



CORSO DI LAUREA IN INFORMATICA

TECNOLOGIE SOFTWARE PER IL WEB

SERVLET – II PARTE

a.a 2022/23

Pagine di errore

```
<error-page>
  <error-code>404</error-code>
  <location>/commons/redirectToError.jsp</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/commons/fatalError.jsp</location>
</error-page>
```

- Si possono gestire anche le eccezioni:

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

- Dalle Servlet 3.0 in poi:

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

Gestione dello stato (di sessione)

- **HTTP è un protocollo stateless:** non fornisce in modo nativo meccanismi per il mantenimento dello stato tra diverse richieste provenienti dallo stesso client
- Le applicazioni Web hanno spesso bisogno di stato. Sono state definite due tecniche per mantenere traccia delle informazioni di stato:
 1. **uso dei cookie: meccanismo di basso livello**
 2. **uso della sessione (session tracking): meccanismo di alto livello**
- La sessione rappresenta un'utile astrazione ed essa stessa può far ricorso a tre meccanismi base di implementazione:
 1. Cookie
 2. URL rewriting
 3. Hidden form

Session Tracking and E-Commerce

- Perché il monitoraggio della sessione?
- Quando un cliente di un negozio on-line (fra i tanti) aggiunge un articolo al proprio carrello, come fa il server a sapere cosa c'è già nel carrello di quel cliente?
- Quando un cliente decide di acquistare i prodotti scelti, come fa il server a determinare in quale carrello sono i suoi prodotti?



Il referto medico

- John va dall'oculista e dopo la visita, riceve una prescrizione.
- Il medico gli chiede quindi di riportargliela la prossima volta che ritorna per il controllo. Perché?



La prescrizione medica

- John va dall'oculista e dopo la visita, riceve una prescrizione.
- Il medico gli chiede quindi di riportargliela tutte le volte che ritornerà per un controllo. Perché?

Probabilmente il medico ha molti clienti e non può ricordare a chi ha fatto quale prescrizione e per fare una nuova visita vuole conoscere la storia del paziente.

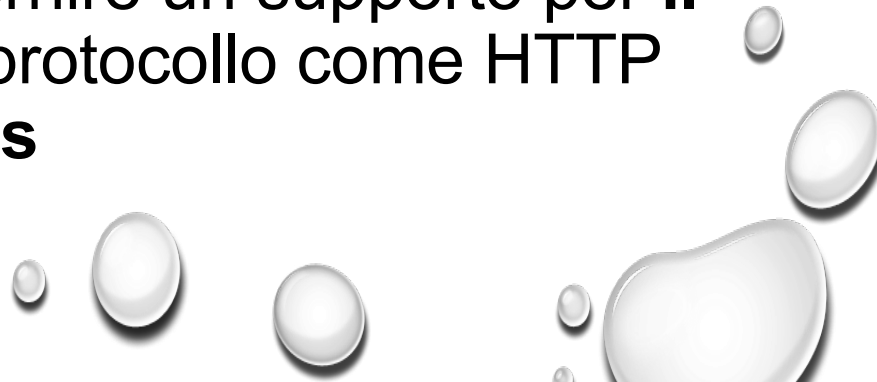
John -> Client

Oculista -> Server

Prescrizione -> cookie




I COOKIE

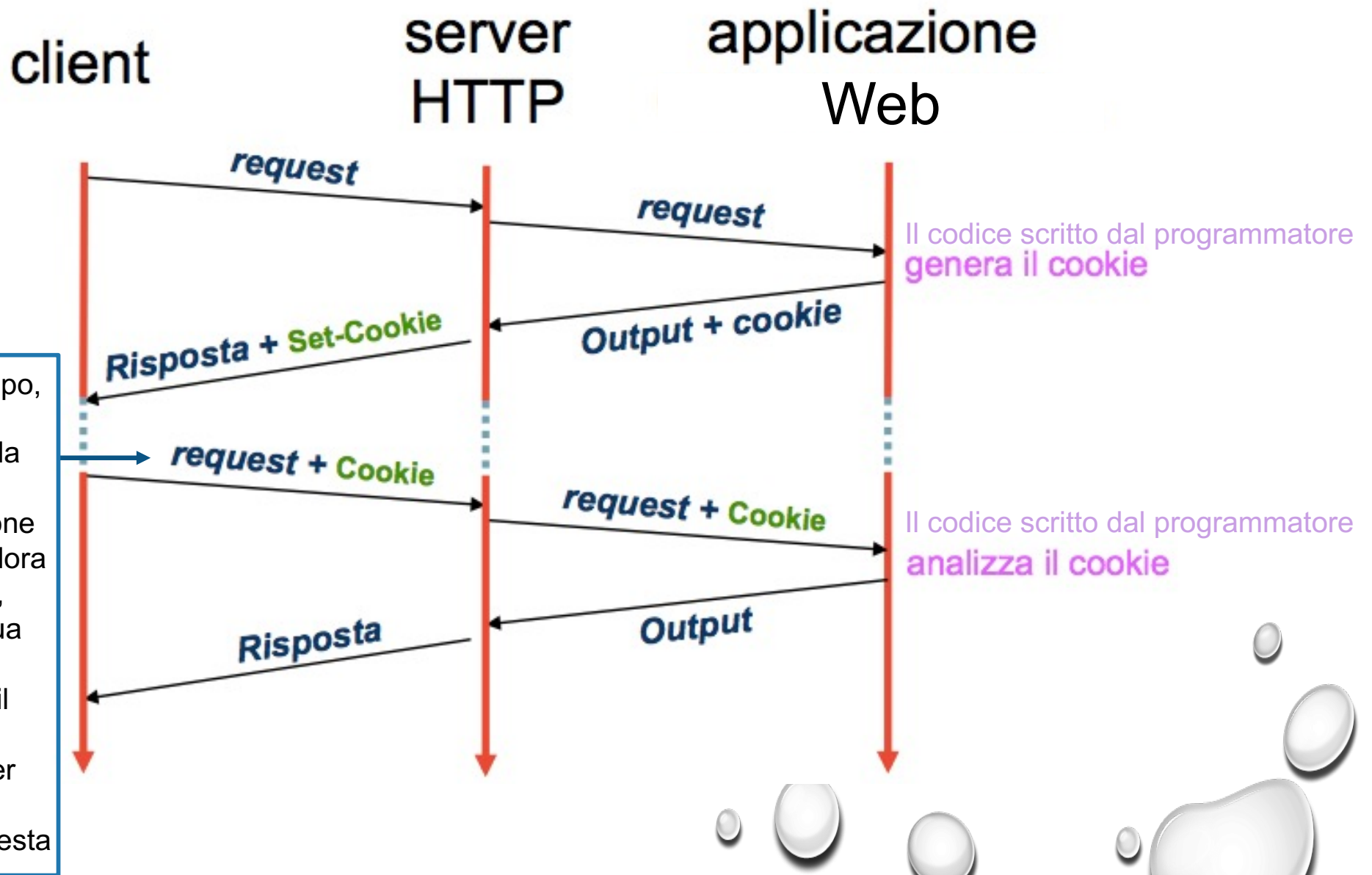
- Parallelamente alle sequenze request/response, il protocollo prevede una struttura dati che si muove come un token, dal client al server e viceversa: **il cookie**
 - I cookie sono di solito **generati** dall'applicazione lato **server** (dal programmatore web) e **memorizzati** e **restituiti** dal **client** (il browser)
 - Dopo la loro creazione, vengono trasmessi ad ogni scambio di request e response
 - Hanno come scopo quello di fornire un supporto per **il mantenimento di stato** in un protocollo come HTTP che è essenzialmente **stateless**
- 



HEADER DEI cookie

- I cookies devono potere essere memorizzati sia nella **risposta**, che nelle **richieste** successive:
 - **Set-Cookie:** header nel messaggio di **HTTP response**. Il client legge il cookie dalla risposta, lo memorizza e può rispeditirlo alla prossima request
 - **Cookie:** header nel messaggio di **HTTP request**. Il client decide se speditirlo sulla base del nome del documento, dell'indirizzo IP del server, e dell'età del cookie
- 

Interazione con i cookie



La classe cookie

<https://jakarta.ee/specifications/servlet/6.0/apidocs/jakarta.servlet/jakarta/servlet/http/cookie>

- Un cookie contiene un certo numero di informazioni, tra cui:
 - una coppia **nome/valore**
 - **Caratteri non utilizzabili:** [] () = , " / ? @ : ;
 - il **dominio Internet** dell'applicazione che ne fa uso
 - **path** dell'applicazione fino alla risorsa
 - una **expiration date** espressa in secondi (60*60*24 indica 24 ore)
 - un valore booleano per definirne il **livello di sicurezza**
- La classe Cookie modella il cookie HTTP
 - L'applicazione web recupera i cookie dalla **request** utilizzando il metodo **getCookies()**
 - L'applicazione web aggiunge i 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());  
    }  
}
```

Cancellazione di un cookie

- Per eliminare un cookie è sufficiente seguire i seguenti tre passi:
 1. Leggere un cookie già esistente e memorizzarlo nell'oggetto Cookie
 2. Impostare l'età del cookie a zero utilizzando il metodo `setMaxAge()`
 3. Aggiungere di nuovo questo cookie nell'intestazione della risposta

- Es:

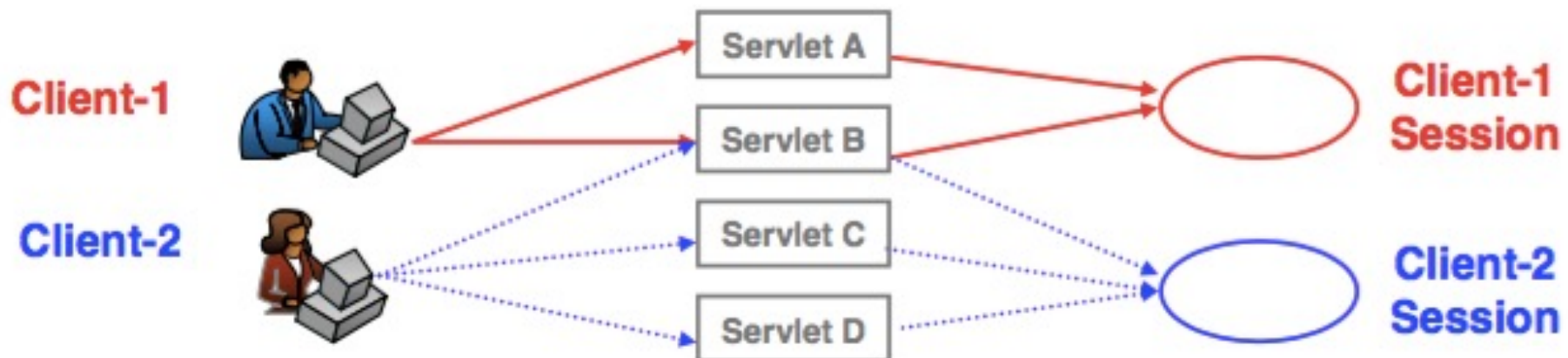
```
Cookie[] cookies = null;  
cookies = request.getCookies();  
Cookie cookie = cookies[i] //i-esimo Cookie  
cookie.setMaxAge(0);  
response.addCookie(cookie);
```

NOTA:

`setMaxAge(-1)` istruisce il browser a non memorizzare in modo persistente il cookie e di cancellarlo alla prima chiusura del browser.

Uso della sessione

- 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)*
- Ogni sessione consiste di un **oggetto** contenente 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

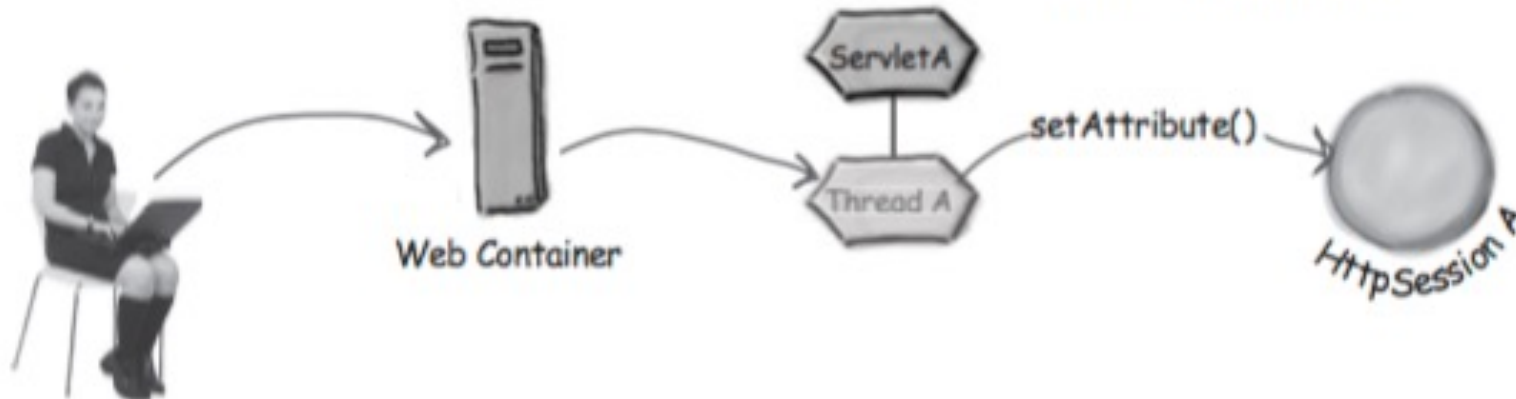


①

Diane selects "Dark" and hits the submit button.

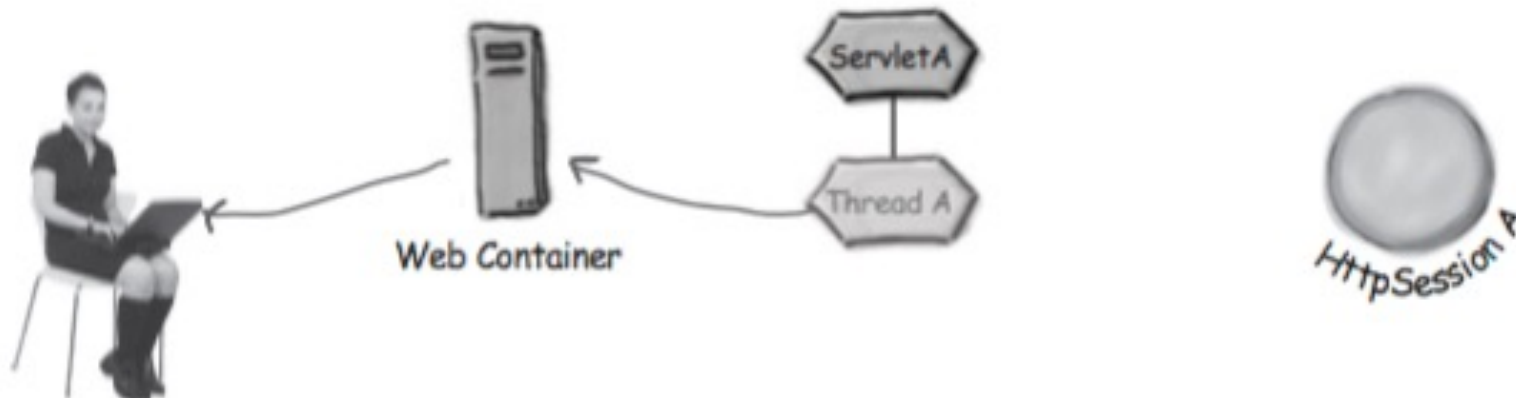
The Container sends the request to a new thread of the BeerApp servlet.

The BeerApp thread finds the session associated with Diane, and stores her choice ("Dark") in the session as an attribute.



②

The servlet runs its business logic (including calls to the model) and returns a response... in this case another question, "What price range?"



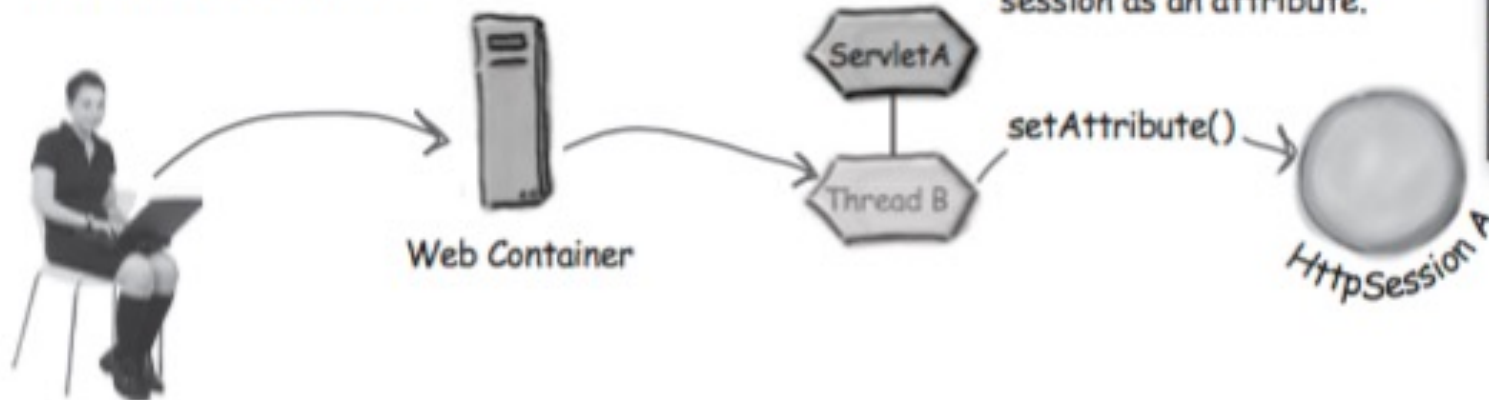
3

Diane considers the new question on the page, selects "Expensive" and hits the submit button.

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

The BeerApp thread finds the session associated with Diane, and stores her new choice ("Expensive") in the session as an attribute.

Same client
Same servlet
Different request
Different thread
Same session



4

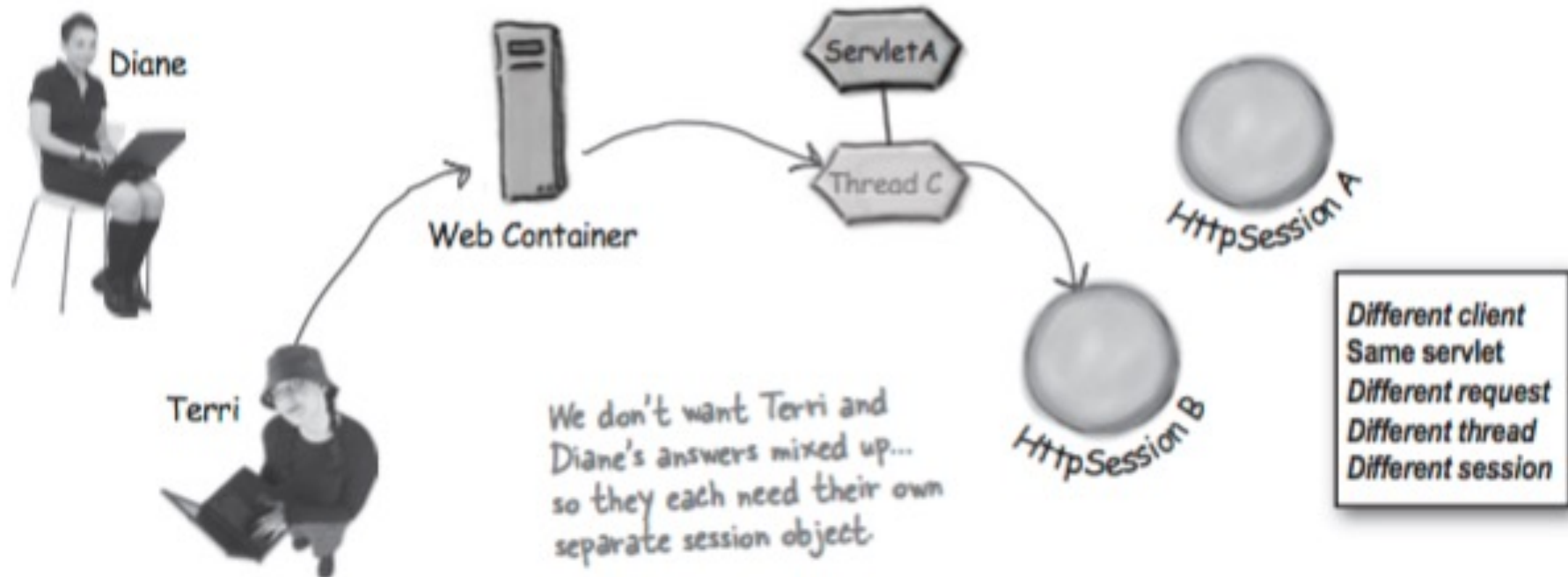
The servlet runs its business logic (including calls to the model) and returns a response... in this case another question.



- ⑤ Diane's session is still active, but meanwhile Terri selects "Pale" and hits the submit button.

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

The BeerApp thread starts a new Session for Terri, and calls `setAttribute()` to store her choice ("Pale").



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 (default)**: ritorna la sessione esistente o, se non esiste, ne crea una nuova
 - **false**: ritorna, se possibile, la sessione esistente, altrimenti ritorna null
- Uso del metodo in una Servlet:

HttpSession session = request.getSession(true);

- Per recuperare l'ID della sessione:

Perché getSession viene applicato sulla richiesta e non sulla risposta?

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

NOTA: una sessione «esiste» se è attivo il suo sessionId

Gestione del contenuto di una sessione

- Si possono memorizzare **dati specifici dell'utente negli attributi della sessione** (coppie nome/valore)
- Consentono di memorizzare e recuperare oggetti

```
Cart sc = (Cart) session.getAttribute("shoppingCart");  
sc.addItem(item);  
...  
session.removeAttribute("shoppingCart");  
...  
session.setAttribute("shoppingCart", new Cart());  
...  
Enumeration e = session.getAttributeNames();  
while(e.hasMoreElements())  
    out.println("Key; " + (String)e.nextElement());
```

Altre operazioni con le sessioni

- **String getId()** restituisce l'ID di una sessione
- **boolean isNew()** dice se la sessione è nuova
- **void invalidate()** permette di invalidare (*distruggere*) una sessione
- **long getCreationTime()** dice da quanto tempo è attiva la sessione (in millisecondi)
- **long getLastAccessedTime ()** 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());
```

Session tracking basics

...

```
HttpSession session = request.getSession();
synchronized(session) {
    SomeClass value = (SomeClass) session.getAttribute("someID");
    if (value == null) {
        value = new SomeClass();
    }
    doSomethingWith(value);
    session.setAttribute("someID", value);
}
```

To Synchronize or not to Synchronize?

- Non ci sono **race conditions** sulle sessioni fra più utenti diversi quando questi accedono contemporaneamente alla stessa applicazione (perchè?)
- Sembra praticamente impossibile che lo stesso utente acceda concorrentemente alla sua sessione
- L'uso di Ajax rende necessaria la sincronizzazione
 - Con le chiamate Ajax, è abbastanza probabile che due richieste dallo stesso utente possano arrivare concorrentemente

Accumulating a list of user data

...

```
HttpSession session = request.getSession();
synchronized (session) {
    @SuppressWarnings("unchecked")
    List<String> previousItems = (List<String>) session.getAttribute("previousItems");
    if (previousItems == null) {
        previousItems = new ArrayList<String>();
    }
    String newItem = request.getParameter("newItem");
    if (newItem != null) {
        previousItems.add(newItem);
    }
    session.setAttribute("previousItems", previousItems);
}
```

...


```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
    throws ServletException, IOException {
```

```
    response.setContentType("text/html");
```

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

```
    synchronized (session) {
```

```
        String heading;
```

```
        Integer accessCount = (Integer) session.getAttribute("accessCount");
```

```
        if (accessCount == null) {
```

```
            accessCount = 0;
```

```
            heading = "Welcome, Newcomer";
```

```
        } else {
```

```
            heading = "Welcome Back";
```

```
            accessCount = accessCount + 1;
```

```
        }
```

```
        session.setAttribute("accessCount", accessCount);
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println
```

```
        ("<!DOCTYPE html>" +
```

```
        "<html>" +
```

```
        "<head><title>Session Tracking Example</title></head>" +
```

```
        "<body>" +
```

```
        "<h1>" + heading + "</h1>" +
```

```
        "<h2>Information on Your Session:</h2>" +
```

```
        "<table border='1'>" +
```

```
        "<tr>" +
```

```
        "  <th>Info Type</th><th>Value</th>" +
```

```
        "</tr><tr>" +
```

```
        "  <td>ID</td><td>" + session.getId() + "</td>" +
```

```
        "</tr><tr>" +
```

```
        "  <td>Creation Time</td><td>" + new Date(session.getCreationTime()) + "</td>" +
```

```
        "</tr><tr>" +
```

```
        "  <td>Time of Last Access</td><td>" + new Date(session.getLastAccessedTime()) + "</td>" +
```

```
        "</tr><tr>" +
```

```
        "  <td>Number of Previous Accesses</td><td>" + accessCount + "</td>" +
```

```
        "</table>" +
```

```
        "</body></html>");
```

```
    }
```

```
}
```

Si analizzi questo codice due volte:

1. Si supponga che si è alla prima chiamata – la sessione non esiste, (verrà creata ed il suo sessionId è inviato al browser)
2. Si supponga che si è ad una chiamata successiva – la sessione esiste (infatti il browser avrà inviato il sessionId al server)

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**
 - Usato per garantire una gestione corretta della sessione
 - Se il server sta usando i cookie, ritorna l'URL non modificato
 - Se il server sta usando l'URL rewriting, prende in input un URL, e se l'utente ha i cookie disattivati, codifica l'id di sessione nell'URL
- Il metodo encodeURL() dovrebbe essere usato per:
 - hyperlink ()
 - form (<form action="...">)
- Es:

```
String url = "order-page.html";  
url = response.encodeURL(url);
```


URL-Rewriting

- Idea
 - Client appends some extra data on the end of each URL that identifies the session
 - Server associates that identifier with data it has stored about that session
 - Es: **http://host/path/file.html;jsessionid=1234**
- Advantage
 - *Works even if cookies are disabled or unsupported*
- Disadvantages
 - Must encode all URLs that refer to your own site
 - All pages must be dynamically generated
 - Fails for bookmarks and links from other sites

Hidden form fields

- Idea:

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

- Advantage

- *Works even if cookies are disabled or unsupported*

- Disadvantages

- Lots of tedious processing
- **All pages must be the result of form submissions**