# Java

*progetto applicazioni Web -- servlets*

G. Prencipe
prencipe@di.unipi.it

## Introduzione

- Un tipico modo per consentire a clienti l'accesso a dati e risorse è quello di utilizzare Internet
  - Tipicamente, viene creata una pagina HTML con un pulsante *submit*
  - Il cliente compila i vari campi presenti nella pagina, e alla pressione del *submit* i dati vengono inviati

## CGI

- I dati sono sottomessi tramite una URL che comunica al server cosa fare con i dati
  - Viene specificata la locazione di un programma *Common Gateway Interface* (*CGI*) che il server esegue
  - Al programma vengono forniti i dati sottomessi
- Il programma CGI è tipicamente scritto in Perl, Python, C, C++, o qualsiasi linguaggio che può leggere da standard input e scrivere su standard output

## CGI

- Quindi, il Web server ha il *solo* compito di invocare il programma CGI corretto, e gli stream standard sono utilizzati per input e output
- Il programma CGI si occupa di tutto il resto
  - Controlla i dati e decide se il loro formato è corretto
  - Se non lo è, il programma deve produrre una pagina HTML che descrive il problema
    - Questa pagina è passata al Web server (tramite standard output dal programma CGI) che la visualizza all'utente
    - L'utente deve a questo punto ritornare alla pagina iniziale e riprovare
  - Se i dati sono corretti, il programma CGI li processa in modo opportuno, e produce una pagina HTML da far visualizzare all'utente dal Web server

## Servlets

- La soluzione basata completamente su Java per fare tutto questo sono le servlets
- Rimpiazzano del tutto la programmazione CGI
- Hanno diverse similitudini con le applets
  - Hanno un "ciclo di vita"
  - Non hanno un **main()**
  - Rivediamo rapidamente le applet e analizziamo le differenze con le servlets

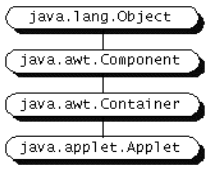
## Le Applet Java

- Le Applet sono piccole applicazioni Java che vengono eseguite all'interno di un web browser
- Forniscono funzionalità simili a quelle delle applicazioni Java
- Vengono eseguite in un *ambiente protetto* (la *sandbox*)
  - Le limitazioni dell'ambiente assicurano che un'Applet maligna non possa fare troppo danno
- Spesso usano i JavaBeans per la GUI e per altre funzionalità

## Le Applet Java

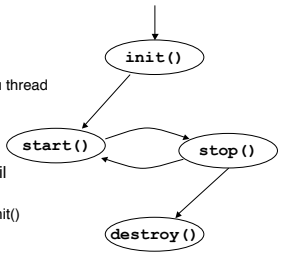
- Per le applicazioni Java, il primo metodo eseguito è il **`main()`** di una classe qualunque
- Per le applet, il ciclo di vita è più complicato:
  - l'applet deve essere sottoclasse di **`Applet`** (o di **`JApplet`**, che è una sua sottoclasse)
  - al caricamento, viene chiamato **`init()`**, poi **`start()`**
  - se l'utente cambia pagina e poi vi ritorna, vengono chiamati **`stop()`** e **`start()`** – anche più volte
  - alla fine, viene chiamato **`stop()`**, poi **`destroy()`**

## Le Applet Java



## Ciclo di vita delle Applet

- init() per le inizializzazioni
  - serve come un costruttore
- start() per avviare i lavori
  - direttamente o creando uno o più thread
- stop() per sospendere o chiudere
  - si fermano i thread
- destroy() quando l'utente chiude il browser
  - si liberano le risorse allocate in init()

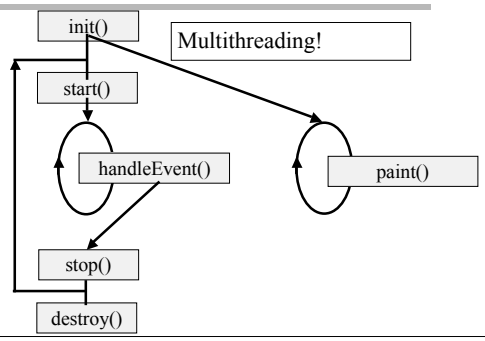



## Ciclo di vita delle Applet

Una Applet può reagire a vari *eventi*. Può:

- ***inizializzarsi*** **init()**
- ***partire*** **start()**
- ***disegnare*** **paint()**
- ***reagire*** a eventi generati dall'utente (Mouse, keyboard, menus…). **handleEvent()**
- ***fermarsi*** **stop()**
- Prepararsi ad essere eliminata **destroy()**

## Ciclo di vita delle Applet




## Accesso ai parametri

- Le Applet possono avere dei *parametri*
- I parametri vengono specificati con dei tag HTML:

```
<APPLET code=Histo.class width=500 height=100>
  <PARAM name=sfondo value=#e0e0ff>
  <PARAM name=colore value=#300010>
  <PARAM name=valori value="10,40,55,53,58,60,68">
</APPLET>
```

- Le Applet recuperano i valori con

```
public String getParameter(String name)
```

## GUI nelle Applet

- Più tipicamente, le Applet non vengono disegnate “a mano”
- Ci si limita a definire il layout manager (es.: GridBagLayout), e poi si aggiungono componenti GUI predefiniti
  - pulsanti, liste, campi testo, ecc.
- La gestione della GUI è uguale a quella delle applicazioni “normali” in Java

## Limitazioni delle Applet

- Per prevenire danni, le applet
  - non possono accedere ai dischi
  - non possono aprire socket verso siti diversi da quello da cui provengono
  - non possono aprire finestre che si “spaccino” per finestre del S.O. locale
  - non possono avviare programmi sulla macchina locale
  - e altro...

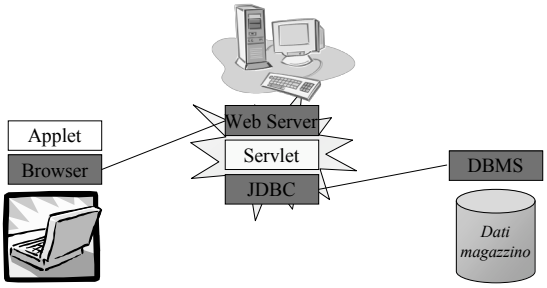
## Limitazioni delle Applet

- Queste limitazioni possono essere superate acquisendo i *diritti* relativi, con il consenso dell’utente
- Il modello di protezione è piuttosto complicato
  - Basato sulla crittografia a chiave pubblica
  - Necessita di certificati e firme digitali

## Servlet, chi era costui?

- Le Servlet sono piccole applicazioni Java che vengono eseguite all’interno di un Web server
- Forniscono funzionalità simili a quelle dei CGI
- Primo Teorema delle Servlet:
  - Le Servlet stanno a un Web Server come le Applet stanno a un Web Browser
- A differenza delle Applet, le Servlet non hanno interfaccia grafica
- Possono però usare comunque i componenti **JavaBeans** (per esempio, per l’accesso ai dati)

## Esempio: gestione magazzino




## Applet & Servlet

- Le applet si usano spesso senza servlet
  - Applet che non necessitano di processing server-side: ricevono un po' di parametri all'avvio, e si limitano a mostrarli
  - In altri casi, le applet si collegano a un server (non servlet) scritto in Java o altri linguaggi, e usano un protocollo privato
  - Volendo, le applet possono comunicare con il server via HTTP (simulano una FORM)

## Applet & Servlet

- Le servlet si usano spesso senza applet
  - Spesso processano input proveniente da FORM tradizionali
  - In altri casi, i dati possono essere generati da una pagina con Javascript, memorizzati in campi hidded di una FORM "invisibile" e inviati al servlet
  - È anche possibile che un servlet processi input proveniente da altre sorgenti

## Applet & Servlet

- Applet e servlet si possono anche usare insieme
  - L'applet raccoglie e pre-elabora i dati, offrendo una bella GUI all'utente
  - Quando i dati sono pronti, li manda al server via HTTP
  - Il server li passa alla servlet per l'elaborazione server-side
- È però un uso poco comune

## Applets vs. Servlets

| | Applet | Servlet |
|---|---|---|
| Gira: | Client | Server |
| Ha un main: | NO | NO |
| Estende: | java.applet.Applet | javax.servlet.http.HttpServlet |
| Grafica | SI | NO |
| Cuore: | handleEvent() | service() |

## Differenza fra Servlet e CGI

- Gli script CGI vengono eseguiti dal S.O., quindi sono potenzialmente meno portabili
- Le Servlet vengono eseguite dalla JVM integrata nel Web Server, quindi sono "isolate" dal S.O. e dunque più portabili

## Differenza fra Servlet e CGI

- Gli script CGI vengono caricati ed eseguiti una volta per ogni richiesta – quindi, il costo di avvio (latenza) è alto
- Le Servlet vengono caricate solo una volta; poi si crea un Thread (di Java) per ogni richiesta – operazione meno costosa, e dunque si ha una latenza più bassa

## Differenza fra Servlet e CGI

- Gli script CGI possono essere scritti in qualunque linguaggio: potete scegliere il linguaggio più adatto al particolare scopo
- Le Servlet devono essere scritte necessariamente in Java: spesso va bene, ma a volte non è conveniente

## Differenza fra Servlet e CGI

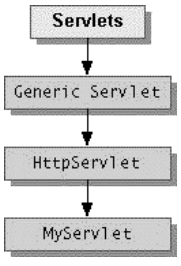
- Il protocollo CGI è supportato da tutti i Web server (anche se a volte con qualche piccola differenza)
- Le Servlet sono supportate solo da alcuni Web server (spesso detti "application server")
  - C'è comunque una buona scelta di prodotti commerciali

## Server che supportano le Servlet

| Prodotto | Versione Servlet supportata | Versione JSP supportata |
|---|---|---|
| Apache & Sun – Tomcat | 2.3 | 1.1 |
| Apache + Jserv | 2.0 | No |
| Borland AppServer | 2.2 | 1.1 |
| IBM WebSphere Application Server | 2.2 | 1.1 |
| IONA iPortal Application Server | 2.2 | 1.1 |
| iPlanet Application Server<br>iPlanet Web Server | 2.2 | 1.1 |
| Oracle 9i JServer | 2.3 | 1.1 |
| ... e parecchi altri ... | | |

## Struttura delle Servlet

- Le Servlet sono **classi Java** che implementano l'interfaccia **`Servlet`**
- Una classe può implementare **`Servlet`** direttamente
- Ma più comunemente estende **`HttpServlet`**




## HTTP: Request & Response

- I clienti inviano una *richiesta* HTTP (*request*) specificando il tipo di azione che deve essere eseguita
  - GET
  - POST
- Il server invia una *risposta* (*response*)

## HTTP: GET & POST

- Metodo GET
  - Per ottenere informazioni dal server
  - Può includere una *query string*, sequenza di informazioni addizionali aggiunte alla fine della URL
    - È possibile fare il bookmark della query string
    - Limitata in lunghezza (240 caratteri)

Esempio: http://www.whoami.com/Servlet/Search?name=Inigo+Montoya

## HTTP: GET & POST

- Metodo POST
  - Per inviare informazioni al server
  - Tutte le informazioni addizionali (lunghezza illimitata) passate come parte della richiesta
    - Invisibile all'utente
    - Non può essere ricaricata (reloaded)

## Dialogo con le Servlet

- Il dialogo fra Servlet e il browser è di tipo client/server
- client → server
  - interfaccia ServletRequest
- server → client
  - interfaccia ServletResponse

## Servlet e HttpServlet

- L'interfaccia delle Servlet è piuttosto generica
- La classe di sistema HttpServlet fornisce una versione di Servlet specializzata per il protocollo HTTP
  - ci sono HttpServletRequest e HttpServletResponse associate
- Nel seguito ci concentreremo sulle servlet HTTP

## Interfaccia HttpServlet
metodi principali

| Metodo | Descrizione |
|---|---|
| doGet() | chiamata quando arriva una richiesta di tipo GET |
| doPost() | chiamata quando arriva una richiesta di tipo POST |
| doPut() | chiamata quando arriva una richiesta di tipo PUT |
| doDelete() | chiamata quando arriva una richiesta di tipo DELETE |

## Dialogo con le Servlet

- Tutti i metodi hanno segnatura

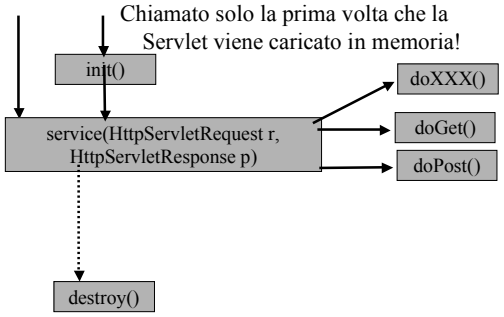
```
void doXXX(  ServletRequest richiesta,
             ServletResponse risposta  )
```

- Il metodo legge da **`richiesta`** tutti i dettagli della richiesta che è arrivata
- Esegue il compito vero e proprio
- Infine, scrive in **`risposta`** la risposta che vuole inviare al client

## Un esempio (una HttpServlet)

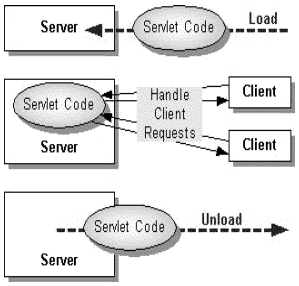
```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest richiesta,
                    HttpServletResponse risposta)
  throws ServletException, IOException
  {
    String titolo = "Risposta dalla mia Servlet";
    risposta.setContentType("text/html");
    PrintWriter out = risposta.getWriter();
    out.println("<HTML><HEAD><TITLE>");
    out.println(titolo);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + titolo + "</H1>");
    out.println("<P>... resto della pagina ...");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

## Ciclo di vita di una Servlet



## Ciclo di vita di una Servlet

- Al primo caricamento viene chiamato **init()**
- Per ogni richiesta, viene chiamato uno dei do*XXX*()
- Alla fine, viene chiamato **destroy()**
  - es: quando il web server termina




## Cosa si può leggere dalla richiesta

Alcuni metodi di HttpServletRequest danno accesso alle informazioni inviate insieme alla richiesta HTTP

| Tipo richiesta | Metodo | Descrizione |
|---|---|---|
| tutti | getParameter(s) | Restituisce il valore del parametro *s* (es.: da una FORM) |
| | getHeader(s) | Restituisce il valore dell'header HTTP *s* |
| GET | getQueryString() | Restituisce la stringa dei parametri (URL encoded) |
| POST, PUT e DELETE | getReader() | Restituisce un BufferedReader da cui leggere i dati (testuali) inviati |
| | getInputStream() | Restituisce un InputStream da cui leggere i dati (binari) inviati |

## Altri metodi di HttpServletRequest

- getAuthType()
- getCookies()
- getDateHeader(String)
- getHeader(String)
- getHeaderNames()
- getIntHeader(String)
- getMethod()
- getPathInfo()
- getPathTranslated()
- getQueryString()
- getRemoteUser()
- getRequestedSessionId()
- getRequestURI()
- getServletPath()
- isRequestedSessionIdFromCookie()
- isRequestedSessionIdFromURL()
- isRequestedSessionIdValid()

## Cosa si può scrivere nella risposta

Alcuni metodi di HttpServletResponse consentono di inviare i dati che costituiscono la risposta verso il client

| Metodo | Descrizione |
|---|---|
| getWriter() | Restituisce un Writer su cui stampare i dati (testuali) da inviare al client |
| getOutputStream() | Restituisce un OutputStream su cui scrivere i dati (binari) da inviare al client |
| setHeader(h,v) | Imposta l'header HTTP *h* al valore *v* |
| setStatus(c,d) | Imposta lo stato HTTP che deve essere restituito; *c* è il codice numerico, *d* è la descrizione testuale |
| addCookie(k) | Invia al client un Cookie *k* |

## Altri metodi di HttpServletResponse

- containsHeader(String)
- encodeRedirectURL(String)
- encodeURL(String)
- sendError(int)
- sendError(int, String)
- sendRedirect(String)
- setDateHeader(String, long)
- setIntHeader(String, int)
- setStatus(int)

## Buffering

- La servlet bufferizza i dati in uscita
- I dati scritti sul PrintWriter (o OutputStream) di HttpServletResponse vengono effettivamente inviati al browser quando
  - Il flusso viene chiuso con out.close()
  - Si chiama HttpServletResponse.flushBuffer()
  - Il buffer è pieno; in questo caso viene svuotato il buffer e si continua
- È possibile cambiare gli header solo finché non viene inviato il primo blocco di dati
  - setHeader(), setStatus(), addCookie(), ecc.

## Buffering

- È possibile intervenire sul buffer con vari metodi di ServletResponse:

| Metodo | Descrizione |
|---|---|
| flushBuffer() | Manda i dati nel buffer al client; svuota il buffer |
| resetBuffer() | Svuota il buffer, cancellando il contenuto corrente, che non viene inviato al browser |
| getBufferSize() | Restituisce la dimensione corrente del buffer |
| setBufferSize() | Imposta la dimensione del buffer |
| isCommitted() | True se gli header sono già stati inviati |

-

## Caratteristiche delle Servlet

- Velocità
  - molto veloci, il codice è compilato, e una volta caricato viene tenuto in memoria dal server – bassa latenza e alta banda
- Portabilità
  - ottima – Java è sempre lo stesso ovunque
- Fattori temporali
  - provare una Servlet richiede (i) modificare il codice (ii) compilare il codice (iii) riconfigurare il web server: non è adatto alla prototipazione rapida

## Caratteristiche delle Servlet

- Competenze
  - con poche aggiunte, basta conoscere Java
  - Per servlet complesse, occorre conoscere Java *bene*…
- Supporto
  - il supporto da parte di terzi è ancora un po' limitato (ma in crescita: si aspetta che diventi standard in Apache)
  - la tecnologia è portabile, ma al momento dipende molto da Sun/IBM/Borland/Oracle
  - Ci sono ambienti di sviluppo molto belli e completi (inclusi Eclipse e IBM WebSphere Studio)

## Strumenti per le servlet

- Installare un *Application Server*
  - Uno dei più diffusi è Apache Tomcat
  - Gratuito e performante per la maggiorparte delle applicazioni che tipicamente girano su server
- Una volta installato Tomcat, è possibile visualizzare la pagina di test accedendo

  **localhost:8080**
- Questo vale se viene utilizzato il file di configurazione di default
  - Modificandolo, è possibile modificare questo inidirizzo di accesso

## Configurazione dell'application server

- Punto dolente!
- Ogni server ha modalità proprie per la configurazione
  - Tomcat
  - Apache
  - J2SDK
- Non possiamo farci nulla: tocca studiare i manuali caso per caso

## Strumenti per le servlet

- Le classi per le servlet sono contenute nella distribuzione Enterprise Edition di Java (j2ee)
  - Una volta installata, bisogna includere nel BuildPath del progetto Eclipse gli archivi .jar relativi a j2ee (j2ee.rar e simili)
    - In Eclipse: pulsante destro sul progetto, Properties, Java Build Path, Libraries, Add ExternalJars
  - È possibile utilizzare un plug-in

## Strumenti per le servlet

- È possibile utilizzare un plug-in di Eclipse per facilitare la scrittura e il deployment delle servlet
  - Utilizzare Eclipse 3.2
  - Dal sito di Eclipse (www.eclipse.org), selezionare i plug-in relativi al *Web Development*, e individuare la URL da cui scaricarli
  - Da Eclipse: Help, SoftwareUpdates, FindAndInstall, SearchForNewFeatures, NewRemoteSites, e inserire la URL individuata
  - Oppure scaricare gli update proposti in Callisto
- Una volta installato il plugin, è possibile utilizzarlo creando progetti Web (New, Project, e poi selezionare Web)

## Strumenti per le servlet

- Dopo aver scritto una servlet, bisogna farne il *deployment*
  - In altre parole, va installate in modo che possa essere acceduta
- Utilizzando la configurazione di default di Tomcat, i file **.class** della nostra applicazione vanno copiati nella cartella

  **Apache-Tomcat/webapps/ROOT/WEB-INF/classes**
  - Se la cartella **classes** non esiste, va creata
- A questo punto, l'applicazione può essere acceduta da **localhost:8080/servlet/NomeApplicazione.class**

## Esercizio 1

- Scrivere una servlet **showParameters()** che stampi i parametri (coppia nome—valore) passati nella richiesta
  - Es: localhost:8080/servlet/showParamaters?pippo=pluto → mostrerà (pippo, pluto)

## Esercizio 2

- Scrivere una servlet che accetta due interi, li somma, e visualizza il risultato.
  - Scrivere una pagina web che invoca la servlet (usare una FORM)

## Gestione della persistenza

- Il protocollo HTTP non ha "memoria"
  - Non si può decidere se una pagina in un server è stata visitata dallo stesso utente
- **Ogni singola richiesta HTTP** viene trattata separatamente da tutte le altre
  - Come fa una povera servlet a gestire sessioni che comprendono più di una richiesta?
  - Come fanno più servlet che costituiscono una sola applicazione a passarsi dati e comunicare fra di loro?

## Gestione della persistenza

- Usando i **cookie**
  - La servlet scrive i dati che vuole rendere persistenti tramite i cookie
  - Successive richieste – alla stessa servlet o a servlet diverse – si porteranno dietro i cookie scritti nel client
- Usando **sessionID**
  - La servlet associa un identificatore unico ad ogni sessione
  - Il sessionID viene usato come chiave per accedere a un DB sul server
  - Successive richieste accedono al DB con la stessa chiave

## Gestione del session ID

1. Si estrae la sessione dalla richiesta, creandola se necessario (primo accesso):
   ```
   HttpSession sess=
               richiesta.getSession(true);
   ```
   - **`getSession()`≡`getSession(true)`:**
     - Il server restituisce la sessione corrente, se esiste; altrimenti ne crea una
   - **`getSession(false)`**
     - Viene restituita la sessione corrente o null
2. Si memorizzano/leggono i dati:
   a. da un DB, usando **`sess`** come chiave
   b. dalla sessione stessa, usando **`sess.setAttribute(chiave,valore)`** e **`sess.getAttribute(chiave)`**

## Gestione del session ID

- Ogni sessione contiene una coppia nome—valore: nome è una **String**, valore un **Object** (deve implementare Serializable)
  - **setAttribute(String name, Object value)** aggiunge un oggetto alla sessione
  - **getAttribute(String name)** legge dalla sessione
  - **removeAttribute(String name)** elimina un oggetto dalla sessione

## Trasferire l'esecuzione

- Come abbiamo visto negli esempi precedenti, quando da una pagina HTML si vuole trasferire il controllo a una servlet, è sufficiente invocarla dalla ACTION della FORM
- Se si vuole trasferire il controllo da una servlet S1 a una servlet S2 bisogna
  - Invocare in S1 il metodo **sendRedirect()** della classe **HttpServletResponse**, specificando S2 come argomento
  - Implementare il metodo **service()** o **doGet()** in S2
  - In questo modo, la **sendRedirect()** avrà come effetto quello di invocare **service()** o **doGet()**

## Esercizio 3

- Scrivere una servlet che gestisca un "carrello della spesa"
  - Utilizzare i **SessionID**
  - Selezionare tra pere e mele
  - Selezionare la quantità (in Kg)
  - Uscire (checkout)
- Suggerimenti….

## Esercizio 3 -- suggerimenti

- Scrivere una pagina HTML **CarrelloSpesa.html** con una interfaccia minimale (Pere, Mele, Esci)
  - Alla pressione di **Submit**, viene invocata una servlet **Selezione**
  - Implementare **Selezione**
    - Usa **SessionID** per ricordare la selezione fatta
    - Se era stata selezionata l'uscita, viene invocata una servlet **Uscita**
    - Altrimenti **Selezione** lascia inserire un peso e permette di scegliere se tornare al carrello o andare all'uscita
    - Alla pressione di **Submit** viene invocata una servlet **Peso** o **Uscita**
- Implementare **Peso**, che aggiunge la quantità di prodotto selezionata e torna al carrello o va all'**Uscita**
- Implementare **Uscita**, che legge dalla sessione la quantità dei prodotti selezionati, e visualizza un prospetto riassuntivo

## Cookies

- Forniscono un altro metodo per memorizzare informazioni
- Mentre le Session memorizzano dati per la durata di una visita, le informazioni memorizzate con i Cookies possono essere utilizzate anche per visite successive
- Un Cookie è una coppia nome—valore: entrambe sono String
- Ogni Cookie è memorizzato in un file mandato dal server al cliente, e letto dal server in successive visite
- Un cookie può essere conservato solo per una sessione, o per più sessioni

## Gestione dei cookie – scrittura

1. Si crea un cookie con
   `cookie = new Cookie(nome,valore);`
2. Si impostano gli attributi del cookie
   `cookie.setComment(...);`
   `cookie.setDomain(...);`
   `cookie.setMaxAge(...);`
   ...eccetera
3. Si invia il cookie con la risposta
   `risposta.addCookie(cookie);`

## Gestione dei cookie – lettura

1. Si recuperano i cookie dalla richiesta
   `k = rich.getCookies();`
   - Restituisce un vettore
2. Si cerca il "nostro" cookie
   ```
   for (i=0; i<k.length; i++)
        if (k[i].getName.equals("nostro")
           { nostro=k[i]; break; }
   ```
3. Si estrae il valore memorizzato nel cookie
   `valore=nostro.getValue();`

## Impostare dei Cookies

```
import java.io.*;   import javax.servlet.*; import javax.servlet.http.*;
/** Imposta 6 cookies: 3 durano solo per la sessione corrente, e  3 durano per un ora
*/
public class SetCookies extends HttpServlet {
  public void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
      for(int i=0; i<3; i++) {
      // Default maxAge is -1, che indica che il Cookie vale solo per questa sessione
      Cookie cookie = new Cookie("Session-Cookie-" + i,"Cookie-Value-S" + i);
      response.addCookie(cookie);
      cookie = new Cookie("Persistent-Cookie-" + i,"Cookie-Value-P" + i);
      // Il Cookie è valido per un ora, anche se il browser viene chiuso o fatto il reboot
      cookie.setMaxAge(3600);
      response.addCookie(cookie);
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Setting Cookies";
    out.println (ServletUtilities.headWithTitle(title) +  "<BODY BGCOLOR=\"#FDF5E6\">\n" +"<H1
       ALIGN=\"CENTER\">"
      + title + "</H1>\n" +"Ci sono 6 Cookies associati a questa pagina.\n" + "</BODY></HTML>");
  }
}
```

## Leggere i Cookies

```
import java.io.*;  import javax.servlet.*;   import javax.servlet.http.*;
/** Crea una tabella che mostra i Cookies associati alla pagina */
public class ShowCookies extends HttpServlet {
      public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Cookies Attivi";
      out.println(ServletUtilities.headWithTitle(title) +
              "<BODY BGCOLOR=\"#FDF5E6\">\n" +
              "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
              "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
              "<TR BGCOLOR=\"#FFAD00\">\n" +
              "  <TH>Cookie Name\n" + "  <TH>Cookie Value");
      Cookie[] cookies = request.getCookies();
      Cookie cookie;
      for(int i=0; i<cookies.length; i++) {
      cookie = cookies[i];
      out.println("<TR>\n" +
                "  <TD>" + cookie.getName() + "\n" +
                "  <TD>" + cookie.getValue());
  } out.println("</TABLE></BODY></HTML>"); }}
```

## Esercizio 4

- Scrivere una servlet che
  - Chieda due interi da sommare
  - Controlli i cookies
    - Se ce n'è uno **NonPrimaVisita**, allora nella pagina di risposta stampa *Bentornato*
    - Altrimenti stampa *Benvenuto*

## Altri temi legati alle Servlet

- Per applicazioni semplici, le servlet sono semplici
- Il sistema "scala": le Servlet supportano anche caratteristiche più complesse
  - Single threading / multi threading
  - URL rewriting come sostituto per i cookie
  - Inoltro delle richieste ad altre servlet o ad altri script CGI

## Caratteristiche delle Servlet

- Velocità
  - Molto veloci, il codice è compilato, e una volta caricato viene tenuto in memoria dal server – bassa latenza e alta banda
- Portabilità
  - Ottima – Java è sempre lo stesso ovunque
- Fattori temporali
  - Provare una Servlet richiede (i) modificare il codice (ii) compilare il codice (iii) riconfigurare il web server: non è adatto alla prototipazione rapida

## Caratteristiche delle Servlet

- Competenze
  - Con poche aggiunte, riutilizzate quello che sapete su Java
- Supporto
  - Il supporto da parte di terzi è limitato (ma in crescita: si aspetta il nuovo Apache)
  - La tecnologia è portabile, ma al momento dipende molto da Sun/IBM/Borland

Java

*progetto applicazioni Web -- servlets*

fine