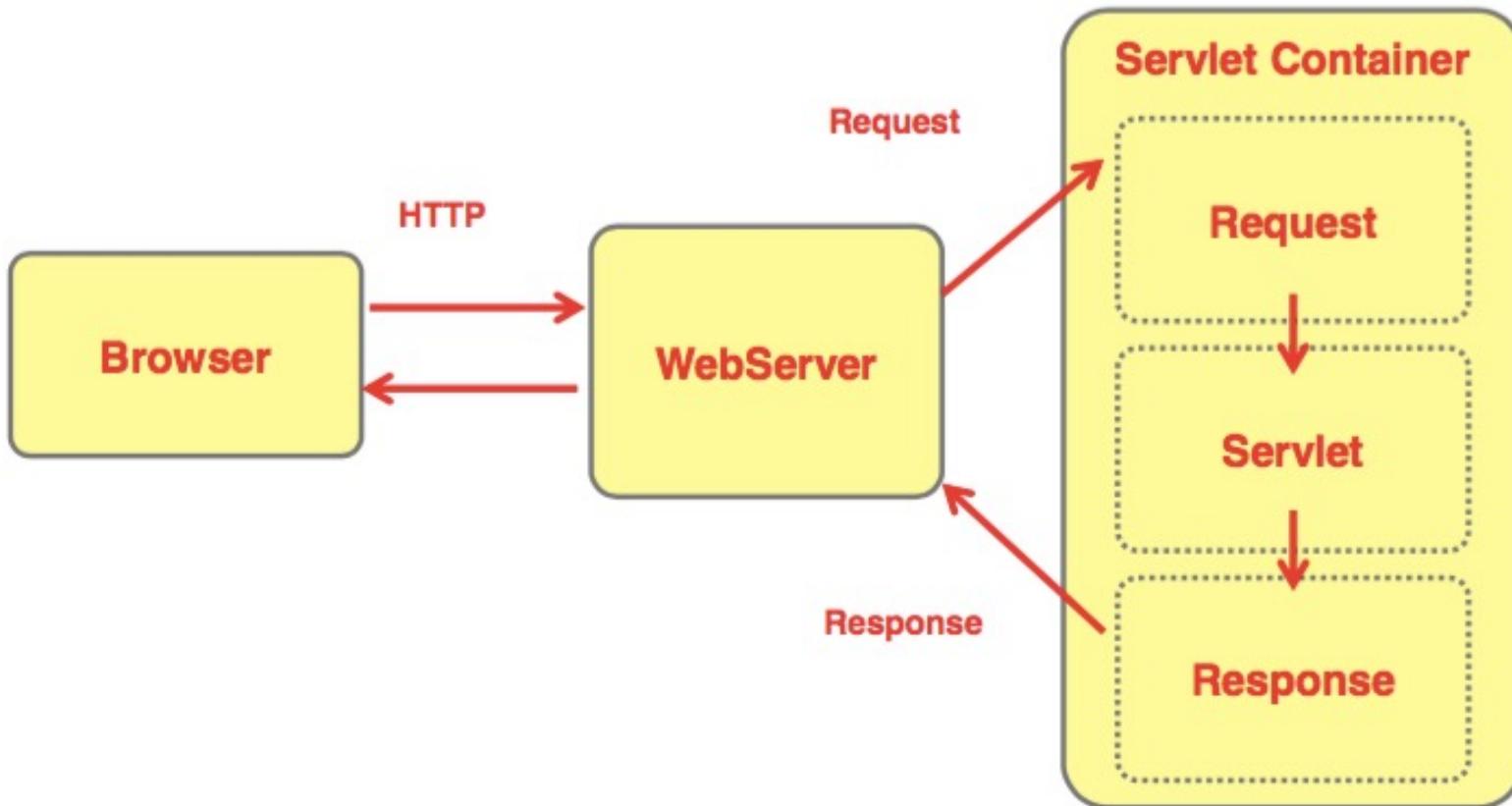
Gerarchie delle Servlet

- 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
javax.servlet.http.*



The request-response model

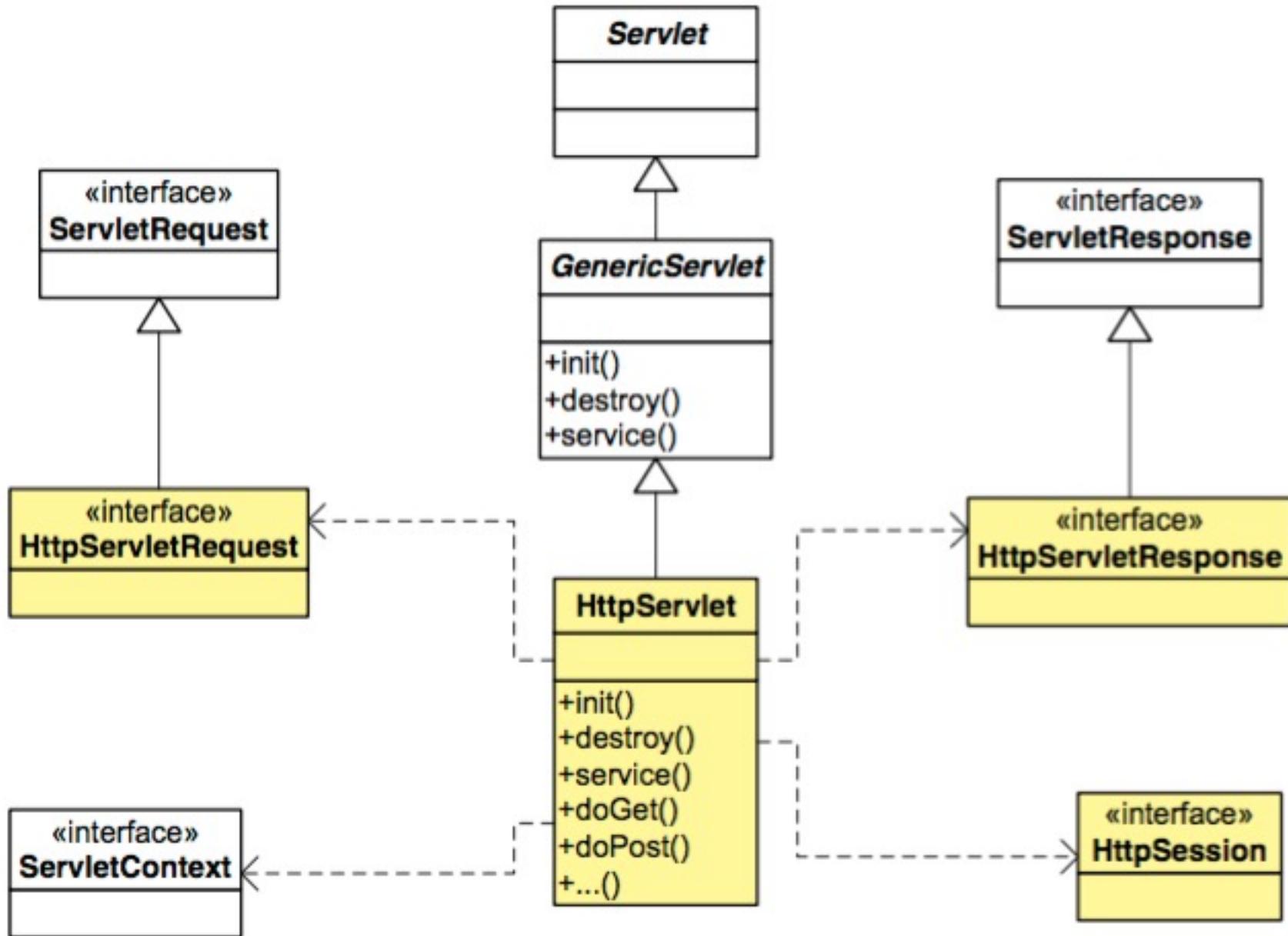
- All'arrivo di una richiesta HTTP il Servlet Container (Web Container) crea un oggetto **request** e un oggetto **response** e li passa alla Servlet:



Request e Response

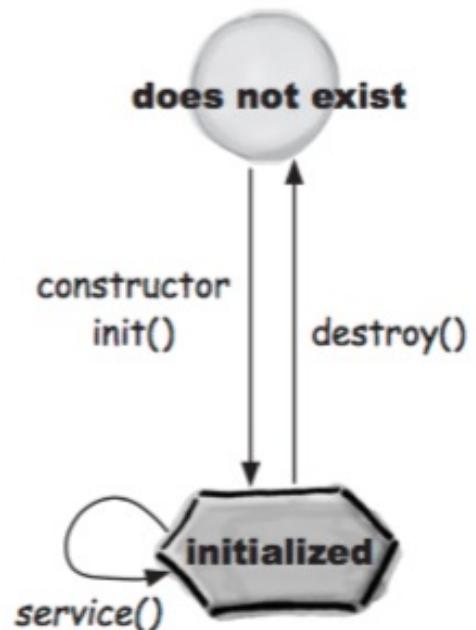
- Gli oggetti di tipo **Request rappresentano la chiamata al Server effettuata dal Client**
- Sono caratterizzati da varie informazioni
 - Chi ha effettuato la Request
 - Quali parametri sono stati passati nella Request
 - Quali header sono stati passati
- Gli oggetti di tipo **Response rappresentano 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

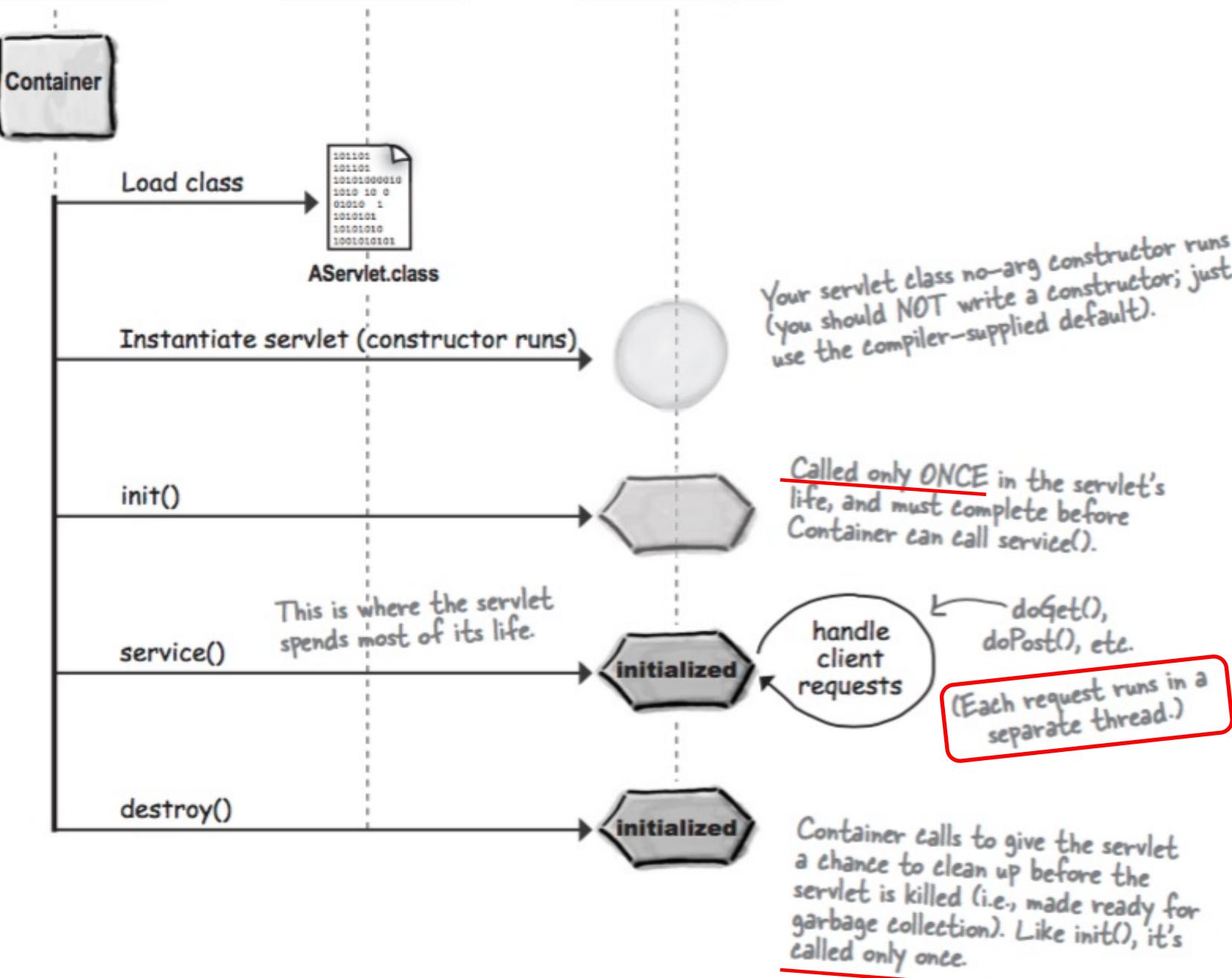
- Il Servlet Container controlla e supporta automaticamente il ciclo di vita di una Servlet
- Esiste un solo stato principale: **inizialized**
- Se la Servlet non è inizialized, può essere:
 - being initialized** (eseguendo il suo constructor o metodo **init()**)
 - being destroyed** (eseguendo il suo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



Web Container

Servlet Class

Servlet Object



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())

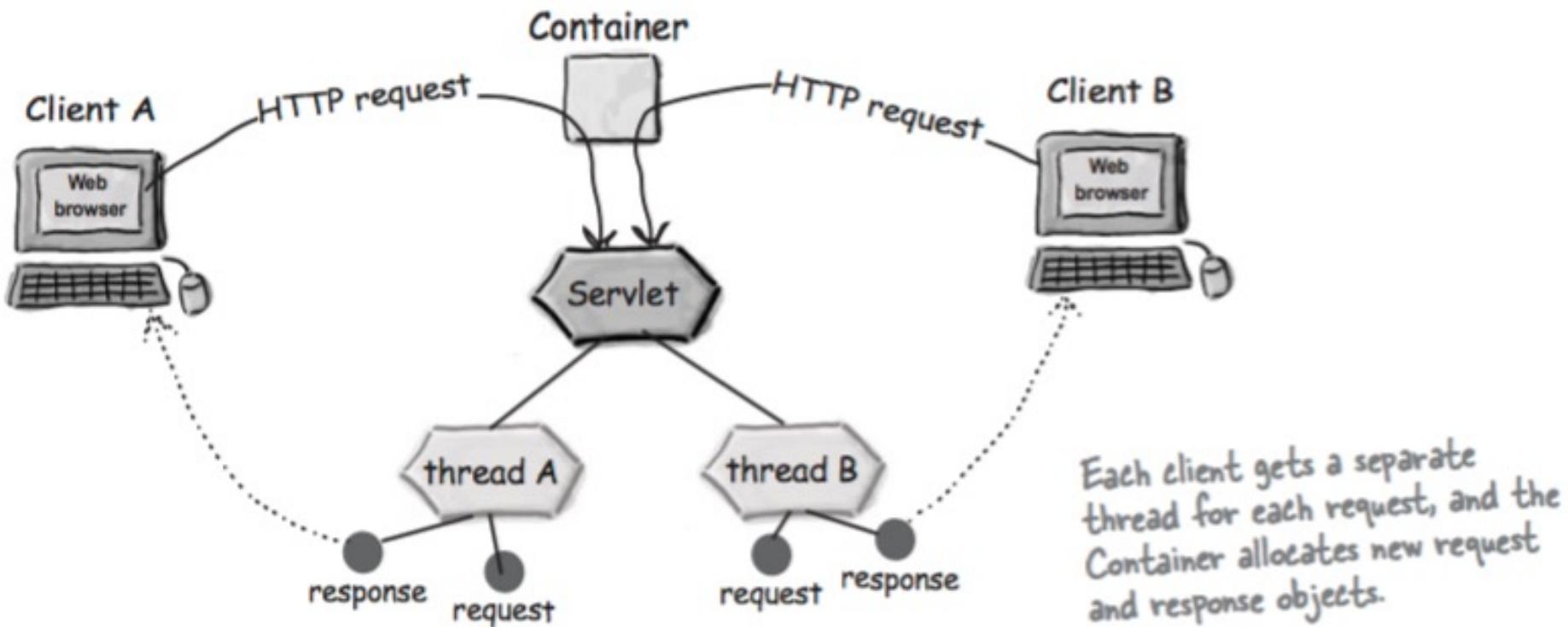
Whichever one(s) you override tells the Container what you support. If you don't override doPost(), for example, then you're telling the Container that this servlet does not support HTTP POST requests.

Servlet & Multithreading

- **Modello “normale”:** una sola istanza di Servlet e un thread assegnato ad ogni richiesta HTTP per Servlet, anche se richieste per quella Servlet sono già in esecuzione
- 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 da numerosi client in modo concorrente ed è quindi necessario gestire le sezioni critiche (a completo carico del programmatore dell'applicazione Web):
 - Uso di blocchi synchronized
 - Semafori
 - Mutex (mutua esclusione)

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 marker

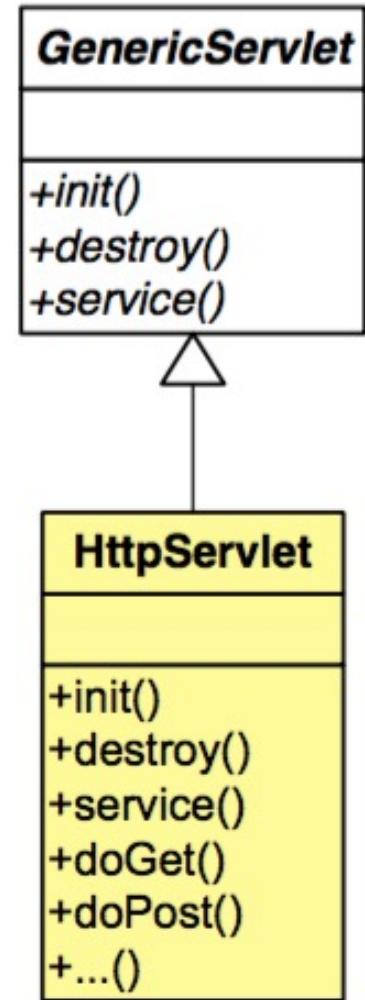
SingleThreadModel

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()**

```
import java.io.*  
import javax.servlet.*  
import javax.servlet.http.*;  
  
public class HelloServlet extends HttpServlet  
{  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
    {...}  
}
```

Hello World: 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  
{  
    ...  
}
```

Hello World: 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);  
}
```

ServletRequest interface

(javax.servlet.ServletRequest)

<<interface>> ServletRequest

```
getAttribute(String) : Object  
getContentLength() : int  
getInputStream() : ServletInputStream  
getLocalPort() : int  
getParameter(String) : String  
getParameterNames() : Enumeration  
// MANY more methods...
```

HttpServletRequest interface

(javax.servlet.http.HttpServletRequest)

<<interface>> HttpServletRequest

```
getContextPath() : String  
getCookies() : Cookie[]  
getHeader(String) : String  
getQueryString() : String  
getSession() : HttpSession  
getMethod() : String  
// MANY more methods...
```

ServletResponse interface

(javax.servlet.ServletResponse)

<<interface>> ServletResponse

```
getBufferSize() : int  
setContent-Type(String) : void  
setContentLength(int) : void  
getOutputStream() : ServletOutputStream  
getWriter() : PrintWriter  
// MANY more methods...
```

HttpServletResponse interface

(javax.servlet.http.HttpServletResponse)

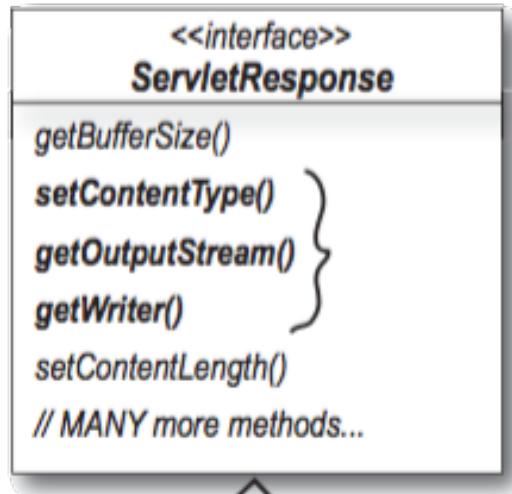
<<interface>> HttpServletResponse

```
addCookie(Cookie) : void  
addHeader(String name, String value) : void  
encodeRedirectURL(String url) : String  
sendError(int) : void  
setStatus(int) : void  
// MANY more methods...
```

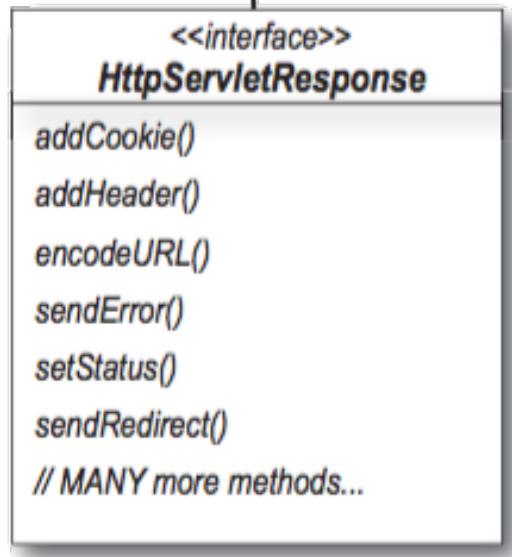
ServletResponse interface (javax.servlet.ServletResponse)

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



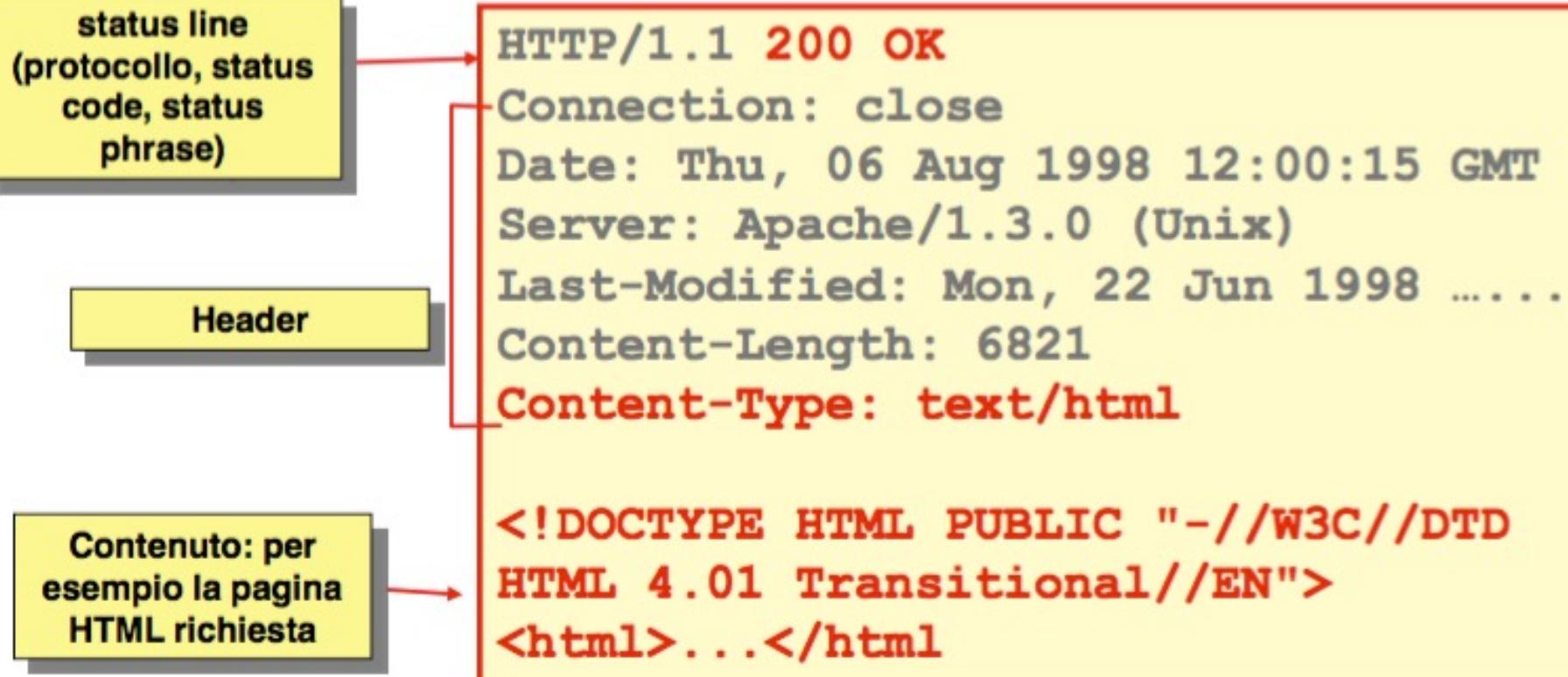
HttpServletResponse interface (javax.servlet.http.HttpServletResponse)



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**)
 - Otttenere un **output stream** in cui scrivere il contenuto da restituire
 - Indicare se l'output è bufferizzato
 - Gestire i cookie
 - ...

Formato della risposta HTTP



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

Gestione dello status code

- Per definire lo status code HttpServletResponse fornisce il metodo

public void setStatus(int statusCode)

- Esempi di status Code

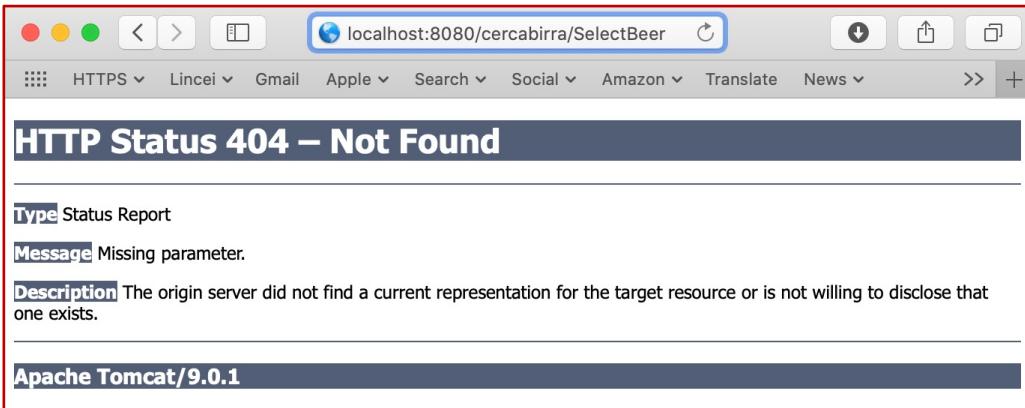
- **200 OK**
- **404 Page not found**
- **...**

- Per inviare errori possiamo anche usare:

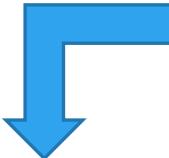
public void sendError (int sc)

public void sendError(int code, String message)

Example (cercabirraM1.zip)



SelectBeer.java



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

String c = request.getParameter("color");

BeerExpert be = new BeerExpert();
if(be.isValid(c)) {
    List<String> result = be.getBrands(c);
    request.setAttribute("styles", result);
} else {
    response.sendError(HttpServletRequest.SC_NOT_FOUND,
                      "Missing parameter.");
    return;
}

RequestDispatcher view = request.getRequestDispatcher("result.jsp");
view.forward(request, response);
}
```

```
public class BeerExpert {
    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) {
        if(colors.contains(color)) return true;
        return false;
    }
}
```

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.setIntHeader("Expires", 0);
```

- **setContentType** configura il content-type (*si usa quasi sempre*)
- **setContentLength** utile per la gestione di connessioni persistenti
- **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!*

Implementazione di doGet()

- 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>"

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**

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/
```

ServletRequest interface
(javax.servlet.ServletRequest)

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();
```

<<interface>> ServletRequest	
getAttribute(String)	
getContentLength()	
getInputStream()	
getLocalPort()	
getRemotePort()	
getServerPort()	
getParameter(String)	
getParameterValues(String)	
getParameterNames()	
// MANY more methods...	



<<interface>> HttpServletRequest	
getContextPath()	
getCookies()	
getHeader(String)	
getIntHeader(String)	
getMethod()	
getQueryString()	
getSession()	
// MANY more methods...	

Request URL

- Una URL HTTP ha la sintassi

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

- La **request path** è composta dal contesto e dal nome della Web application
- 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:

< a href="/bkstore1/catg?add=101">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 valore di un parametro individuato per nome
- **Enumeration getParameterNames()**
 - restituisce tutti i nomi dei parametri sotto forma di enumerazione di stringhe
- **String[] getParameterValue(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 stringa di query
- **String getPathInfo()**
 - per ottenere il path
- **String getPathTranslated()**
 - per ottenere informazioni sul path nella forma risolta

Metodi per accedere all'Header

- **String getHeader(String name)**
 - restituisce il 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

```
Enumeration<String> names = request.getHeaderNames();
while (names.hasMoreElements()) {
    String name = (String) names.nextElement();
    Enumeration<String> values = request.getHeaders(name);

    if (values != null) {
        while (values.hasMoreElements()) {
            String value = (String) values.nextElement();
            System.out.println(name + ": " + value);
        }
    }
}
```

Header Name	Header Value
host	localhost:8080
cookie	JSESSIONID=FE8FC061C396D7BA00D1C18EFDE8146B
connection	keep-alive
upgrade-insecure-requests	1
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
user-agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0.3 Safari/605.1.15
referer	http://localhost:8080/cercabirraIV/
accept-language	en-us
accept-encoding	gzip, deflate

Autenticazione, sicurezza e cookies

- **String getRemoteUser()**
 - nome di user se la Servlet ha accesso autenticato, null altrimenti
- **String getAuthType()**
 - nome dello schema di autenticazione usato per proteggere la Servlet
- **boolean isUserInRole(java.lang.String role)**
 - restituisce true se l'utente è associato al ruolo specificato
- **Cookie[] getCookies()**
 - restituisce un array di oggetti cookie che il client ha inviato alla request

Il metodo doGet con request

```
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>"
```

Esempio di doPost: gestione del form

- I form dichiarano i campi utilizzando l'attributo **name**
- Quando il form viene inviato al server, il nome dei campi e loro valori sono inclusi nella request:
 - agganciati alla URL come query string (GET)
 - inseriti nel body del pacchetto HTTP (POST)

```
<form action="myServlet" method="post">
    First name: <input type="text" name="firstname"/><br/>
    Last name: <input type="text" name="lastname"/>
</form>
```

```
public class MyServlet extends HttpServlet
{
    public void doPost(HttpServletRequest rq, HttpServletResponse rs)
    {
        String firstname = rq.getParameter("firstname");
        String lastname = rq.getParameter("lastname");
    }
}
```

Echo: reading all request parameters (REchoBase.zip)

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                     throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("text/plain");

    Enumeration<String> parameterNames = request.getParameterNames();

    while (parameterNames.hasMoreElements()) {

        String paramName = parameterNames.nextElement();
        out.write(paramName);
        out.write(" = \n");

        String[] paramValues = request.getParameterValues(paramName);
        for (int i = 0; i < paramValues.length; i++) {
            String paramValue = paramValues[i];
            out.write("\t" + paramValue);
            out.write("\n");
        }
        out.write("\n");
    }
    out.close();
}
```

Altri aspetti di request (REchoBase.zip)

- HttpRequest espone anche il metodo **InputStream getInputStream();**
- Consente di leggere il body della richiesta (ad esempio dati di post)

```
public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException
{
    PrintWriter out = response.getWriter();
    InputStream is = request.getInputStream();
    BufferedReader in =
        new BufferedReader(new InputStreamReader(is));
    out.println("<html>\n<body>");
    out.println("Contenuto del body del pacchetto: ");
    while ((String line = in.readLine()) != null)
        out.println(line)
    out.println("</body>\n</html>");
}
```

Deployment Descriptor

- Un'applicazione Web deve essere installata e questo processo prende il nome di **deployment descriptor (DD)**
- Il deployment comprende:
 - La definizione del runtime environment di una Web Application
 - La mappatura delle URL sulle Servlet
 - La definizione delle impostazioni di default di un'applicazione, ad es. welcome page e pagine di errore
 - La configurazione delle caratteristiche di sicurezza dell'applicazione

web.xml (API 4.0)

- È un file di configurazione (in formato XML) che contiene una serie di elementi descrittivi
 - Descrive la struttura dell'applicazione Web
- Contiene 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:**
 - **@WebServlet**
 - **@ServletFilter**
 - **@WebListener**
 - **@WebInitParam**

Mappatura Servlet-URL

- Esempio di descrittore con mappatura:

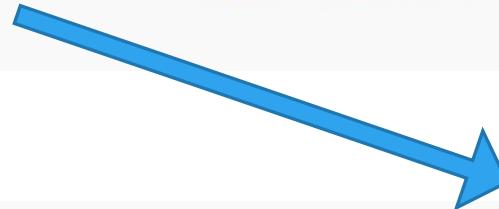
```
<web-app>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>myPackage.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myURL</url-pattern>
  </servlet-mapping>
</web-app>
```

- 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>

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)**

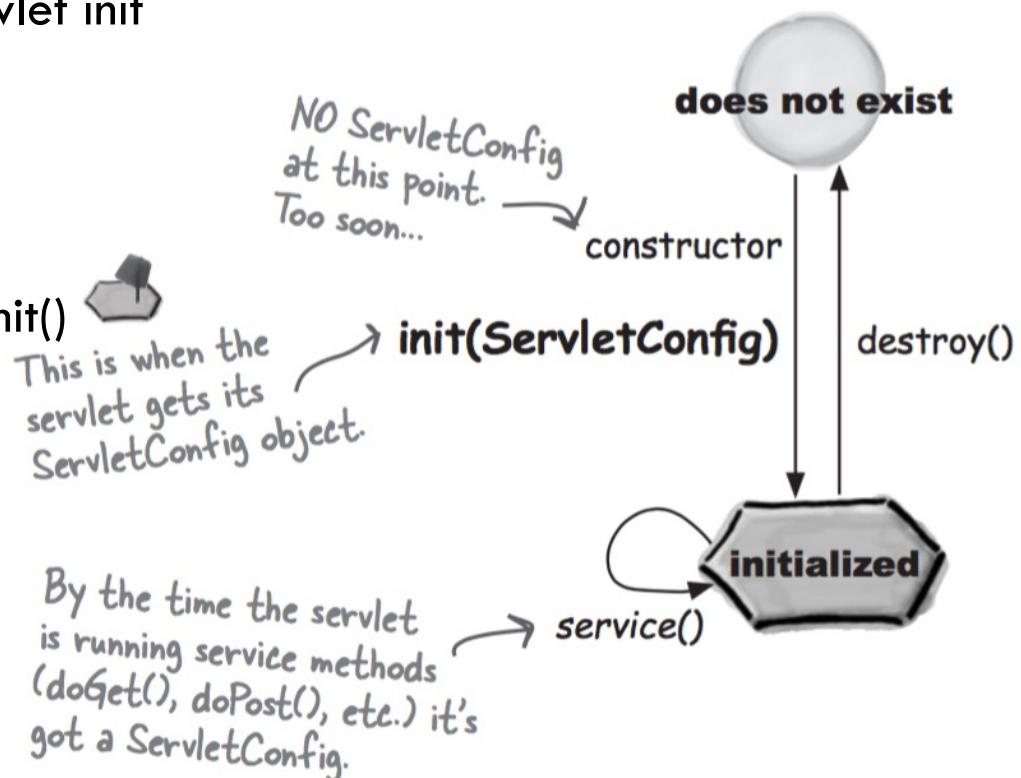
Esempio di parametro
di configurazione

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

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

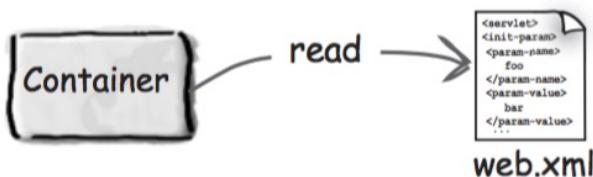
You can't use Servlet init parameters until the Servlet is initialized

- 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

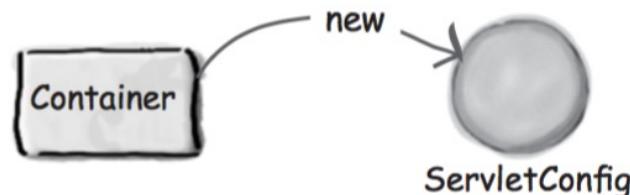


The Servlet init parameters are read only ONCE

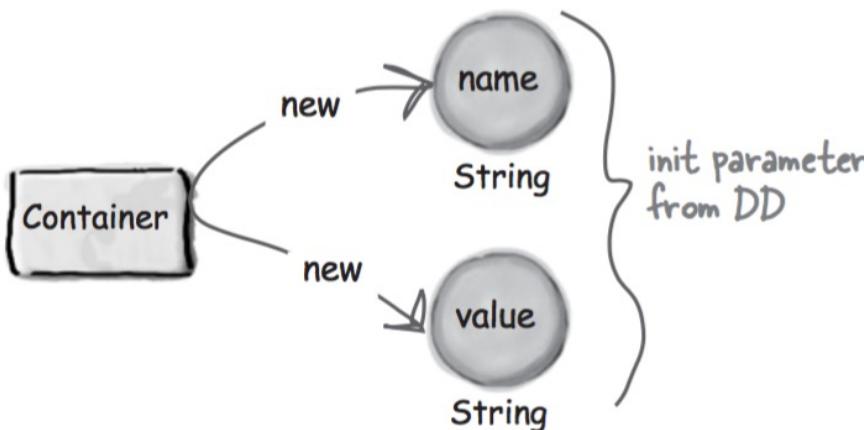
- ① Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



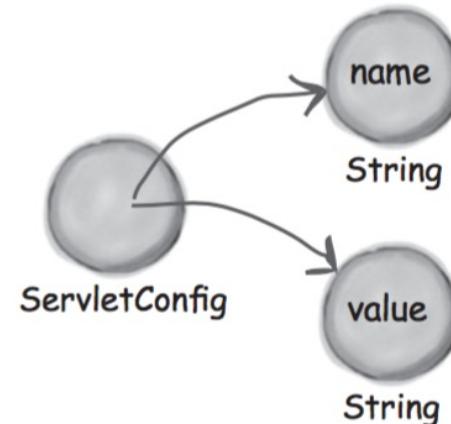
- ② Container creates a new ServletConfig instance for this servlet.



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

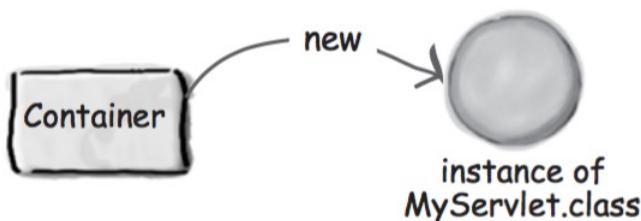


- ④ Container gives the ServletConfig references to the name/value init parameters.

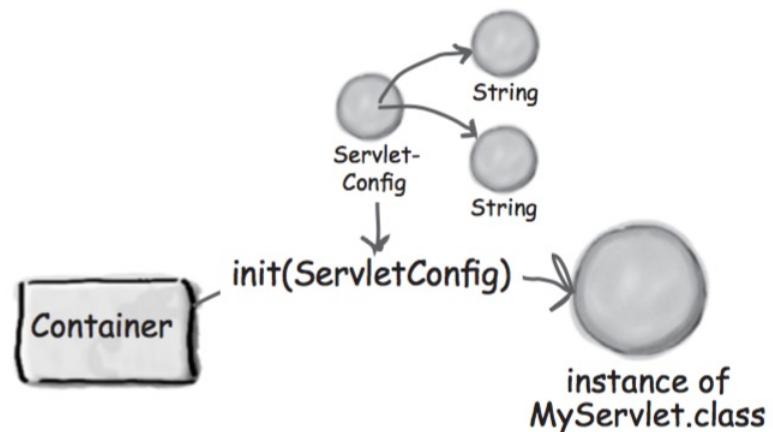


The Servlet init parameters are read only ONCE (2)

- 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

- Estendiamo il nostro esempio rendendo parametrico il titolo della pagina HTML e la frase di saluto:

```
<web-app>
  <servlet>
    <servlet-name>HelloServ</servlet-name>
    <servlet-class>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>HelloServ</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

```
@WebServlet(name = "/HelloServ", urlPatterns = { "/hello" },
  initParams = {@WebInitParam(name = "title", value = "Hello Page"),
    @WebInitParam(name = "greeting", value = "Ciao") })
```

HelloServlet parametrico

- Ridefiniamo quindi anche il metodo init(): memorizziamo i valori dei parametri in due attributi

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

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

Notare l'effetto della
mappatura tra l'URL hello e la servlet

```
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>"+title+"</title></head>");  
    out.println("<body>"+greeting+" "+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>"

Pagine di errore (cercabirraIV.zip)

```
<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>
```

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

- Si possono gestire anche le eccezioni:

```
<error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/error.html</location>
</error-page>
```

- Dalle Servlet 3.0:

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

Servlet context

- Ogni Web application esegue in un **contesto**:
 - corrispondenza 1:1 tra una Web application e suo contesto
- L'interfaccia **ServletContext** è la vista della Web application (del suo contesto) da parte della Servlet
- Si può ottenere un'istanza di tipo ServletContext 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 (serializzazione/deserializzazione)

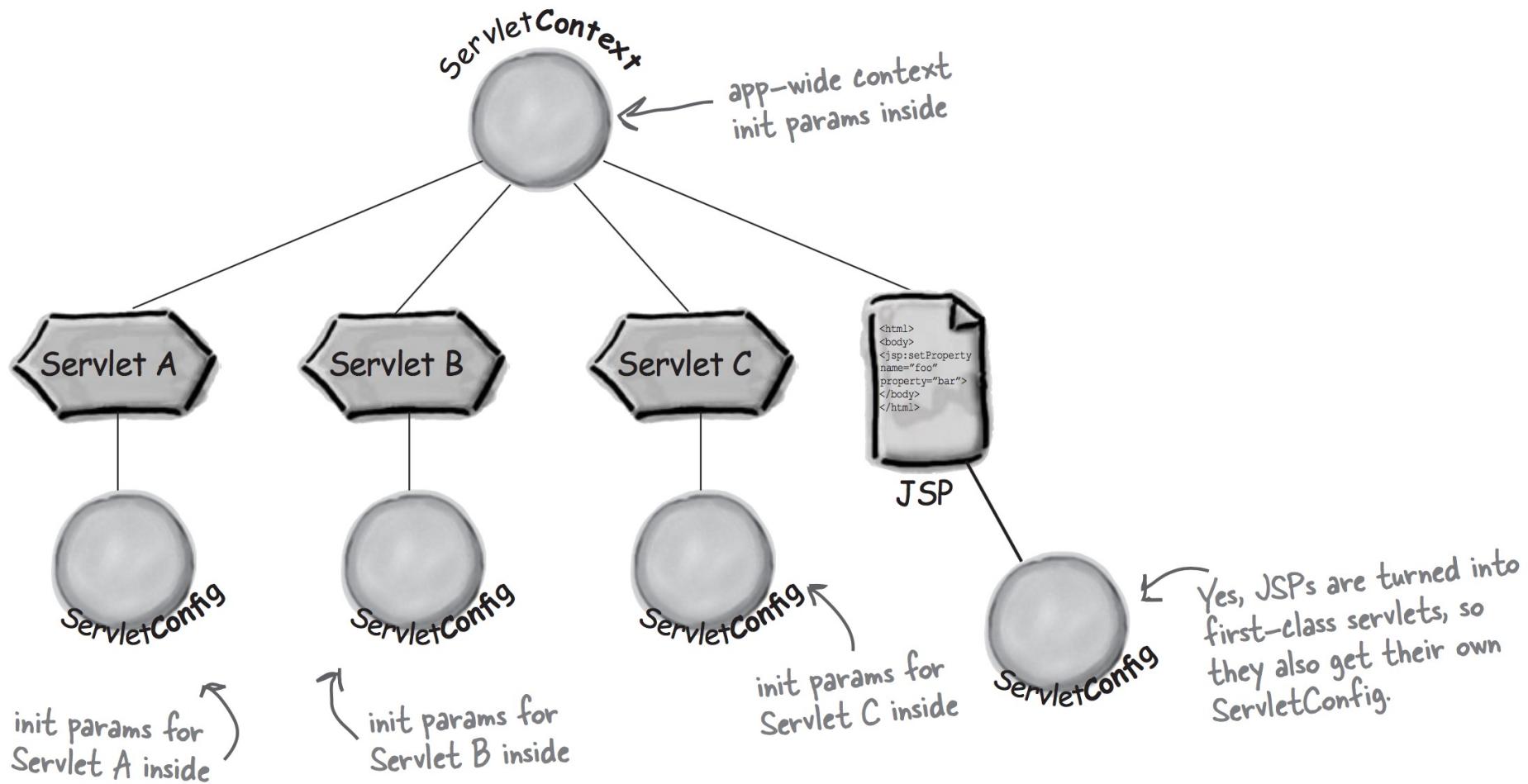
scrittura

```
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

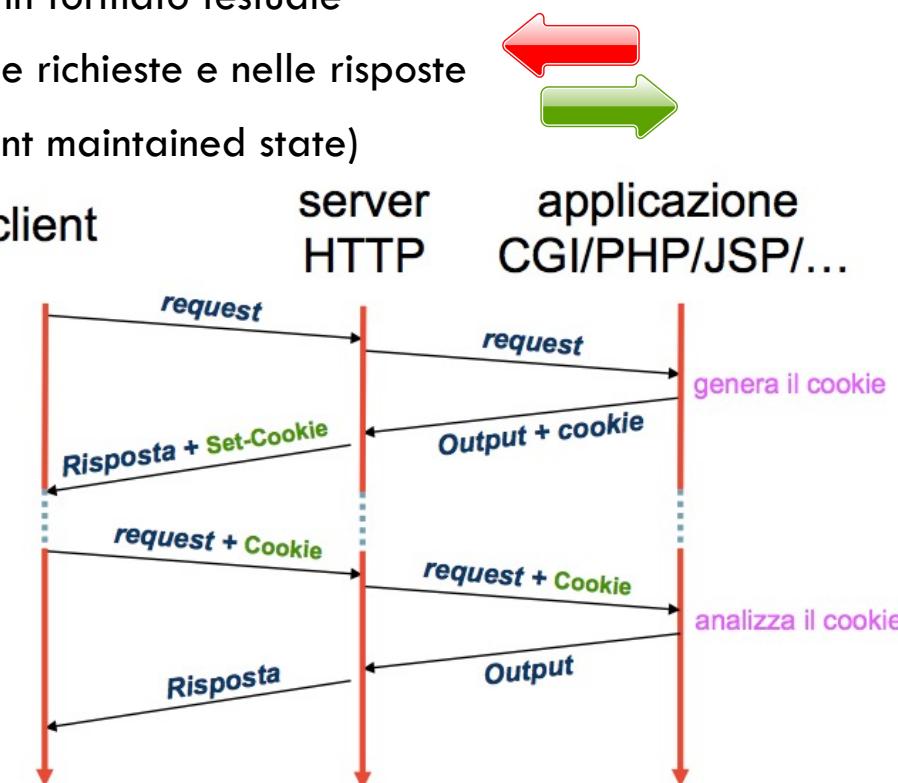
Session Tracking and e-Commerce

- **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 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)
- Sono spesso considerati un fattore di rischio



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

The image displays two side-by-side screenshots of Microsoft Internet Explorer windows, both showing search results for the query "servlets jsp book" on the AltaVista search engine.

Top Window (Left): This window shows the AltaVista homepage. A red circle highlights the top banner advertisement for "POINT, CLICK ... EARN FREE STUFF!" which includes a "go!" button. Below the banner, the AltaVista search interface is visible, featuring a search bar and various navigation links like "Search Home", "Comparison Shop", and "Channels".

Bottom Window (Right): This window shows the search results page for the query. A large red circle highlights the search bar at the top, which contains the query "servlets jsp book". The results page features a prominent yellow banner for "JavaServer Pages & Java Servlets" with the text "HASSLE-FREE". The AltaVista search interface is also visible below the banner.

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);
```

Sending Cookies to Browser

- Standard approach:

```
Cookie c = new Cookie("name", "value");
c.setMaxAge(...); // Means cookie persists on disk
// Set other attributes
response.addCookie(c);
```

- Simplified approach (use LongLivedCookie class):

```
import javax.servlet.http.*;
public class LongLivedCookie extends Cookie {
    private static final long serialVersionUID = 1L;
    public static final int SECONDS_PER_YEAR = 60 * 60 * 24 * 365;
    public LongLivedCookie(String name, String value) {
        super(name, value);
        setMaxAge(SECONDS_PER_YEAR);
    }
}
```

Reading Cookies from Browser

- Standard approach:

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        Cookie c = cookies[i];
        if (c.getName().equals("someName")) {
            doSomethingWith(c);
            break;
        }
    }
}
```

Simple Cookie-Setting Servlet

```
public class SetCookies extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        for(int i=0; i<3; i++) {
            Cookie cookie = new Cookie("Session-Cookie-" + i, "Cookie-Value-S" + i);
            response.addCookie(cookie);
            cookie = new Cookie("Persistent-Cookie-" + i, "Cookie-Value-P" + i);
            cookie.setMaxAge(3600);
            response.addCookie(cookie);
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body><a href='ShowCookies'>Show Cookies</a>");
        out.println("</body></html>");
    }
}
```

Simple Cookie-Viewing Servlet

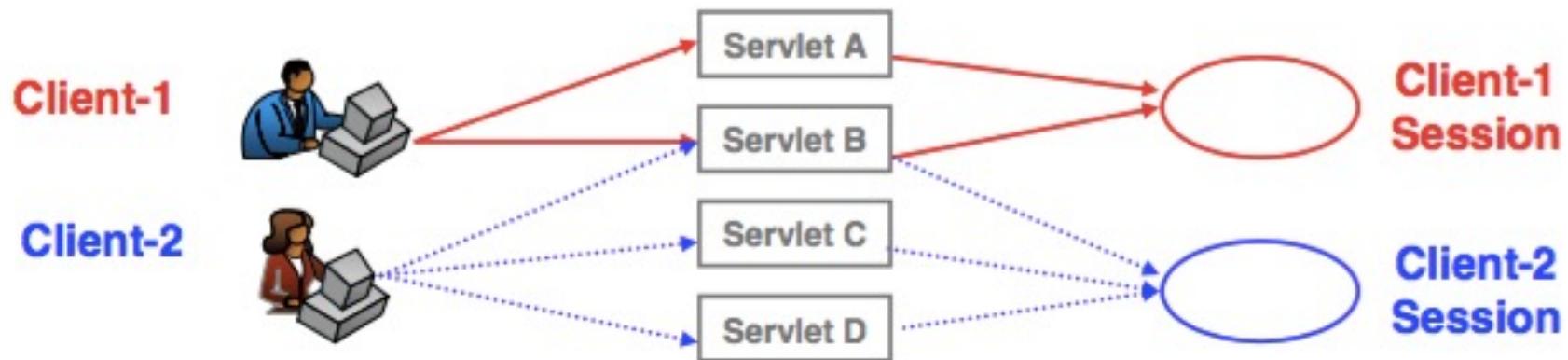
```
public class ShowCookies extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<html><body>" +  
            "<h1>Active Cookies </h1>" +  
            "<table border=1>" +  
            "<tr><th>Cookie Name</th><th>Cookie Value</th></tr>");  
  
        Cookie[] cookies = request.getCookies();  
        if (cookies != null) {  
            Cookie cookie;  
            for(int i=0; i<cookies.length; i++) {  
                cookie = cookies[i];  
                out.println("<tr>" +  
                    "<td>" + cookie.getName() + "</td>" +  
                    "<td>" + cookie.getValue() + "</td>");  
            }  
        }  
        out.println("</table></body></html>");  
    }  
}
```

```
public class ClientAccessCounts extends HttpServlet {  
  
    public static 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);  
    }  
  
    public 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) { }  
  
        LongLivedCookie c = new LongLivedCookie("accessCount", String.valueOf(count+1));  
        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>");  
    }  
}
```

AccessCount Servlet with Cookies

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)

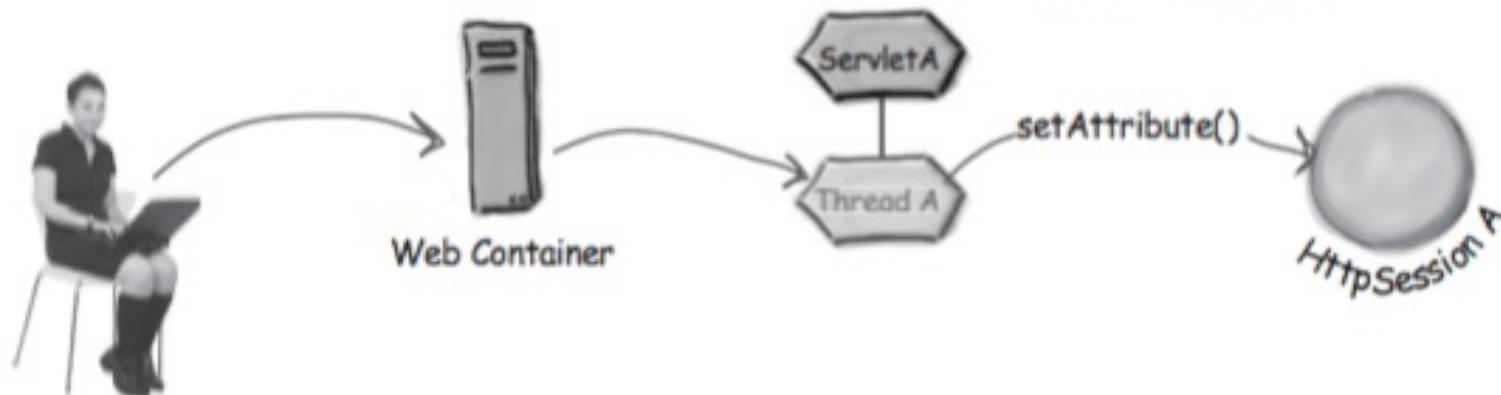


①

Diane selects "Dark" and hits the submit button.

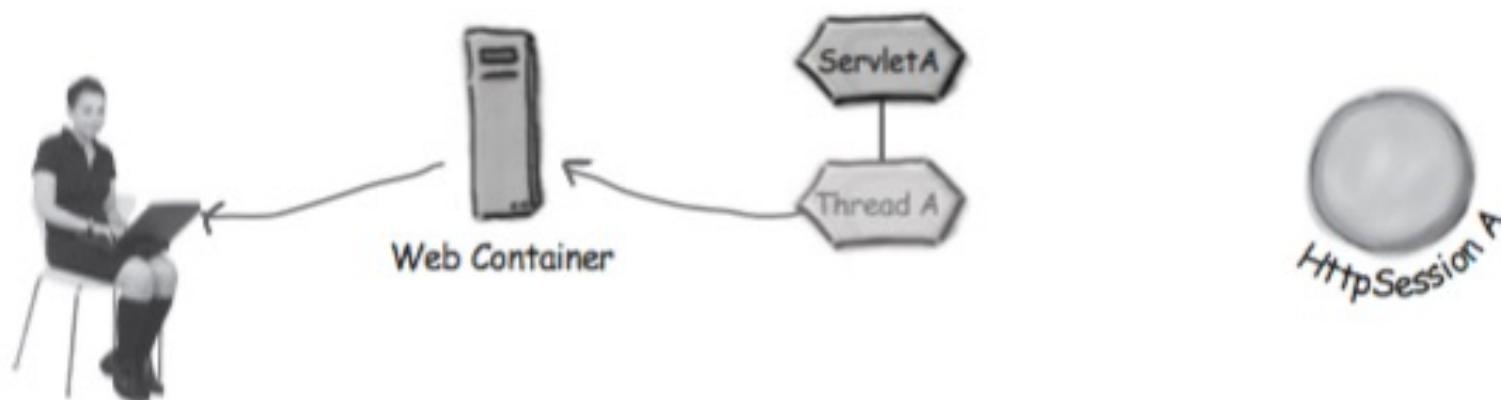
The Container sends the request to a new thread of the BeerApp servlet.

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?"

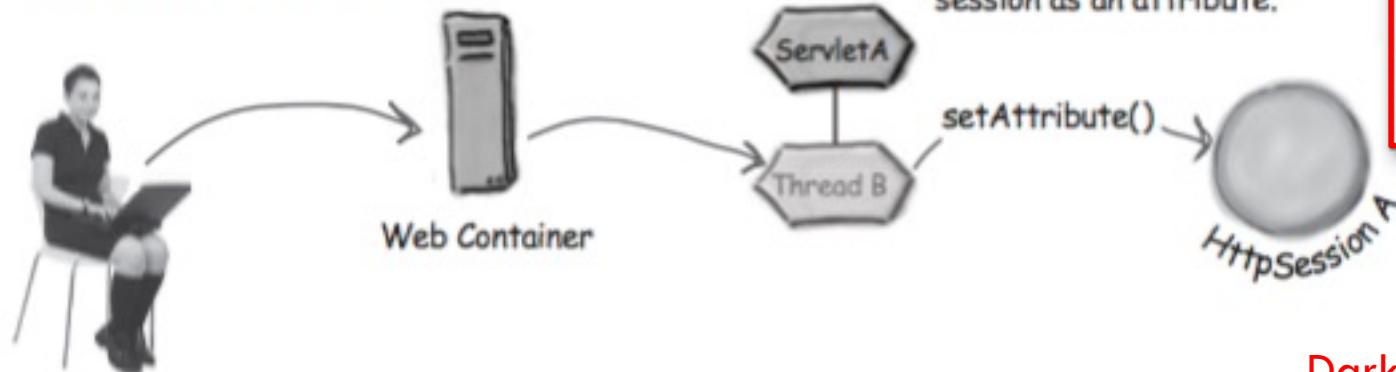


③

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.

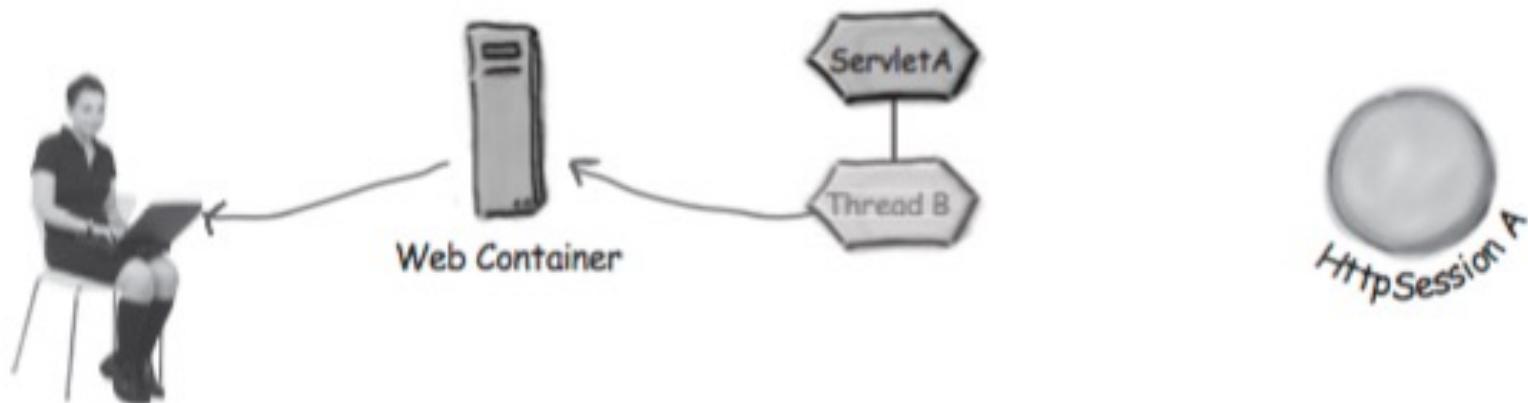


Same client
Same servlet
Different request
Different thread
Same session

Dark and Expensive

④

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

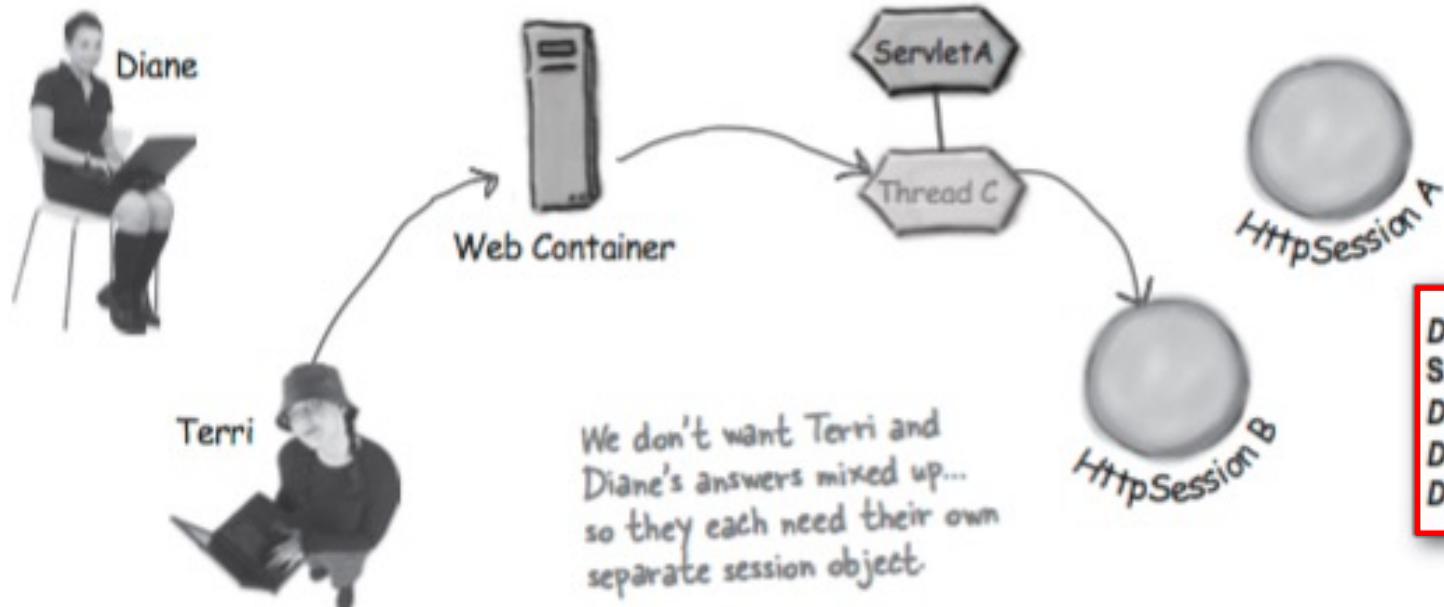


⑤

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").



Different client
Same servlet
Different request
Different thread
Different session

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()** 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**
- Uso del metodo in una Servlet:
HttpSession session = request.getSession(true);
- Per recuperare l'ID della sessione:

null! ?

```
HttpSession ssn = request.getSession();
if(ssn != null){
    String ssnId = ssn.getId();
    System.out.println("Your session Id is : "+ ssnId);
}
```

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 nella 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
 - Fails for bookmarks and links from other sites

URL-Rewriting (2)

- È buona prassi codificare sempre le URL generate 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

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);
```

Hidden form field

- Idea:

```
<input type="hidden" name="session" value="7199A6...">
```

- Advantage
 - Works even if cookies are disabled or unsupported
- Disadvantages
 - Lots of tedious processing
 - All pages must be the result of form submissions

Hidden form field (2)

- In the Servlet:

```
Map<String, HttpSession> sessions = (Map<String, HttpSession>)
    request.getServletContext().getAttribute("sessionmap");
if(sessions == null) {
    sessions = new HashMap<String, HttpSession>();
    request.getServletContext().setAttribute("sessionmap", sessions);
}
HttpSession session = request.getSession();
sessions.putIfAbsent(session.getId(), session);
```

- In the JSP:

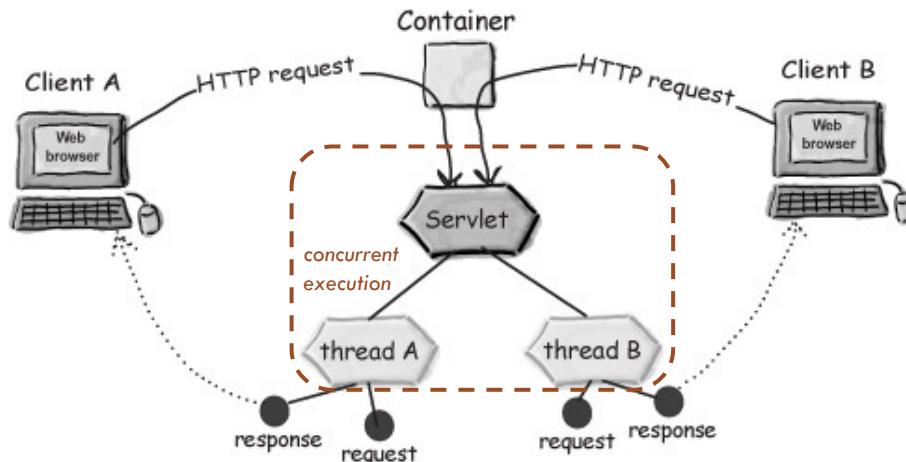
```
<form method="post" action="ShowSession">
    <input type="hidden" name="session" value="<%request.getSession().getId()%>">
    <!-- ... -->
    <br>
    <input type="submit" value="Add">
</form>
```

- In the Servlet (ShowSession):

```
String sessionID = request.getParameter("session");
if(sessionID != null) {
    Map<String, HttpSession> sessions = (Map<String, HttpSession>)
        request.getServletContext().getAttribute("sessionmap");
    if(sessions != null) {
        HttpSession session = sessions.get(sessionID);
        /* ... */
    }
}
```

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



Accumulating a list of user data

...

```
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);
}
...
```

```
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>" +
        "<table border='1'>" +
        "<tr>" +
        "  <th>Info Type</th><th>Value</th>" +
        "</tr><tr>" +
        "  <td>ID</td><td>" + session.getId() + "</td>" +
        "</tr><tr>" +
        "  <td>Creation Time</td><td>" + new Date(session.getCreationTime()) + "</td>" +
        "</tr><tr>" +
        "  <td>Time of Last Access</td><td>" + new Date(session.getLastAccessedTime()) + "</td>" +
        "</tr><tr>" +
        "  <td>Number of Previous Accesses</td><td>" + accessCount + "</td>" +
        "</table>" +
        "</body></html>");
}
}
```

AccessCount Servlet with Session (Sessions.zip)

```
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);

String link = response.encodeURL("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>"
+ "<table border='1'>"
// ...
+ "</tr><tr>"
+ "<td>Number of Previous Accesses</td><td>" + accessCount + "</td>"
+ "</table>"
+ "<br>Link: " + link
+ "<br><a href='"+ link +">Refresh</a>"
+ "</body></html>");
}

}
```

AccessCount Servlet with Session and URLRewriting (SessionRewrite.zip)

Attenzione: scope DIFFERENZIATI (scoped objects)

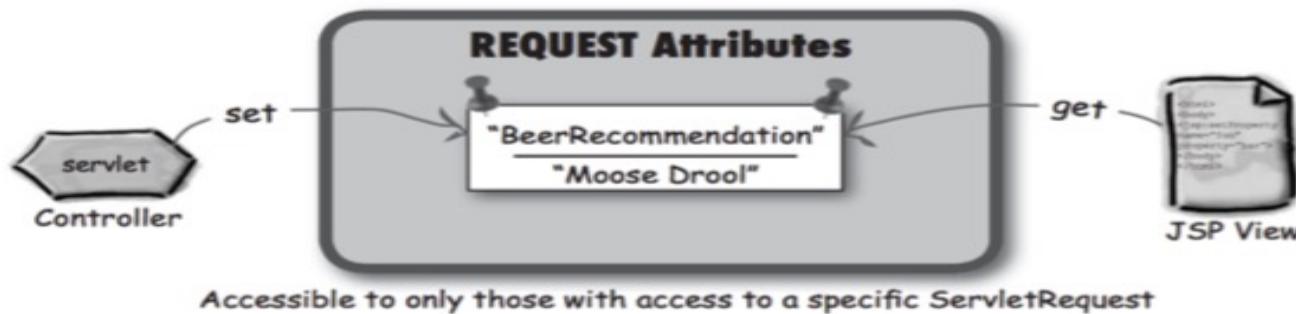
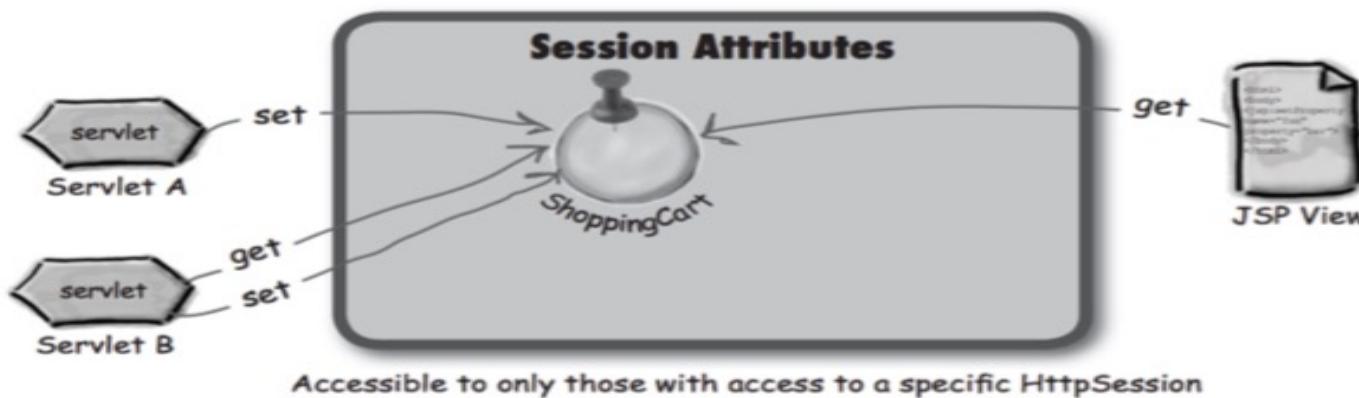
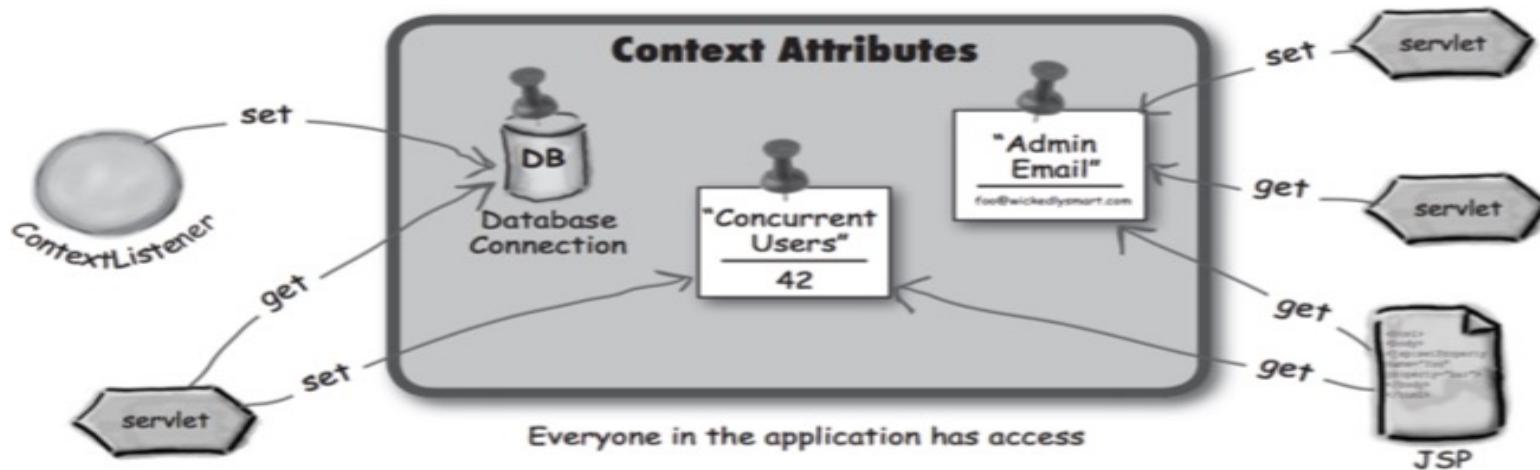
- Gli oggetti di tipo `HttpServletRequest` , `HttpSession` , `ServletContext` forniscono metodi per immagazzinare e ritrovare oggetti nei loro rispettivi ambiti (scope)
- Lo scope è definito dal **tempo di vita (lifespan)** e dall'**accessibilità** da parte delle Servlet

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();**

The Three Scopes: Context, Request, and Session



Ridirezione del browser

- È possibile inviare al browser una risposta che lo forza ad accedere ad un'altra pagina/risorsa (**ridirezione**)
- Possiamo ottenere agendo sull'oggetto **response** invocando il metodo
 - **public void sendRedirect(String url);**
- Scenario: *When the Servlet shows the client a JSP with the results, and if you don't want the user to be able to refresh and re-submit the form, then use a sendRedirect*
- **sendRedirect** can be used to communicate between two servlets present on the same or different server. The output will be the same as the **Request Dispatcher forward** but the URL of the page will be changed to the redirected page

SendRedirect

- Es:

```
String name=request.getParameter("name");
response.sendRedirect("https://www.google.co.in/#q="+name);
```

```
String userAgent = request.getHeader("User-Agent");
if (userAgent != null && userAgent.contains("MSIE")) {
    response.sendRedirect("http://home.mozilla.com");
} else {
    response.sendRedirect("http://www.microsoft.com");
}
```

SendRedirect example (Redirect.zip)

index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <body>
        <h2>Servlet SendRedirect Example</h2>
        <form id="loginFormId" method="post" action="loginServlet">
            Username:<input id="userInput" type="text" name="username" required/>
            <br>
            Password:<input id="passInput" type="password" name="password" required/>
            <br>
            <input id="btn" type="submit" value="Login" />
            <%if(request.getAttribute("error") != null) {%>
                <p style='color: red;'>You are not an authorized user!</p>
            <% } %>
        </form>
    </body>
</html>
```

SendRedirect example (2)

```
@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);
    }
    /* ... */
}

@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 style='color: green;'>");
        out.println("Congratulations! " + param1 + ", You are an authorised login!");
        out.println("</p></body></html>");
    }
    /* ... */
}
```

The diagram illustrates the flow of control between two Java Servlets. A blue arrow originates from the `response.sendRedirect("welcomeServlet");` line in the `doGet` method of the `Login` servlet. This arrow points to the `Welcome` servlet, which is shown in a separate code block below. The `Welcome` servlet contains its own `doGet` method implementation.

Includere una risorsa (include)

- Per includere una risorsa si ricorre a un oggetto di tipo **RequestDispatcher** che può essere richiesto al contesto indicando la risorsa da includere
- Si invoca quindi il metodo **include** passando come parametri **request** e **response** che vengono così condivisi con la risorsa inclusa
 - Se necessario, l'URL originale può essere salvato come un attributo di request

può essere anche una pagina JSP

```
RequestDispatcher dispatcher =  
    getServletContext().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:in questo caso la 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** da request passando come parametro il nome della risorsa
- Si invoca quindi il metodo **forward** passando anche in questo caso **request e response**

può essere anche una pagina JSP

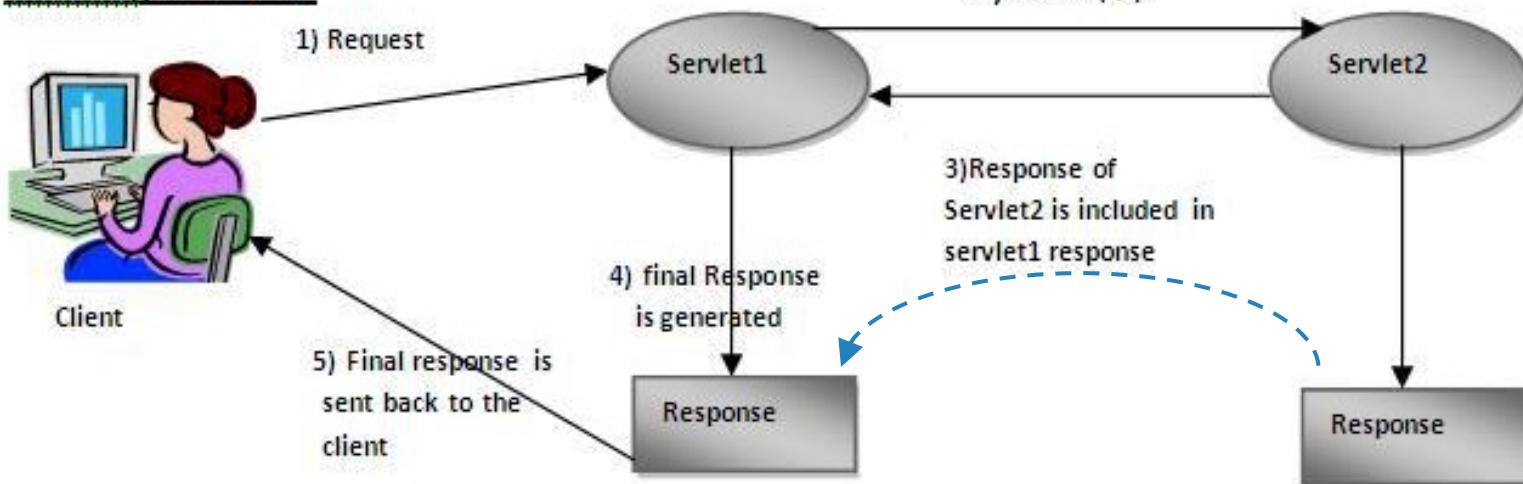
```
RequestDispatcher dispatcher =  
    getServletContext().getRequestDispatcher("/inServlet");  
dispatcher.forward(request, response);
```

Include e forward

- **Scenario (include):** When the current Servlet has done some of its job and is using another Servlet (or JSP) to help with display or other work. If the Servlet using the RequestDispatcher has already written to the response, then you will need to use the include to send control to another Servlet, not the forward. Also, if you want to spread the display of a page across multiple pages, use an Include for each part of a page
 - **Example:** Servlet handles a request from the client and does the necessary work. It then uses RequestDispatcher#`include` to add a **Header** response to all pages, since the header and banner on a page is the same for all pages on the site. It then includes a second JSP which displays the **content** that is specific to this request. It finally includes a third page that acts as a **Footer** for all pages on the site
- **Scenario (forward):** When the current Servlet is done its job, and hasn't done anything with the response, you forward the request and response to another Servlet (or JSP) to finish any work and to display results
 - **Example:** A Servlet retrieves information from a database and stores it in the request object. It then forwards the request to a JSP to display the data. User presses refresh, and the control goes to the Servlet, which again retrieves the data from the database and puts it in the request, forwards to the JSP, and the JSP sees the data

Include e forward

include() method



forward() method:

