Interazione Uomo-Macchina e tecnologie web

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

Programmazione server-side: Java Servlets
parte 1

Applicazioni Web

- Le applicazioni web sono applicazioni accessibili in rete. Esse vengono eseguite su un server remoto:
  - non richiedono installazione locale sul computer dell'utente
  - l'utente interagisce con le applicazioni tramite browser (l'interfaccia utente delle applicazioni è costituita di pagine web dinamiche)
- Per far girare un'applicazione java Web serve:
  - Un Web server (che cattura richieste HTTP e invia le risposte)
  - Un Web Container: **java runtime** che ospiti ed esegua le applicazioni web sul server

Web Container

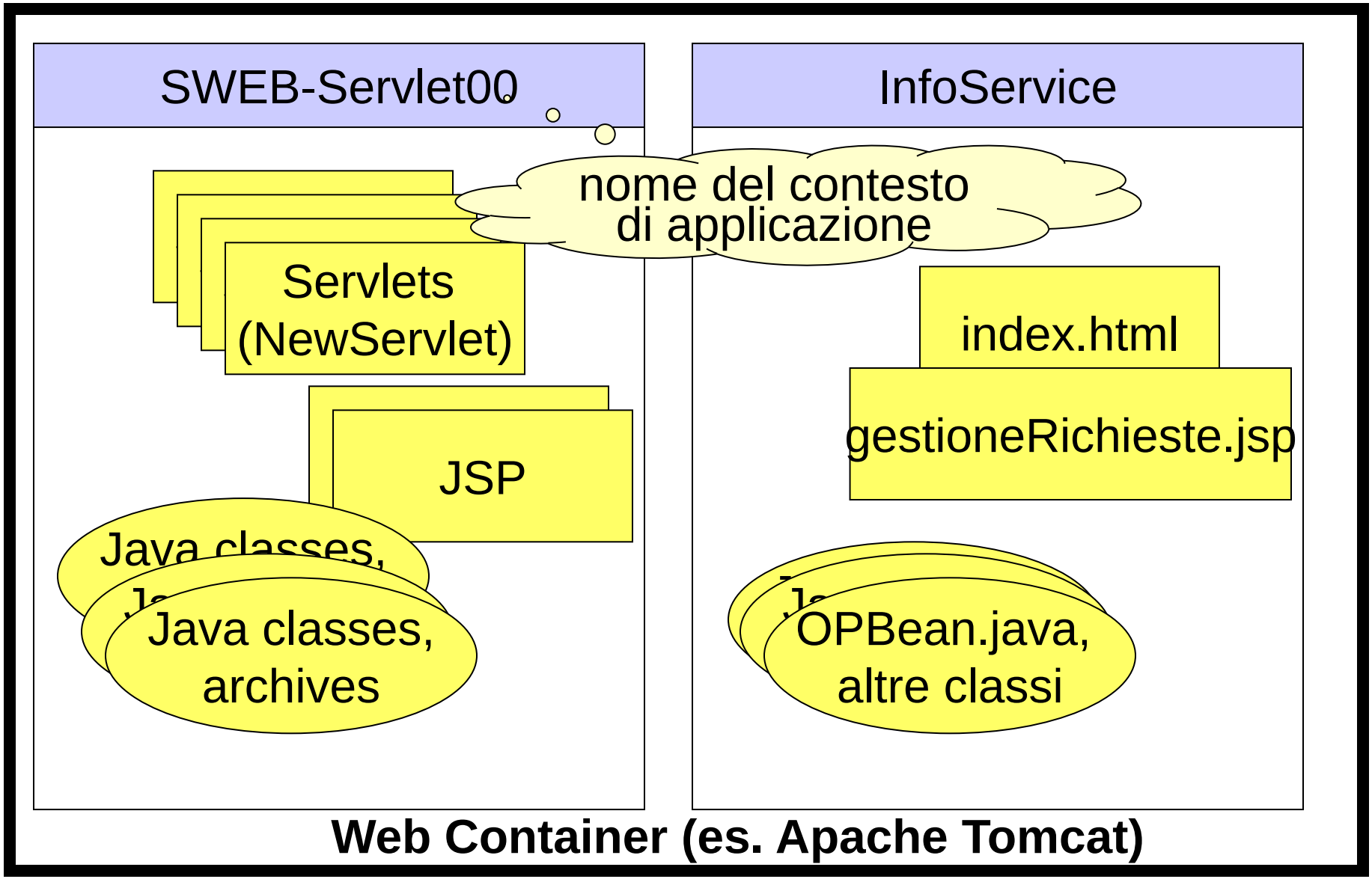
Tipi di web container

- **Web container incluso nel web server**: i web server scritti in Java includono il web container
  Es: **Apache Tomcat** o Application server **GlassFish** (GlassFish è embedded in NetBeans IDE - oppure si può installare **Tomcat**, più leggero e veloce)
- **Web Container in J2EE application server**: application server che gestiscono Enterprise Java Beans includono anche il web server
- **Web container in runtime separato**: web servers come Apache e Microsoft IIS necessitano di runtime separato per gestire codice java, più plug-in che gestisce la comunicazione tra web server e web container

Applicazioni Web-based in ambiente java

- **Java Servlets e JSP**: strumenti base per sviluppare le applicazioni (**Web components**)
  - un'applicazione Web è composta da Servlets, JSPs (pagine web dinamiche, tipo PHP o ASP), librerie di classi (Java Bean e altre classi), risorse statiche (HTML, XML, …), immagini, ...
- Ogni applicazione web è identificata dal suo **contesto di applicazione** (di solito questo corrisponde al nome del progetto Web; es: SWEB-Servlet00, InfoService, …)

Struttura di applicazioni Java Web-based



Modello a eventi – cenni (estratto dalle slides del corso di Programmazione III)

- I programmi tradizionali hanno un comportamento funzionale: essi ricevono un input, eseguono la propria computazione e restituiscono un risultato seguendo il proprio flusso di controllo
- Nel caso delle applicazioni web il comportamento deve essere reattivo per rispondere alle richieste degli utenti
- Un programma basato sul modello a eventi consiste di un insieme di procedure (**event handlers**), ciascuna delle quali specifica cosa fare quando si verifica un certo evento. Quando l'evento si verifica, verrà eseguito l'event-handler associato
- Il flusso di controllo con cui il programma viene eseguito non è determinato a priori, ma dipende dall'ordine con cui gli eventi si verificano. Il programma termina quando si verifica un evento che ne richiede la terminazione

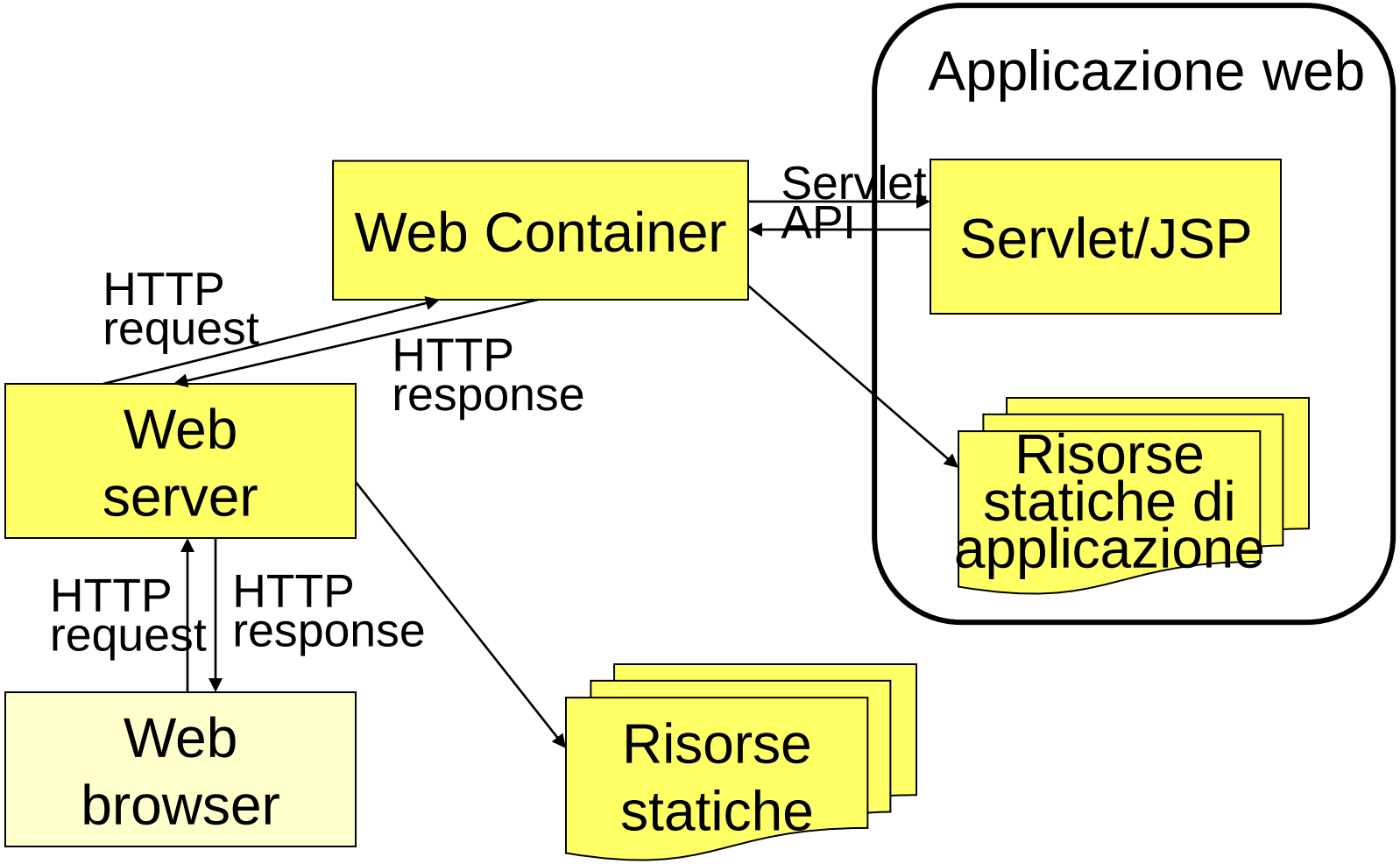
Schema di un programma guidato dagli eventi (estratto dai lucidi di Progr. III)

L'esecuzione delle Servlet Java è basata sul modello ad eventi. Gli eventi sono le richieste HTTP dirette alle Servlet.

Servlet Java

- **Le Servlet sono classi java eseguite in un processo del server web**
- In fase di deployment il Web Container carica le Servlet nel processo web server. Il Web Container interagisce con le Servlet via **Java Servlet API**
- Le Servlet vengono **invocate mediante HTTP request** e producono il loro output sotto forma di **HTTP response**
- **Documentazione: http://www.oracle.com/technetwork/java/index-jsp-135475.html**

Applicazioni server-side – esempio in ambiente Java



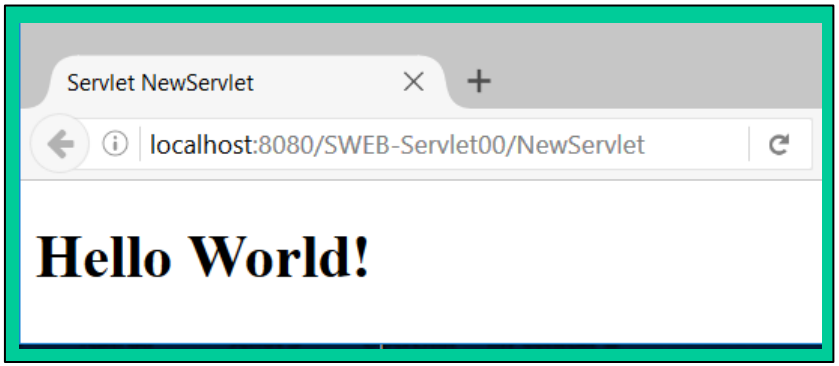
Esempio: SWEB-Servlet00/NewServlet

```
public class NewServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,
HttpServletResponse response)  throws ServletException,
IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html>");
            out.println("<html><head>");
            out.println("<title>Servlet NewServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Hello world!</h1>");
            out.println("</body></html>");
        } finally {out.close();}
    }
}
```

doGet() è eseguito a seguito di GET HTTP (evento)

PrintWriter rappresenta l'output stream verso il browser



Invocazione di una Servlet Java - I

- **Da browser digitando il suo URL** *(solo HTTP request di tipo GET).*
  - Es: http://localhost:8080/SWEB-Servlet00/NewServlet
- **Da pagina HTML/JSP** mediante link all'URL della risorsa (*solo GET*). Es:
  - Link a <A HREF = *"/SWEB-Servlet00/NewServlet"*>*NewServlet*</A>
- **Da form HTML**, con bottone di invocazione (*GET/POST*). Es:

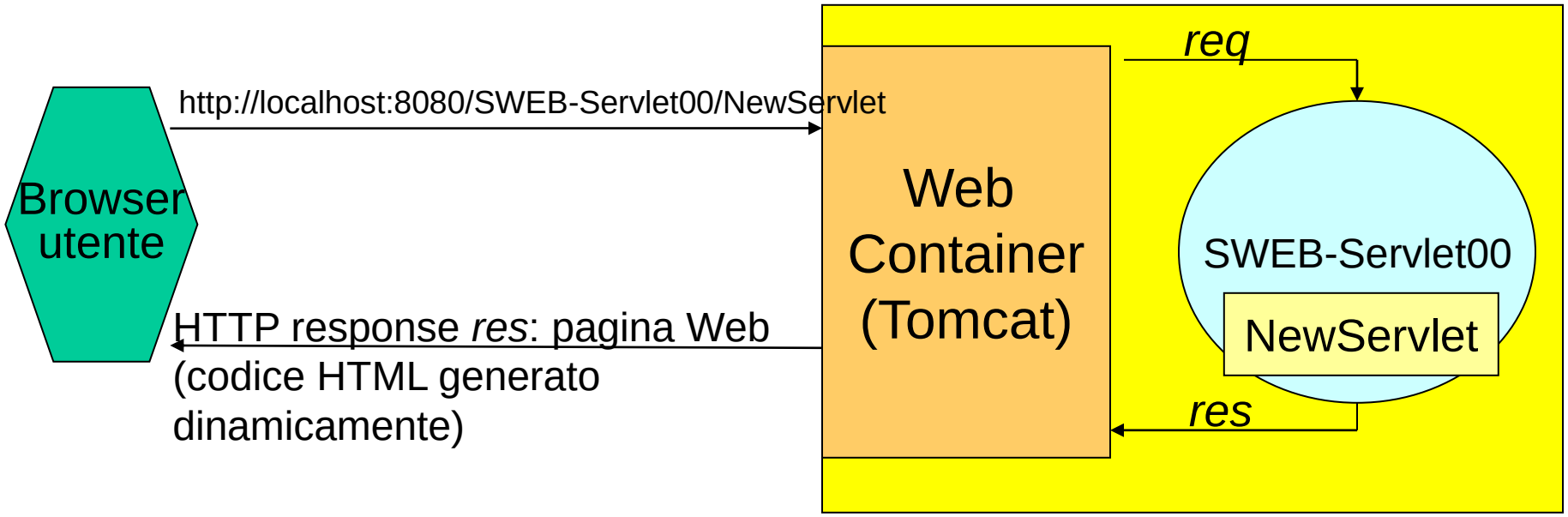
```
<form action="/SWEB-Servlet00/NewServlet" method="get">
    <p><input type="submit" name="submit" value="OK"></p>
<form>
```

Invocazione di una Servlet Java - II

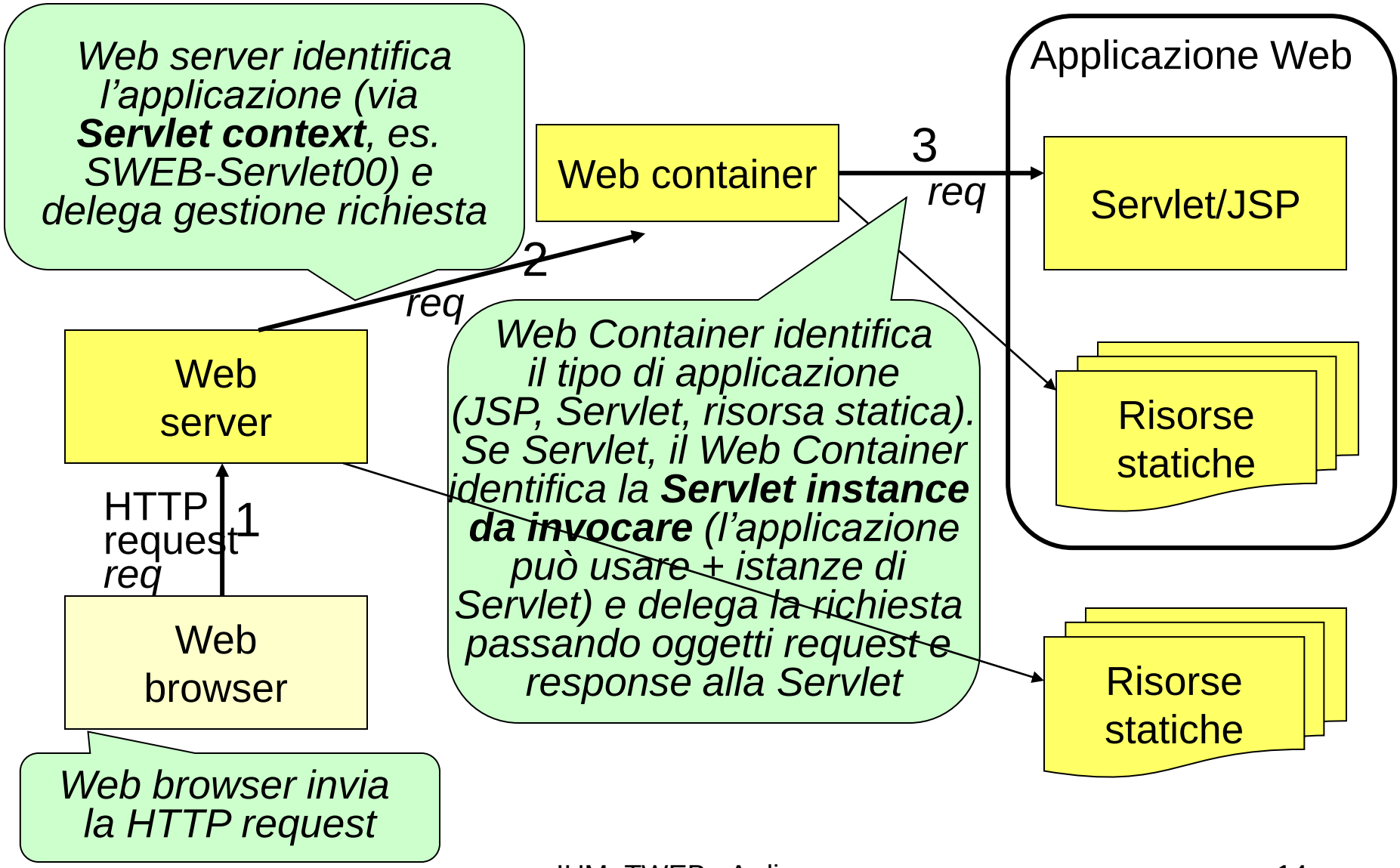
Una volta "caricata" nel Web Container, **una Servlet rimane attiva** in attesa di richieste dai browser:

- **Qualunque client può invocarla**
- La si può **invocare volte quante si vuole**. Ad ogni invocazione, la Servlet esegue il proprio codice e genera la pagina Web per il client specifico
- L'esecuzione del codice avviene in un **thread separato del Web Container** per permettere di servire più client in parallelo

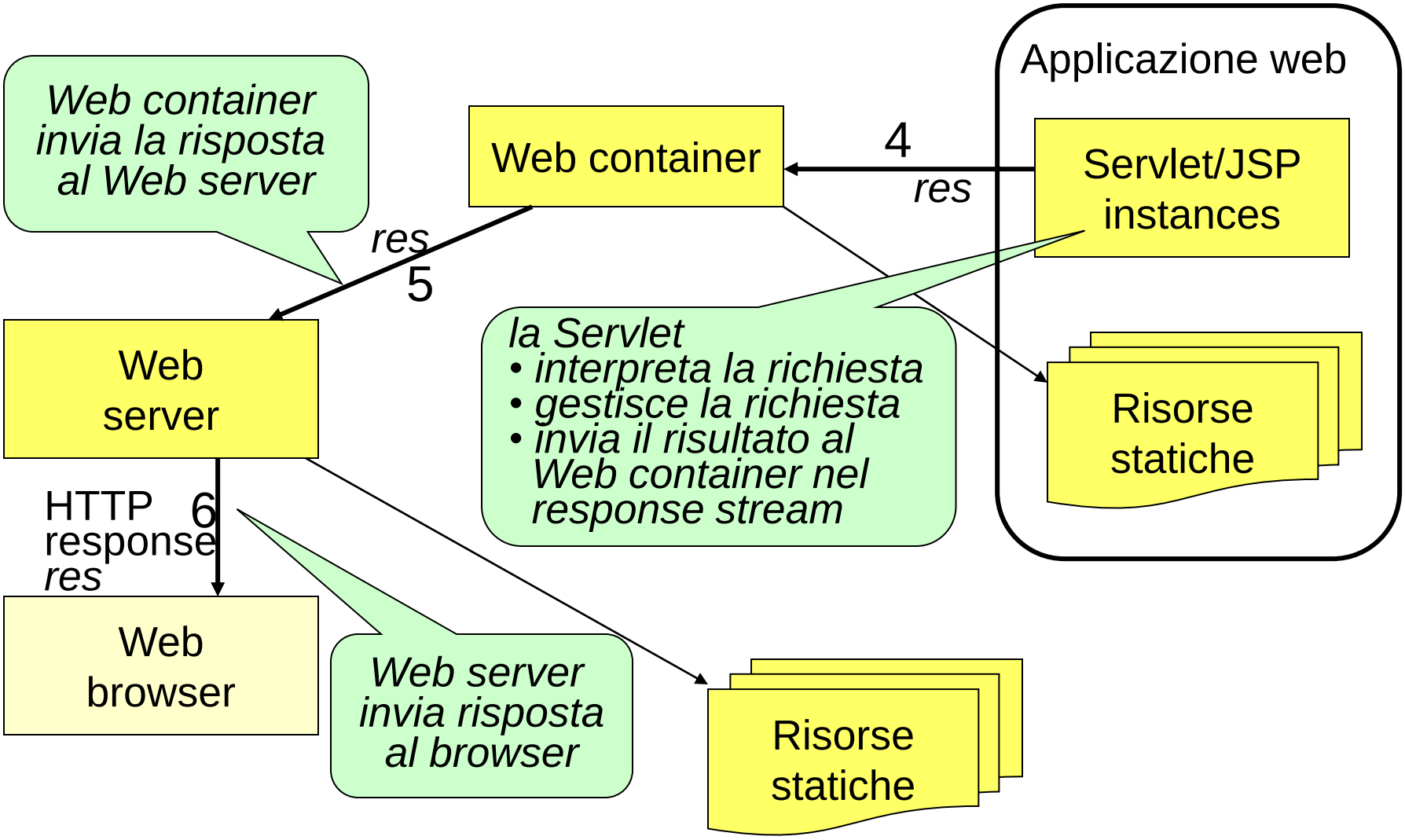
Esecuzione di NewServlet (progetto SWEB-Servlet00). URL: http://localhost:8080/SWEB-Servlet00/NewServlet



Invocazione di applicazione basata su Servlet - I



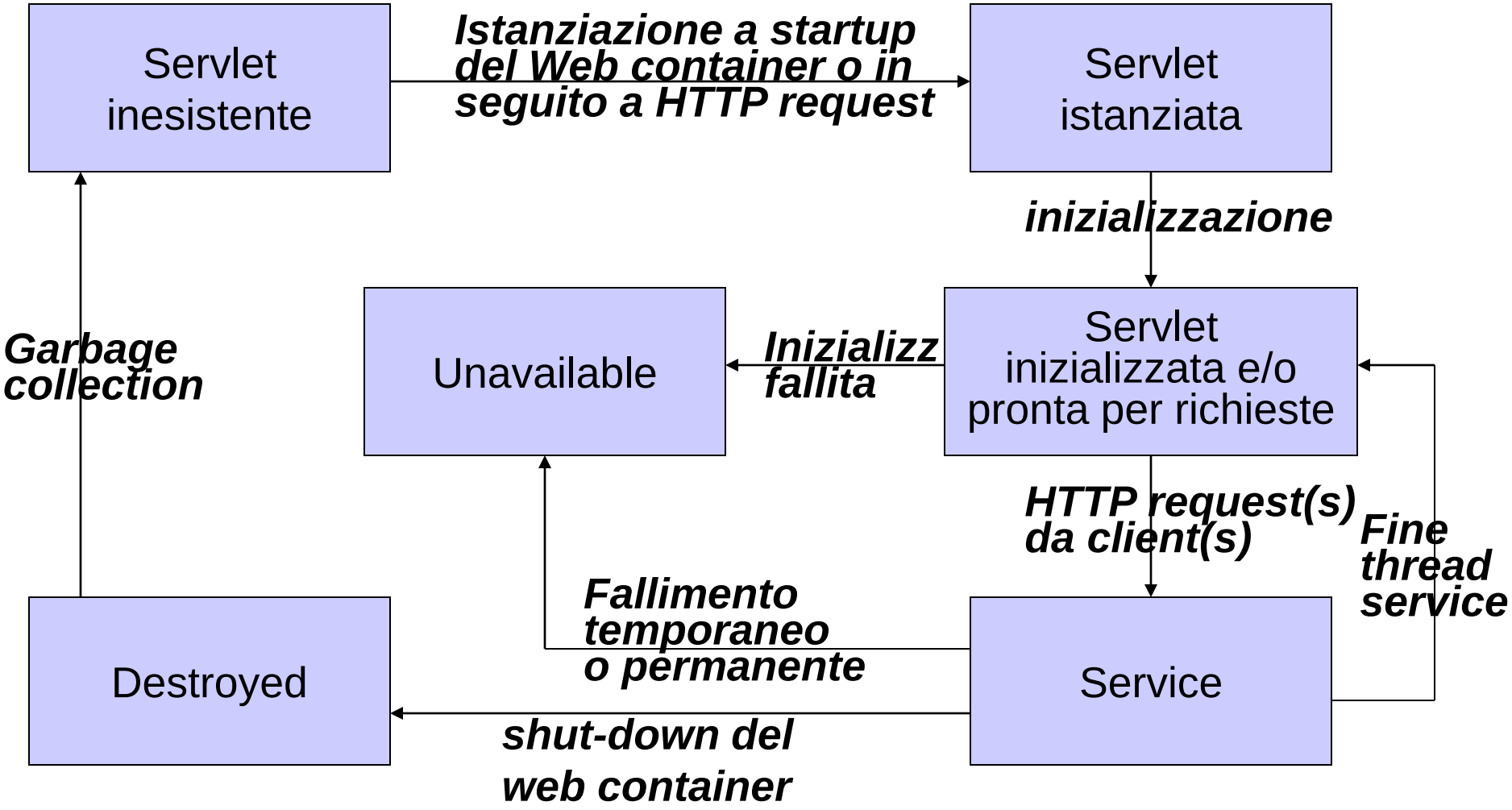
Invocazione di applicazione basata su Servlet - II



Java Servlet API

- **package javax.servlet**: include classi e interfacce per la gestione di Servlet indipendenti dal protocollo di comunicazione
- **package javax.servlet.http**: include classi e interfacce specifiche per l'uso di Servlet con il protocollo HTTP
- alcune classi e interfacce di javax.servlet.http estendono le rispettive di javax.servlet

Ciclo di vita di una Servlet java



Porzione della gerarchia delle classi - Servlet

interface **javax.servlet.Servlet**

- init(ServletConfig c)
- service(ServletRequest req, ServleResponse res)
- destroy(), … *altri metodi* ...

interface **javax.servlet.ServletConfig**

Mantiene i dati di configurazione della Servlet

abstract class **javax.servlet.GenericServlet** implements Servlet, ServletConfig, Serializable

offre un'implementazione dell'interface Servlet indipendente dal protocollo di comunicazione (HTTP, …)

abstract class **javax.servlet.http.HttpServlet** extends GenericServlet implements Serializable

*Offre un'**implementazione** di interface Servlet **specifica per HTTP***

javax.Servlet - Interface Servlet

*Specifica i **metodi di gestione** della Servlet*

- **public void init(ServletConfig c)** throws ServletException
  - **invocato dal Web Container** dopo aver istanziato la Servlet
  - usato per inizializzare le variabili della Servlet (prima di accettare HTTP requests). Es: leggere dati di configurazione, registrare database driver, …
- **public void service(ServletRequest req, ServletResponse res)** throws ServletException, IOException
  - **invocato dal Web Container** a ogni HTTP request (oggetto **req** contiene i dati della richiesta, **res** contiene la risposta)
- **public void destroy()**
  - **invocato dal Web Container** prima di rimuovere l'istanza della Servlet (es: prima di uno shut down del Web server, o se serve spazio di memoria, …)

javax.Servlet.http - abstract class HttpServlet - I

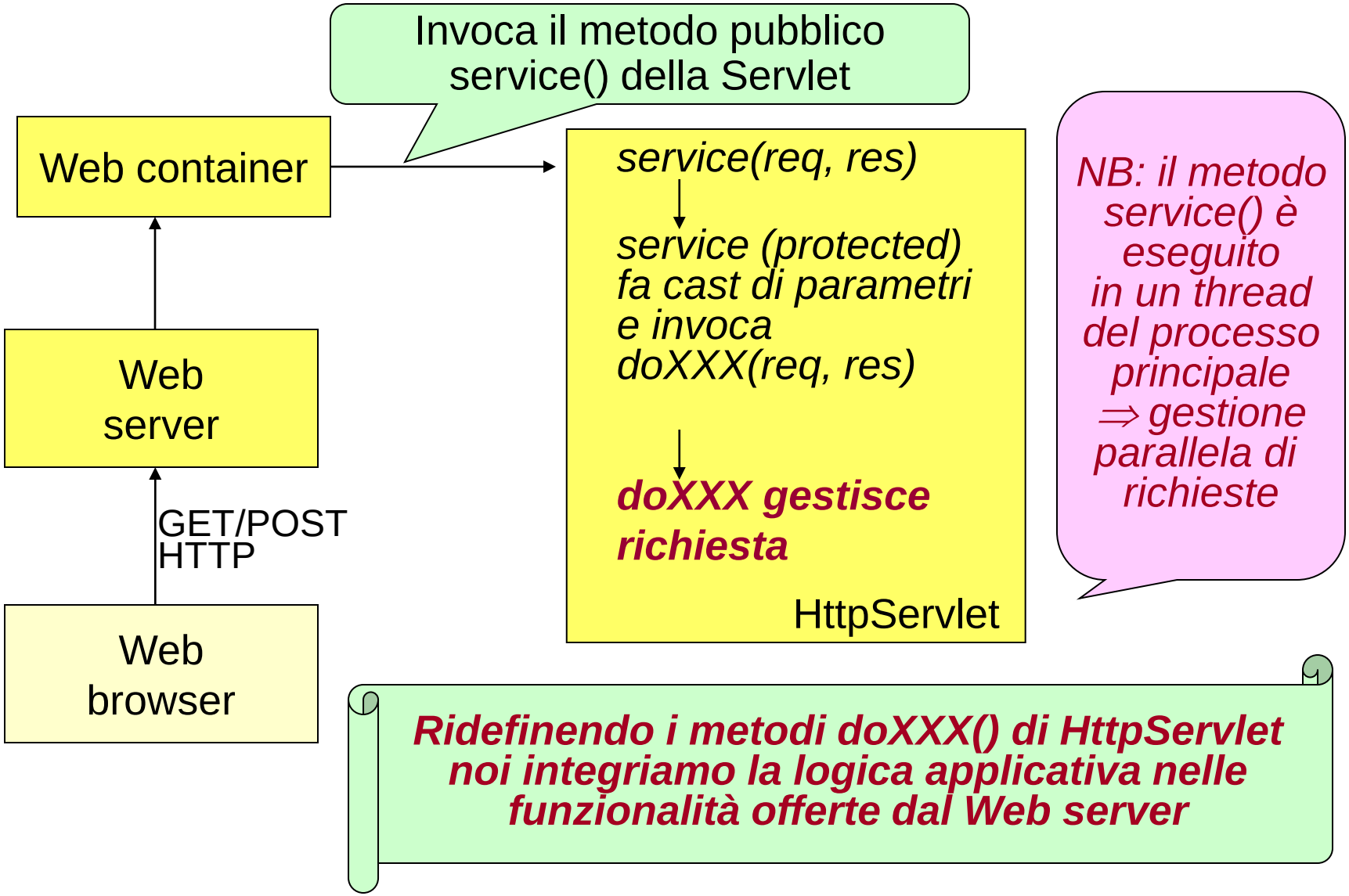
- **public void service(ServletRequest req, ServleResponse res)**
  - implementa il metodo service() di GenericServlet. Service fa un cast di ServletRequest/ServletResponse ad HttpServiceRequest/HttpServiceResponse e invoca il metodo service() protected.
  - **Non ridefinire (no overriding)!!**

- **protected void service(HttpServletRequest req, HttpServleResponse res)**
  - invoca req.getMethod() per sapere se è GET, POST, HEAD, … e delega gestione della richiesta al metodo doXXX() specifico….
  - **Non ridefinire!!!**

javax.Servlet.http - abstract class HttpServlet - II

- **protected void doGet(HttpServletRequest req, HttpServleResponse res)** throws ServletException, IOException
  - gestisce HTTP request di tipo GET. **Ridefinibile**
- **protected void doPost(HttpServletRequest req, HttpServleResponse res)** throws ServletException, IOException
  - gestisce HTTP request di tipo POST. **Ridefinibile**
- **protected void doHead(HttpServletRequest req, HttpServleResponse res)** throws ServletException, IOException
  - gestisce HTTP request di tipo HEAD. **Ridefinibile**
- **...**

Invocazione di una Servlet Java



Porzione di gerarchia delle classi - Request

Interface **javax.servlet.ServletRequest**

specifica
- *metodi per accedere a connessione client*
- *input stream associata a connessione*

supporta lettura di dati da input stream

→ InputStream **ServiceInputStream**

↓

interface **javax.servlet.http.HttpServletRequest**

*offre **implementazione** di ServletRequest **specifica per HTTP***

Porzione di gerarchia delle classi - Response

Interface **javax.servlet.ServletResponse**

specifica
- *metodi per accedere a connessione client*
- *output stream associata a connessione*

supporta scrittura dati per client

OutputStream
ServiceOutputStream

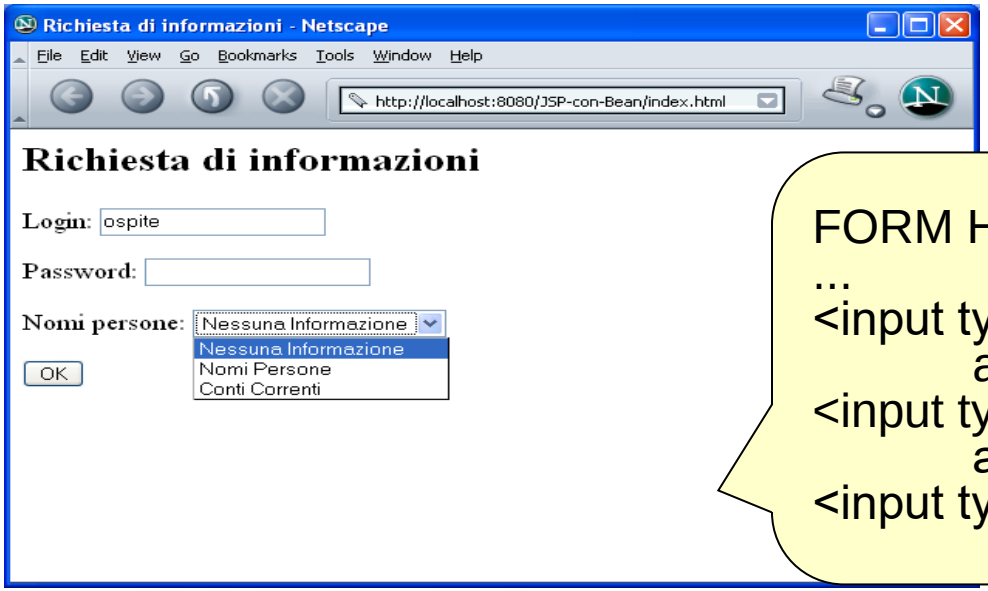
interface **javax.servlet.http.HttpServletResponse**

*offre **implementazione** di ServletResponse **specifica per HTTP***

Metodi per la comunicazione con il client - I

HttpServletRequest

- **String getParameter(String parameterName)**
  - analizza sintatticamente il contenuto della request HTTP e estrae il valore del parametro del metodo. Es:
    - String tipoInfo = req.getParameter("tipoInfo");
    - String loginName = req.getParameter(“login");




```
FORM HTML
…
<input type="text" name=“login"
          align="left" size="25"/>
<input type="password" name="passwd"
          align="left” size=“25”/>
<input type=“tipoInfo” … >
```

Metodi per la comunicazione con il client - II

HttpServletRequest

- **Object getAttribute(String name)**
  - restituisce valore dell'attributo “name”, se esiste. Null altrimenti
- **Enumeration getAttributeNames()**
  - restituisce l’enumerazione dei nomi degli attributi memorizzati nell'HttpRequest
- **void setAttribute(String name, Object o)**
  - inserisce nella HttpRequest un attributo di nome *name* e valore *o*
- ….

Metodi per la comunicazione con il client - III

HttpServletResponse

- **void setContentType(String mimeType)**
  - Specifica il formato MIME di output – importante per permettere al client di interpretare correttamente i dati

- **PrintWriter getWriter()**
  - restituisce un oggetto PrintWriter costruito dal Web Container a partire dall'OutputStream associato alla connessione col client
  - il PrintWriter serve per scrivere dati sull'output stream (i dati sono il contenuto della pagina di risposta per il browser)

Es.: SWEB-Servlet00/NewServlet: dettaglio - I

```
import …;
```

***Java annotation**: specifica il nome della Servlet e il suo path di invocazione all'interno dell'applicazione SWEB-Servlet00*

```
@WebServlet(name = "NewServlet",
                    value = {"/NewServlet"})
public class NewServlet extends HttpServlet {

@Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)  throws
ServletException, IOException {
        processRequest(request, response);
    }
```

Es.: SWEB-Servlet00/NewServlet: dettaglio - II

```
private void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter(); // output stream
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html><head>");
        out.println("<title>Servlet NewServlet</title></head>");
        out.println("<body>");
        out.println("<h1>Hello world!</h1>");
        out.println("</body></html>");
    } finally {
        out.close();
    }
}
```

NB: processRequest() non è un API:
esso serve solo per fattorizzare
il codice di doGet() e doPost()

Es.: SWEB-Servlet00/NewServlet in dettaglio - III

```
@Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

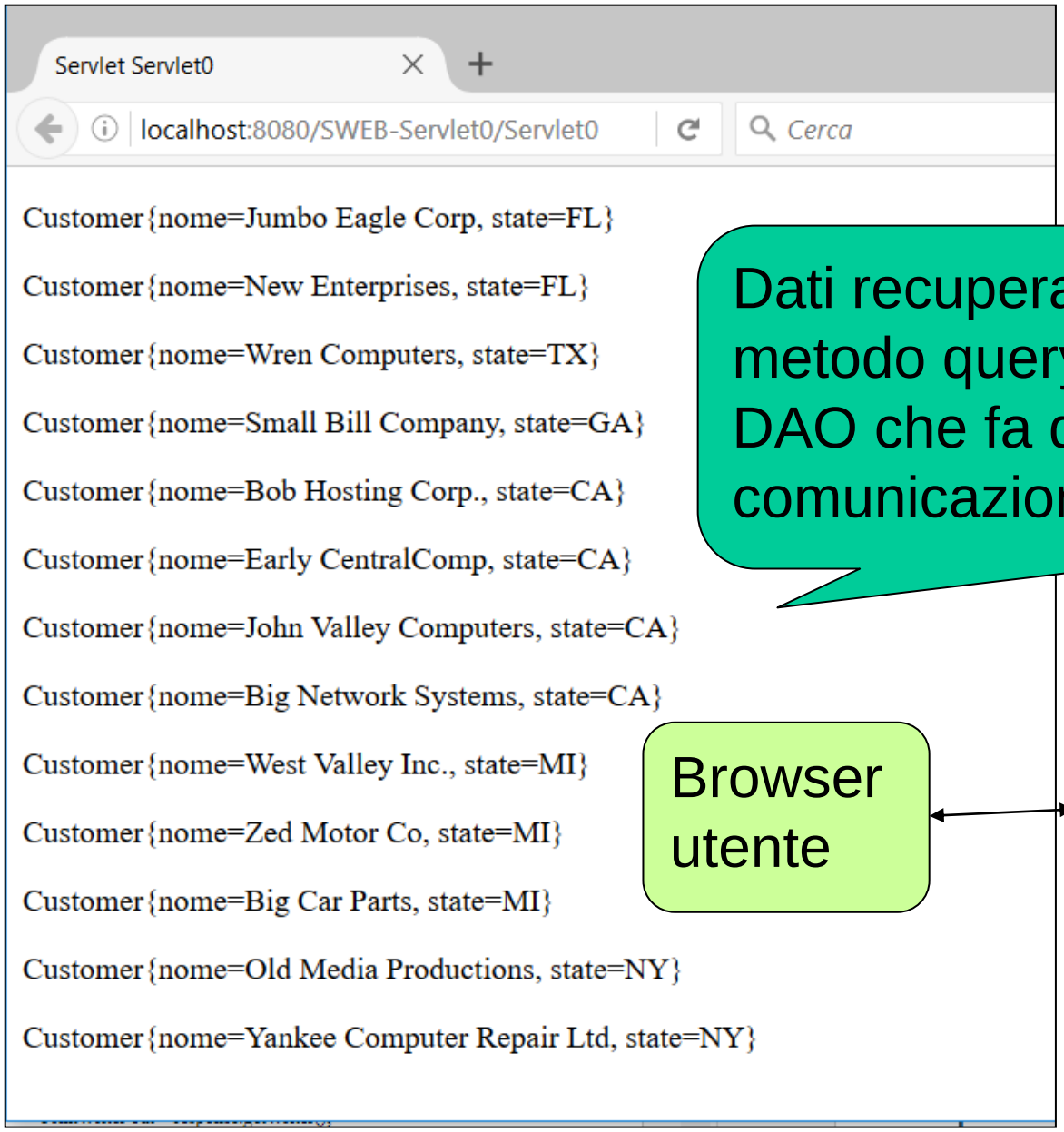
Usare un metodo private comune, invocato in entrambi i metodi doGet() e doPost(), aiuta a fattorizzare il codice della servlet.

Applicazioni basate su Servlet Java

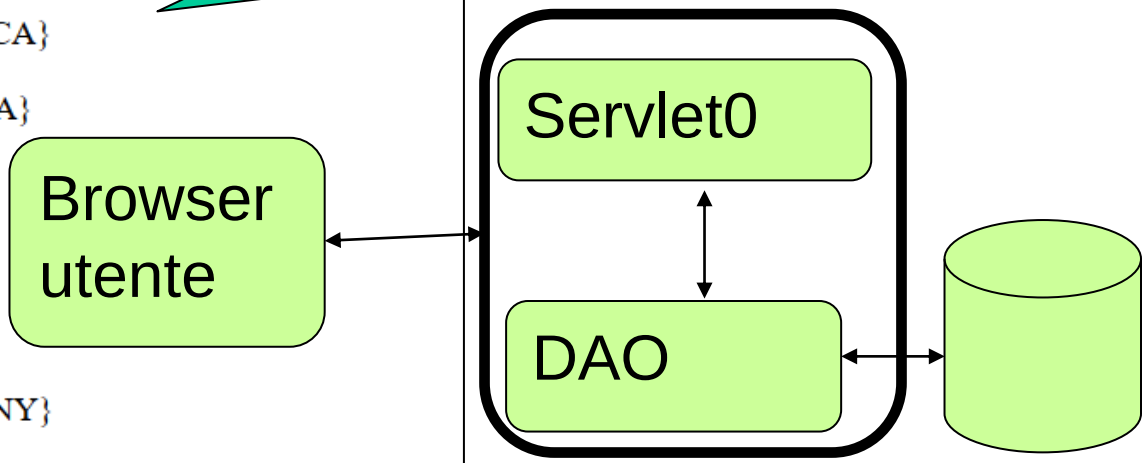
- Le Servlet Java sono normali classi Java: hanno solo la peculiarità di comunicare via HTTP

  → è facile invocare codice applicativo al loro interno
- Vediamo un esempio di Servlet che usa un DAO per accedere a dati in modo strutturato (come abbiamo fatto nell'applicazione stand alone di prova di JDBC)

Servlet0 - I



Dati recuperati dal DB, usando il metodo queryDB() di un oggetto DAO che fa da intermediario per la comunicazione con il DB



Porzione di codice di Servlet0

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response)  throws ServletException,
IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<!DOCTYPE html>"); out.println("<html><head>");
        out.println("<title>Servlet Servlet0</title></head>");
out.println("<body>");
        List<Customer> customers = DAO.queryDB();
        for (int i=0; i<customers.size(); i++)
            out.println("<p>" + customers.get(i) + "</p>");
        out.println("</body></html>");
    } finally {out.close();}
}
```

DAO: strato di software che accede al data source
Il Modello dei dati è Customer

Init() e doGet()/doPost()

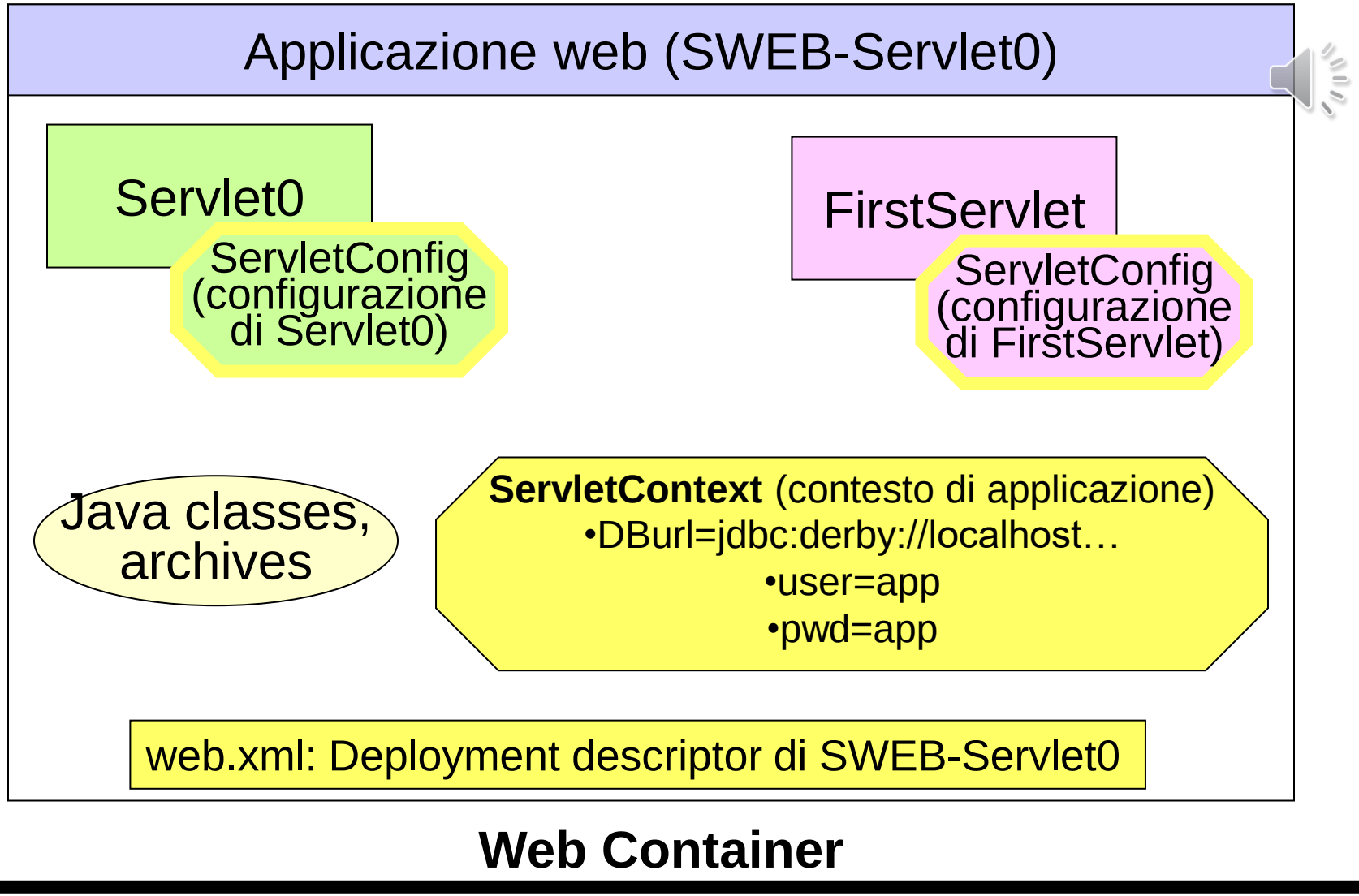
Quando si sviluppa una Servlet java

- Nel metodo **init()** si può inserire **codice da eseguire una sola volta** nel ciclo di vita della Servlet (es: inizializzazione di variabili di classe e di istanza della Servlet, registrazione del driver JDBC, etc.)
- Nel metodo **doGet()/doPost()** inserire **codice che va eseguito ad ogni richiesta utente** (thread paralleli)

NB: Esecuzione del metodo service()

- Ad ogni HTTP request il Web Container crea un **thread** per l'esecuzione richiesta ⇒ le richieste di client diversi sono gestite in parallelo nell'ambito del processo principale, con poco spreco di risorse computazionali
- **MA: attenti agli accessi a risorse in mutua esclusione.** Es:
  - variabili globali della Servlet, da non modificare in parallelo
  - accessi a DB che non garantiscono modifiche in mutua esclusione

Struttura di un'applicazione Java Web



Web Container

ServletConfig – parametri di configurazione di una Servlet - I

Il contesto di configurazione di una Servlet (**ServletConfig**)

- mantiene **i parametri di inizializzazione della Servlet a cui è associato**
- **contiene il riferimento al contesto di applicazione (ServletContext)**

ServletConfig – parametri di configurazione di una Servlet- II

- ServletConfig serve per specificare parametri di configurazione che interessano una specifica Servlet e che noi non vogliamo inserire nel codice della Servlet stessa → approccio dichiarativo alla configurazione del SW
- Durante il deployment dell'applicazione, il web container inizializza il ServletConfig di ciascuna Servlet basandosi sul contenuto del **web.xml,** o sulle **Java Annotations** associate alla Servlet

web.xml è memorizzato in WEB_INF

Porzione di gerarchia delle classi - Servlet

interface **javax.servlet.Servlet**

- init(ServletConfig c)
- service(ServletRequest req, ServleResponse res)
- destroy(), … *altri metodi* ...

interface **javax.servlet.ServletConfig**

mantiene dati di configurazione della Servlet

abstract class **javax.servlet.GenericServlet** implements Servlet, ServletConfig, Serializable

offre un'implementazione dell'interface Servlet indipendente dal protocollo di comunicazione (HTTP, …)

abstract class **javax.servlet.http.HttpServlet** extends GenericServlet implements Serializable

*offre **implementazione** di interface Servlet **specifica per HTTP***

javax.Servlet - Interface Servlet

Metodi per accedere a informazioni generiche sulla Servlet e all'oggetto ServletConfig

- **ServletConfig getServletConfig()**
  - restituisce l'oggetto ServletConfig associato alla Servlet
- **String getServletInfo()**
  - restituisce informazioni generali su Servlet (autore, data di creazione, ...)

javax.Servlet - Interface ServletConfig

- **String getInitParameter(String name)**
  - restituisce il valore del parametro di inizializzazione “name”, se esiste. Null altrimenti
- **Enumeration getInitParameterNames()**
  - restituisce l’enumerazione dei nomi dei parametri di inizializzazione della Servlet
- **String getServletName()**
  - Restituisce il nome della Servlet
- **ServletContext getServletContext()**
  - restituisce il riferimento al contesto dell’applicazione a cui appartiene la Servlet
- …

Configurazione di Servlet tramite Java annotations - I

I parametri di inizializzazione di una Servlet possono essere definiti tramite java annotations nella Servlet stessa. Esempio:

```
@WebServlet(name = "ProvaParametriInizializzazione",
                value = "/ProvaParametriInizializzazione", initParams = {
                        @WebInitParam(name = "param1", value = "val1"),
                        @WebInitParam(name = "param2", value = "val2")})
public class ProvaParametriInizializzazione extends HttpServlet {
   private String par1 = ""; private String par2 = "";

   public void init(ServletConfig conf) throws ServletException {
      super.init(conf);
      // prendo parametri da servlet config perchè le java annotations li inseriscono lì
      par1 = conf.getInitParameter("param1");
      par2 = conf.getInitParameter("param2");
   }
```

Recupero i parametri di configurazione della Servlet dal suo ServletConfig

Configurazione di Servlet tramite Java annotations - II

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
    IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
          out.println("<!DOCTYPE html>");
          out.println("<html><head>");
          out.println("<title>Servlet ProvaParametriInizializzazione</title>");
          out.println("</head>");
          out.println("<body>");
          out.println("param1 = " + par1 + "</br>");
          out.println("param2 = " + par2);
          out.println("</body></html>");
      } finally {
          out.close();
}}
```

ServletContext – parametri di configurazione di un'applicazione - I

In un’applicazione Web, più pagine potrebbero utilizzare gli stessi dati. Per esempio, due pagine potrebbero interrogare il DB (usando url, user and password comuni)

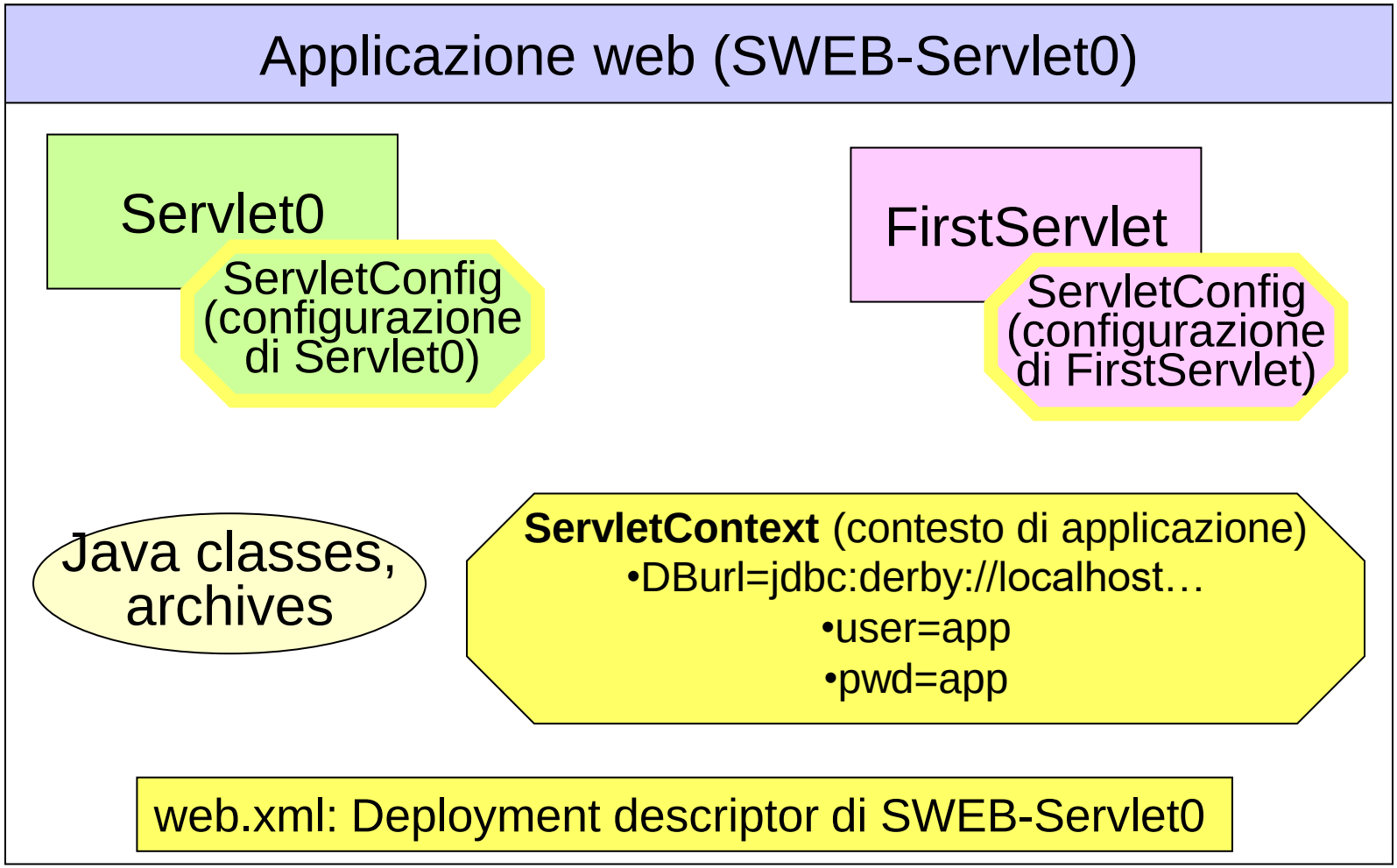
Per separare il codice di un’applicazione Web dai dati di configurazione generali (che possono variare di installazione in installazione, o nel tempo) si usano:

- un file di configurazione testuale (XML), ove si memorizzano tali dati: **web.xml**. Questo file si chiama **deployment descriptor** dell’applicazione
- *oppure le **java annotations**: annotazioni direttamente inserite in forma testuale nel codice sorgente - @*

SWEB-Servlet0 – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0“ xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ... ">
   <context-param>
      <description>url del DB </description>
      <param-name>DB-URL</param-name>
      <param-value>jdbc:derby://localhost:1527/sample</param-value>
   </context-param>
   <context-param>
      <description>account per accedere a DB</description>
      <param-name>user</param-name>
      <param-value>app</param-value>
   </context-param>
..... </webapp>
```

Struttura di un'applicazione Java Web-based



Web Container

javax.Servlet - Interface ServletContext

- **String getInitParameter(String name)**
  - restituisce valore del parametro di inizializzazione “name”, se esiste. Null altrimenti.
- **Enumeration getInitParameterNames()**
  - restituisce l’enumerazione dei nomi dei parametri di inizializzazione dell'applicazione.
- **Object getAttribute(String name)**
  - restituisce valore dell'attributo “name”, se esiste. Null altrimenti.
- **Enumeration getAttributeNames()**
  - restituisce l’enumerazione dei nomi degli attributi memorizzati nel ServletContext.
- **…**

Esempio: SWEB-Servlet0 – I

```
public class Servlet0 extends HttpServlet {
    DAO dao = null;

    public void init(ServletConfig conf) throws ServletException {
        super.init(conf);
        ServletContext ctx = conf.getServletContext();
        String url = ctx.getInitParameter("DB-URL");
        String user = ctx.getInitParameter("user");
        String pwd = ctx.getInitParameter("pwd");
        dao = new DAO(url, user, pwd);
    }
```

Inizializzo il DAO con i dati di accesso al DB

Nel metodo init() inizializzo **(1 sola volta)** le variabili della Servlet per accedere ai dati

Esempio: SWEB-Servlet0 – II

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
                        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html><head><title>Servlet Servlet0</title></head>");
        out.println("<body>");
        List<Customer> customers = dao.queryDB();
        for (int i = 0; i < customers.size(); i++) {
            out.println("<p>" + customers.get(i) + "</p>");
        }
        out.println("</body></html>");
    } finally { out.close(); }}
```

Esempio: SWEB-Servlet0 – DAO - I

```
public class DAO {
    private final String URL; // url del DB
    private final String USER; // login utente da usare per connettersi
    private final String PWD; // password utente
    public DAO(String url, String user, String pwd) {
        URL = url; USER = user; PWD = pwd;
        registerDriver();
    }
    public static void registerDriver() {
        try {
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        } catch (SQLException e) {System.out.println(e.getMessage());}
    }
```

Esempio: SWEB-Servlet0 – DAO - II

```
public ArrayList<Customer> queryDB() {
    ArrayList<Customer> out = new ArrayList();
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PWD);
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM CUSTOMER");
        while (rs.next()) {
            Customer c = new Customer(rs.getString("NAME"),
                                      rs.getString("STATE"));
            out.add(c);
        }
        rs.close(); st.close(); conn.close();
    } catch (SQLException e) {System.out.println(e.getMessage());};
    return out;
}}
```

NB: Gestione di eccezioni durante esecuzione di Servlet java

La **firma dei metodi doPost(), doGet(),** etc. prevede la throws di ServletException e IOException

- **non si può modificare la signature per lanciare altre eccezioni** se no si definisce un nuovo metodo della Servlet, che non verrà mai invocato – *la modifica non fa overriding bensí overloading!*
- negli accessi a DB (o di altro codice che può lanciare eccezioni) bisogna catturare una SQLException (o altro) e
  - **gestirla localmente**, in modo che non causi il lancio di un'eccezione non prevista dalla signature dei metodi, **oppure**
  - **rilanciare una ServletException il cui message sia quello dell'eccezione originale** (così chi riceve l'eccezione può visualizzare il messaggio)

Gestione di eccezioni durante esecuzione di Servlet java - esempio

```
... try { apro connessione a DB;
          eseguo query SQL; .... altro codice ... ;
          chiudo oggetti Resultset, Statement e Connection;
        } catch (SQLException e)
          { ServletException e1 =
                        new
ServletException(e.getMessage());
                  throw e1;}
 ...
out.println("</body></html>");
 out.close(); }
```

*La throw rilancia **ServletException** (rispettando la signature) ma usa il message dell'eccezione originale per descrivere l'errore. Provare a fare query sbagliata a DB per vedere…*