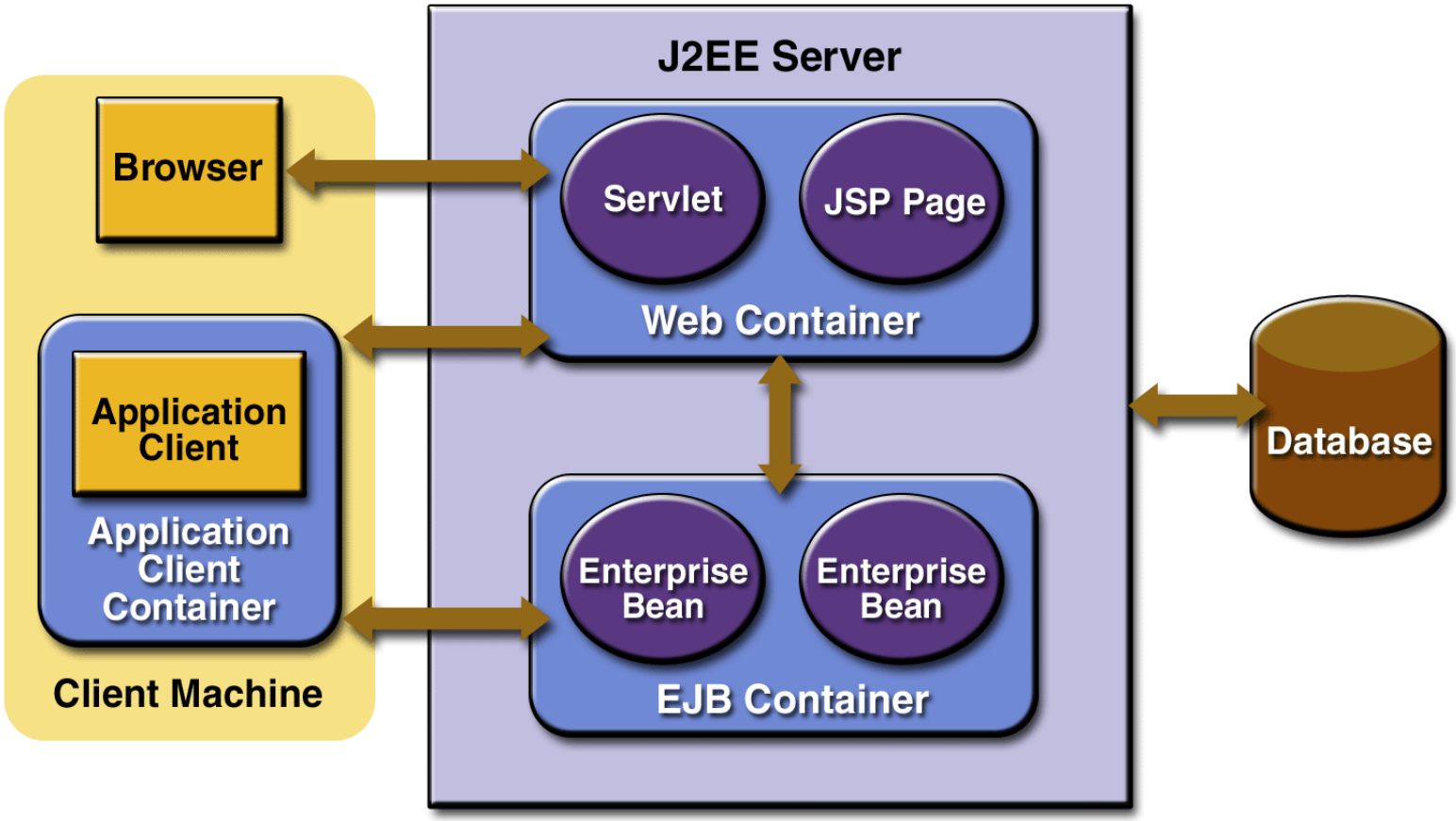
Servlet

Su Virtuale:
Versione 1 pagina per foglio = 2.02.Servlet.pdf
Versione 2 pagine per foglio = 2.02.Servlet-2p.pdf

L'architettura Java Enterprise Edition (JEE o J2EE)

Modello a Componente-Container

- Architetture multi-tier
- Servizi ad alta scalabilità per applicazioni distribuite enterprise, anche Web-based



Web Client

I Web Client hanno sostituito, in molte situazioni di applicazioni client-server, i più tradizionali "fat client"

- sono spesso costituiti dal semplice browser Web senza bisogno di alcuna installazione ad hoc
- comunicano via HTTP e HTTPS con il server (come sapete, browser è, tra le altre cose, un client HTTP)
- effettuano il rendering della pagina in HTML (o altre tecnologie mark-up come, ad es. XML e XSL)
- possono essere sviluppati utilizzando varie tecnologie
- sono spesso implementati come ***parti di architetture multi-tier***

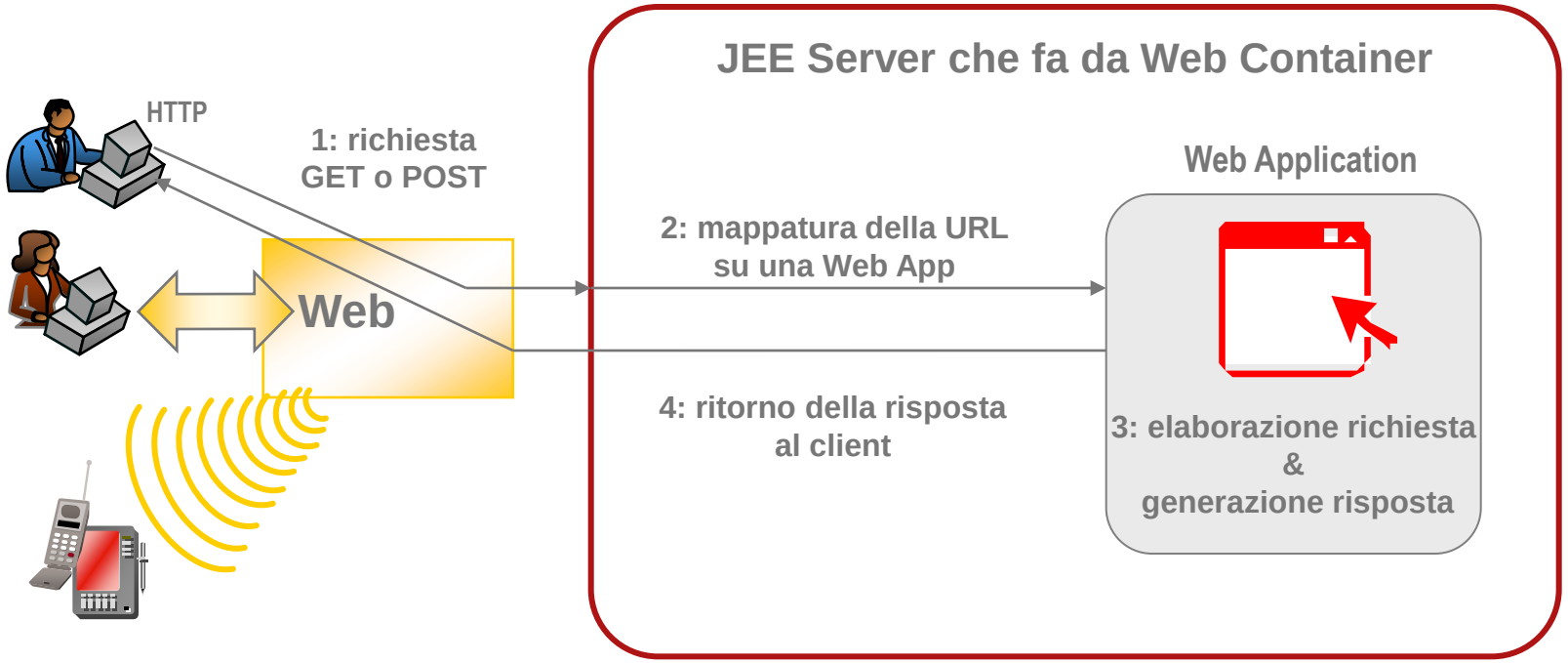
Vedi anche trend verso tecnologie RESTful…

JEE Web Application e Web Container

- Una **Web Application** è un gruppo di risorse server-side che nel loro insieme creano una applicazione interattiva fruibile via Web
- Le **risorse server-side** includono:
  - **Classi server-side** (Servlet e classi standard Java)
  - **Java Server Pages** (le vedremo in seguito)
  - **Risorse statiche** (documenti HTML, immagini, css, ...)
  - **Applet, Javascript** e/o altri componenti che diventeranno attivi client-side
  - **Informazioni di configurazione** e deployment
- I **Web Container** forniscono un ambiente di esecuzione per Web Application
- In generale, Container garantiscono servizi di base alle applicazioni sviluppate secondo un **paradigma a componenti**

Accesso ad una Web Application

L'accesso a Web Application è un processo multi-step:



Che cos'è una Servlet (vi ricordate CGI?)

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 e superando i classici limiti delle applicazioni CGI***
- **Eseguono direttamente in un Web Container**
  - In termini pratici sono classi che derivano dalla classe HttpServlet
  - HttpServlet implementa vari metodi che possiamo ridefinire

Esempio di Servlet: Hello World!

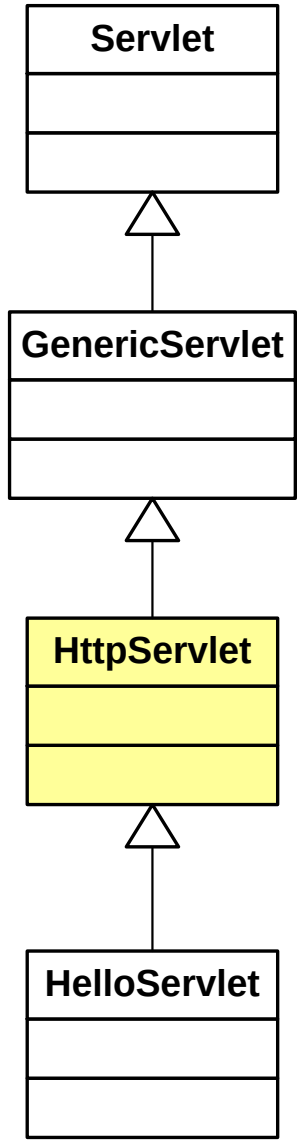
- Ridefiniamo `doGet()` e implementiamo la logica di risposta a HTTP GET
- Produciamo in output un testo HTML che costituisce la pagina restituita dal server HTTP:

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

Gerarchia delle Servlet

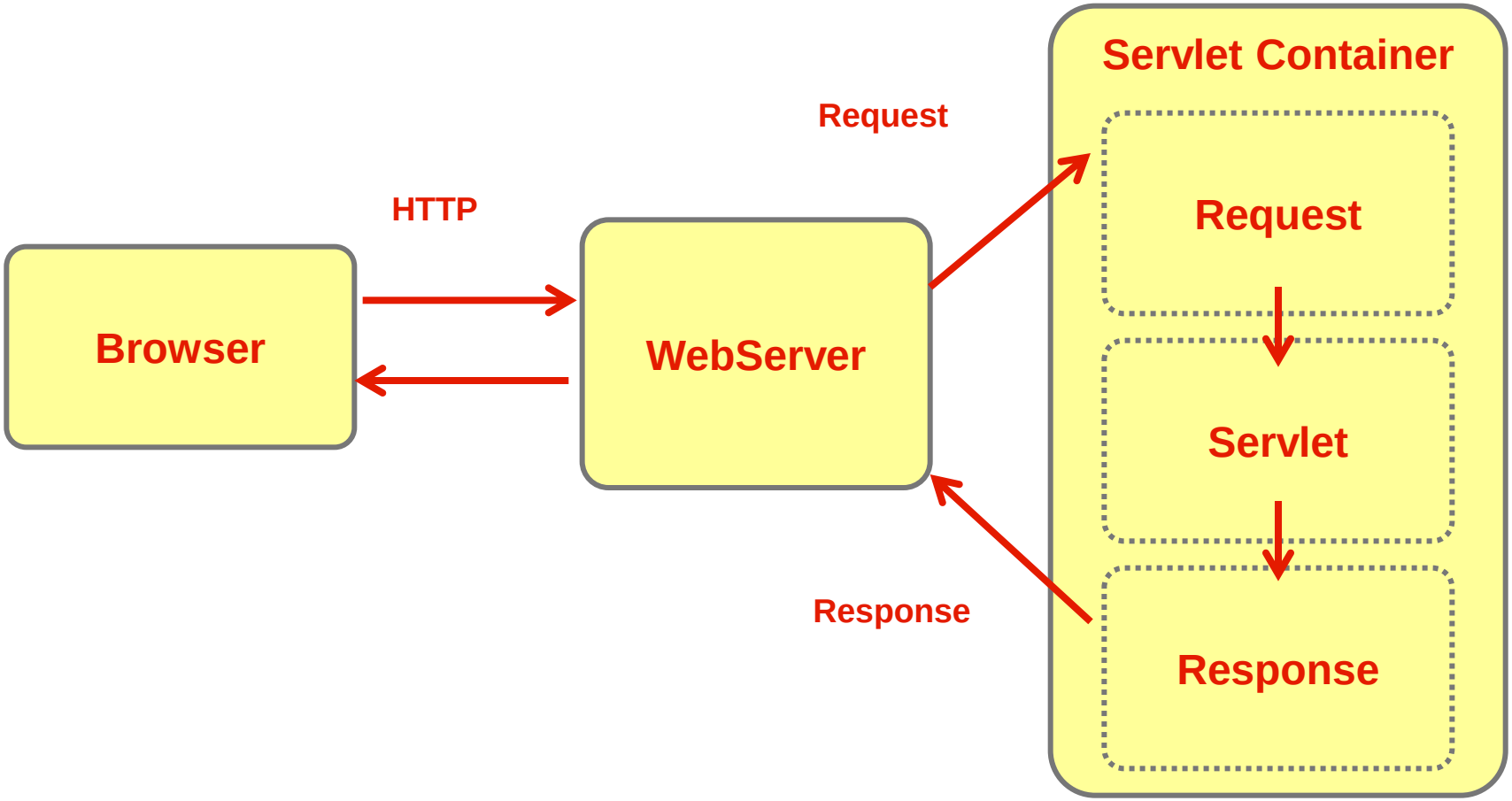
- Le servlet sono classi Java che elaborano richieste seguendo un protocollo condiviso
- ***Le servlet HTTP sono il tipo più comune di servlet e possono processare richieste HTTP, producendo response HTTP***
- Abbiamo quindi la catena di ereditarietà mostrata a lato
  - Nel seguito ragioneremo sempre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

```
javax.servlet.http.*
```



Il modello request-response

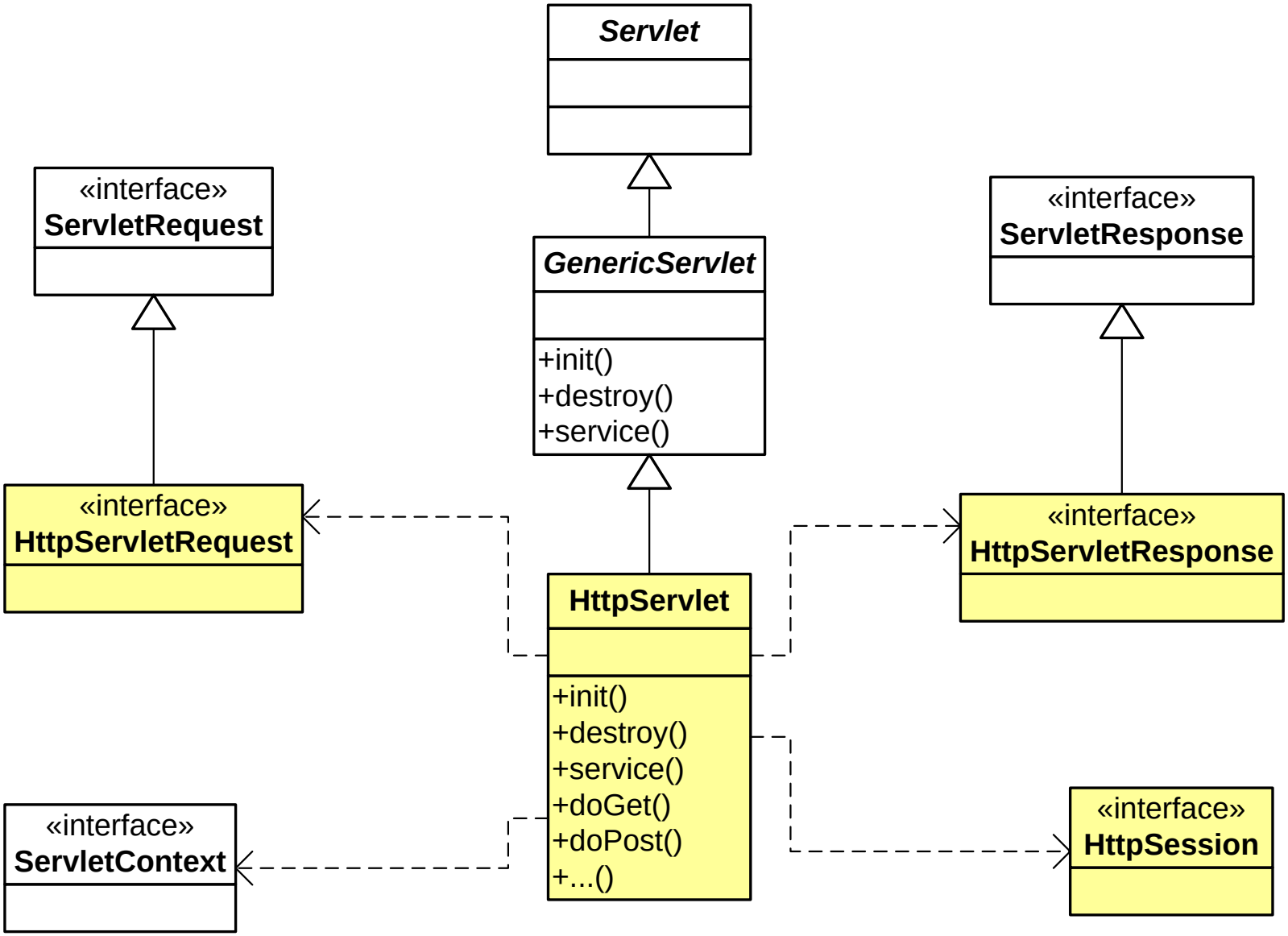
All'arrivo di una richiesta HTTP il Servlet Container crea un ***oggetto request e un oggetto response e li passa alla servlet***:



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



Il ciclo di vita delle Servlet

- *Servlet container* controlla e supporta automaticamente il ciclo di vita di una servlet
- Se non esiste una istanza della servlet nel container
  - Carica la classe della servlet
  - Crea una istanza della servlet
  - Inizializza la servlet (invoca il metodo `init()`)
- Poi, a regime:
  - Invoca la servlet (`doGet()` o `doPost()` a seconda del tipo di richiesta ricevuta) passando come parametri due oggetti di tipo `HttpServletRequest` e `HttpServletResponse`

Quante istanze di servlet? Quanti thread sono associati ad una istanza di servlet? Quale modello di concorrenza? Con quali pericoli?

Servlet e 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**

Modello single-threaded (deprecated)

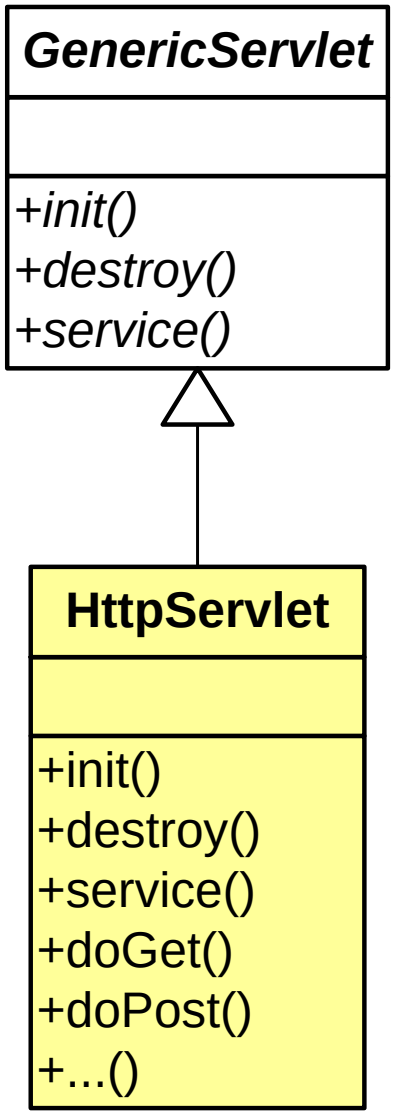
- *Alternativamente si può indicare al container di creare un'istanza della servlet per ogni richiesta concorrente*
- Questa modalità prende il nome di **Single-Threaded Model**
  - È onerosa in termine di risorse ed è deprecata nelle specifiche 2.4 delle servlet
- Se una servlet vuole operare in modo single-threaded deve implementare l'interfaccia marker `SingleThreadModel`
  - Altri esempi di interfacce marker in Java? *Serializable*

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

- 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

- Importiamo i package 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)
     ...
}
```

Hello World - doGet

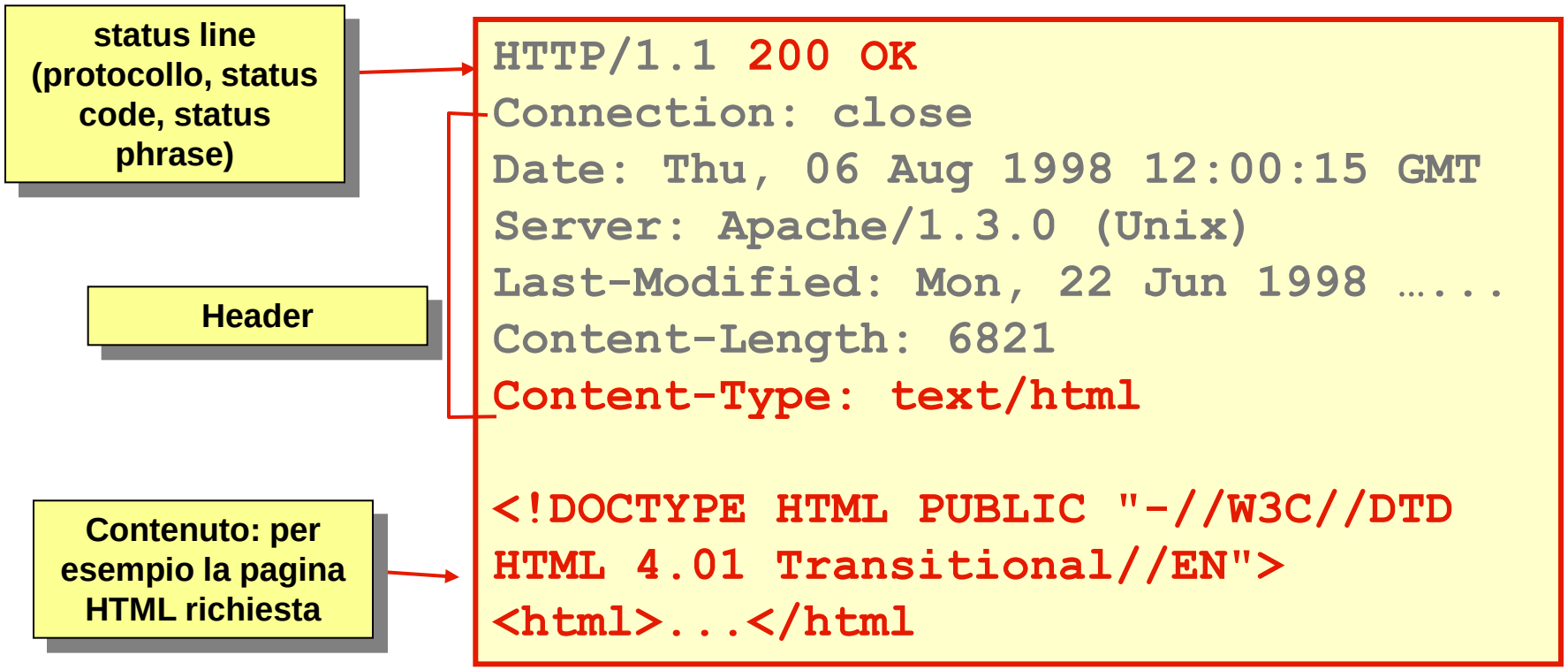
- Dobbiamo tener conto che in `doGet()` possono essere sollevate **eccezioni** di due tipi:
  - quelle specifiche delle Servlet
  - quelle legate all'input/output
- Decidiamo di non gestirle per semplicità e quindi ricorriamo alla clausola `throws`
- In questo semplice esempio, non ci servono informazioni 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
{
  ...
}
```

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 `HttpResponse` che 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
  - …

Il formato della risposta HTTP



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

Gestione dello status code

Per definire lo status code HttpServletResponse fornisce il metodo

```
public void setStatus(int statusCode)
```

- 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)
```

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 (**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 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)

Implementazione di doGet()

Abbiamo tutti gli elementi per implementare correttamente il metodo `doGet()` di `HelloServlet`:

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

Risposta generata

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

Rendiamo Hello World un poco 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 richiesta 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/
```

Request URL

Come sapete bene, 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</a>
```

- **Il metodo `getParameter()` di `request` ci permette di accedere ai vari parametri**
  - Ad esempio se scriviamo:

```
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

Metodi per accedere agli 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

Autenticazione, sicurezza e cookie

- `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 dei 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>");
}
```

Ridefinizione di `service()`

Se non viene ridefinito, ***il metodo service effettua il dispatch delle richieste ai metodi doGet, doPost, … a seconda del metodo HTTP usato nella request***

- Ad es. se si vuole trattare in modo uniforme get e post, si può ridefinire il metodo service facendogli elaborare direttamente la richiesta:

```
public void service(HttpServletRequest req,
                    HttpServletResponse res)
{
     int reqId = Integer.parseInt(req.getParameter("reqID");
     switch(reqId)
     {
       case 1: handleReq1(req, res); break;
       case 2: handleReq2(req, res); break;
       default : handleReqUnknown(req, res);
     }
}
```

Deployment

Prima di proseguire con l'esame delle varie caratteristiche delle servlet vediamo come fare per far funzionare il nostro esempio

- 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 Archives

Gli Archivi Web (Web Archives) sono file con estensione ".war".

- Rappresentano la modalità con cui avviene la distribuzione/deployment delle applicazioni Web
- Sono file jar con una struttura particolare
- Per crearli si usa il comando jar:

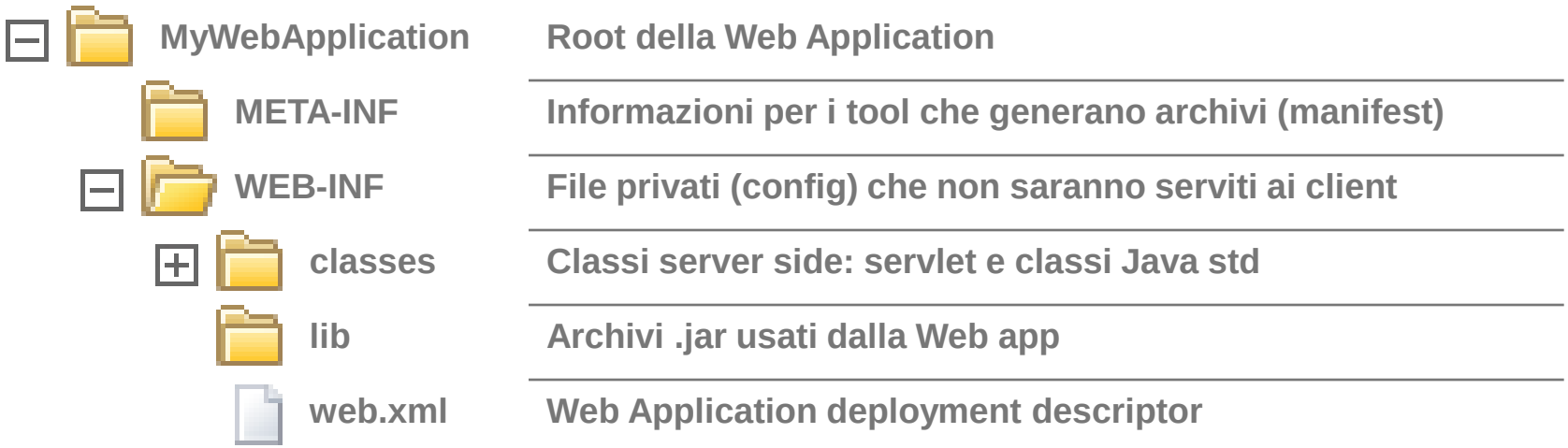
```
jar {ctxu} [vf] [jarFile] files

-ctxu: create, get the table of content, extract, update content
-v: verbose
-f: il JAR file sarà specificato con jarFile option
-jarFile: nome del JAR file
-files: lista separata da spazi dei file da includere nel JAR
```

Esempio

```
jar -cvf newArchive.war myWebApp/*
```

Struttura interna del war

- La struttura di directory delle Web Application è basata sulle **Servlet 2.4 specification**



- **web.xml** è in sostanza un file di configurazione (XML) che contiene una serie di elementi descrittivi
- Contiene *l'elenco delle servlet* attive sul server, il loro mapping verso URL, e per ognuna di loro permette di definire una *serie di parametri come coppie nome-valore*

Il descrittore di deployment

web.xml è un file di configurazione (in formato XML) che descrive la struttura dell'applicazione Web

- Contiene l'elenco delle servlet e per ogni servlet permette di definire
  - nome
  - classe Java corrispondente
  - una serie di parametri di configurazione (coppie nome-valore, valori di inizializzazione)
- IMPORTANTE: contiene **mappatura fra URL e servlet** che compongono l'applicazione

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:
  - Il parametro di tipo ServletConfig passato al metodo `init()`
  - 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>
```

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>
```

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>"
```

Servlet context

- Ogni Web application esegue in un **contesto**: corrispondenza 1:1 tra una Web-app 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 utlizzando 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 `getResourceAsStream(String path)`

IMPORTANTE: servlet context viene condiviso tra tutti gli utenti, tutte le richieste e tutti i componenti server-side (servlet, …) 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 in lettura 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);
}
```

Gestione dello stato (di sessione)

- Come abbiamo già detto più volte, ***HTTP è un protocollo stateless*: non fornisce in modo nativo meccanismi per il mantenimento dello stato tra diverse richieste provenienti dallo stesso client**
- ***Applicazioni Web hanno spesso bisogno di stato*. Sono state definite due tecniche per mantenere traccia delle informazioni di stato**
  - uso dei cookie: meccanismo di basso livello
  - 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:**
  - Cookie
  - URL rewriting

Cookie (riepilogo per smemorati…)

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

La classe cookie

Un cookie contiene un certo numero di informazioni, tra cui:

- una coppia nome/valore
- 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)
- 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);
c.setPath("/");
response.addCookie(c);
```

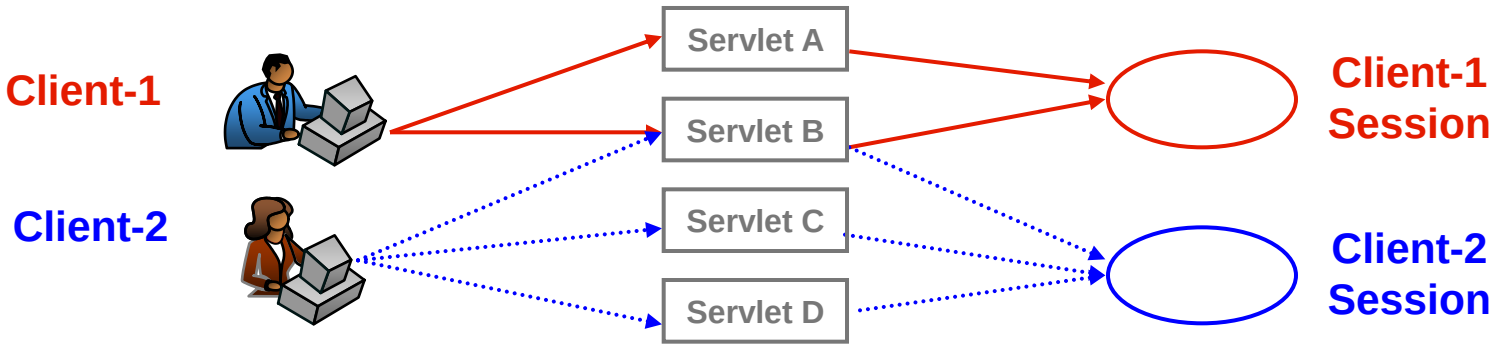
lettura

```
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());
  }
}
```

Uso della sessione in Web Container

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



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);
```

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.setAttribute("shoppingCart", new Cart());
session.removeAttribute("shoppingCart");

Enumeration e = session.getAttributeNames();
while(e.hasMoreElements())
   out.println("Key; " + (String)e.nextElements());
```

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 ()` dà 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());
```

Come identificare una sessione?

Proposte?

- Ad esempio, possiamo usare IP cliente?
- Ad esempio, dalla vostra esperienza di utenti Web, le applicazioni di uso comune utilizzano IP cliente? Come verificarlo?
- Possiamo chiedere, a ogni request, di inserire di nuovo info di autenticazione all'utente?
- Altre idee…

Session ID e URL Rewriting

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

- *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
- *Un'alternativa è rappresentata dall'inclusione del session ID nella URL*: si parla di **URL rewriting**
  - È buona prassi codificare sempre le URL generate dalle servlet usando il metodo encodeURL() di `HttpServletResponse`
  - Il metodo encodeURL() dovrebbe essere usato per:
    - hyperlink (<a href="…">)
    - form (<form action="…">)

Attenzione: scope DIFFERENZIATI (scoped objects)

- Gli oggetti di tipo `ServletContext`, `HttpSession`, `HttpServletRequest` 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	`HttpServletRequest`	Fino all'invio della risposta	Servlet corrente e ogni altra pagina interrogata tramite include o forward
Session	`HttpSession`	Durata della sessione utente	Ogni richiesta dello stesso cliente
Application	`ServletContext`	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()`

Inclusione di risorse Web

- Includere risorse Web (altre pagine, statiche o dinamiche) può essere utile quando si vogliono aggiungere contenuti creati da un'altra risorsa (ad es. un'altra servlet)
- **Inclusione di risorsa statica:**
  - includiamo un'altra pagina nella nostra (ad es. banner)
- **Inclusione di risorsa dinamica:**
  - la servlet inoltra una request ad un componente Web che la elabora e restituisce il risultato
  - Il risultato viene incluso nella pagina prodotta dalla servlet
- **La risorsa inclusa può lavorare con il response body (problemi comunque con l'utilizzo di cookie)**

Come si fa l'inclusione

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**

```
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 perché 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`

Come si fa un forward

Anche in questo caso 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`
- Se necessario, l'URL originale può essere salvato come un attributo di request

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

Completam DIVERSA: Ridirezione del browser

- È anche possibile inviare al browser una risposta che lo forza ad accedere ad un'altra pagina (ridirezione)
- Si usa uno dei codici di stato di HTTP: sono i codici che vanno da 300 a 399 e in particolare
  - `301 Moved permanently:` URL non valida, il server indica la nuova posizione

Possiamo ottenere questo risultato in due modi, agendo sull'oggetto `response`:

- Invocando il metodo
  ```
  public void sendRedirect(String url)
  ```
- Lavorando più a basso livello con gli header:
  ```
  response.setStatus(response.SC_MOVED_PERMANENTLY);
  response.setHeader("Location", "http://...");
  ```