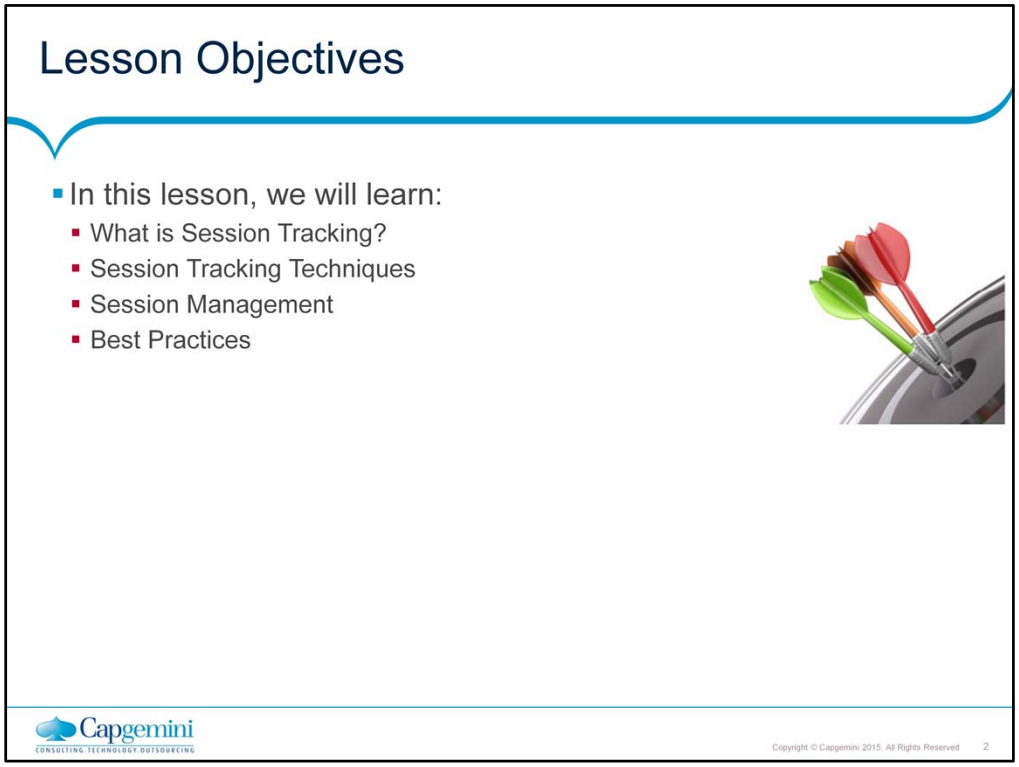
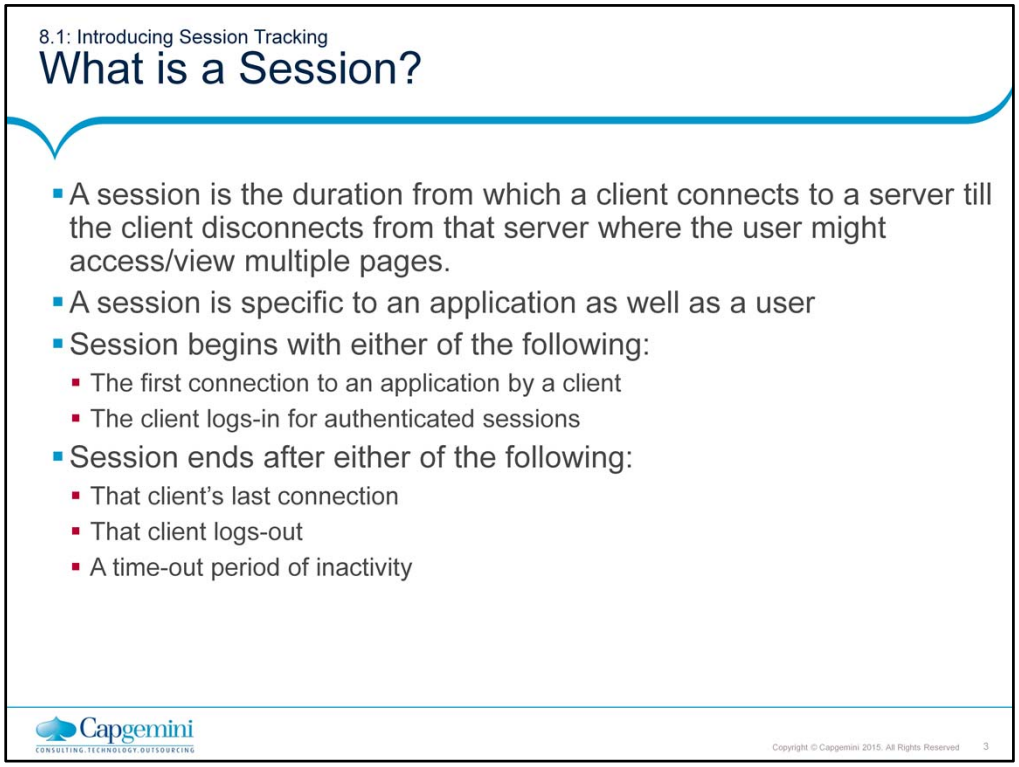
Java Servlets 3.0

Lesson 8: Session Tracking

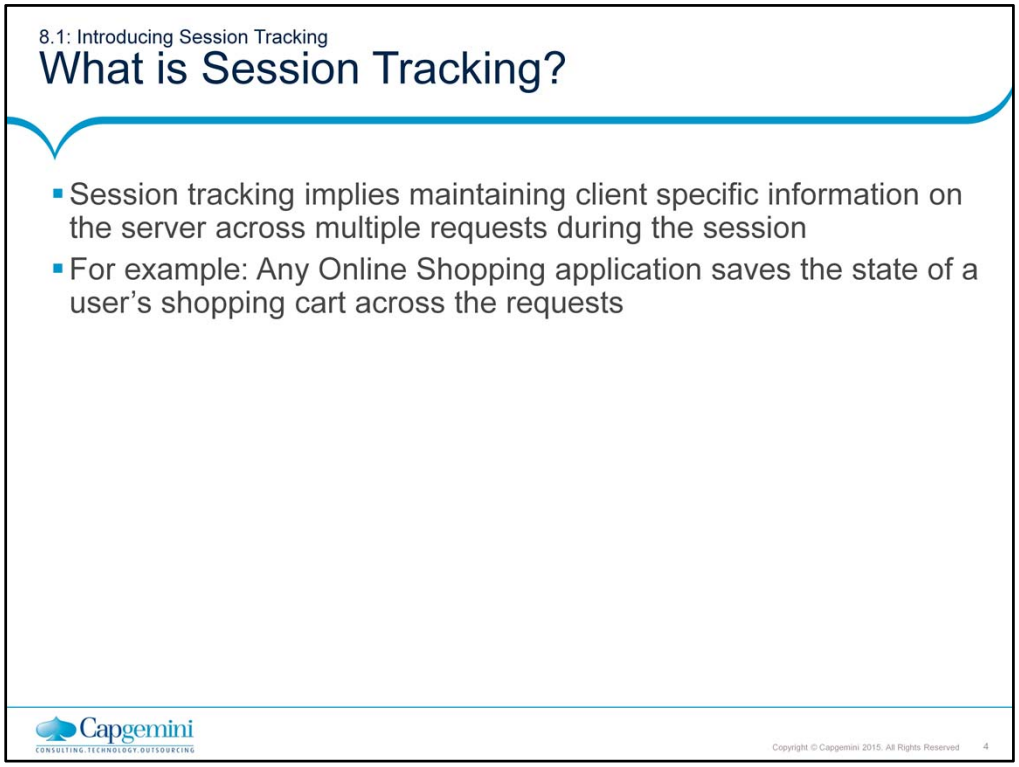


Lesson Objectives:
This lesson introduces Session Management. The lesson contents are:
Lesson 08: Session Management

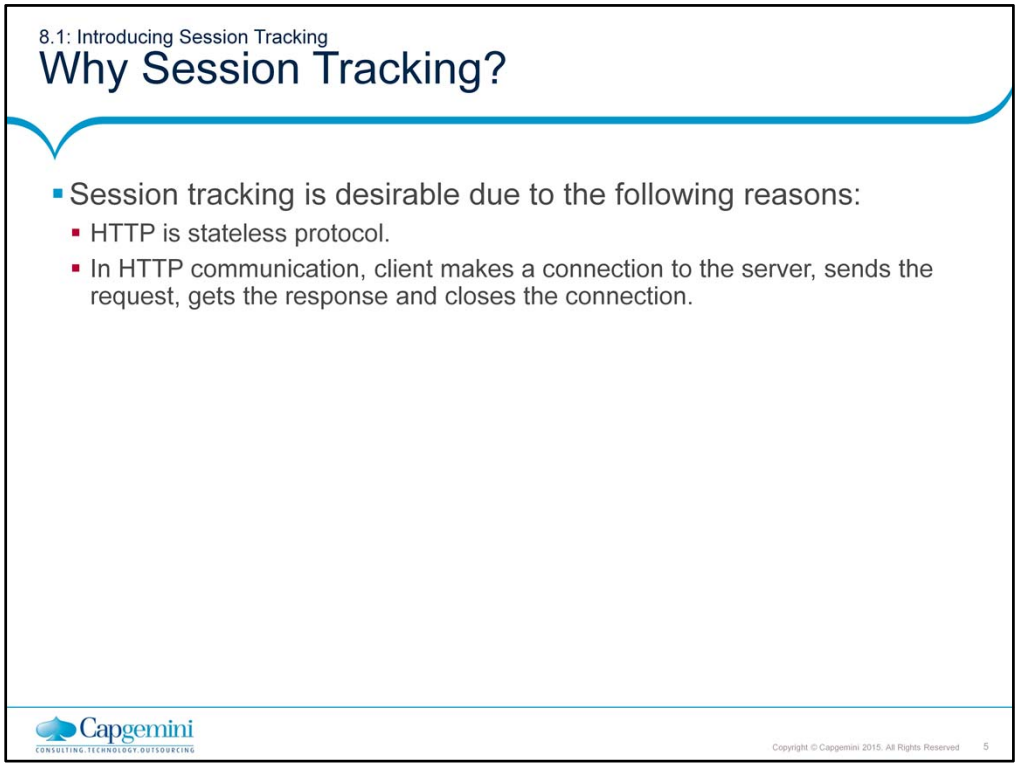
- 8.1: Introducing Session Tracking
- 8.2: Ways of Session Tracking in JEE
- 8.3: Session Management
- 8.4: Best Practices



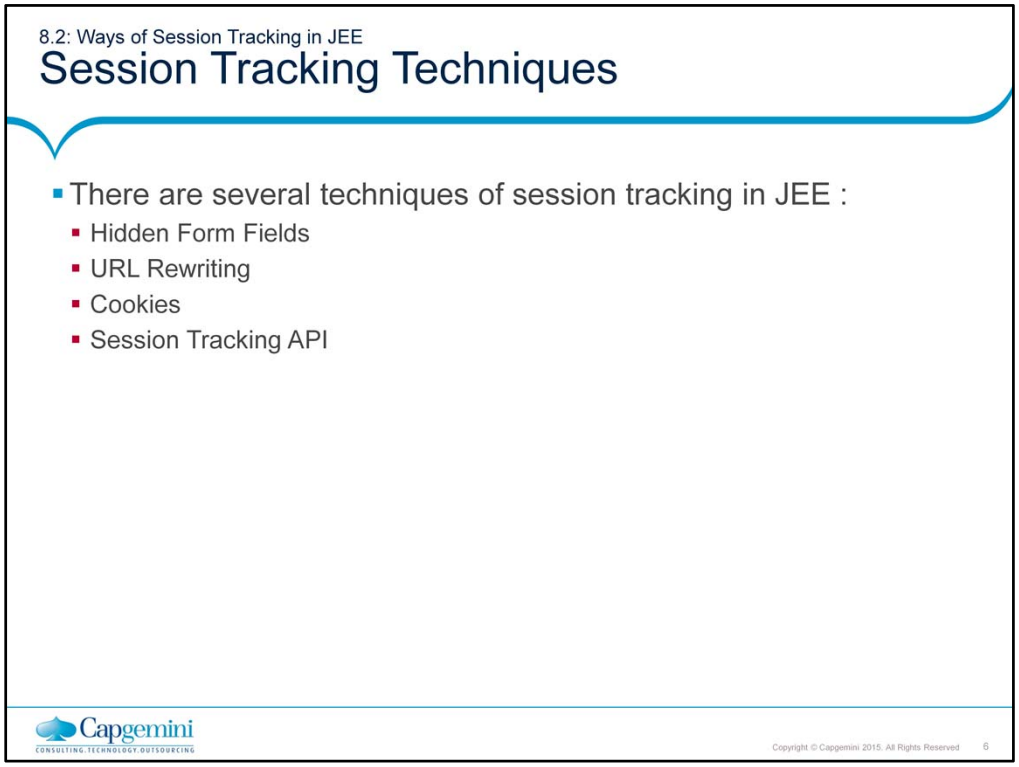
Introducing Session Tracking:
A session refers to all the connections that a single client might make to a server in the course of viewing any pages associated with a given application.
Sessions are specific to both the individual user and the application. Therefore every user of an application has a separate session. The user has access to a separate set of session variables.
Logically a session begins with the first connection to an application by a client and ends after that client's last connection.
Session can be an authenticated session also where a user has to log-in. In this case, sessions begin when the user logs in and ends when the user log-out.
Session also ends after a time-out period of inactivity.



Introducing Session Tracking:
What is Session Tracking?
In Session tracking, client first makes a request for any servlet or any page, the container receives the request.
The container then generates a unique identification, called Session ID, for that client and gives it back to the client along with the response. This ID gets stores on the client machine. Thereafter when the client sends a request again to the server, it also sends the session Id with the request. The container sees the Id and sends back the response. While seeing the ID the container recognizes that it is the same previous client making a request.
Thus container identifies the client.



Introducing Session Tracking:
Why Session Tracking?
HTTP is a stateless protocol, that is it cannot persist the information. HTTP always treats each request as a new request. Each request made by a Web browser (client), for an image, an HTML page, or other Web object, is made via a new connection. In other words, in HTTP, the client makes a connection to the server, sends the request, gets the response, and closes the connection. Hence to maintain the state across pages we need some sort of session tracking mechanism.



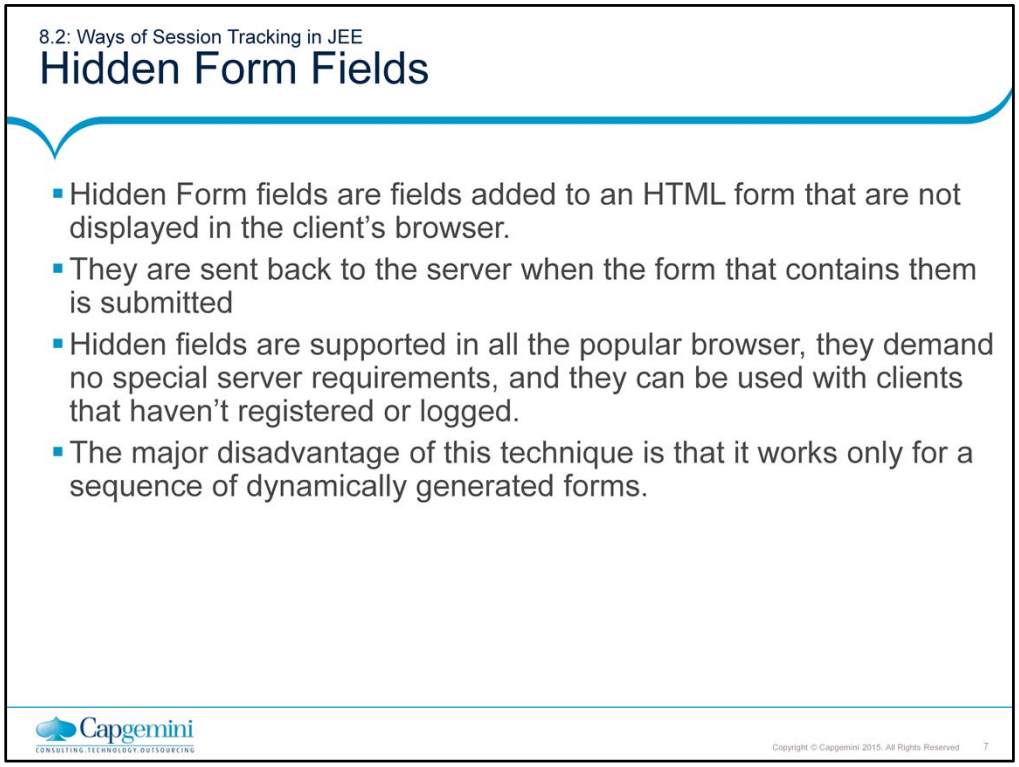
Ways of Session Tracking in JEE:
Session Tracking Techniques:
From the last slide we understand that since HTTP is stateless protocol it becomes developer's job to work around saving client specific information on server in a web application.
There are several ways in which session tracking can be done in JEE.

Hidden Form Fields: It is the simplest way of saving user-specific information where hidden parameters are encoded inside an HTML form, such as the username and type of transaction being made

URL Rewriting: URL rewriting involves placing a session id in the URL.

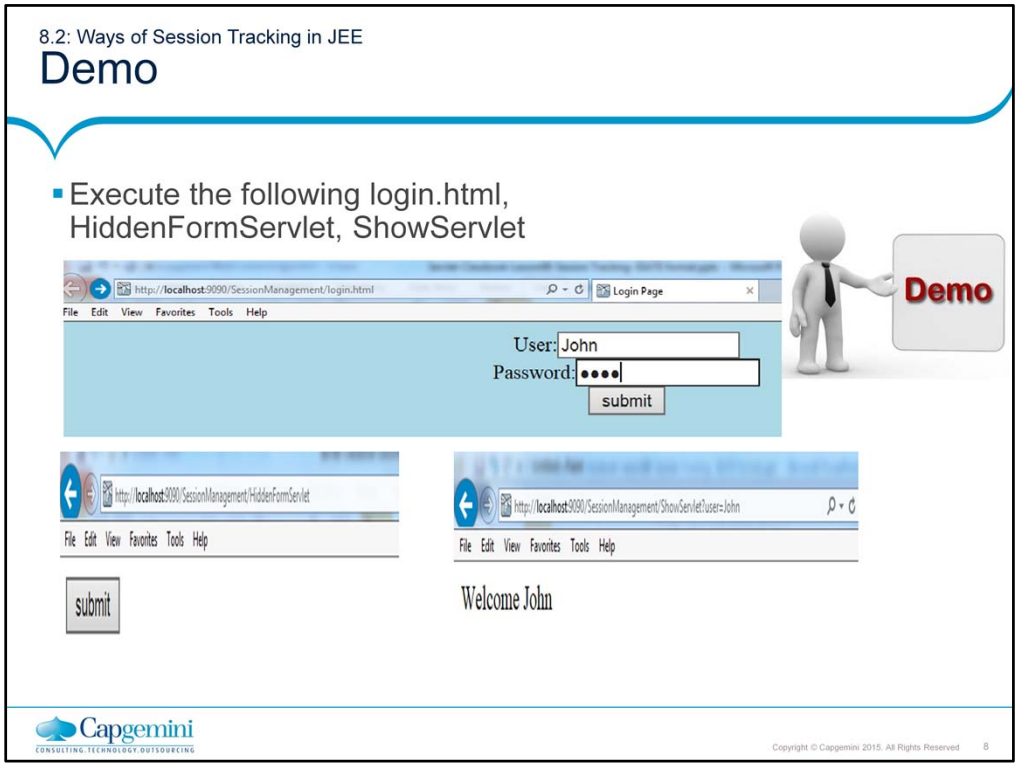
Cookies: It is a small text file storing client information, cookies are created by the server and saved on client's machine.

Session Tracking API: Servlets API provides Session Tracking API



Ways of Session Tracking in JEE:
Hidden Form Fields:
In this technique the fields are added to an HTML form which are not displayed in the client's request. The hidden form fields are sent back to the server when the form is submitted. In hidden form fields, the HTML entry will be as shown below:

This means that when form is submitted ,the specified name and value will be get included in get or post method.



Execute the following servlets:
http://localhost:9090/SessionManagement/login.html

This would lead to servlet:

http://localhost:9090/SessionManagement/HiddenFormServlet

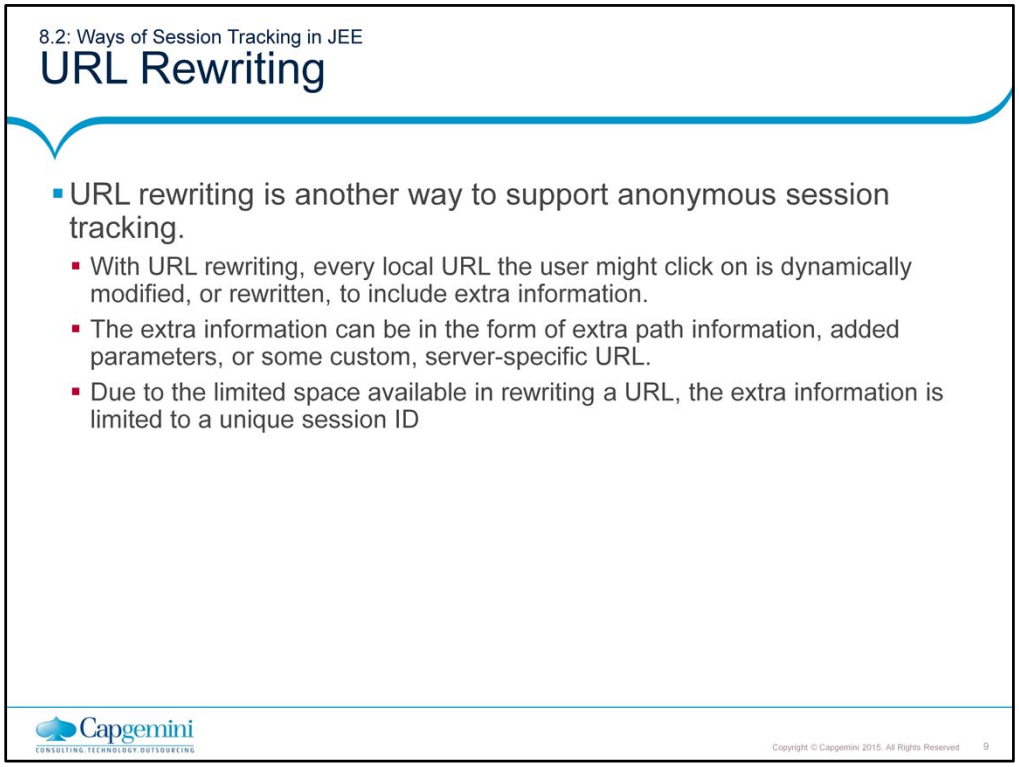
In HiddenFormServlet, username is added as Hidden Field in html form as shown below in partial listing.

```
out.println("<form action='ShowServlet'>");
out.println("<input type='hidden' name='user' value='" + user + "'>");
out.println("<input type='submit' value='submit' >");
```

Then username is retrieved in ShowServlet.

http://localhost:9090/SessionManagement/ShowServlet?user=John.

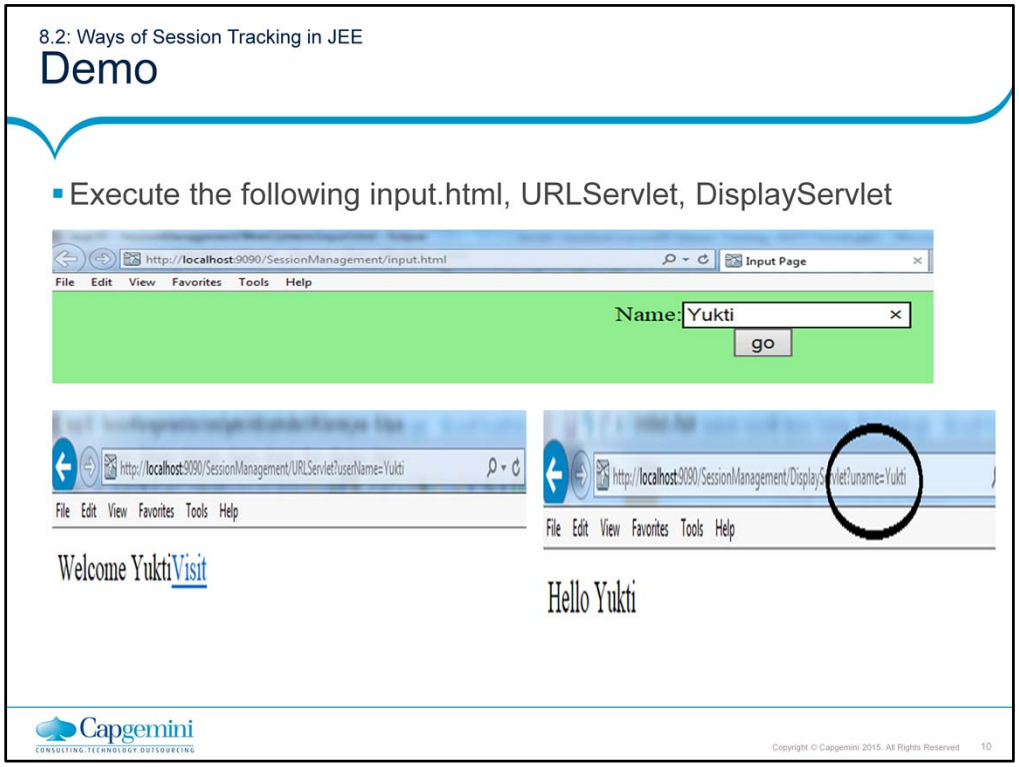
Query parameters get appended due to the GET method.



Ways of Session Tracking in JEE:
URL Rewriting:
The explanation for session Id would be covered in the HttpSession section. URL rewriting is another way to support anonymous session tracking. With URL rewriting, every local URL the user might click on is dynamically modified, or rewritten, to include extra information. The extra information can be in the form of extra path information, added parameters, or some custom, server-specific URL. Due to the limited space available in rewriting a URL, the extra information is limited to a unique session ID. For example, the following URLs have been rewritten to pass session id 123:

Original URL :
http://server:port/servlet/Rewritten
Extra path information :
http://server:port/servlet/Rewritten/123
Added parameter :
http://server:port/servlet/Rewritten?sessionid=123
Added parameter change :
http://server:port/servlet/Rewritten;$sessionid$123

Each rewriting technique has its advantages and disadvantages. Using extra path information works on all servers, and it works as a target for forms that use both the GET and POST methods. It doesn't work well if a servlet has to use the extra path information as true path information.
An added parameter works on all servers too, but fails as a target for forms those uses the POST method, and it causes parameter-naming collisions.
Using a custom, server-specific change works under all conditions for servers that support the change. Unfortunately, it doesn't work at all servers that don't support the change.



Execute the following servlets:
http://localhost:9090/SessionManagement/input.html

This would lead to servlet:

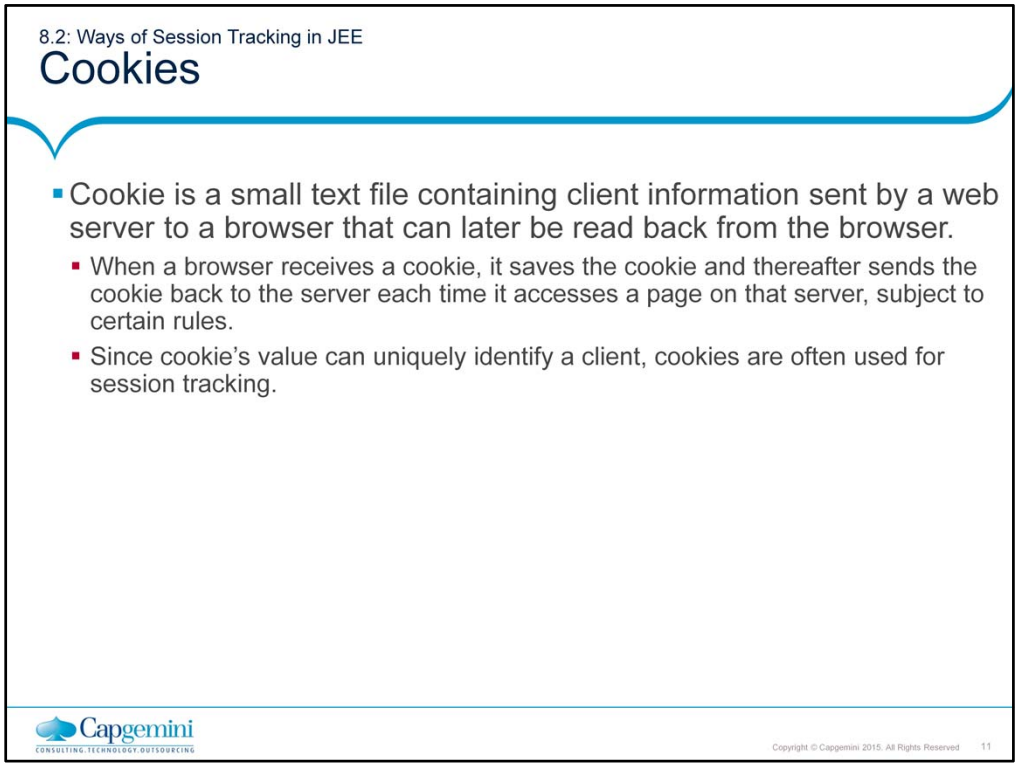
http://localhost:9090/SessionManagement/URLServlet

In URLServlet, username is appended as query string as shown below in partial listing.

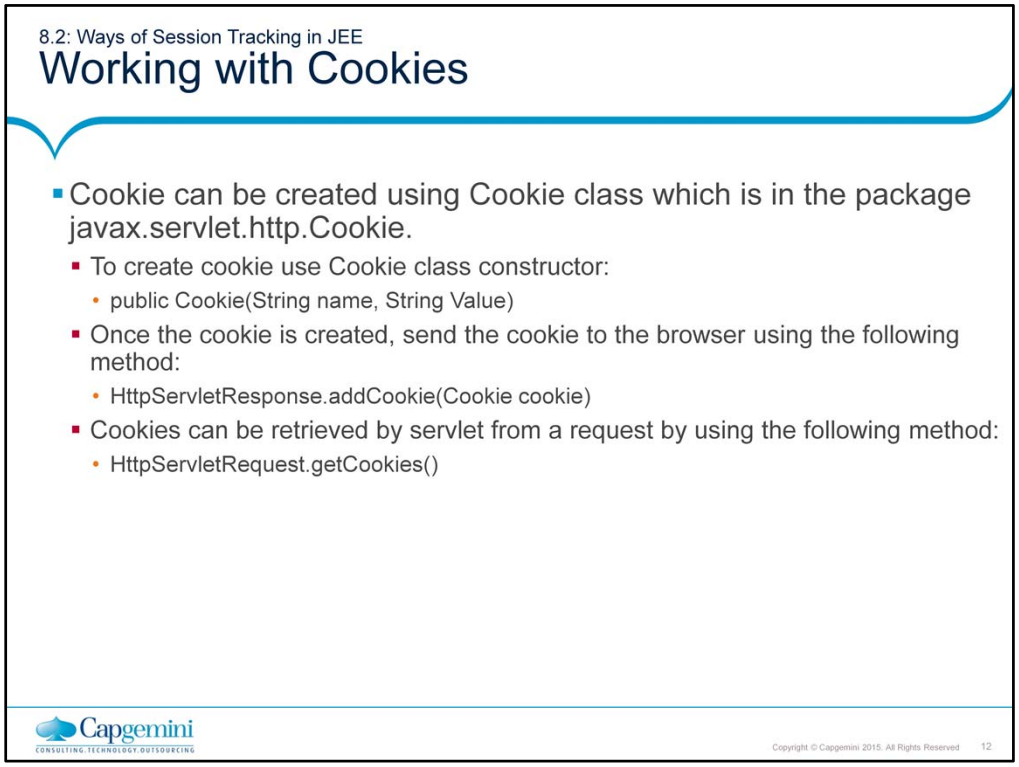
out.print("visit");

Then username is retrieved in DisplayServlet.

http://localhost:9090/SessionManagement/DisplayServlet?user=Yukti.



Ways of Session Tracking in JEE:
Cookies:
Cookie is a bit of information stored in a small text file sent by a web server to a browser that can later be read back from the browser. When a browser receives a cookie, it saves the cookie and there-after sends the cookie back to the server each time it accesses a page on that server, subject to certain rules. Since cookie's value can uniquely identify a client, cookies are often used for session tracking.
The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.



Ways of Session Tracking in JEE:
Working with Cookies
Create a cookie with the Cookie( ) constructor:

public Cookie(String name, String value)

This creates a new cookie with an initial name and value. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
The name of the cookie must be an HTTP/1.1 token. Tokens are strings that contain none of the special characters listed in. (Alphanumeric strings qualify as tokens.)
The value of the cookie can be any string, though null values are not guaranteed to work the same way on all browsers. In addition, if a cookie is sent that complies with Netscape's original cookie specification, do not use whitespace or any of these characters:

[ ] ( ) = , " / ? @ : ;

A servlet can send a cookie to client by passing a Cookie object to the addCookie( ) method of HttpServletResponse:

public void HttpServletResponse.addCookie(Cookie cookie)

The method adds the specified cookie to the response. Additional cookies can be added with subsequent calls to addCookie( ).

Ways of Session Tracking in JEE:

Working with Cookies:

- Since cookies are sent using HTTP headers, they should be added to response before any content is sent.
- The code to set a cookie looks like this:

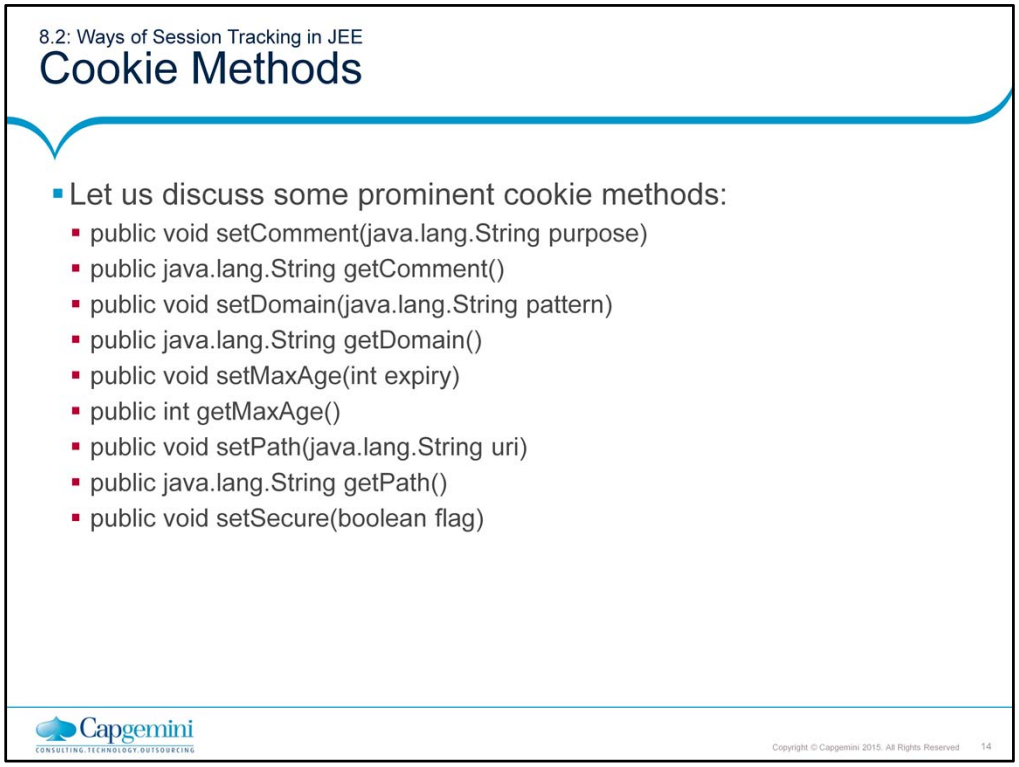
```
Cookie cookie=new Cookie(“ID” , “123” );
res.addCookie(cookie);
```

- A servlet retrieves cookies by calling the getCookies( ) method of HttpServletRequest.

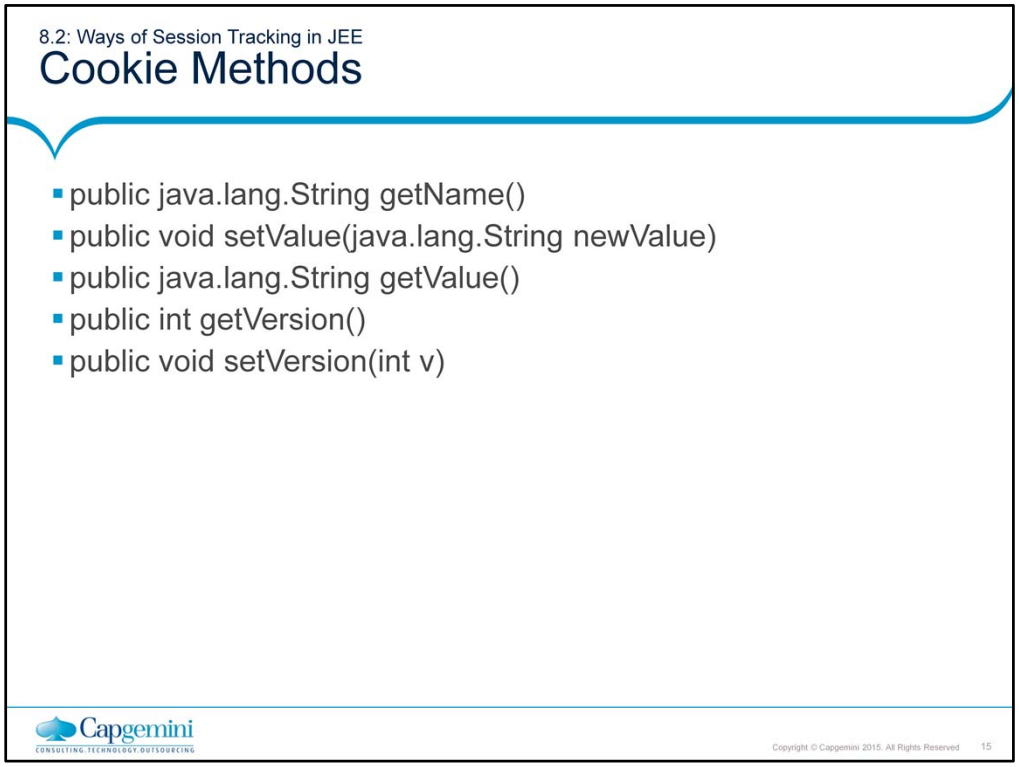
```
public Cookies[ ] HttpServletRequest.getCookies( )
```

- This method returns an array of Cookie objects that contains all the cookies sent by the browser as part of the request or null if no cookies were sent. Several cookies might have the same name but different path attributes. The code to fetch the cookies looks like this:

```
Cookie[] cookies = req.getCookies();
   if (cookies != null) {
     for (int i = 0; i < cookies.length; i++) {
       if (cookies[i].getName().equals("sessionid")) {
              String name =cookies[i].getName();
              String value =cookies[i].getValue();
       }
     }
}
```



Ways of Session Tracking in JEE:
Cookie Methods:
public void setComment(java.lang.String purpose) :
It specifies a comment that describes a cookie's purpose.
public java.lang.String getComment() :
It returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
public void setDomain(java.lang.String pattern) :
It specifies the domain within which this cookie should be presented.
public java.lang.String getDomain() :
It returns the domain name set for this cookie.
public void setMaxAge(int expiry) :
It sets the maximum age of the cookie in seconds. A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the maximum age when the cookie will expire, not the cookie's current age. A negative value means that the cookie is not stored persistently and will be deleted when the Web browser exits. A zero value causes the cookie to be deleted.
public int getMaxAge() :
It returns the maximum age of the cookie, specified in seconds. If getMaxAge returns a negative value, the cookie was not stored persistently. This method does not return a zero value, because if a cookie's age was set to zero with setMaxAge, the cookie was deleted.



Ways of Session Tracking in JEE:
Cookie Methods:
public void setPath(java.lang.String uri)

It specifies a path for the cookie, which is the set of URIs to which the client should return the cookie. The cookie is visible to all the pages in the directory being specified, and all the pages in that directory's subdirectories. A cookie's path must include the servlet that set the cookie, for example, servlet/dir1, which makes the cookie visible to all directories on the server under dir1.

public java.lang.String getPath()

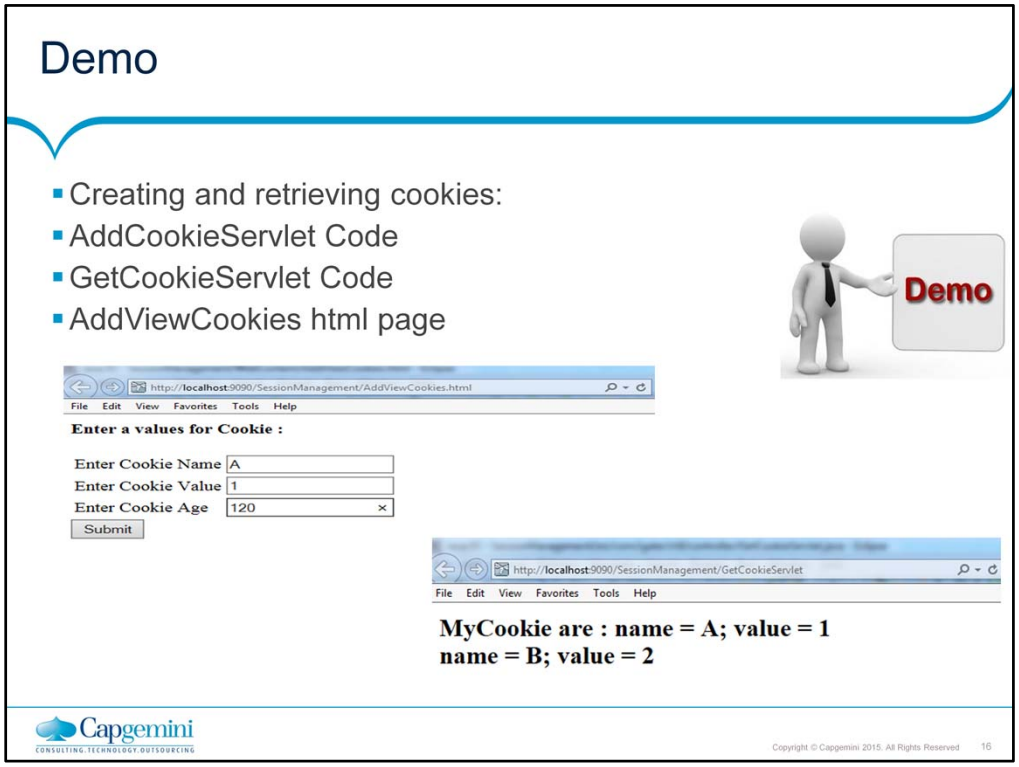
It returns the paths (that is, URIs) on the server to which the browser returns this cookie. The cookie is visible to all subdirectories within the specified path on the server.

public void setSecure(boolean flag)

It indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL. This method should only be used when the cookie's originating server used a secure protocol to set the cookie's value. The default value is false.

public boolean getSecure()

It returns true if the browser is sending cookies only over a secure protocol, or false if the browser can use a standard protocol.



Note:
Refer to the com.igate.ch8.controller.AddCookieServlet.java code, partial listing of doGet() is given here:

```
Cookie cookie =  new Cookie(name,data);
            if ((age==null) || (age.equals(""))) {
                  res.addCookie(cookie);
                  out.println("Set Cookie --- Name= " + name + " Value = " + data);}
            else {  try { int intage=Integer.parseInt(age);
 cookie.setMaxAge(intage);
 res.addCookie(cookie);
 out.println("Set Cookie --- Name= " + name + " Value = " +  data + " age= "+age );
            }  catch(Exception e) {  out.println("Exception " +e; }
             }
```

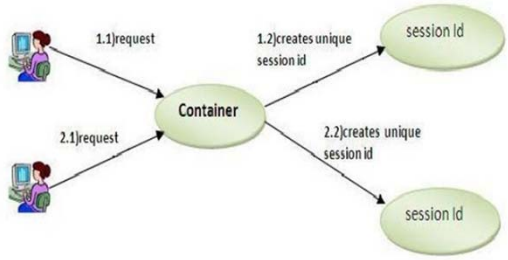
Partial code for com.igate.ch8.controller.GetCookieServlet is here :

```
      Cookie[] cookies =  req.getCookies();
      out.println("<B>MyCookie are  : ");
      for(int i=0;i < cookies.length;i++)  {
            String name = cookies[i].getName();
            String value = cookies[i].getValue();
            out.println("name =  " + name + "; value = " + value +"<BR>"); }
```

8.2: Ways of Session Tracking in JEE

Session Tracking API

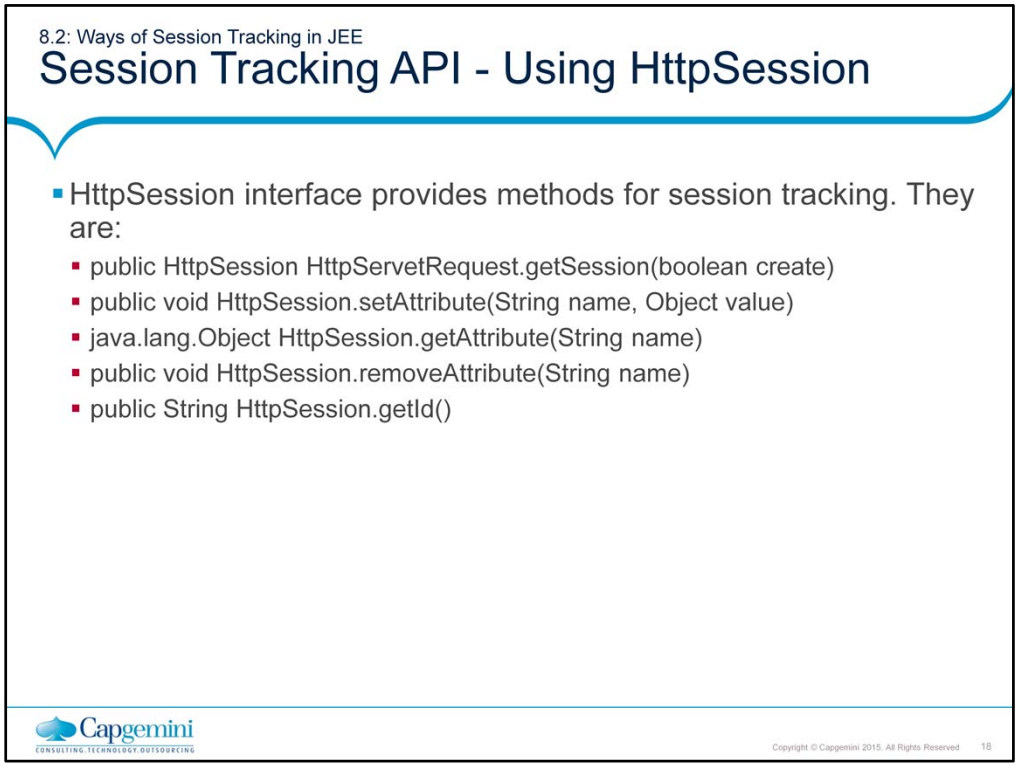
- Servlet API provides HttpSession interface in javax.servlet.http package to track and manage sessions
- HttpSession interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user
- A session usually corresponds to one user, who may visit a site many times



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Ways of Session Tracking in JEE:
Session Tracking API:
One of the ways to track session is by using Session Tracking interfaces provided by Servlet API.
The API provides HttpSession interface to track and manage sessions. The servlet container uses this interface to create a session between an HTTP client and an HTTP server creating an unique identifier called Session ID. The server can maintain a session in many ways such as using cookies or rewriting URLs. This interface allows servlets to view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.
The explanation about Session Id would be explained in the next subsequent slide.



Ways of Session Tracking in JEE:
Session Tracking API - Using HttpSession:

public HttpSession HttpServetRequest.getSession(boolean create) : retrieve the current HttpSession object. This method returns the current HttpSession object associated with the request or, if necessary, create a new session for the request. Use true to create a new session if none exists.

If create is false, and the request has no valid session then this method returns null else returns the existing session. To make sure the session is properly maintained, need to call this method at least once before writing any output to the response.

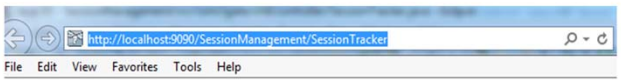
public void HttpSession.setAttribute(String name, Object value) :add information in form of an object to session object. This method binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced. The putValue() method is sometimes used instead of setAttribute() however it is deprecated now.

public Object HttpSession.getAttribute(String name) : retrieve information object stored by setAttribute() method in form of an object from the session. Returns null if no object of that name exists.

public java.lang.String getId() : This method returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet engine and is implementation dependent.

Demo

- Session tracking to count the number of times a client has accessed the site:
  - SessionTracker.java servlet



Session Tracking Demo

You've visited this page 6 times.

Here is your session data:

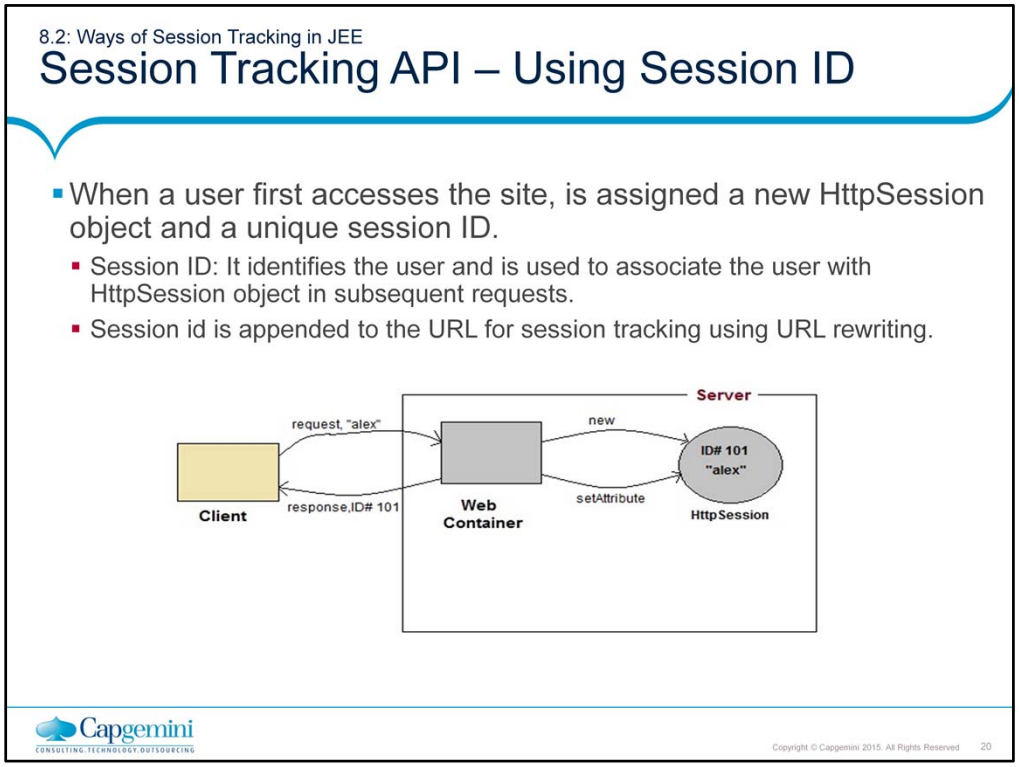
tracker.count: 6

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

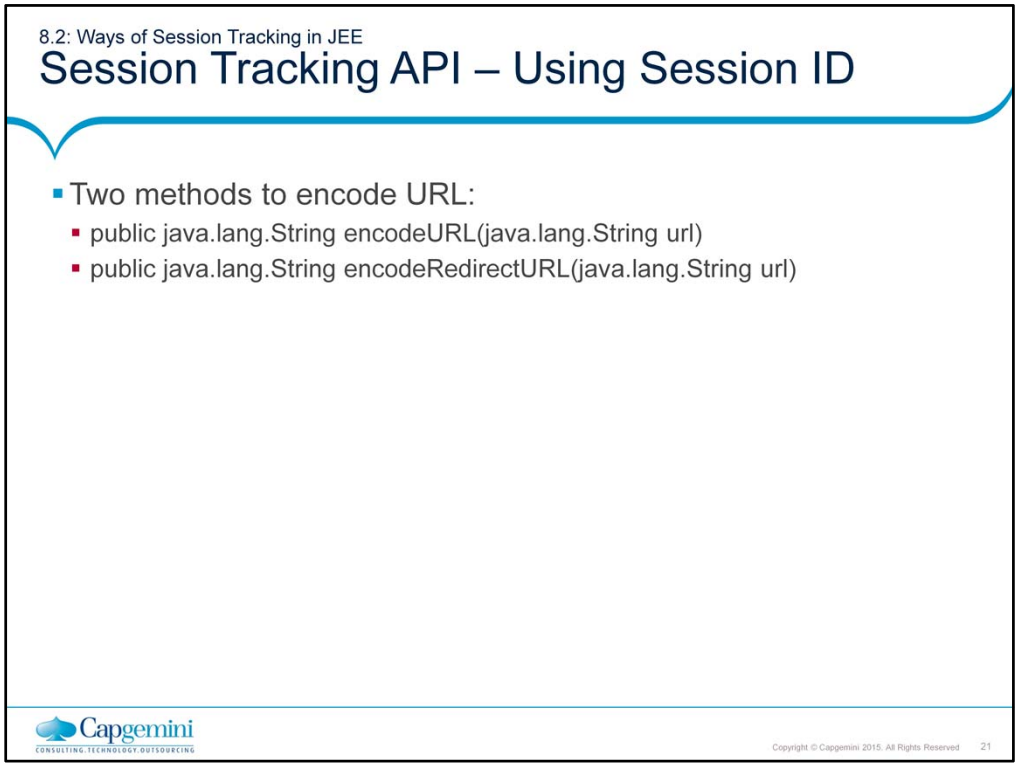
 19

Note:
Refer the com.igate.ch8.controller.SessionTracker.java code; partial listing of doGet() method is given:

```
// Get the current session object, create one if necessary
    HttpSession session = req.getSession(true);
     // Increment the hit count for this page.  The value is saved
    // in this client's session under the name "tracker.count".
Integer count = (Integer)session.getAttribute("tracker.count");
    if (count == null) {
      count = new Integer(1); }
    else {
      count = new Integer(count.intValue() + 1); }
     session.setAttribute("tracker.count", count);
     out.println("<HTML><HEAD><TITLE>SessionTracker</TITLE></HEAD>");
    out.println("<BODY><H1>Session Tracking Demo</H1>");
    // Display the hit count for this page
    out.println("You've visited this page " + count +
      ((count.intValue() == 1) ? " time." : " times."));
     out.println("<P>");
     out.println("<H2>Here is your session data:</H2>");
    String[] names = session.getAttributeNames();
    for (int i = 0; i < names.length; i++) {
      out.println(names[i] + ": " + session.getAttribute(names[i]) + "<BR>"); }
    out.println("</BODY></HTML>");
```

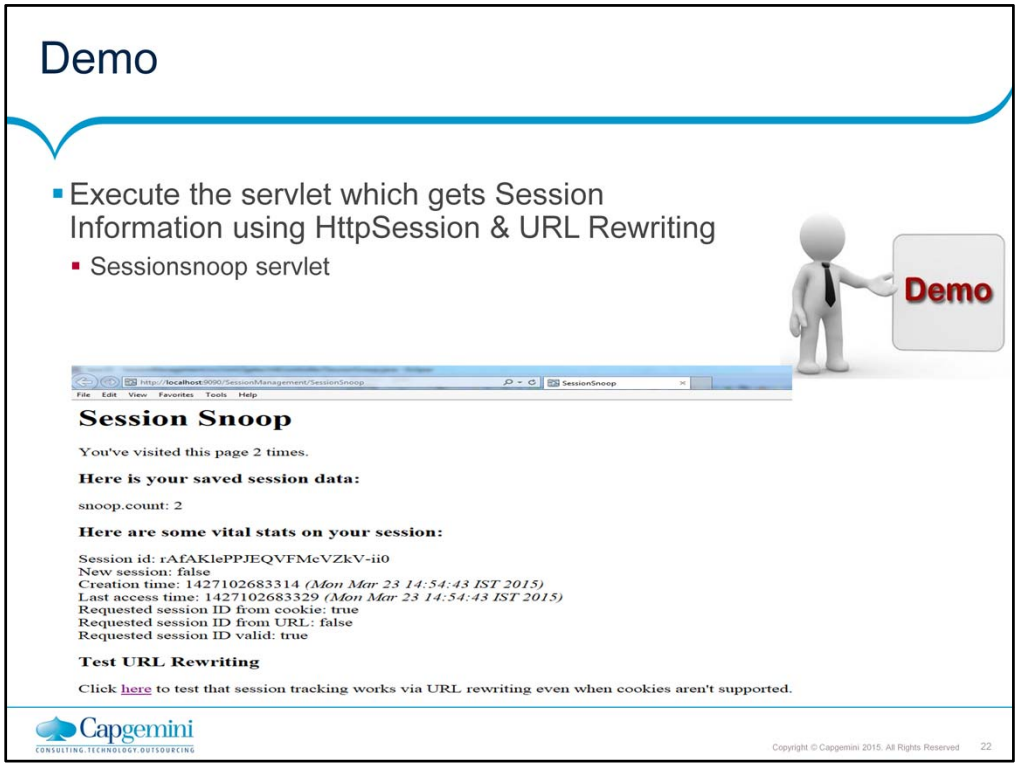


Ways of Session Tracking in JEE:
Session Tracking API – Using Session ID:
When a user first accesses the site, that user is assigned a new HttpSession object and a unique session ID. The Session ID identifies the user and is used to the user with HttpSession object in subsequent requests. Behind the scenes, the session ID is usually saved on the client in a cookie or sent as part of a rewritten URL.
A server can discover session's ID with the getID() method seen earlier. It returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet engine and is implementation dependent. This method throws an IllegalStateException if session is invalid.



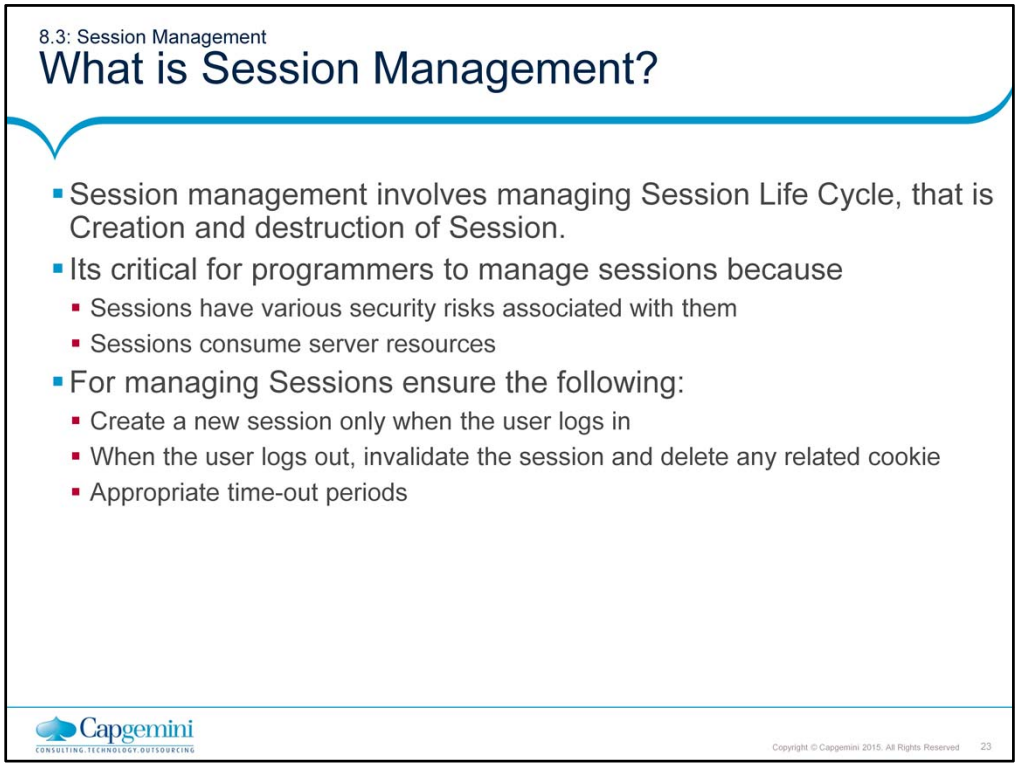
Session Tracking API:

- Session Id is stored as part of URL to support session tracking via URL rewriting. Servlet rewrites every local URL before sending it to the client. The Servlet API's HttpServletRequest interface provides two methods to perform this encoding:
  public java.lang.String **encodeURL(java.lang.String url)**
- It encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged. The implementation of this method should include the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary.
- All URLs emitted by a Servlet should be run through this method. Otherwise, URL rewriting cannot be used with browsers, which do not support cookies.



Note:
Refer to com.igate.ch8.controller.SessionSnoop.java servlet code; partial code in doGet() method is here :

```
out.println("<H3>Here are some vital stats on your session:</H3>");
out.println("Session id: " + session.getId() + "<BR>");
out.println("New session: " + session.isNew() + "<BR>");
out.println("Creation time: " + session.getCreationTime());
out.println("<I>(" + new Date(session.getCreationTime()) + ")</I><BR>");
out.println("Last access time: " + session.getLastAccessedTime());
out.println("<I>(" + new Date(session.getLastAccessedTime()) +
")</I><BR>");
out.println("Requested session ID from cookie: "
+req.isRequestedSessionIdFromCookie() + "<BR>");
out.println("Requested session ID from URL: " +
req.isRequestedSessionIdFromURL() + "<BR>");
out.println("Requested session ID valid: " +
req.isRequestedSessionIdValid() + "<BR>");
out.println("<H3>Test URL Rewriting</H3>");
out.println("Click <A HREF=\"" +
      res.encodeURL(req.getRequestURI()) + "\">here</A>");
out.println("to test that session tracking works via URL");
out.println("rewriting even when cookies aren't supported.");
out.println("</BODY></HTML>");
```

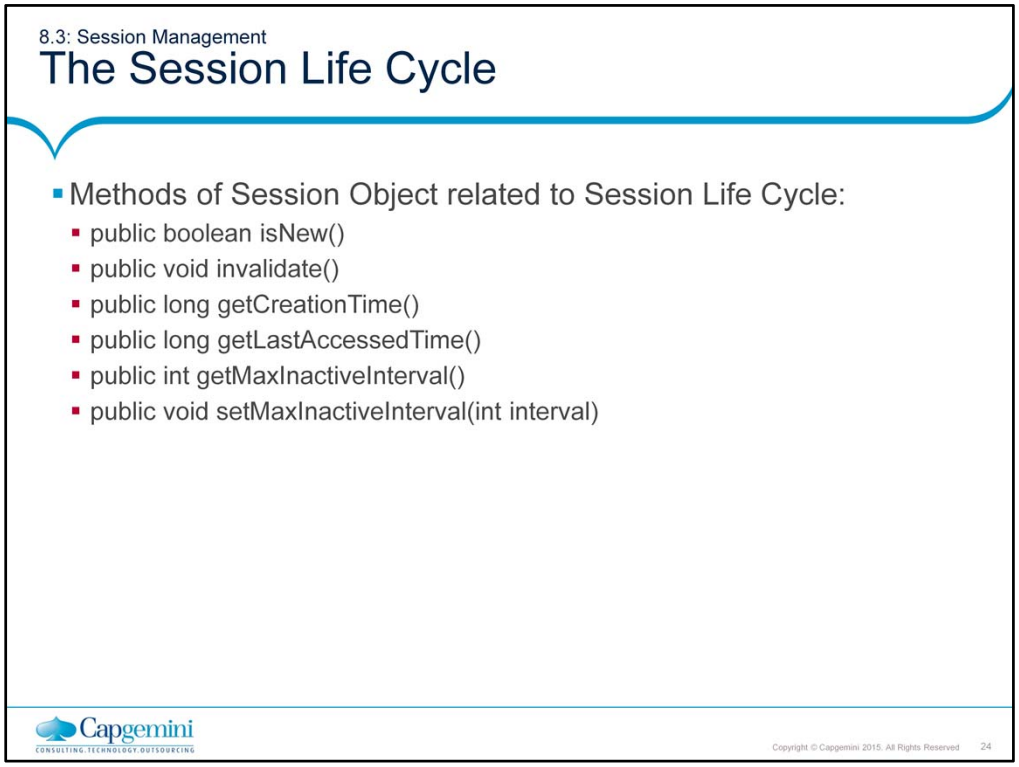


Session Management:
What is Session Management?
As we have seen in the first slide of this lesson, logically a session begins with the first connection to an application by a client and ends after that client's last connection. Session can be an authenticated session also, where a user has to log-in. In this case sessions begins when the user logs in and ends when the user logs-out. Session also ends after a time-out period of inactivity. We as programmers have to manage this life cycle of session, i.e. Creation and destruction of session.
Managing the life cycle of session is very important due to following reasons:
The Servlet API is rather liberal in creating sessions. Various tools have default behaviours which can implicitly create sessions in the background. It's very easy for an application to "accidentally" create a session, even when one was not explicitly requested.
Sessions have various security risks associated with them.
Sessions consume server resources, and should likely be avoided if possible.
Ensure the following for session management :

Create a new session only when the user logs in
Invalidate the session and delete any related cookie, session data
Set appropriate time-out period in web.xml like follows :

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
```

Here time-out is set to 30 minutes
In Servlet 3.0 this is a global way of setting the Session time out period



Session Management:
The Session Life Cycle:
Since there is no way for an HTTP client to signal that it no longer needs a session, each session has an associated timeout so that its resources can be reclaimed. The timeout period can be accessed by using a session's [get|set]MaxInactiveInterval methods.
Methods of Session Object related to Session Life Cycle :
public boolean isNew()
It returns true if the Web server has created a session but the client has not yet joined. For example, if the server used only cookie-based sessions, and the client had disabled the use of cookies, then a session would be new.
public void invalidate()
It invalidates this session and unbinds any objects bound to it.
public long getCreationTime()
It returns the time in long integer specifying when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

Session Management:

The Session Life Cycle:

- **public long getLastAccessedTime()**
  It returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
  Actions that the application takes, such as getting or setting a value associated with the session do not affect the access time. The last accessed time can help to manage sessions. For example, the sessions can be sorted according to age to optimize some task.
- **public int getMaxInactiveInterval()**
  It returns the maximum time interval, in seconds, as integer that the servlet engine will keep this session open between client requests. It is recommended to set the maximum time interval with the setMaxInactiveInterval method.
- **public void setMaxInactiveInterval(int interval)**
  It specifies the maximum length of time, in seconds, that the servlet engine keeps this session if no user requests have been made of the session and takes an integer as a parameter specifying the number of seconds.

Demo

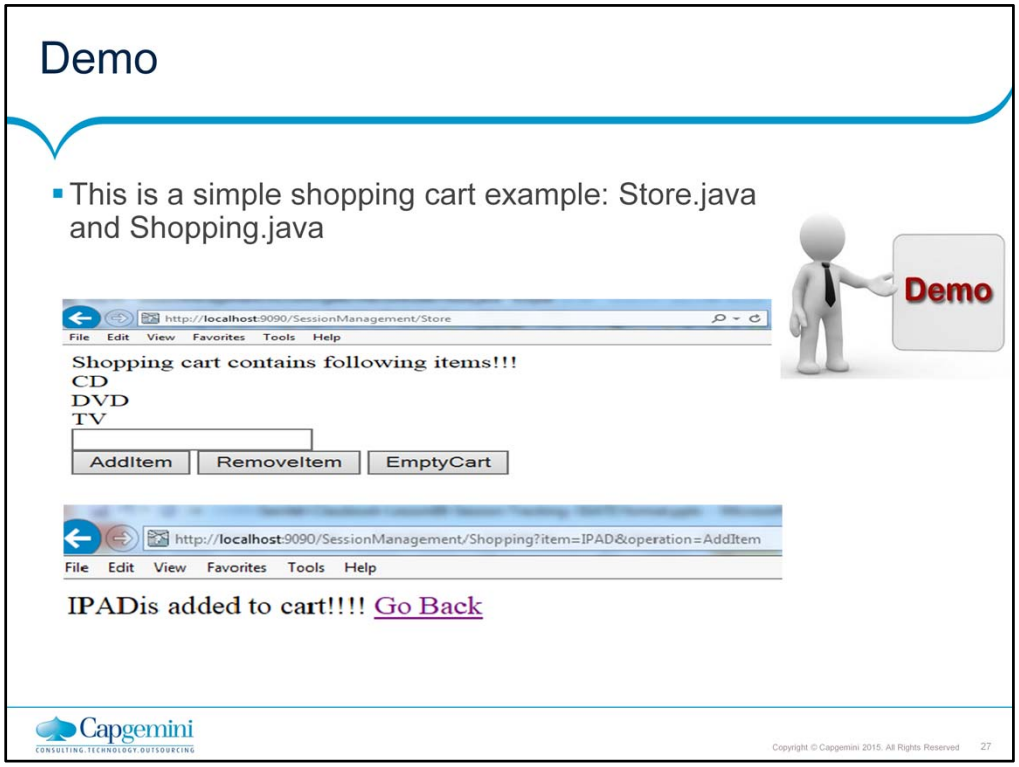
- Execute a servlet that manually invalidates a session if it is more than a day old or has been inactive for more than an hour.
  - Manual Invalidate servlet

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26

Note:
Refer to com.igate.ch8.controller.ManualInvalidate.java servlet code; partial code of doGet() method is given here:

```
// Get the current session object, create one if necessary
    HttpSession session = req.getSession(true);
    // Invalidate the session if it's more than a day old or has been
    // inactive for more than an hour.
    if (!session.isNew()) {  // skip new sessions
      Date dayAgo = new Date(System.currentTimeMillis() - 24*60*60*1000);
      Date hourAgo = new Date(System.currentTimeMillis() - 60*60*1000);
      Date created = new Date(session.getCreationTime());
      Date accessed = new Date(session.getLastAccessedTime());
      if (created.before(dayAgo) || accessed.before(hourAgo)) {
        session.invalidate();
        session = req.getSession(true);  // get a new session
      }
```



Note:
Execute Servlets:

http://localhost:9090/SessionManagement/Store

This servlet maintains a list of shopping items

http://localhost:9090/SessionManagement/Shopping?item=CD&operation=AddItem
This servlet does different operations like adding item to cart, removing item from cart and making the cart as empty.

8.4: Best Practices

Best Practices in Session Tracking

- An application that uses URL rewriting to track sessions must adhere to certain programming guidelines as URL rewriting has high security risks.
- The application developer needs to:
  - Program session servlets to encode URLs
  - Supply a servlet or JSP file as an entry point to the application
  - Avoid using plain HTML files in the application
  - All emitted links must be consistently rewritten

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 28

Summary

- In this lesson, we have learnt:
  - The concept of Session Tracking
  - Session Tracking Techniques
  - The concept of Session Management

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

Review Questions

- Question 1: Web Application developer needs to maintain client's state on server because:
  - Option 1: To remind client of his information
  - Option 2: Http is a stateless protocol
  - Option 3: Http is unreliable protocol
  - Option 4: No need as its taken care of by the Http Server

- Question 2: Browsers required to accept ___ cookies per site:
  - Option 1: 1
  - Option 2: 40
  - Option 3: 20
  - Option 4: unlimited

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review Questions

- Question 3: ___ method is used to end the session
  - Option 1: invalidate()
  - Option 2: logout()
  - Option 3: invalidSession()

- Question 4: Global timeout for all the sessions is maintained by.
  - Option 1: HttpSession.setMaxInactiveInterval(600)
  - Option 2: iHttpSession.logout(600)
  - Option 3: mapping in web.xml
  - Option 4: All Options are true

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31