WKW SCHOOL

2020

CI6206 Internet Programming

HTTP (GET/POST/Session)

Wong Twee Wee

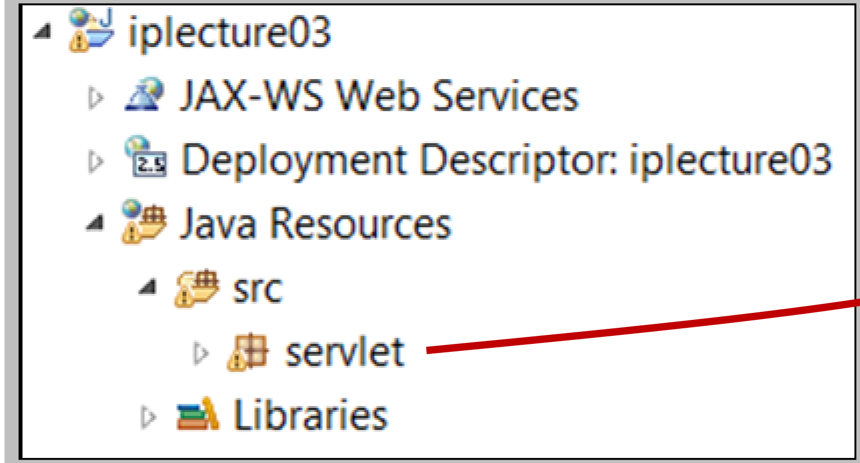
Ver1.1

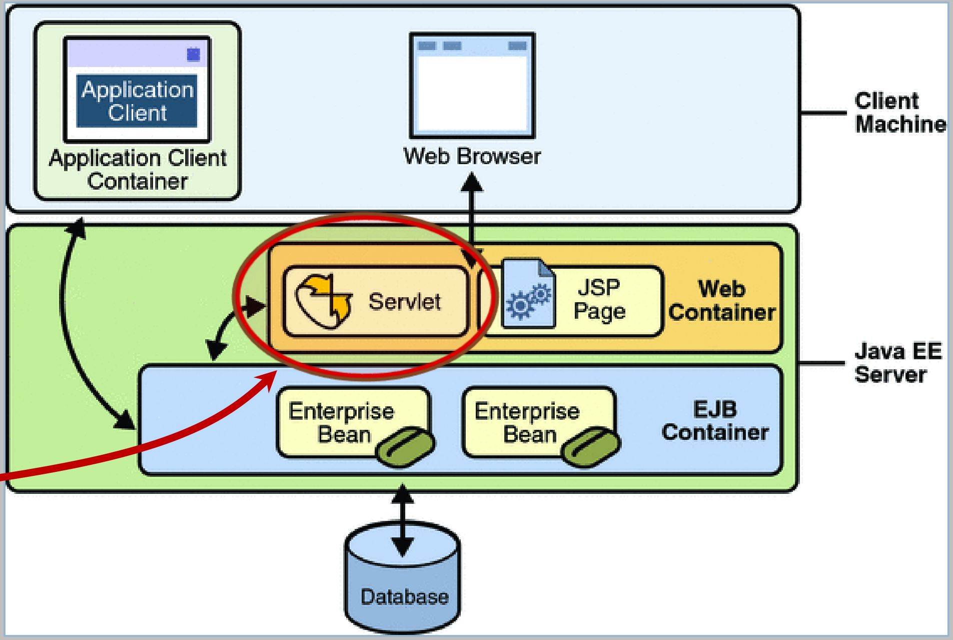
TOPICS

- Java Servlets (Web Application)
- Handling HTTP requests and form data
- Generating HTTP responses
- Session tracking

JAVA SERVLETS

- Java-based Web component that generates dynamic content
- Managed by a container/servlet engine
  - Part of a Web server or application server
  - Processes HTTP requests and responses





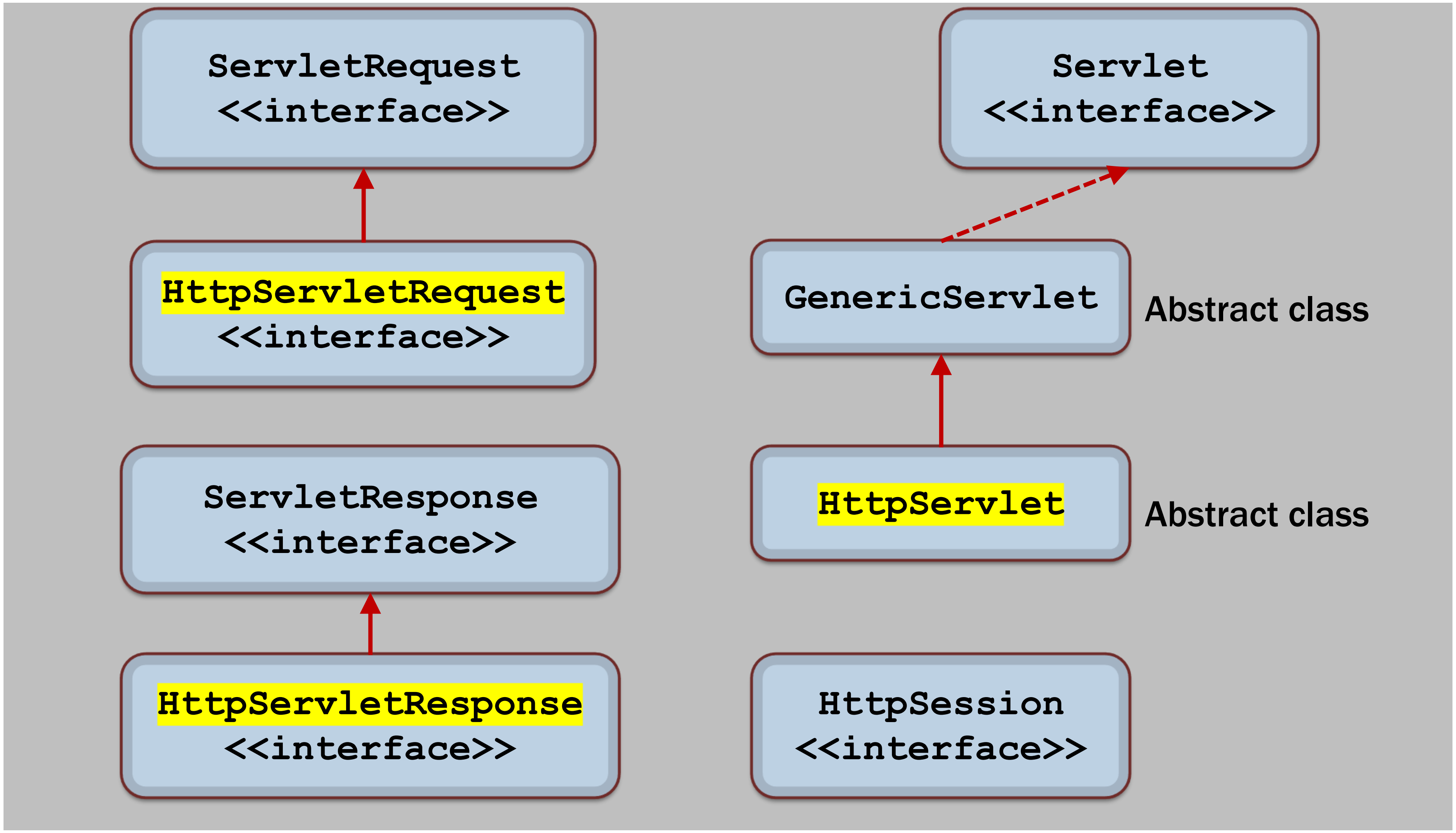
JAVA SERVLETS

- Subclass of HttpServlet
  - Override doGet(), doPost(), doXXX() methods
  - Arguments
    - HttpServletRequest
    - HttpServletResponse
  - Exceptions thrown
  - ServletException
  - IOException

THE SERVLET INTERFACE

- A Servlet, in its most general form, is an instance of a class, which <u>**implements**</u> the **javax.servlet.Servlet interface**.
- Most Servlets, <u>**extend**</u> one of the standard implementations of that interface, usually **javax.servlet.http.HttpServlet**.
  - HTTP Servlets extend the javax.servlet.http.HttpServlet class
  - [API DOCS : https://docs.oracle.com/javaee/7/api/toc.htm](https://docs.oracle.com/javaee/7/api/toc.htm)

JAVA SERVLETS - INTERFACES AND CLASSES


ServletRequest <<interface>>
HttpServletRequest <<interface>>
ServletResponse <<interface>>
HttpServletResponse <<interface>>
Servlet <<interface>>
GenericServlet
Abstract class
HttpServlet
Abstract class
HttpSession <<interface>>

SERVLET CLASS

javax.servlet.GenericServlet

Signature: ***public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable***

- ***GenericServlet*** defines a generic, protocol-independent servlet.
- ***GenericServlet*** gives a blueprint and makes writing servlet easier.
- ***GenericServlet*** provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface.
- ***GenericServlet*** implements the log method, declared in the ServletContext interface.

javax.servlet.http.HttpServlet

Signature: ***public abstract class HttpServlet*** *extends GenericServlet* ***implements java.io.Serializable***

- ***HttpServlet*** defines a HTTP protocol specific servlet.
- ***HttpServlet gives a blueprint for Http servlet and makes writing them easier.***
- ***HttpServlet*** extends the ***GenericServlet*** and hence inherits the properties ***GenericServlet***.

INTERFACE CLASS

```
// I say all motor vehicles should look
like this:
interface MotorVehicle
{
    void run();
    int getFuel();
}

// Implement a vehicle looks  that way
class Car implements MotorVehicle
{
    int fuel;
    void run()
    {
        print("Wrroooooooom");
    }

    int getFuel()
    {
        return this.fuel;
    }
}
```

- **An interface is not a class.**
- **An interface has no implementation**
- **It is an empty shell whose methods <u>MUST</u> be implemented.**

ABSTRACT CLASS

```
// I say all animals should behave like
this:
public abstract Animal
{
   public void eat(Food food)
   {
        // do something with food....
   }
   public void sleep(int hours)
   {
      ...
      Thread.sleep();
   }

   public abstract void makeNoise();
}

public Cow extends Animal
{
   public void makeNoise() {
   System.out.println ("Moo! Moo!"); }
}
```

- **Abstract classes may or may not contain abstract methods.**
- **if a class has at least one abstract method, then the class <u>must</u> be declared abstract.**
- **An abstract class contains one or more abstract methods**
- **An <u>abstract method</u> contains no implementation.**
- **Abstract classes cannot be instantiated**
- **To use abstract class, you have to inherit it and provide implementations to the <u>all</u> the abstract methods.**

SERVLET SUBCLASS

If a Servlet is subclass of HttpServlet it must override one of the following methods:-

- doGet() – for HTTP GET requests
- doPost() - for HTTP POST requests
- doDelete() - for HTTP DELETE requests
- doPut() - for HTTP PUT requests
- init()/destroy() - for managing the resources during the life of the servlet
- getServletInfo() - this method is used by the servlet to provide information about itself

SERVLET SUBCLASS

A subclass of HttpServlet must override at least one method, usually one of these:

- doGet, if the servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests
- doDelete, for HTTP DELETE requests
- init and destroy, to manage resources that are held for the life of the servlet
- getServletInfo, which the servlet uses to provide information about itself

```
package servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        public MyServlet() {
          super();
        }
        protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
                                             IOException {
        //... Add codes here
        }
        protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException,
                                                     IOException {
        //... Add codes here
        }
}
```

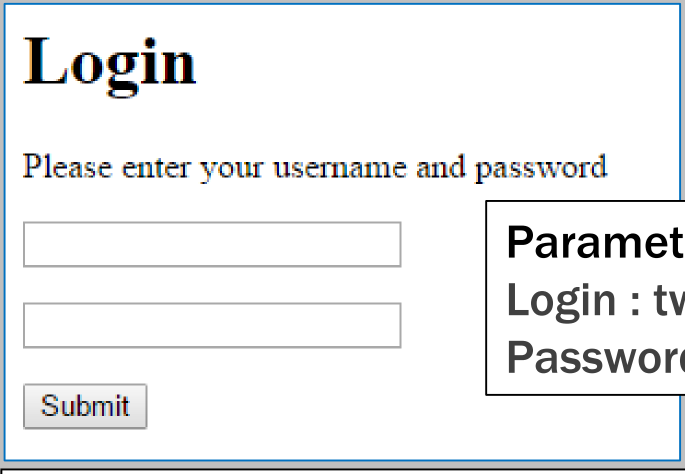
WRITING A SERVLET

- HTML Form (Request)
  - html, url, deployment (web.xml)
- Create a servletclass (Response)
  - extend HttpServlet
- Implement the doGet() or doPost() method
  - Both methods accept two parameters
    - HttpServletRequest
    - HttpServletResponse
  - Handling FORM data from HttpServletRequest Interface using
    - *getParameter(String name)*
  - Obtain the writer from the response object
  - Process input data and generate output (in html form) and write to the writer
  - Close the writer

HANDLING FORM DATA

- Methods in **ServletRequest** interface
  - Typical sources of data: form data, environment variables, HTTP headers
  - ***getParameter()***
    - **to get the value of a form parameter.**
  - ***getParameterValues()***
    - **to get parameter appears more than once and returns multiple values, for example checkbox**
  - ***getParameterNames()***
    - **To get a complete list of all parameters in the current request**
- Example
  - feedback.html
  - ProcessParameters.java

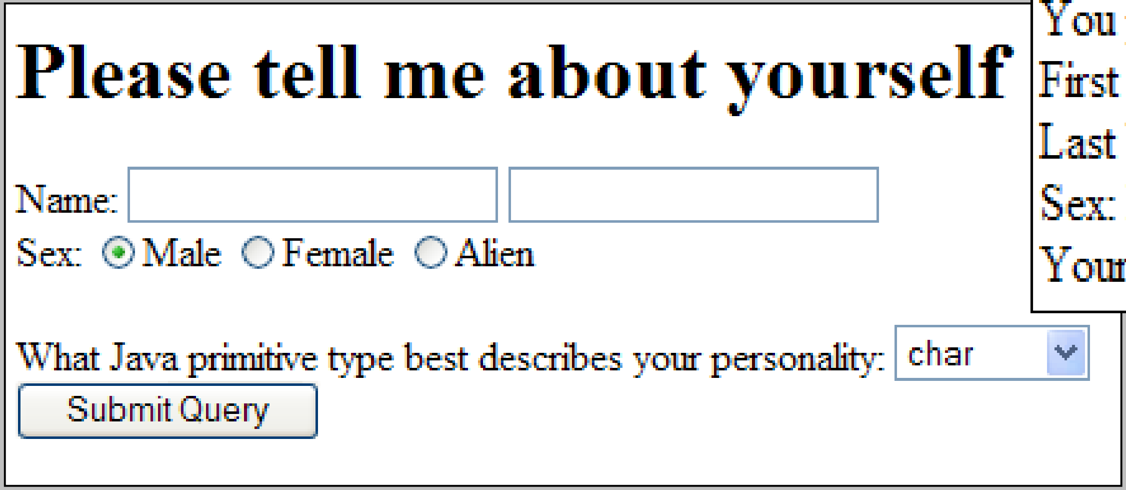
FORM



Login

Please enter your username and password

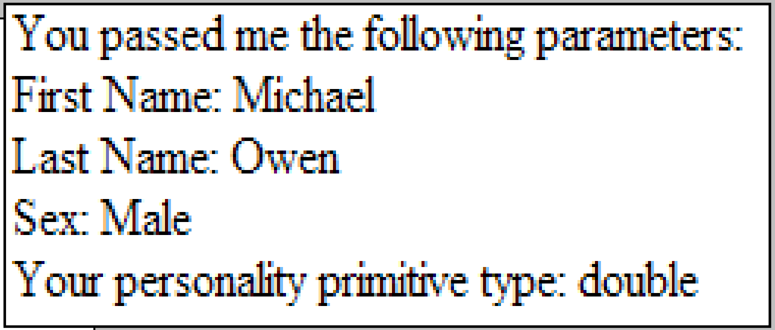
Parameter Pairs
Login : twwong
Password : @pple123



Please tell me about yourself

Name:
Sex: [✓ Male] Female Alien

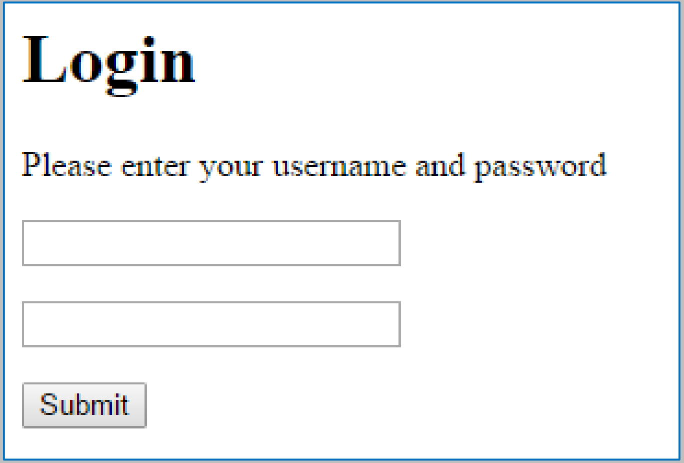
What Java primitive type best describes your personality: char



You passed me the following parameters:
First Name: Michael
Last Name: Owen
Sex: Male
Your personality primitive type: double

EXAMPLE 1
LOGIN.HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Login</title>
  </head>
  <body>
    <h1>Login</h1>

    Please enter your username and password
    <form action="servlet/Login" method="POST">
      <p><input type="text" name="username" length="40">
      <p><input type="password" name="password" length="40">
      <p><input type="submit" value="Submit">
    </form>
  </body>
</html>
```



Login

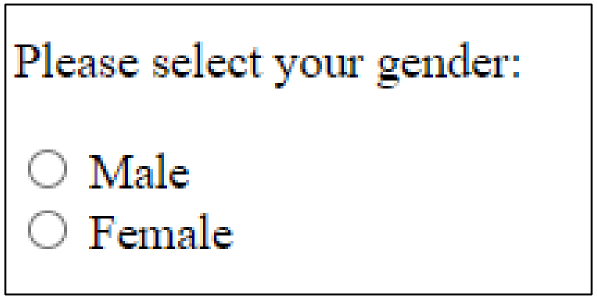
Please enter your username and password

EXAMPLE 1
LOGIN SERVLET

```
package wkwsci.ntu.edu.sg.practicals.servlets;
import javax.servlet.http.*;
import java.io.*;
public class Login extends HttpServlet {
  public void doPost( HttpServletRequest request, HttpServletResponse response)
        {
    // Get the parameter from the request
    String username = request.getParameter("username");
    // Send the response back to the user
    try {
        response.setContentType("text/html");

        PrintWriter writer = response.getWriter();
        writer.println("<html><body>");
        writer.println("Thank you, " + username + ". You are now logged into the system.");
        writer.println("</body></html>");
        writer.close();

      } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

RADIOBUTTON

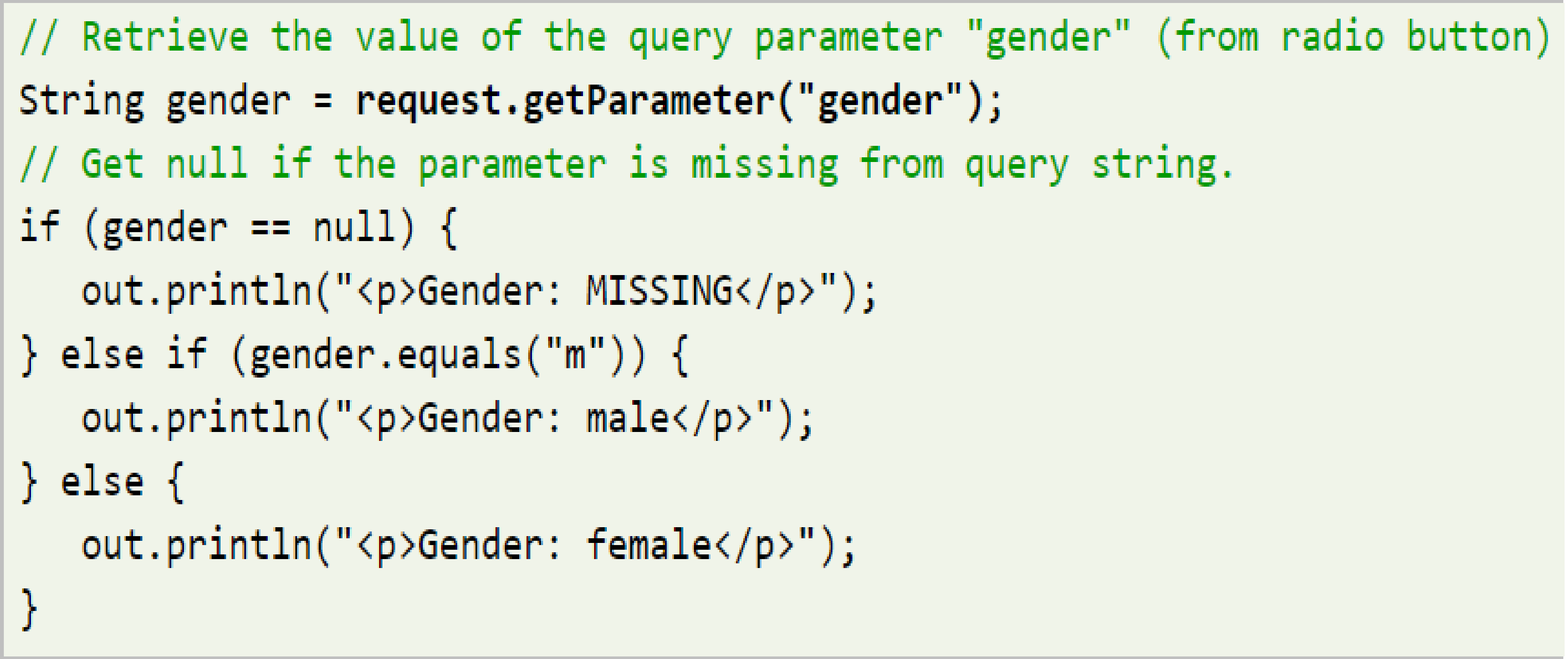
```
<html>
<body>

<h1>Gender</h1>

<form action="/Register">
  <p>Please select your gender:</p>
  <input type="radio" id="male" name="gender" value="m">
  <label for="male">Male</label><br>
  <input type="radio" id="female" name="gender" value="f">
  <label for="female">Female</label><br>
</form>

</body>
</html>
```



Please select your gender:

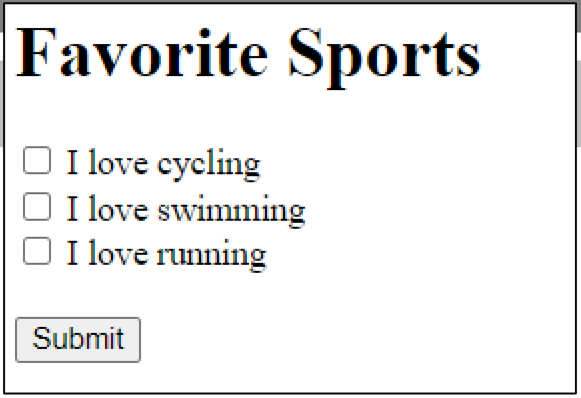
○ Male
○ Female

PROCESSING RADIOBUTTON



```
// Retrieve the value of the query parameter "gender" (from radio button)
String gender = request.getParameter("gender");
// Get null if the parameter is missing from query string.
if (gender == null) {
   out.println("<p>Gender: MISSING</p>");
} else if (gender.equals("m")) {
   out.println("<p>Gender: male</p>");
} else {
   out.println("<p>Gender: female</p>");
}
```

CHECKBOX


Favorite Sports

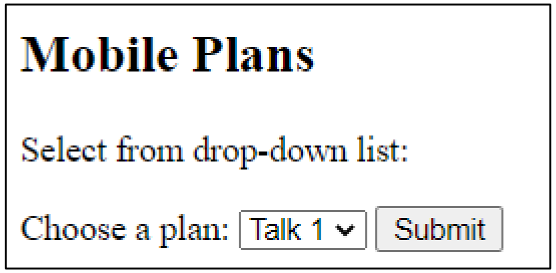
```
<html>
<body>

<h1>Favorite Sports</h1>

<form action="/action_page.php">
  <input type="checkbox" id="Hobby1" name="favsports" value="Cycling">
  <label for="Hobby1"> I love cycling</label><br>
  <input type="checkbox" id="Hobby2" name="favsports" value="Swimming">
  <label for="Hobby2"> I love swimming</label><br>
  <input type="checkbox" id="Hobby3" name="favsports" value="Running">
  <label for="Hobby3"> I love running</label><br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

PROCESSING

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException
{

String[] sports = request.getParameterValues("favsports");
List list = Arrays.asList(sports);
request.setAttribute("sports", list);
RequestDispatcher rd = request.getRequestDispatcher("Register.jsp");
rd.forward(request, response);

}
```

DROPDOWN LIST

```
<html>
<body>

<h2>Mobile Plans</h2>

<p>Select from drop-down list:</p>

<form action="/Register">
  <label for="plans">Choose a plan:</label>
  <select id="plans" name="plans">
    <option value="Talk1">Talk 1</option>
    <option value="Talk2">Talk 2</option>
    <option value="Talk3">Talk 3</option>
    <option value="Talk4">Talk 4</option>
  </select>
  <input type="submit">
</form>

</body>
</html>
```



Mobile Plans

Select from drop-down list:

Choose a plan: Talk 1 Submit

PROCESSING

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException
{

String mobileplans = request.getParameter(“plans");


}
```

HTTPSERVLETREQUEST INTERFACE

- Extends ServletRequest
- Inherited methods from ServletRequest
  - getParameterNames()
  - **getParameter(String name)**
  - getParameterValues(String name)
  - getServerName()
  - getServerPort()
  - **getRequestDispatcher**
- Others
  - getHeader()
  - getPathInfo()
  - getContextPath()
  - getQueryString()

HTTPSERVLETRESPONSE INTERFACE,.

- Extends ServletResponse
- Inherited methods from ServletResponse
  - **getWriter(String name)** //Pen
  - **setContentType()** //Return type

DEPLOYMENT DESCRIPTOR (WEB.XML)

- Configuration and deployment information
  - **ServletContex**t initialization parameters
  - Session configuration
  - Servlet/JSP definitions
  - Servlet/JSP mappings
  - MIME type mappings
  - Welcome file list
  - Error pages
  - Security

DEPLOYMENT DESCRIPTOR (WEB.XML)

- **Base format**

```
<web-app> ... other elements here ... </web-app>
```

- **Naming servlets**

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>test.HelloWorld</servlet-class>
</servlet>
```

- **Mapping servlets**

```
<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Order matters – naming then mapping

DEPLOYMENT DESCRIPTOR (WEB.XML)

- **Servlet initialization parameters**

```
<servlet>
 <init-param>
   <param-name>tempFile</param-name>
   <param-value>/tmp/file.dat</param-value>
 </init-param>
</servlet>
```

- **Context initialization parameters**

```
<context-param>
 <param-name>footerPage</param-name>
 <param-value>/data/foot11-00.html</param-value>
</context-param>
```

`<init-param>`
defines a value available to a **single specific servlet** within a context.
`<context-param>`
defines a value available to **all the servlets** within a context.

```
<web-app>
  <context-param>
        <param-name>email</param-name>
        <param-value>plk1745@hotmail.com</param-value>
  </context-param>
  <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>example.HelloWorld</servlet-class>
        <init-param>
              <param-name>language</param-name>
              <param-value>english</param-value>
        </init-param>
  </servlet>
  <servlet-mapping>
        <servlet-name>HellowWorld</servlet-name>
        <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

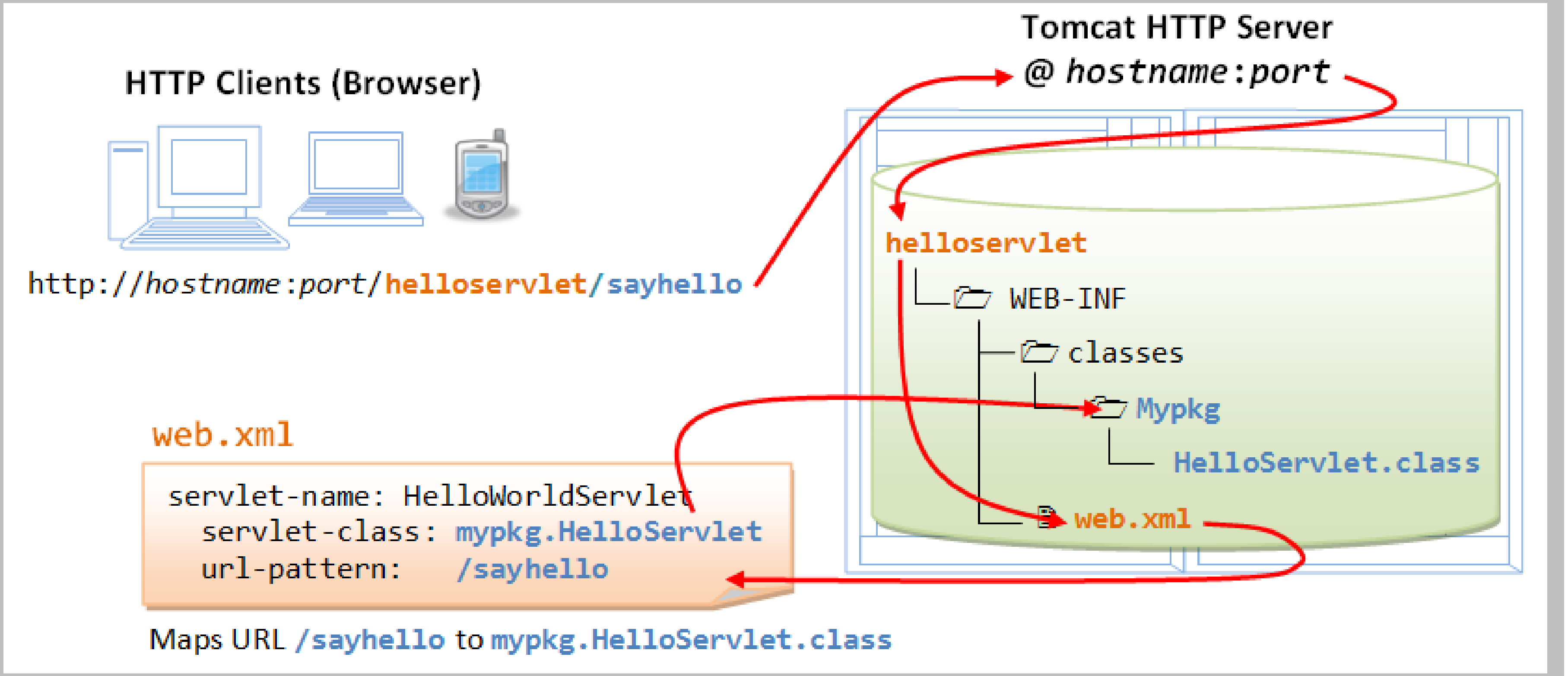
EXAMPLE 1
WEB.XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Login Servlet</display-name>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>
        wkwsci.ntu.edu.sg.practicals.servlets.Login
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/Login</url-pattern>
  </servlet-mapping>
</web-app>
```

Running the servlet

http://localhost:8080/**Login**/login.html

WEB.XML MAPPING

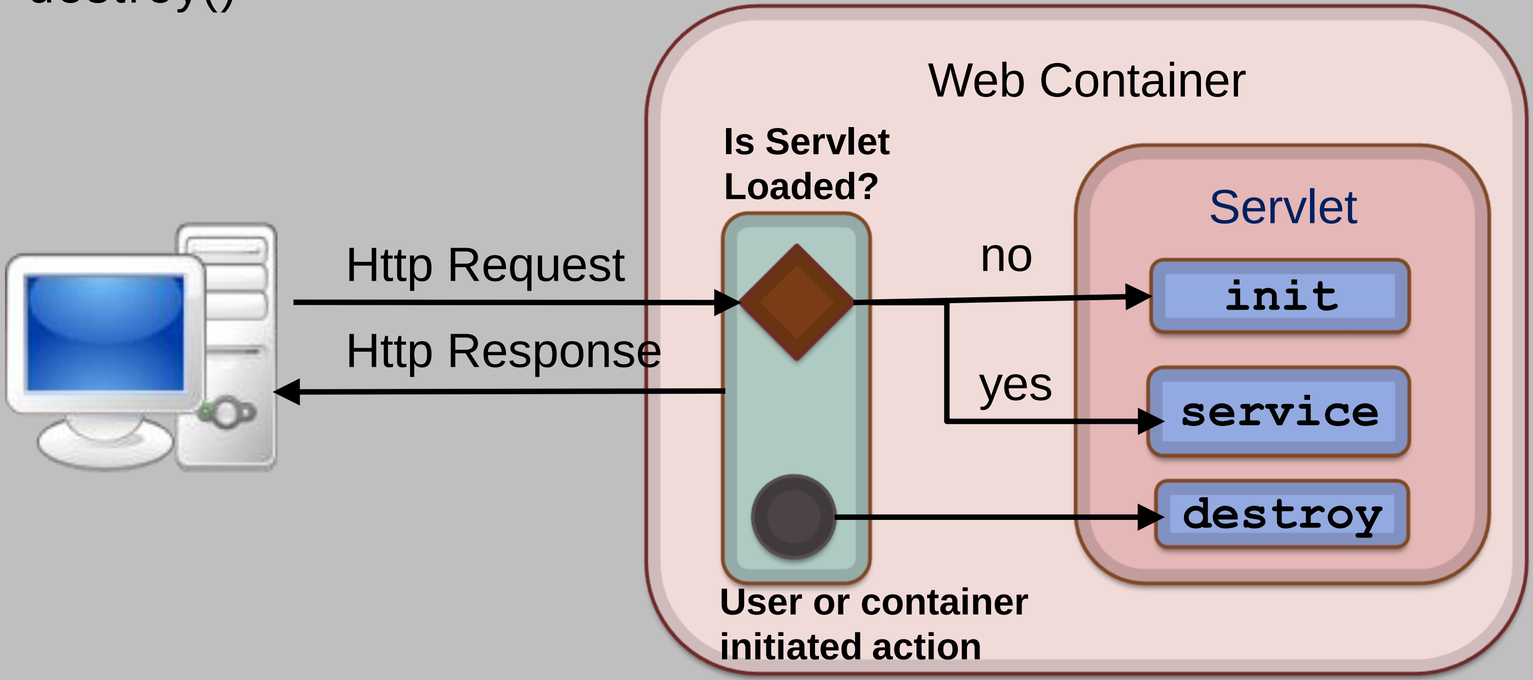


```
servlet-name: HelloWorldServlet
  servlet-class: mypkg.HelloServlet
  url-pattern:    /sayhello
```

Maps URL /sayhello to mypkg.HelloServlet.class

SERVLET LIFE CYCLE

- Life cycle of servlet is controlled by container.
- The various life cycle methods are invoked by container
  - init()
  - service()
  - destroy()



SERVLET LIFE CYCLE

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.

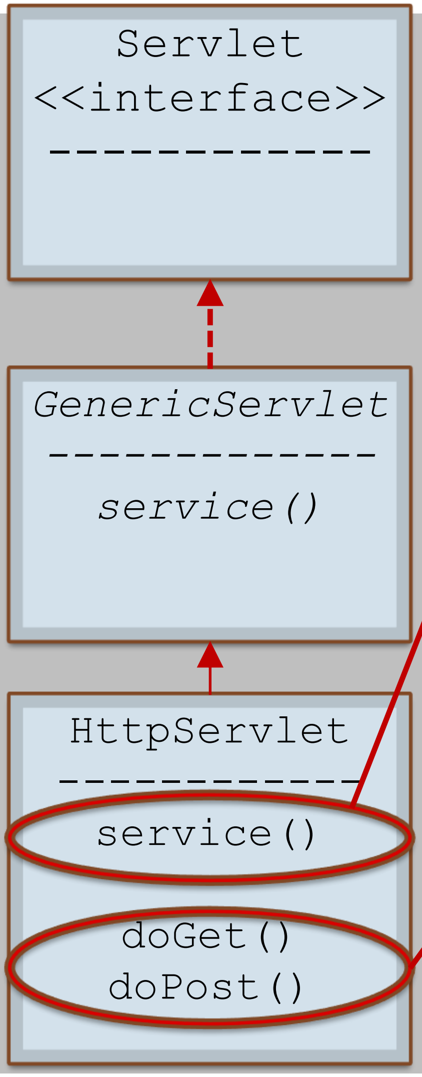
SERVLET LIFE CYCLE

If an instance of the servlet does not exist, the Web container

i. Loads the servlet class.
ii. Creates an instance of the servlet class.
iii. Initializes the servlet instance by calling the init method.
iv. Invokes the service method, passing a request and response object.

- init()
  - Designed to be called only once.
  - It is called when the servlet is first created, and <u>not called again</u> for each user request.
  - Used for one-time initialization.
  - Perform any set-up in this method
- service()
  - Invoked whenever received a request from client
  - Extract client data and perform business operation
  - Populate HTTP response and sent back to client
- destroy()
  - Invoked once before servlet instance is removed
  - Perform any clean-up in this method

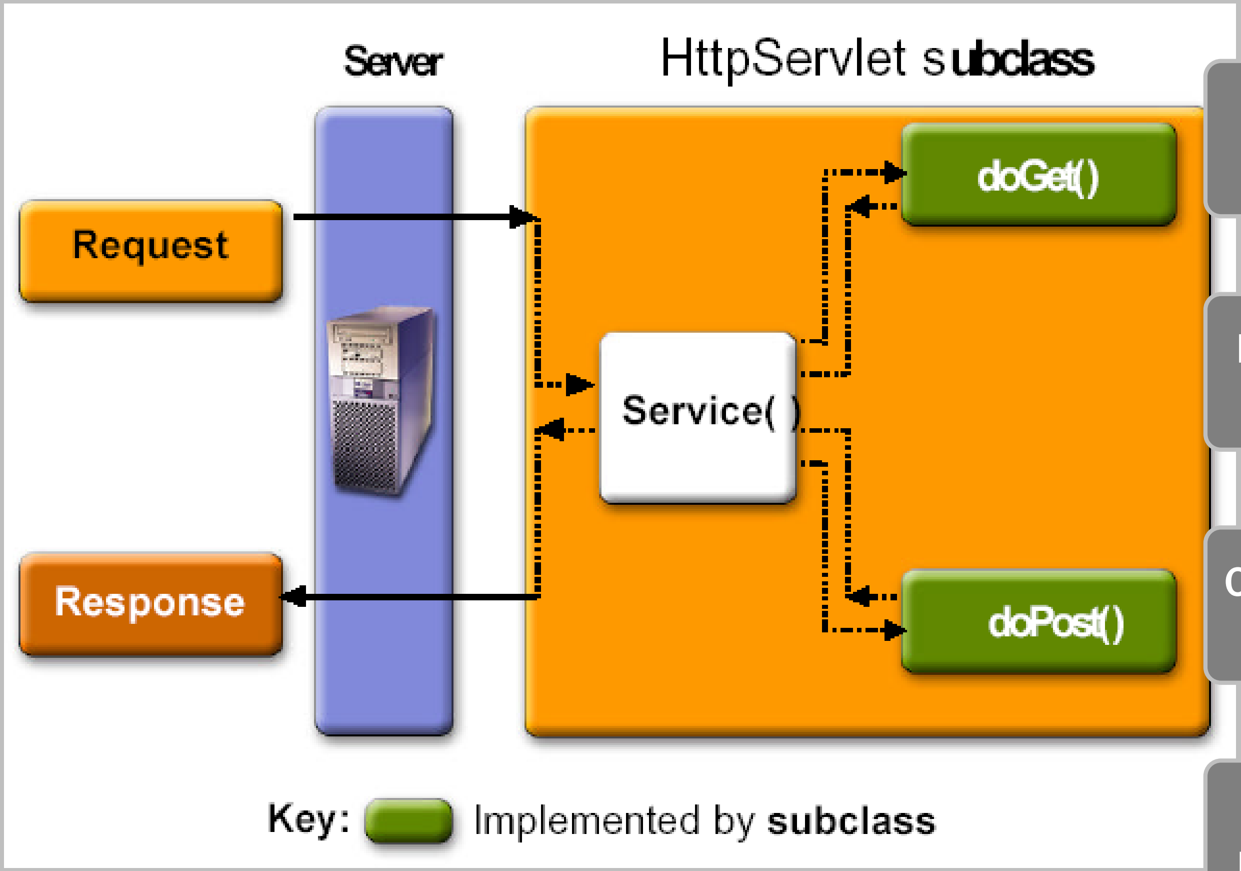
GENERICSERVLET AND HTTPSERVLET



Overrides abstract service() in GenericServlet
Dispatch the HTTP request to the respective doGet(), doPost()
Do not override this method in your servlets

Handles HTTP GET, POST methods
Overrides these methods in your servlets

HTTPSERVLET



Extract client-sent information from HTTP request

Getting or setting the value from/to the scope object

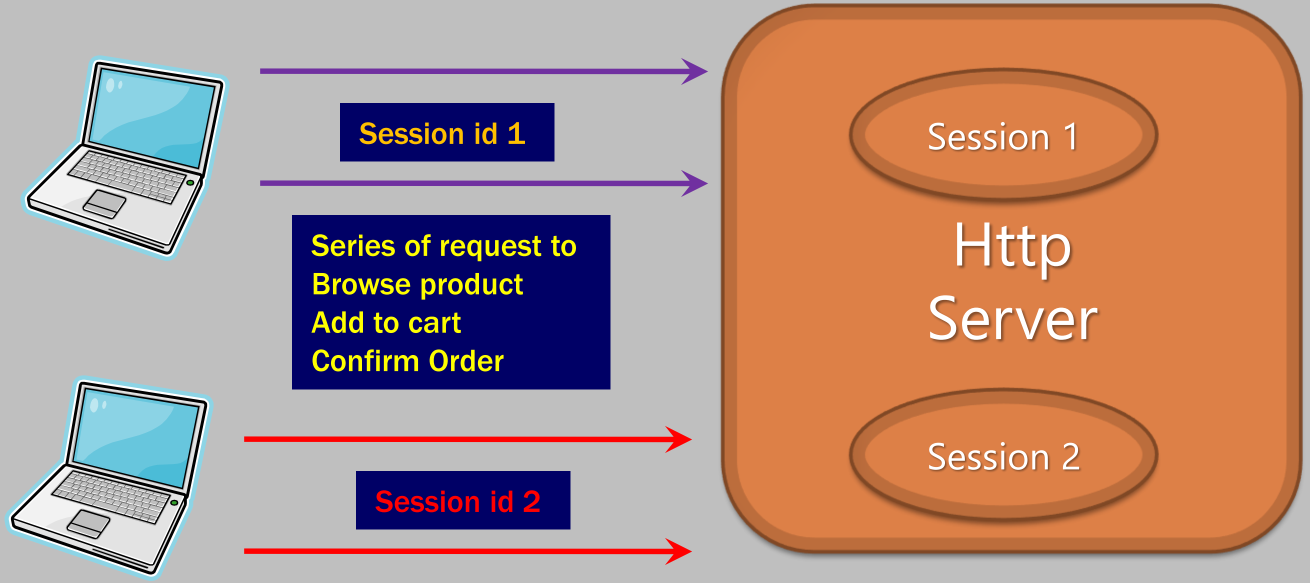
Perform some business logic and database access

Optionally forward the request to other Web components

Populate HTTP response message and sent it to client

HTTP IS STATELESS

- Need to track Session
- Need to maintain state across a series of requests from the same user over some period of time.
- Able to differentiate between client request



SESSION TRACKING

- Goal
  - Assign unique ID to users to maintain state
  - Conceptually, two data structures required
    - ID to user mapping table
    - Storage for users’ state information
  - All pages are dynamic
- Hidden form fields
  - <input type="hidden" name="id" value="1234">
  - ID transmitted using GET or POST
  - All pages result of form submissions

SESSION TRACKING

- Cookies
  - Generating the unique session identifiers
  - Extracting cookie that stores session identifier
- URL rewriting
  - ID transmitted as extra information on URL
    - http://host/path/file.html;uniqueid=1234
  - Works even if cookies are disabled or unsupported
- Session objects
  - Represents user’s session
  - HttpSession session = **request.getSession();**
  - Programmers deal with session objects to store/retrieve information

HTTPSESSION OBJECT

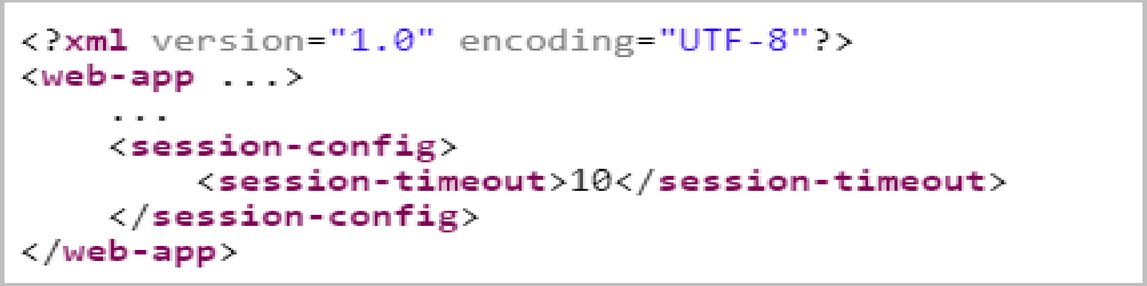
- Mechanism to store state information
  - Access/create with
    - HttpServletRequest.getSession()
  - Manipulate with
    - getAttribute(), setAttribute()
    - removeAttribute(), invalidate()
    - getMaxInactiveInterval(), setMaxInactiveInterval()

HTTPSESSION OBJECT

```
public class ProductServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, …) {
    // create a session
    HttpSession session = req.getSession();
    ….
    // store attribute
    session.setAttribute(“userid”, id);
    ……
    // retrieve attribute
    String id = (String)session.getAttribute(“userid”);
  }
}
```

SESSION TIMEOUT

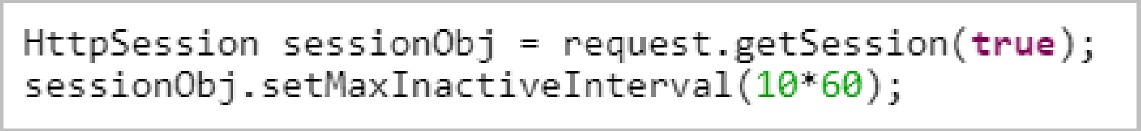
- Used when an user leave the browser without actively logging out of the system
- Session timeout after 30 mins (default)
- The time out value can be set on the deployment descriptor, web.xml file



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
    ...
    <session-config>
        <session-timeout>10</session-timeout>
    </session-config>
</web-app>
```

time in minutes

- Or can be set programmatically using the API setMaxInactiveInterval()



```
HttpSession sessionObj = request.getSession(true);
sessionObj.setMaxInactiveInterval(10*60);
```

time in Seconds

SESSION INVALIDATION

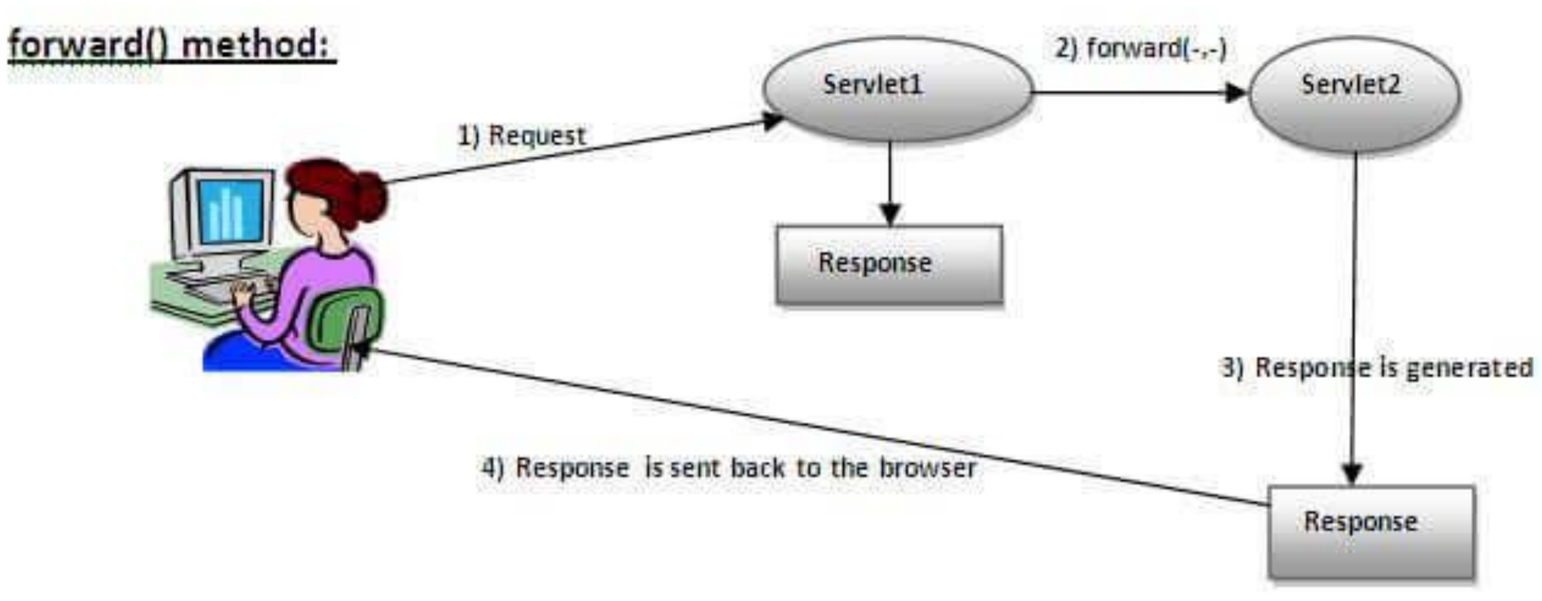
- Used by the programmer to end a session directly.
- Typically provide a logout page to allow user to invalidate their session

```
public class LogoutServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, ...) {
    // retrieve a session
    HttpSession session = req.getSession();
    ....
    session.invalidate();
  }
}
```

INVOKING OTHER WEB RESOURCES

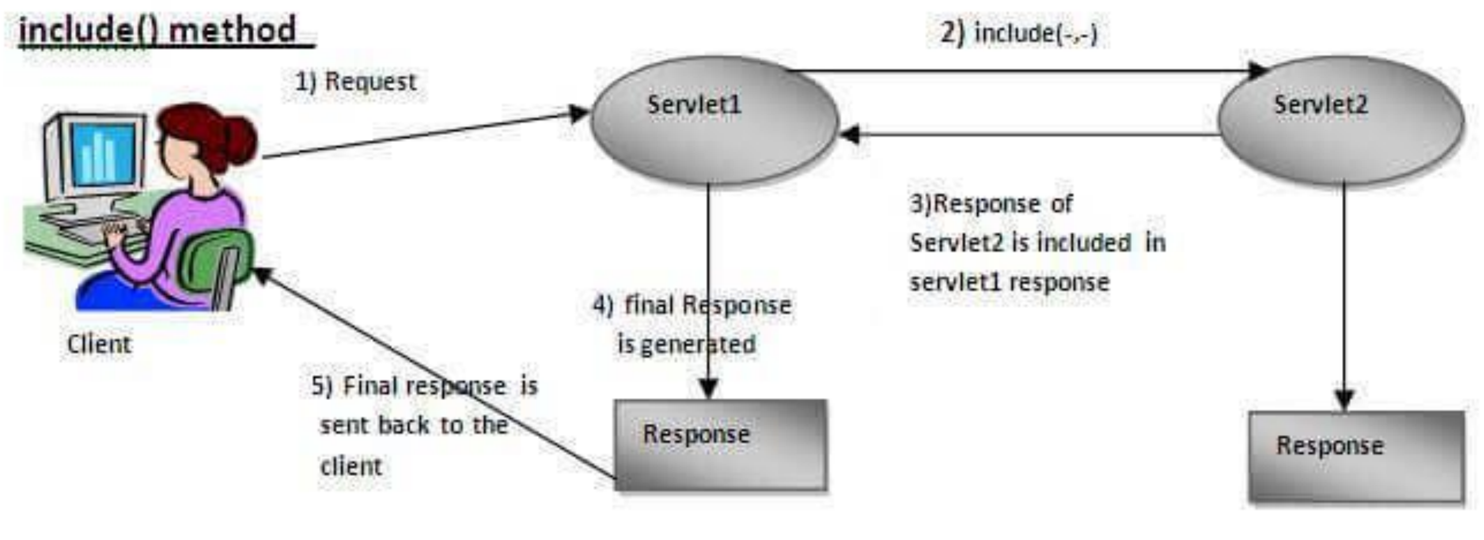
- Servlet Request Dispatcher
  - an interface whose implementation defines that an object can dispatch requests to any resource (such as HTML, Image, JSP, Servlet etc.) on the server.
  - Used in two cases:
    - To include the response of one Servlet into another (i.e. the client gets the response of both Servlets)
    - To forward the client request to another Servlet to honor the request (i.e. the client calls a Servlet but the response to client is given by another Servlet)

FORWARDING



Source : java point

INCLUDE


include() method
1) Request
2) include(-,-)
Servlet1
Servlet2
3)Response of Servlet2 is included in servlet1 response
4) final Response is generated
5) Final response is sent back to the client
Client
Response
Response

Source : java point

INVOKING OTHER WEB RESOURCES

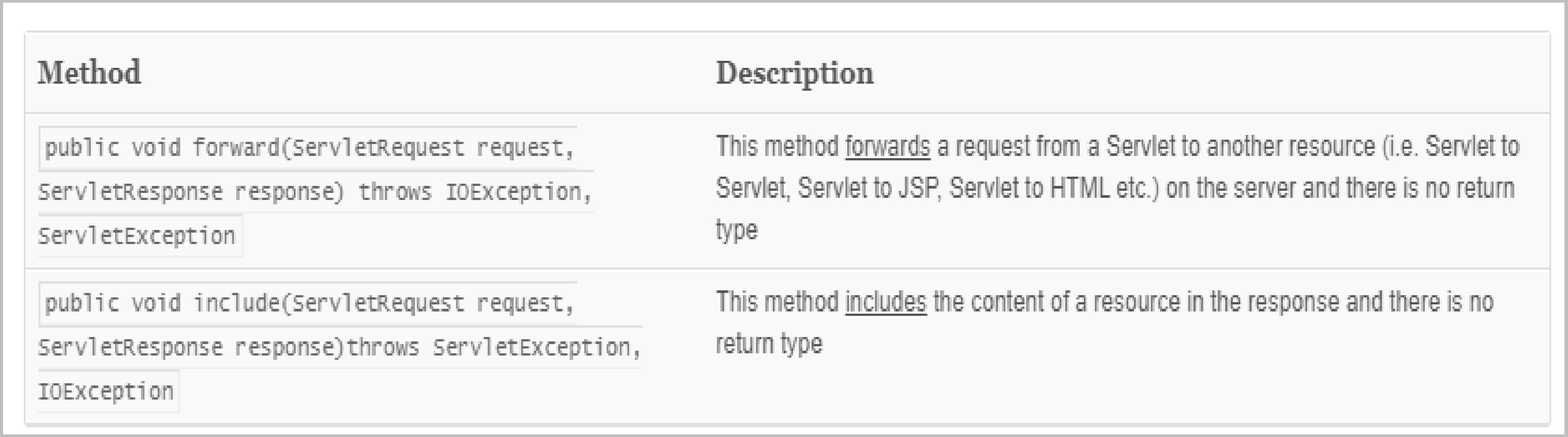
- RequestDispatcher object
  - Retrieve the RequestDispatcher object from the **request** (HttpServletRequest) or **web context** (ServletContext) as follows:

    getRequestDispatcher([*Path of targeted web resource*])

    - From request, the path is relative to current servlet
      - Relative path – does not begin with **“/”**
    - From web context, path must be absolute
      - Must begin **“/”**
    - If resource is not available – null is returned
      - Wrong path or resource not found

METHODS

Both methods can be found in the javax.servlet package:



Method	Description
`public void forward(ServletRequest request, ServletResponse response) throws IOException, ServletException`	This method forwards a request from a Servlet to another resource (i.e. Servlet to Servlet, Servlet to JSP, Servlet to HTML etc.) on the server and there is no return type
`public void include(ServletRequest request, ServletResponse response)throws ServletException, IOException`	This method includes the content of a resource in the response and there is no return type

INVOKING OTHER WEB RESOURCES

- Include is useful for static pages like banner content or header or footer information.
  - Example:Getting RequestDispatcher using **request** object.
  - Example:Getting RequestDispatcher using ServletContext.

```
RequestDispatcher rd = request.getRequestDispatcher(“MyServlet”);
rd.include(request, response);
```

```
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(“/MyServlet”);
rd.include(request, response);
```

INVOKING OTHER WEB RESOURCES

- Forward – transfers control to another resource
  - Should give the destination resource responsibility to reply to user.
  - Should **not** access ServletOutputStream or PrintWriter within Servlet before calling forward
    - Generate a **IllegalStateException**
  - Example:Getting RequestDispatcher using **request** object.
  - Example:Getting RequestDispatcher using **ServletContext**.

```
RequestDispatcher rd = request.getRequestDispatcher(“MyServlet”);
rd.forward(request, response);
```

```
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(“/MyServlet”);
rd.forward(request, response);
```

SUMMARY

- Servlet is based on Request-Response model
- Servlet component is managed by Servlet Container
- Servlet request object encapsulates client data
- Scope objects allow web components to share data across the method invocation