CORSO DI LAUREA IN INFORMATICA

TECNOLOGIE SOFTWARE PER IL WEB

SERVLET

a.a 2022/23

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
- Eseguono direttamente in un Web Container
- In termini pratici sono classi che derivano dalla classe **HttpServlet**
  - HttpServlet implementa vari metodi che possiamo ridefinire

NOTA: Tomcat 10.1.x fa riferimento alla versione 6.0 per le Servlets, 3.1 per le JSP e 5 per l'Expression Language (http://tomcat.apache.org/whichversion.html)

Documentazione per le Jakarta servlets 6.0:

https://jakarta.ee/specifications/servlet/6.0/

https://jakarta.ee/specifications/servlet/6.0/apidocs/

Cos'è una Servlet?

- *È una **classe Java** che fornisce risposte a richieste HTTP e per far ciò ha bisogno di due librerie **servlet-api.jar** e **jsp-api.jar***
- *Dove trovo queste due librerie?*
  - *Distribuzione java?*
  - *IntelliJ o Eclipse?*
  - *Tomcat?* **<===****

Nota per la compilazione ed esecuzione di un qualsiasi programma java (e quindi servlet):

*Se un sorgente java (per es. servlet) utilizza una classe da una **libreria** (per es. HttpServlet presa da servlet-api.jar versione 6) allora:*

1. *Essa deve essere **compilata** avendo a disposizione **quella libreria** (es. servlet-api.jar versione 6) [quindi IntelliJ o qualsiasi **IDE** deve avere accesso ad essa quando compila il sorgente .java]*
2. *Essa deve essere **eseguita** avendo a disposizione la **stessa libreria** (e. servlet-api.jar versione 6) [quindi il **servlet container** (Tomcat 10.1.x) deve avere accesso ad essa (nella cartella Tomcat/lib) quando esegue il compilato .class]*

DOMANDA: posso fare il deploy di una servlet compilata con servlet-api.jar versione 6 su Tomcat 8.5.x ?? Controlla http://tomcat.apache.org/whichversion.html

Esempio di Servlet: Hello world! (non MVC)

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

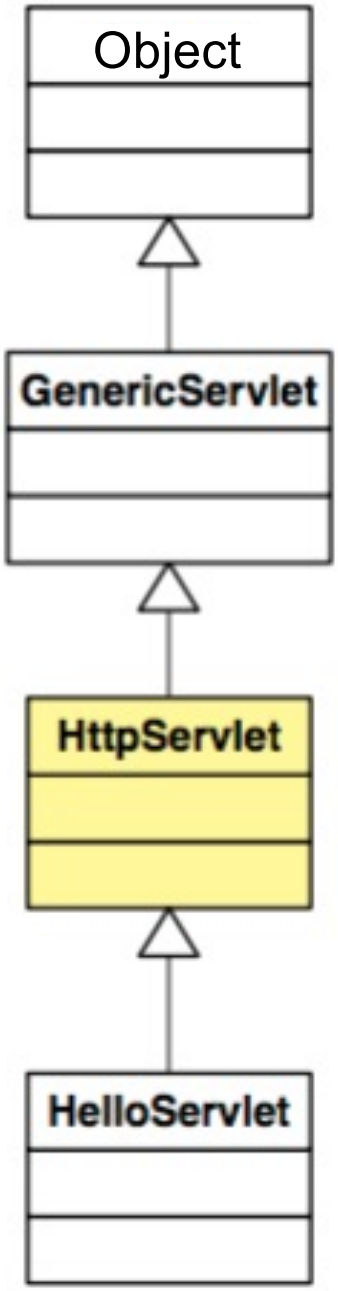
Gerarchie delle Servlet

- 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 richieste HTTP, producendo *response* HTTP
- Abbiamo quindi la catena di ereditarietà mostrata a lato
- Da ora in poi facciamo riferimento solo a HttpServlet
- Le classi che ci interessano sono contenute nel package

  jakarta.servlet.http.*

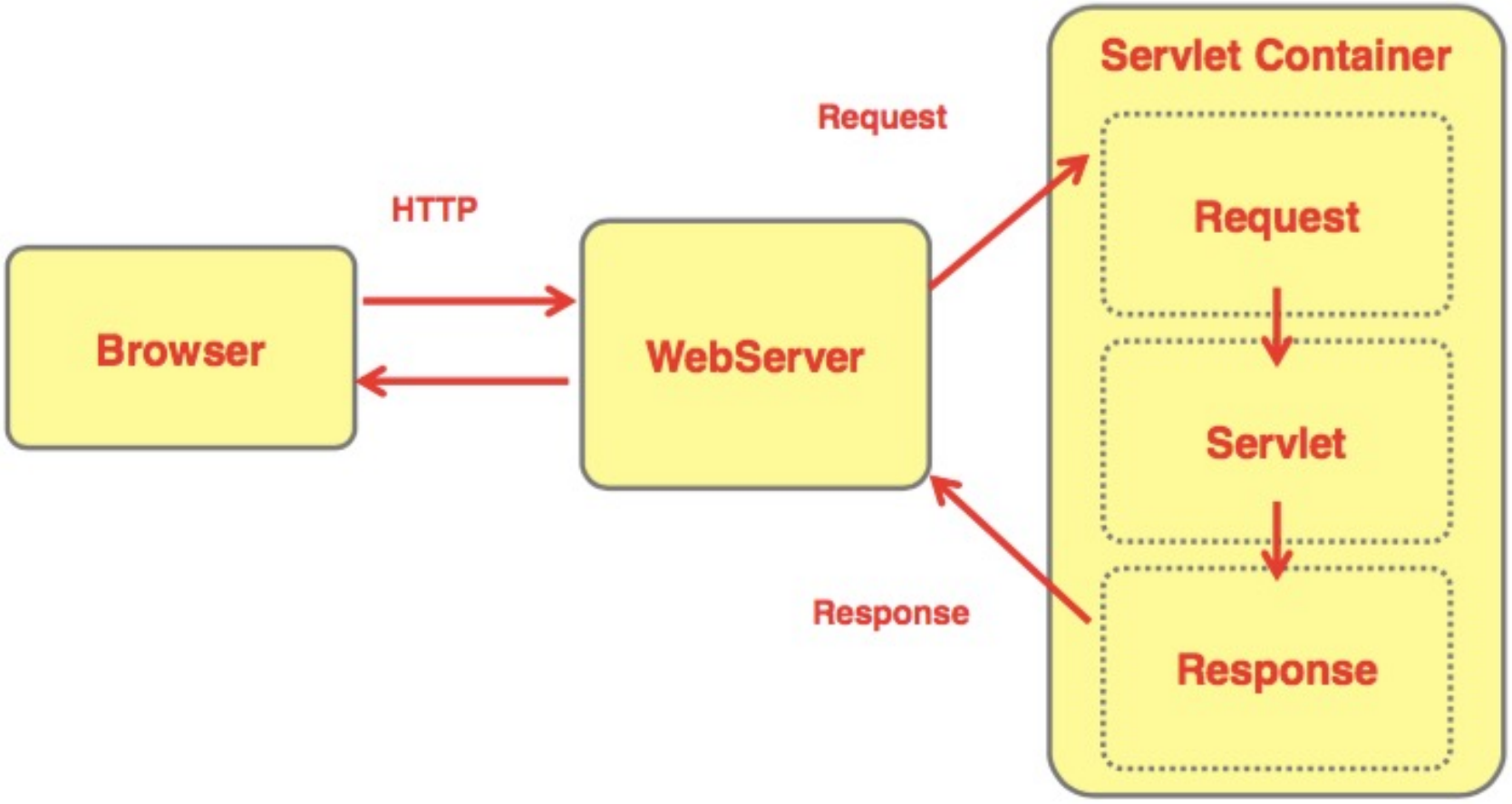
  che è contenuto nella libreria **servlet-api.jar**

  (totalmente descritta in https://jakarta.ee/specifications/servlet/6.0/apidocs)



The request-response model

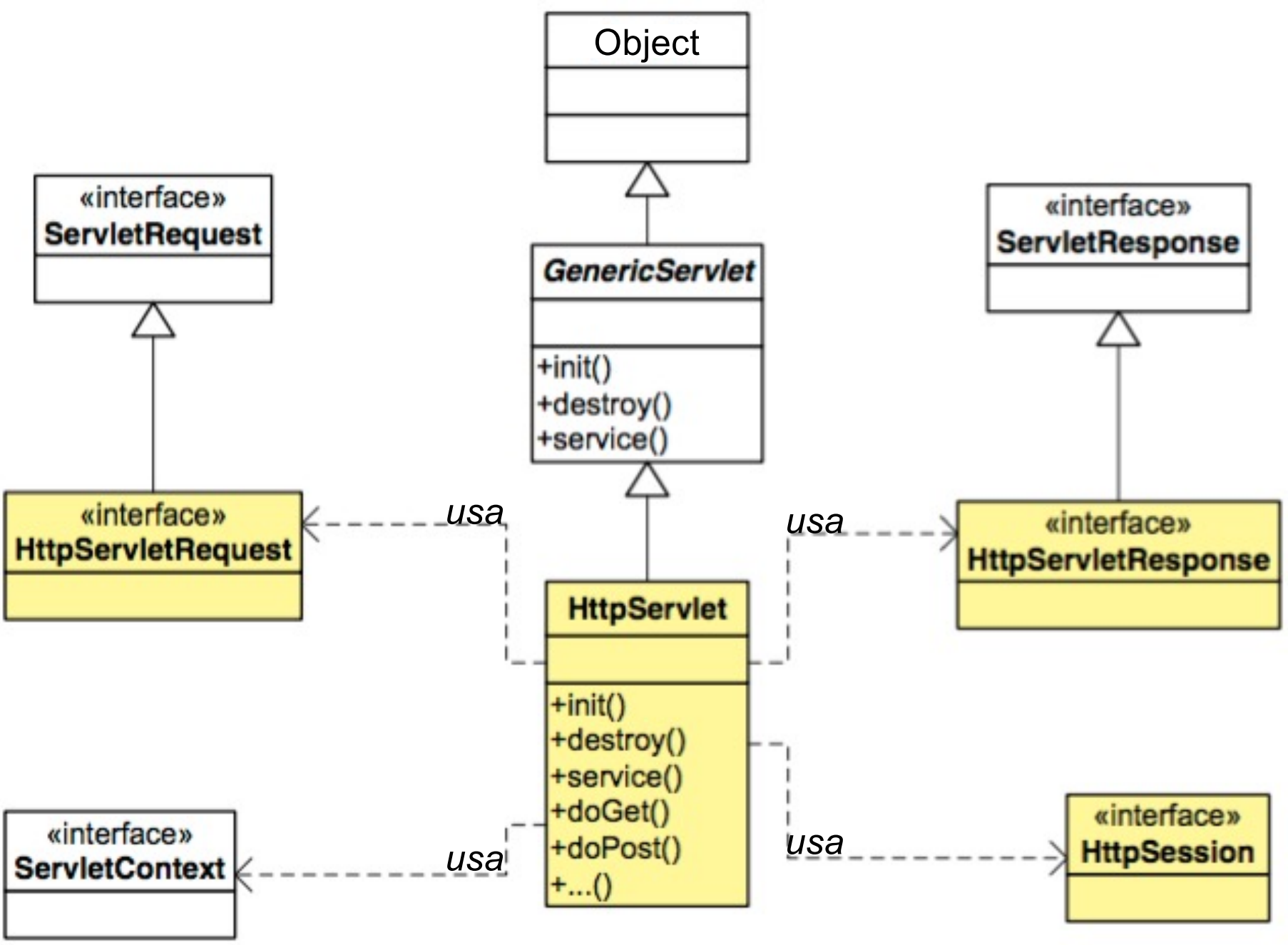
- All'arrivo di una richiesta HTTP il Servlet Container (Web Container) crea un oggetto **request** e un oggetto **response** e li passa alla Servlet:



Request e Response

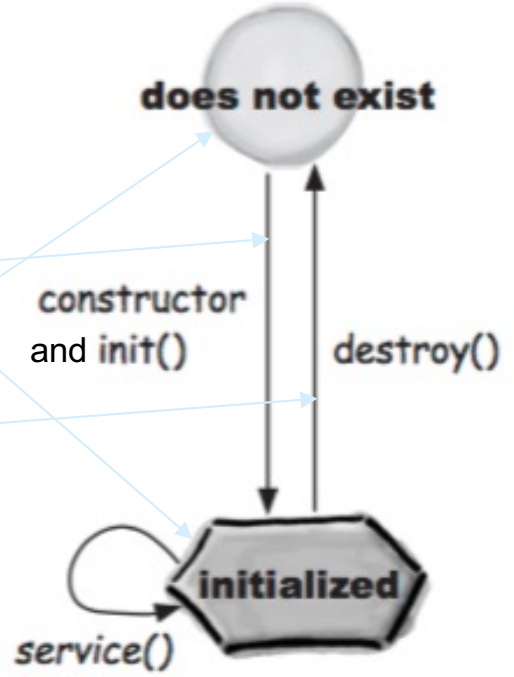
- Gli oggetti di tipo **Request** rappresentano i dati inviati dal Client con la chiamata al Server
- Sono caratterizzati da varie informazioni
  - Chi ha effettuato la Request
  - Quali parametri sono stati passati nella Request
  - Quali header sono stati passati
- Gli oggetti di tipo Response rappresentano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

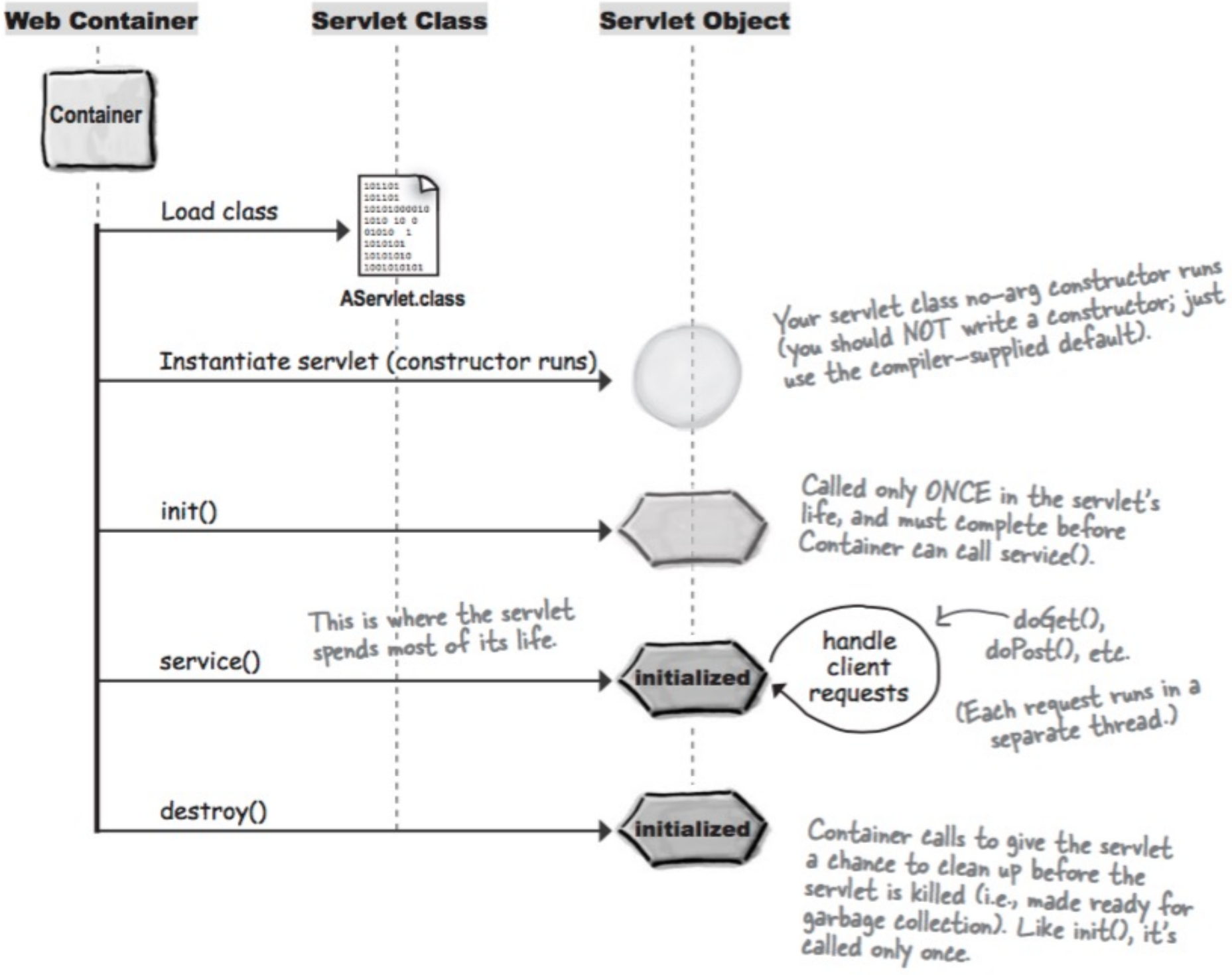


Ciclo di vita delle Servlet

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

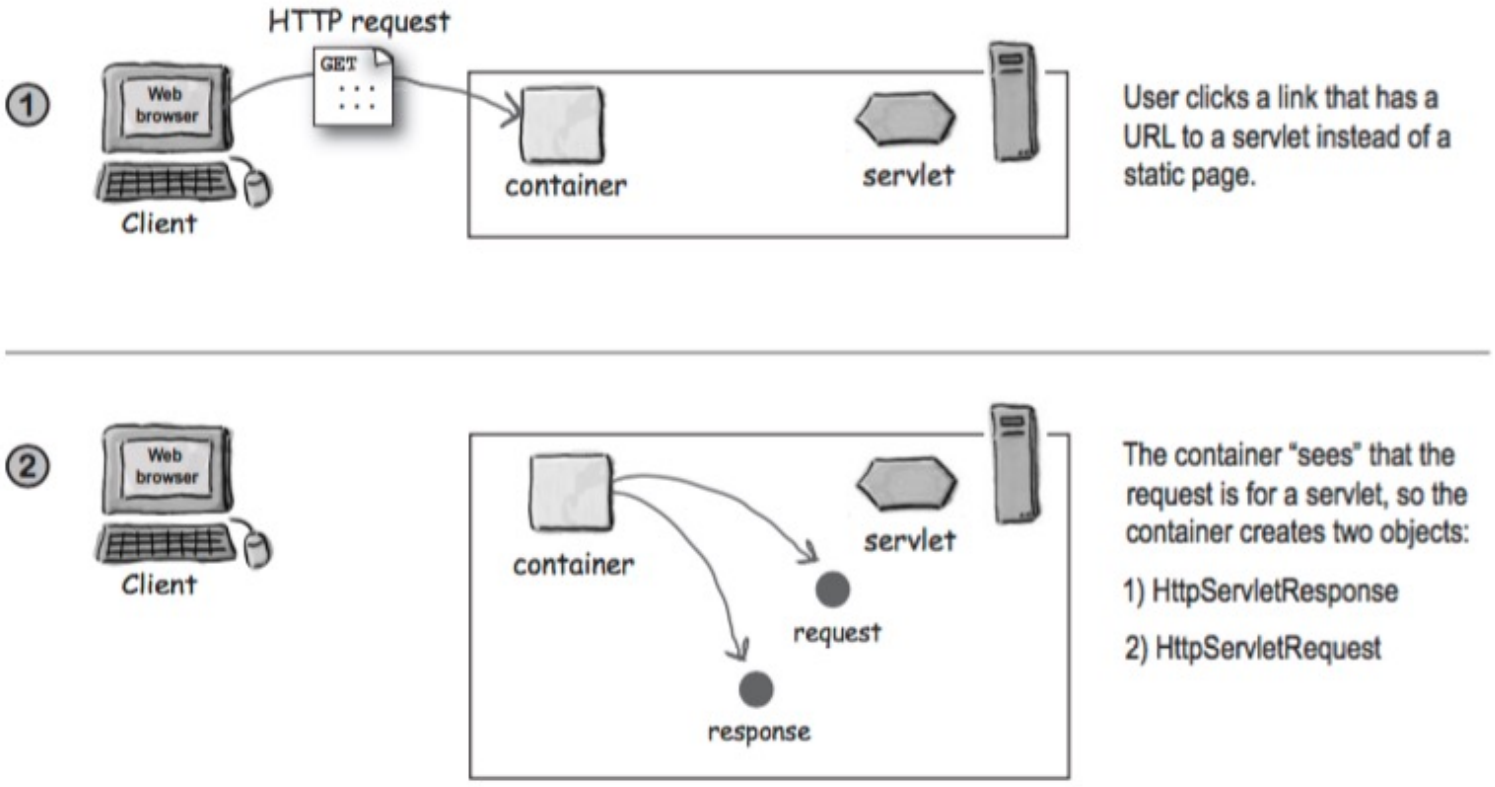


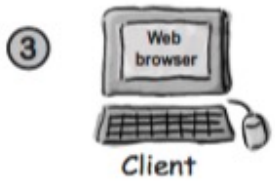
Come leggere il diagramma di stato a destra: lo stato della servlet all'inizio è "non esiste", la ricezione del commando init() la porta nello stato "inizializzato" etc.

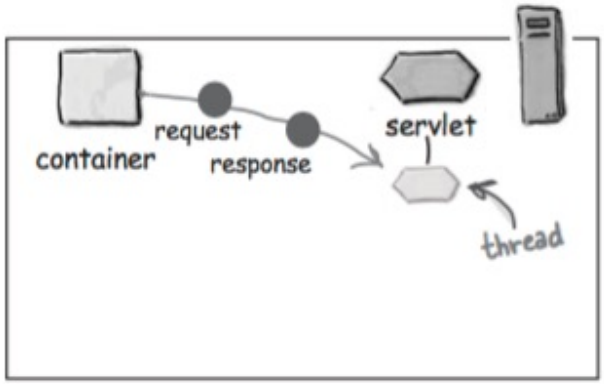

Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
This is where the servlet spends most of its life.
service()
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
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.

Riprendiamo il modello richiesta – risposta dall'ultima lezione

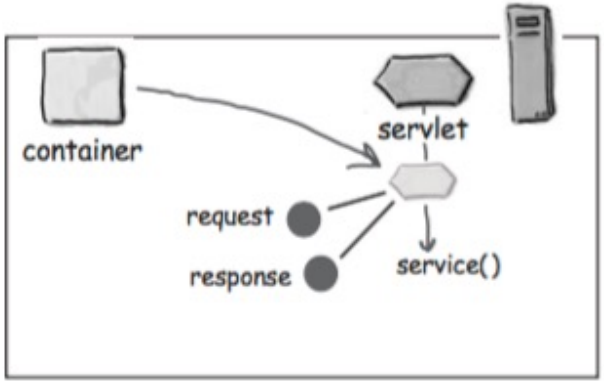
In che stato è la servlet?





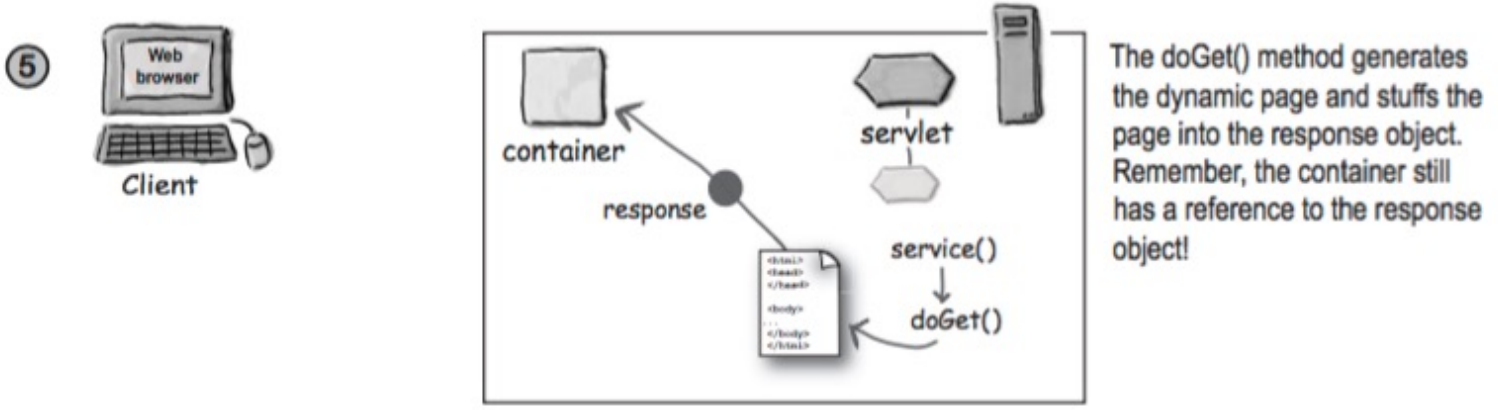


The container finds the correct servlet based on the URL in the request, creates or allocates a thread for that request, and passes the request and response objects to the servlet thread.

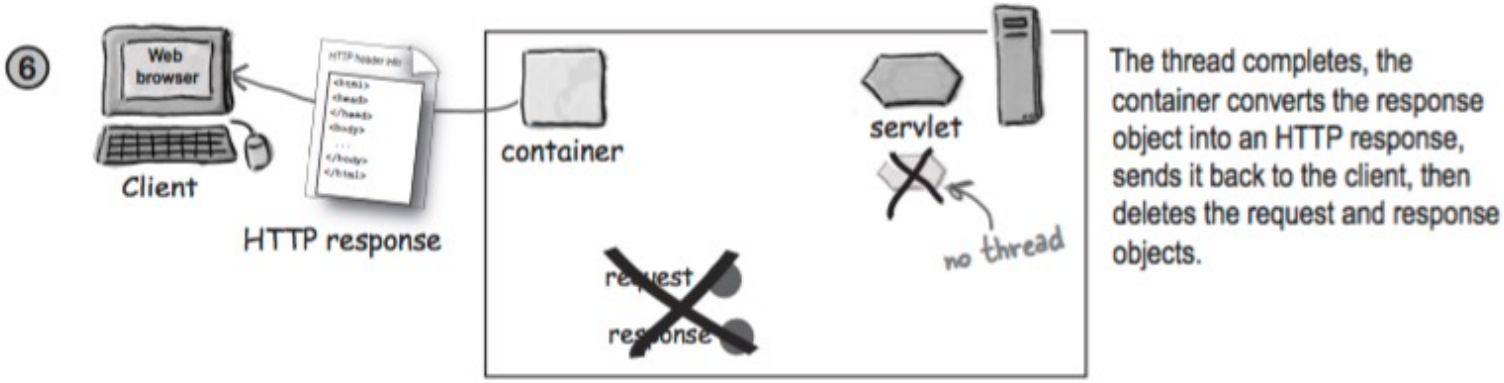


The container calls the servlet's service() method. Depending on the type of request, the service() method calls either the doGet() or doPost() method.

For this example, we'll assume the request was an HTTP GET.



The doGet() method generates the dynamic page and stuffs the page into the response object. Remember, the container still has a reference to the response object!



The thread completes, the container converts the response object into an HTTP response, sends it back to the client, then deletes the request and response objects.

Ma è qui che entra in campo il destroy()? (NO)

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)

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())

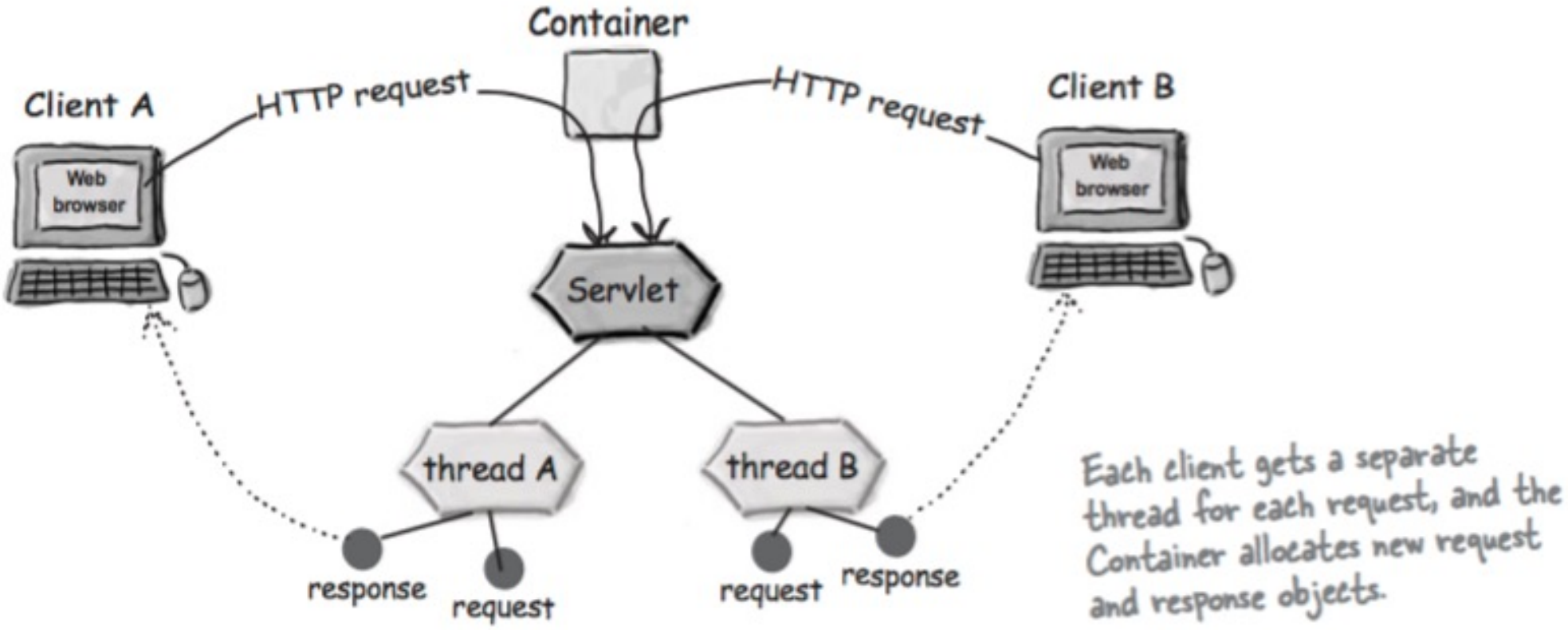
Whichever one(s) you override tells the Container what you support. If you don't override doPost(), for example, then you're telling the Container that this servlet does not support HTTP POST requests.

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 (stesso per destroy alla fine della vita della servlet)
  - 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):
    - Di solito con uso di blocchi di codice synchronized
    - Semafori
    - Mutex (mutua esclusione)

Ogni richiesta in un thread separato

- **Il contenitore esegue più thread per elaborare più richieste a una singola Servlet**
  - e ogni richiesta del client genera una nuova coppia di oggetti richiesta e risposta



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 dalle specifiche 2.4 delle Servlet in poi
  - 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 da parte del Container
  - 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 (possible concorrenza)
- **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, ...)

Anatomia di Hello World basata su tecnologia Servlet

- Usiamo l'esempio "Hello World" per affrontare i vari aspetti della realizzazione di una Servlet
- Importiamo i package di jakarta ed altri necessari
- Definiamo la classe **HelloServlet** che discende da **HttpServlet**
- 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)
     ...
}
```

Ricordarsi di usare jakarta al posto del vecchio javax

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**
- In questo semplice esempio, non ci servono informazioni sulla richiesta e quindi 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*

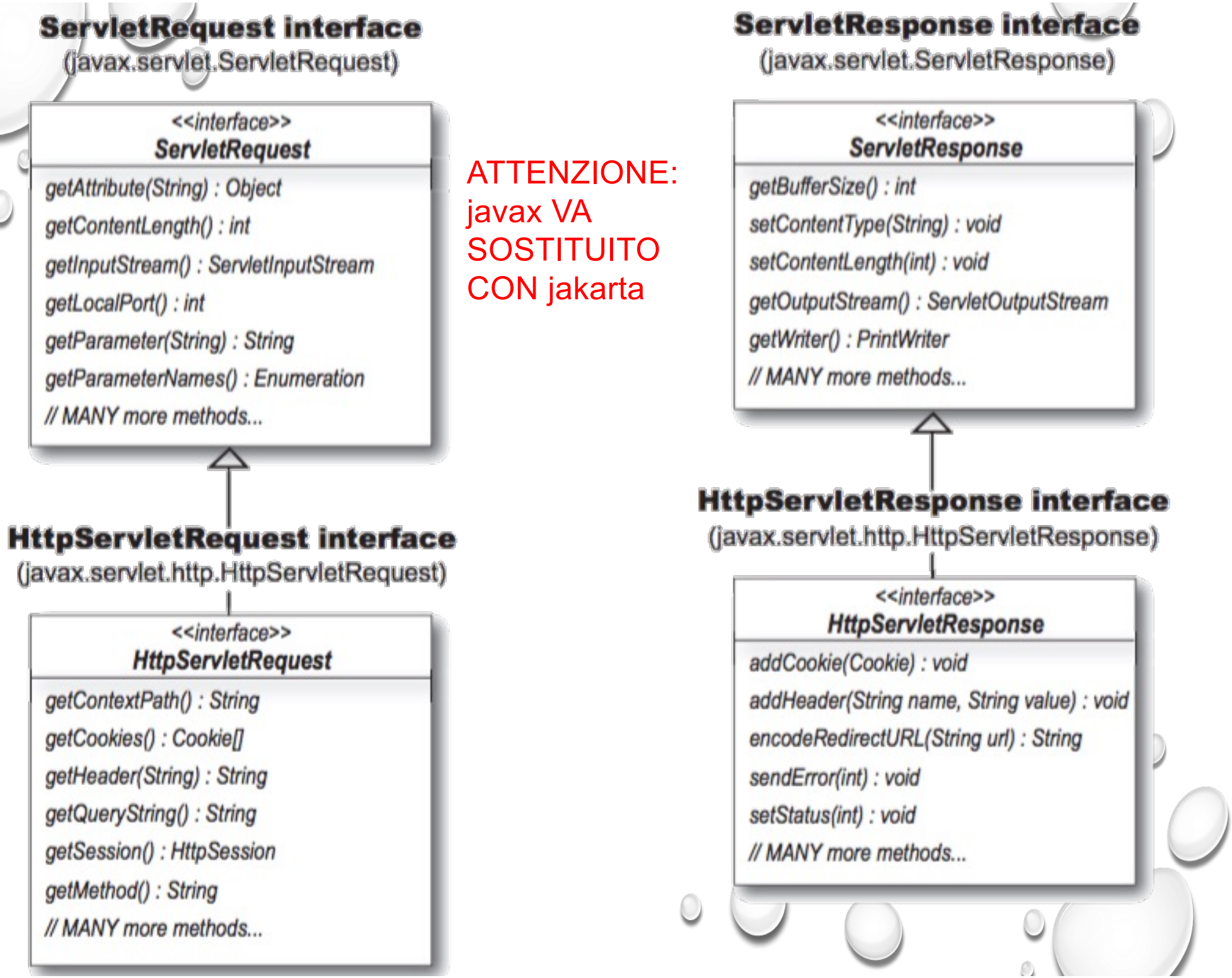
- Una servlet può implementare un metodo doPost che chiama semplicemente doGet
- Gestisce entrambe le richieste GET e POST
- Questo approccio è una buona pratica standard se si vuole che le interfacce HTML abbiano una certa flessibilità nel modo in cui inviano dati alla Servlet
- Attenzione!!! Questi metodi sono intrinsecamente diversi

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

    doGet(request, response);
}
```

L'oggetto response

- https://jakarta.ee/specifications/servlet/6.0/apidocs/jakarta.servlet/jakarta/servlet/http/httpservletresponse
- 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)
- Estende l'interfaccia **ServletResponse** ed espone metodi per:
  - Specificare lo status code della risposta HTTP
  - Indicare il content type (tipicamente **text/html**)
  - Ottenere un output stream in cui scrivere il contenuto da restituire
  - Indicare se l'output è bufferizzato
  - Gestire i cookie
  - ...


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...
ATTENZIONE:
javax VA
SOSTITUITO
CON jakarta
ServletResponse interface
(javax.servlet.ServletResponse)
<<interface>>
ServletResponse
getBufferSize() : int
setContentType(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...

Formato della risposta HTTP

In rosso ciò che può/deve essere specificato

Gestione dello status code

- Per definire lo status code HttpServletResponse fornisce il metodo

  public void setStatus(int statusCode)

(di solito usato per 2xx e 3xx)

- Esempi di status Code
  - **200 OK**
  - **301 Moved Permanently**
  - **...**

USO: *response.setStatus(SC_OK). // setta status code a 200 OK*

- Per inviare errori del tipo 4xx e 5xx possiamo anche usare:

  public void sendError(int sc)

  public void sendError(int code, String message)

Gestione degli header HTTP

- **public void setHeader(String headerName, String headerValue)** imposta un header arbitrario
- **public void setDateHeader(String name, long millisecs)** 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
- **setContentType** configura il content-type *(che è sempre un MIME type) (si usa 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 redirezione

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**)
  - **response.getWriter** -> utile per restituire un testo nella risposta (tipicamente HTML)
- **public ServletOutputStream getOuputStream()**: mette a disposizione uno stream di byte (un'istanza di **ServletOutputStream**)
  - **response.getOutputStream** -> più utile per una risposta con contenuto binario (per esempio un'immagine)

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>"
```

request

- https://jakarta.ee/specifications/servlet/6.0/apidocs/jakarta.servlet/jakarta/servlet/http/httpservletrequest
- **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 della HTTP Request
  - URL
  - HTTP Request header
  - Tipo di autenticazione e informazioni su utente
  - Cookie (scritto dal browser)
  - Session (lo vedremo nel dettaglio in seguito)
  - …

Struttura di una request HTTP

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***

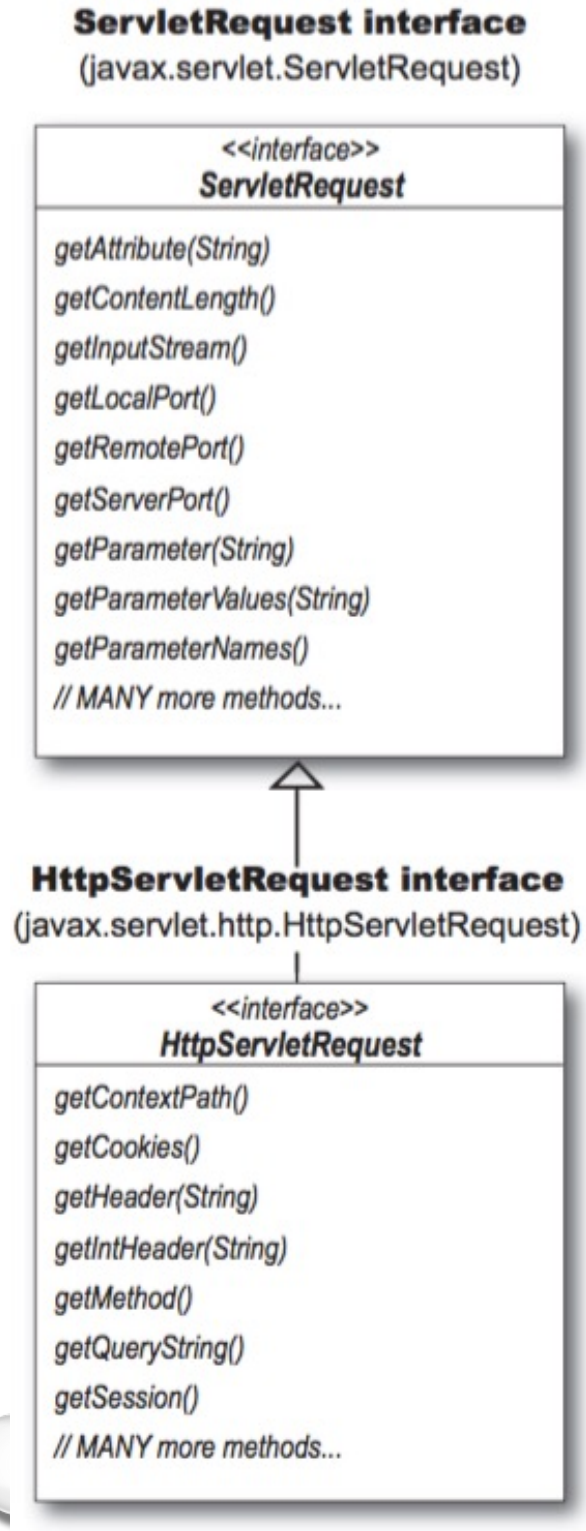
  HttpSession session = **request.getSession();**

- ***The HTTP Method of the request***

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

- ***An input stream from the request (upload)***

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



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:

  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
- **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

PROVATELI!!

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
- **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);
        }
    }
}
```

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, 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"));
   }
}
```

Altri aspetti di request

- 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>");
}
```

Echo: reading all request parameters

```
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();
}
```

Deployment

- Un'applicazione Web deve essere installata e questo processo prende il nome di **deployment**
- 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 2.5)

- È un file di configurazione (in formato XML) che contiene una serie di elementi descrittivi che descrivono la struttura **dell'intera applicazione web**
- Contiene l'elenco delle Servlet e per ognuna di loro permette di definire una serie di parametri:
  - 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
```

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)**

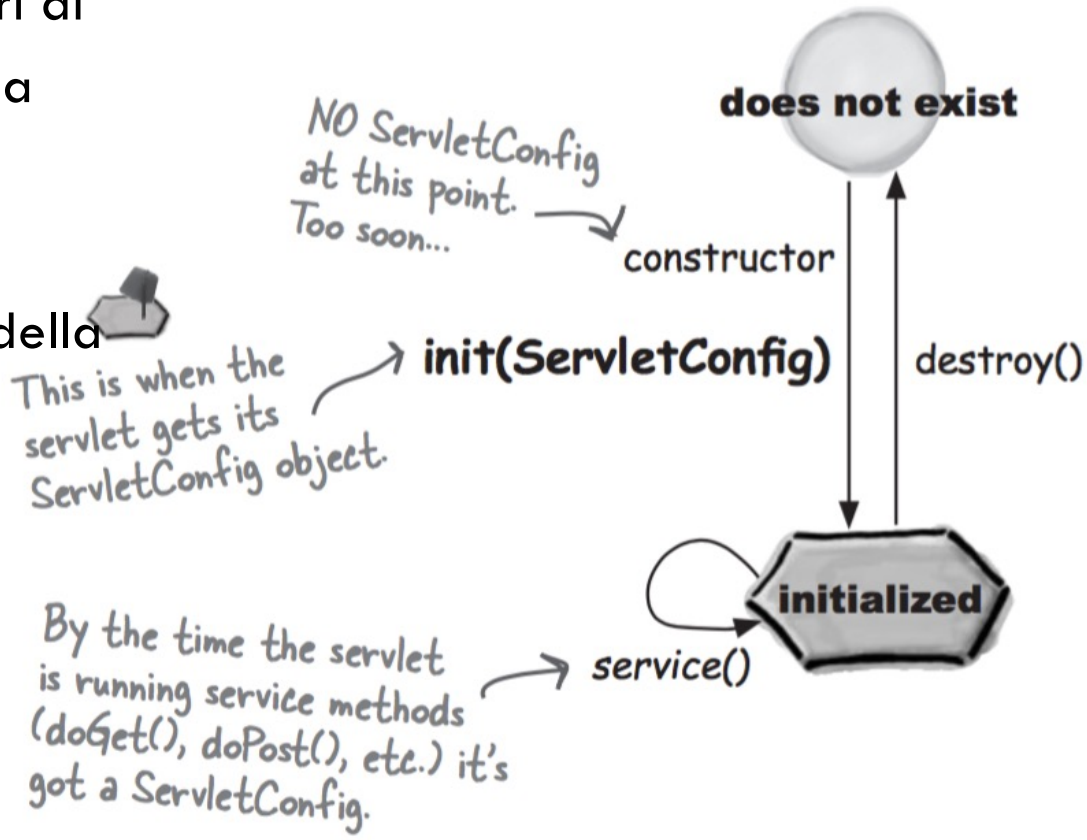
Esempio di parametro di configurazione

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

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

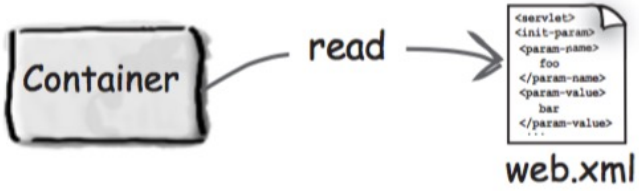
You can't use Servlet init parameters until the Servlet is initialized

- Quando il Container inizializza una Servlet, crea una ServletConfig unica per la Servlet
- Il Container "legge" i parametri di inizializzazione della Servlet da web.xml e li memorizza nel ServletConfig, poi passa il ServletConfig al metodo init() della servlet

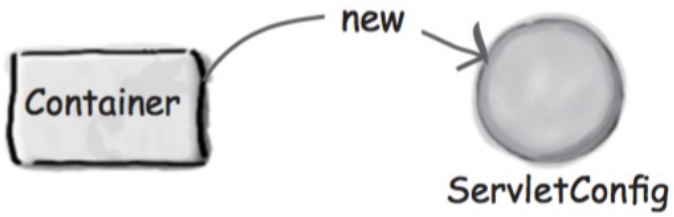


The Servlet init parameters are read only ONCE

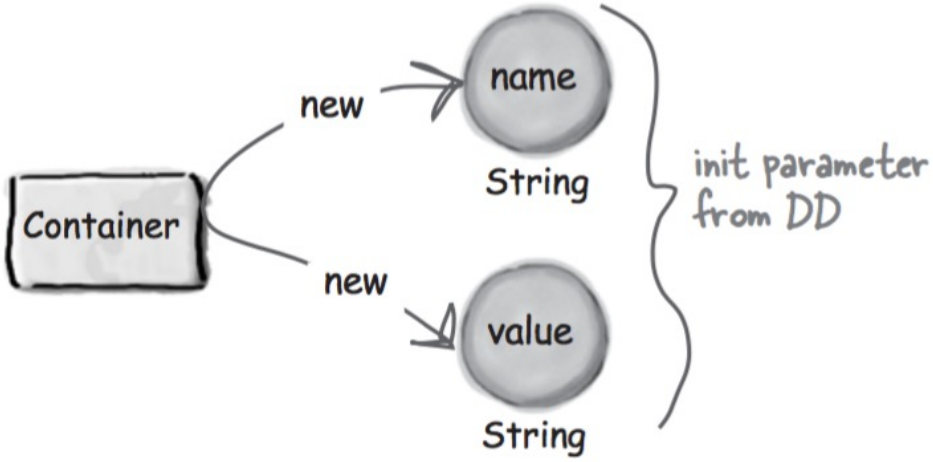
1. Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



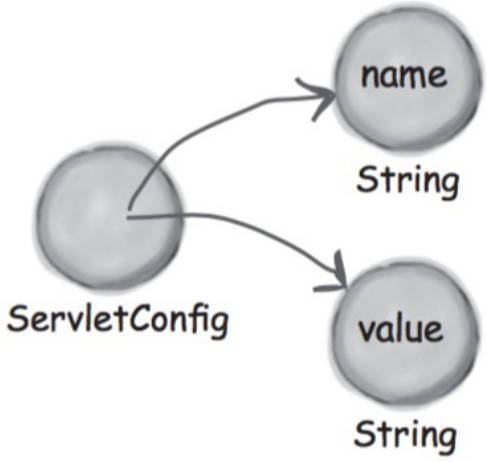
2. Container creates a new ServletConfig instance for this servlet.



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

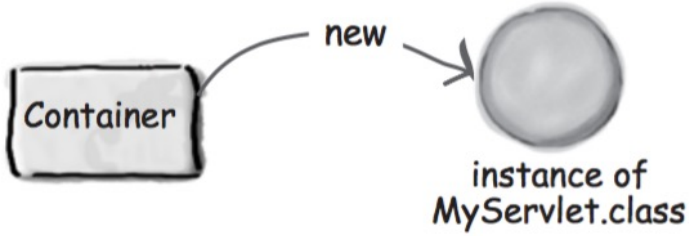


4. Container gives the ServletConfig references to the name/value init parameters.

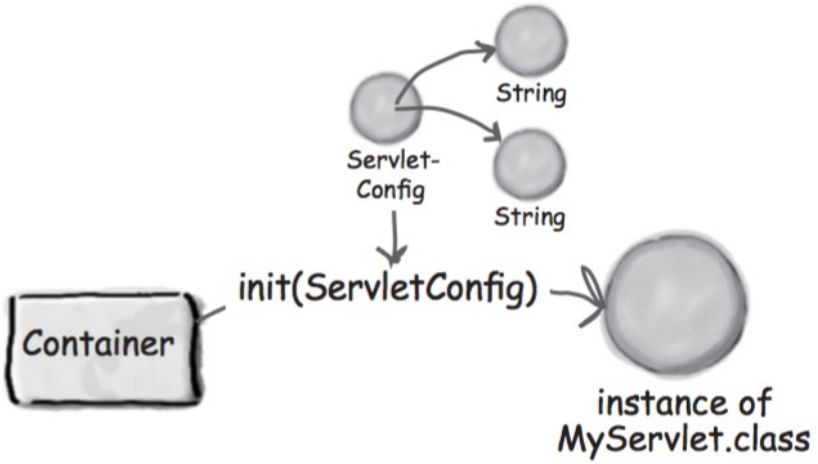


The Servlet init parameters are read only ONCE (2)

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 = "HelloServlet", 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>"
```