INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Programmation Orientée Objet

Sami Yangui, Ph.D.
A. Prof and CNRS LAAS Researcher

Lecture 6: Java Servlets

November 18, 2019

OUTLINE

- Introduction to Servlets
- Architecture
- Characteristics & Common Use Cases
- Communication & Requests Processing Procedures
- Servlets Lifecycle
- Writing Servlets / Servlets Template
- Session Tracking / Management

INTRODUCTION TO SERVLETS

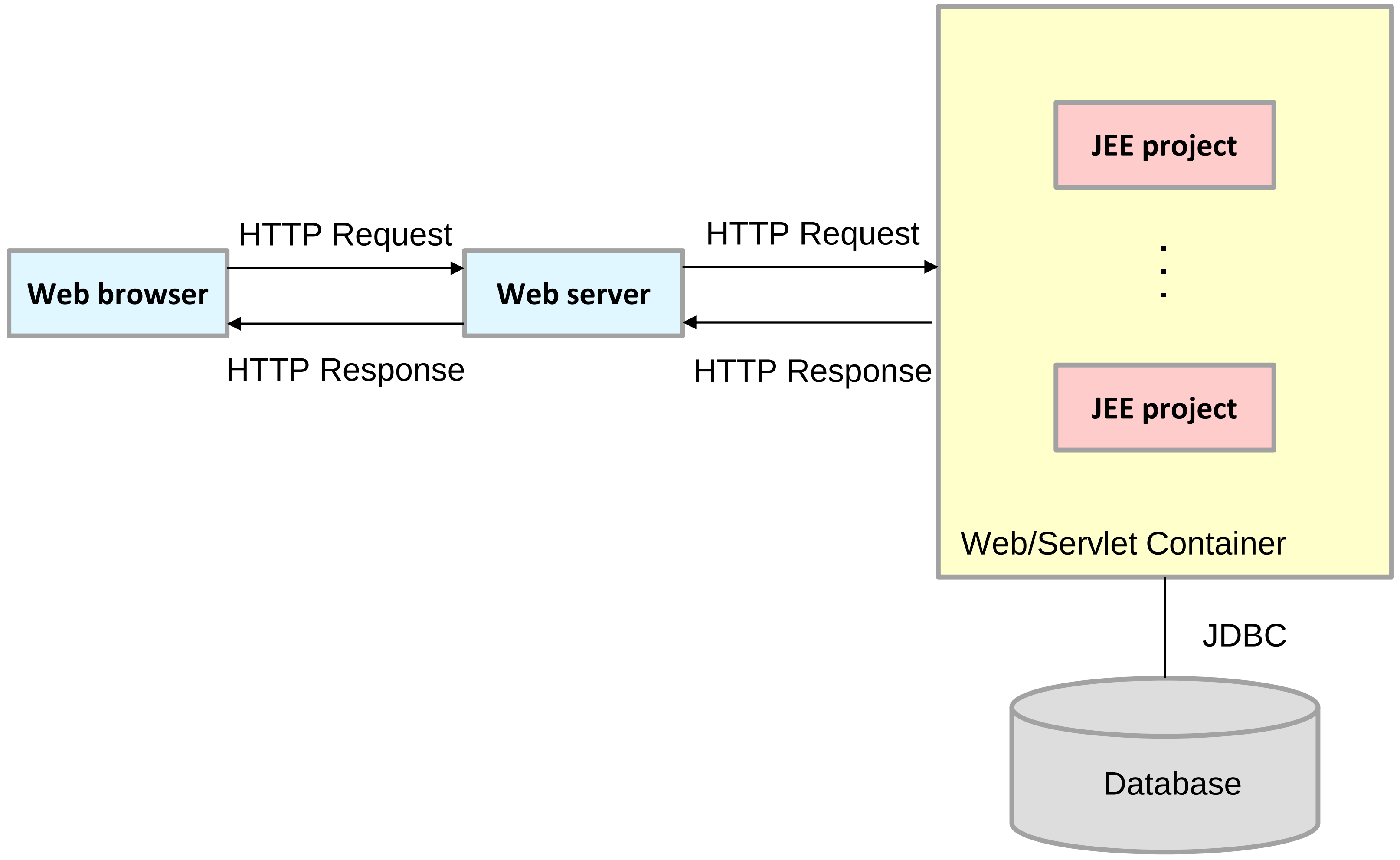
- Implements a java program that extends the capabilities of servers
  - Resides on a Web server
  - Loaded and executed by a Web browser
- Supports multi-threaded coding/execution
  - Each request launches a new thread
- Parses client inputs into a ***Request Variable***

INTRODUCTION TO SERVLETS (CONT.)

- Represents a set of classes that make up a Java Web project
  - E. g. Java Enterprise Edition (JEE)
- Produces dynamic output Web pages following a client request
  - E.g. Java Server Pages (JSP)
- Requires a compiler and a JVM machine in both sides
  - Server side: Servlet container
  - Client side: java-enabled browser

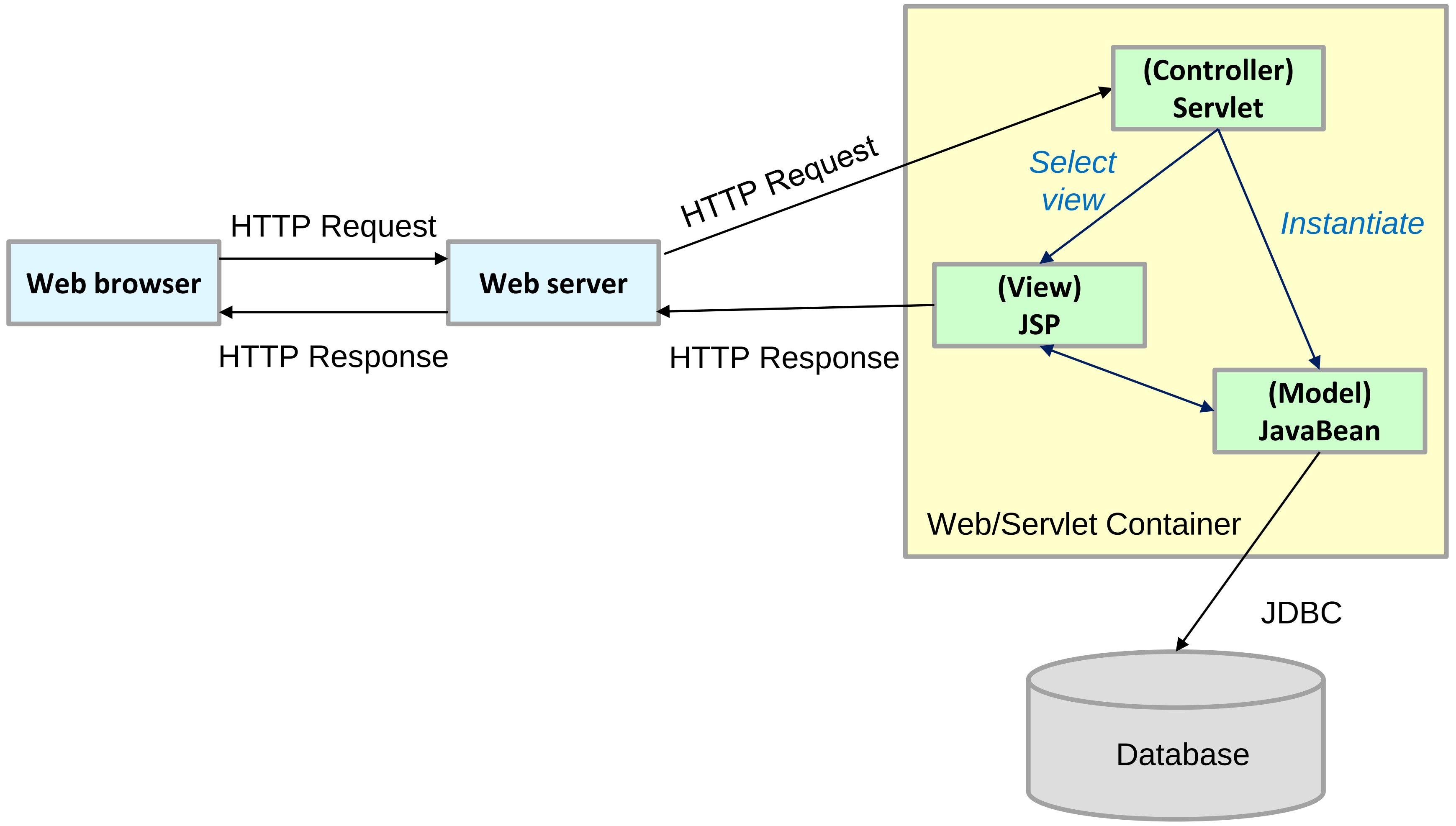
ACHITECTURE

- At-a-glance



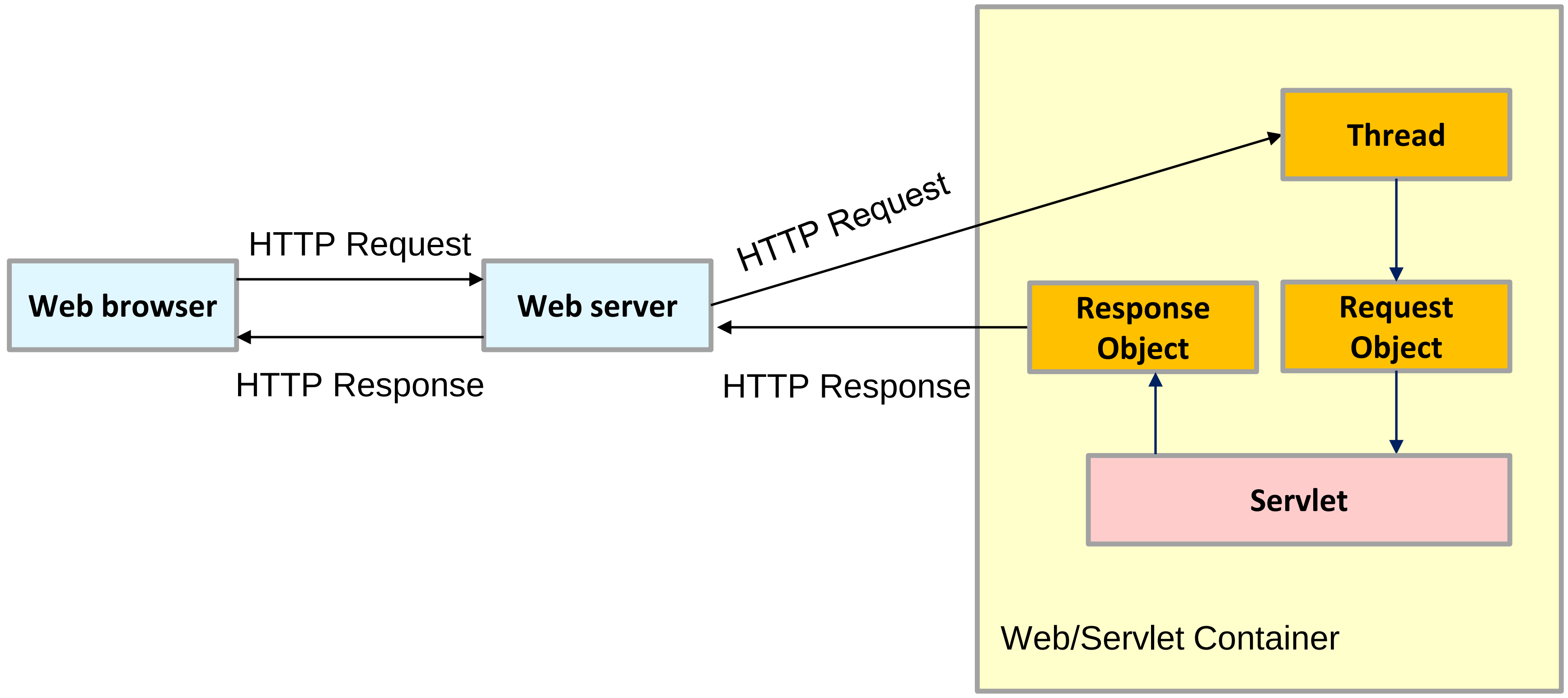
ACHITECTURE

- With focus on MVC pattern



ACHITECTURE

- With focus on HTTP transport protocol



CHARACTERISTICS

- Servlets are:
  - Platform-independent and extensible
    - Legacy scripts (e.g. CGI) are typically written in Perl/C, and are very much tied to a particular server platform
    - Servlets are written in Java, which can easily integrate with existing legacy systems through RMI, CORBA, etc.

CHARACTERISTICS (CONT.)

- Persistent and fast
  - Legacy scripts are transient (e.g. a CGI script is removed from memory after it is complete)
    - For each browser request, the Web server must spawn a new operating system process
  - Servers are loaded only once by the Web server and can maintain services between requests (particularly important for maintaining database connections)

CHARACTERISTICS (CONT.)

- Secure
  - The only way to invoke a Servlet from the outside world is through a Web server, which can be protected behind a firewall, encrypted, etc.

COMMON USE CASES

- Search engines
- E-commerce applications
- Shopping carts
- Product catalogs
- Intranet applications
- Groupware applications (e.g. bulletin boards, file sharing, etc.)

COMMUNICATION PROCEDURE

1. The client makes a request via HTTP
2. The listening Web server receives the requests and forwards it to the servlet
   – If the servlet has not been loaded, the Web server loads it into the JVM and executes it
3. The servlet receives the HTTP request and performs some type of processing
4. The servlet returns a response to the client (via the browser)

STEPS OF SERVLETS PROCESSING

1. Read any data sent by the server
   - E.g. capture data submitted by an HTML form
2. Look up any HTTP information
   - E.g. determine the browser version, host name of client, cookies, etc.
3. Generate the results
   - E.g. execute the code, connect to databases, connect to legacy applications, etc.

STEPS OF SERVLETS PROCESSING (CONT.)

4. Format the results
   – Generate HTML on the fly
5. Set the appropriate HTTP headers
   – Tell the browser the type of document being returned or set any cookies
6. Send the document back to the client

LIFECYCLE

- Create (Servlet Instantiation)
  - Loading the servlet class and creating a new instance
- Initialize (Servlet Initialization)
  - Initialize the servlet using the ***init()*** method
- Service (Servlet Processing)
  - Handling 0 or more client requests using the ***service()*** method

LIFECYCLE (CONT.)

- Destroy (<u>Servlet Death</u>)
  - Destroying the servlet using the ***destroy()*** method

- When HTTP calls for a servlet
  - **Not loaded: *Load()*, *Create()*, *Init()*, *Service()***
  - **Already loaded: *Service()***

WRITING SERVLETS

- Install a Web server capable of launching and managing servlet programs
  - E.g. Apache tomcat, Apache axis
- Install the ***javax.servlet*** package to enable programmers to write servlets
  - Ensure CLASSPATH is changed to correctly reference the ***javax.servlet*** package

WRITING SERVLETS (CONT.)

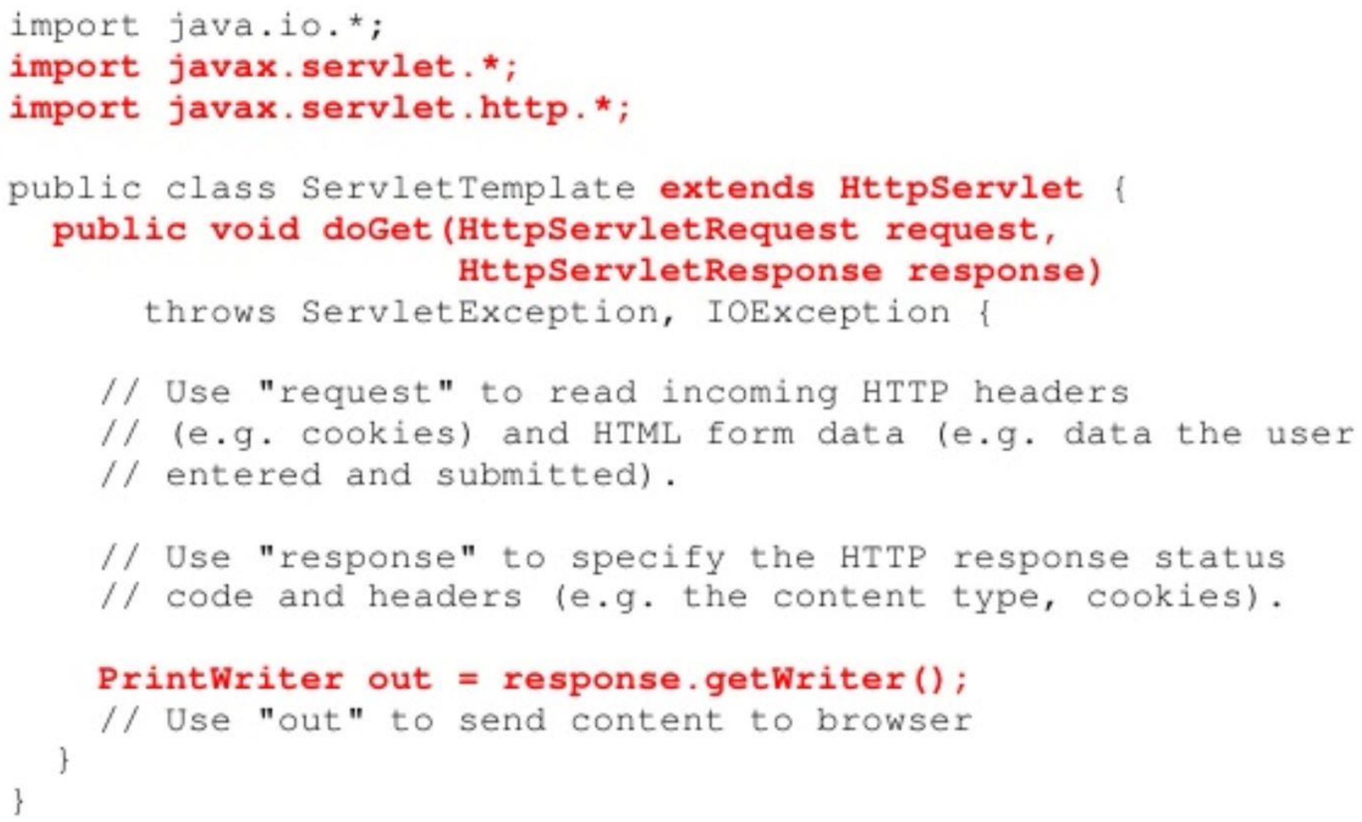
- Define a servlet by
  - Subclassing the ***HttpServlet*** class
  - Adding any necessary code to the ***init()*** method
  - Adding any necessary code to the ***doGet()*** and/or ***doPost()*** methods

WRITING SERVLETS (CONT.)

- Each HTTP Request type has a seperate handler function
  - GET ➡ doGet(HttpServletRequest, HttpServletResponse)
  - POST ➡ doPost(HttpServletRequest, HttpServletResponse)
  - PUT ➡ doPut(HttpServletRequest, HttpServletResponse)
  - DELETE ➡ doDelete(HttpServletRequest, HttpServletResponse)
  - TRACE ➡ doTrace(HttpServletRequest, HttpServletResponse)
  - OPTIONS ➡ doOptions(HttpServletRequest, HttpServletResponse)

WRITING SERVLETS (CONT.)

❑ **A servlet template**



```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

WRITING SERVLETS (CONT.)

❑ Important steps

- Import the servlet API
  - import ***javax.servlet.****;
  - import ***javax.servlet.http.****;
- Extend the ***HTTPServlet*** class
  - Full servlet API available at:

https://javaee.github.io/javaee-spec/javadocs/javax/servlet/Servlet.html

WRITING SERVLETS (CONT.)

❑ Important steps (Cont.)

- Override **at least one** of the request handlers
- Get an output stream to send the response back to the client
  - All output is channeled to the browser

WRITING SERVLETS (CONT.)

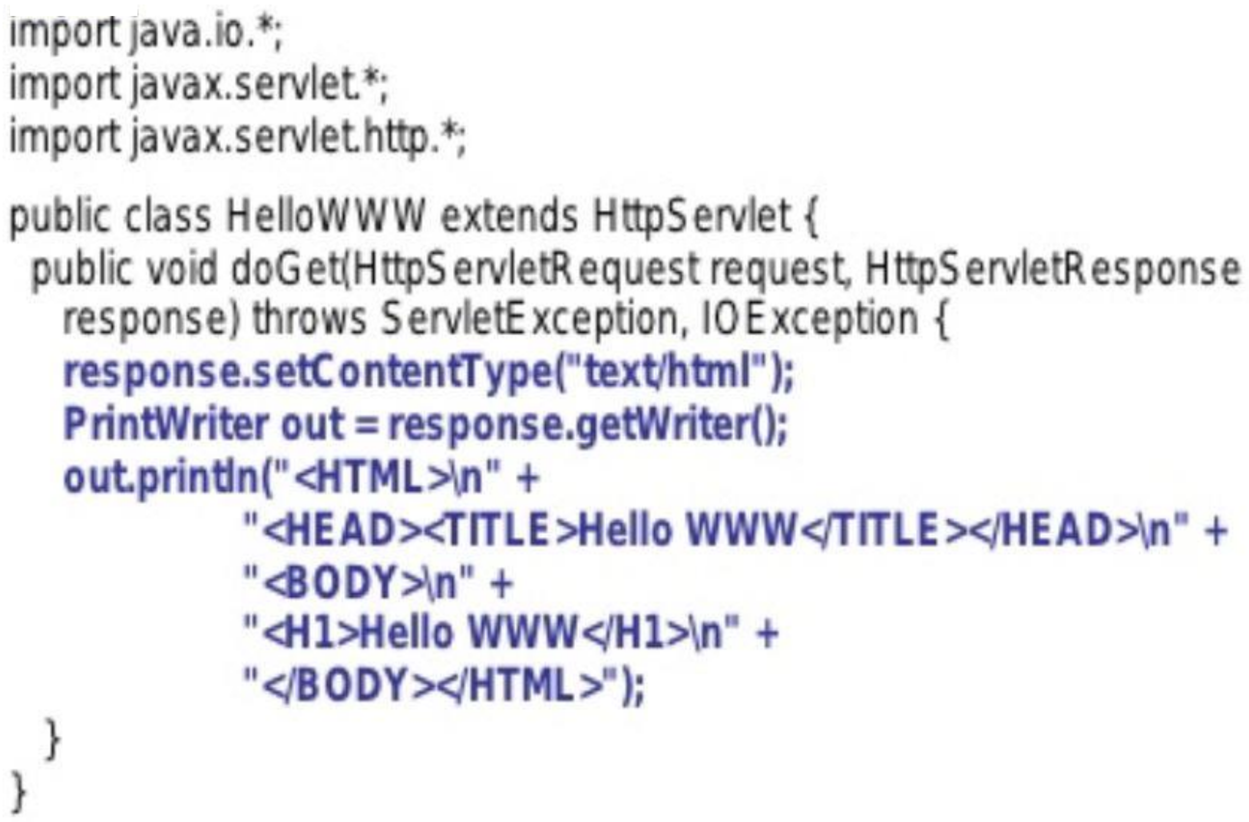
❑ Handlers parameters

- The handler methods each take two parameters:
  - ***HTTPServletRequest***: Encapsulates all information regarding the browser request
    - From data, client host name, HTTP request headers
  - ***HTTPServletResponse***: Encapsulates all information regarding the servlet response
    - HTTP return status, outgpoing cookies, HTML response

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

WRITING SERVLETS (CONT.)

❑ Hello world servlet



```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

WRITING SERVLETS (CONT.)

Single threaded example

- By default, servlet uses shared threads
  - Single instance of servlet shared by all requests
  - One thread created for each request
  - Class and instance variables are thread-unsafe; auto variables are thread-safe

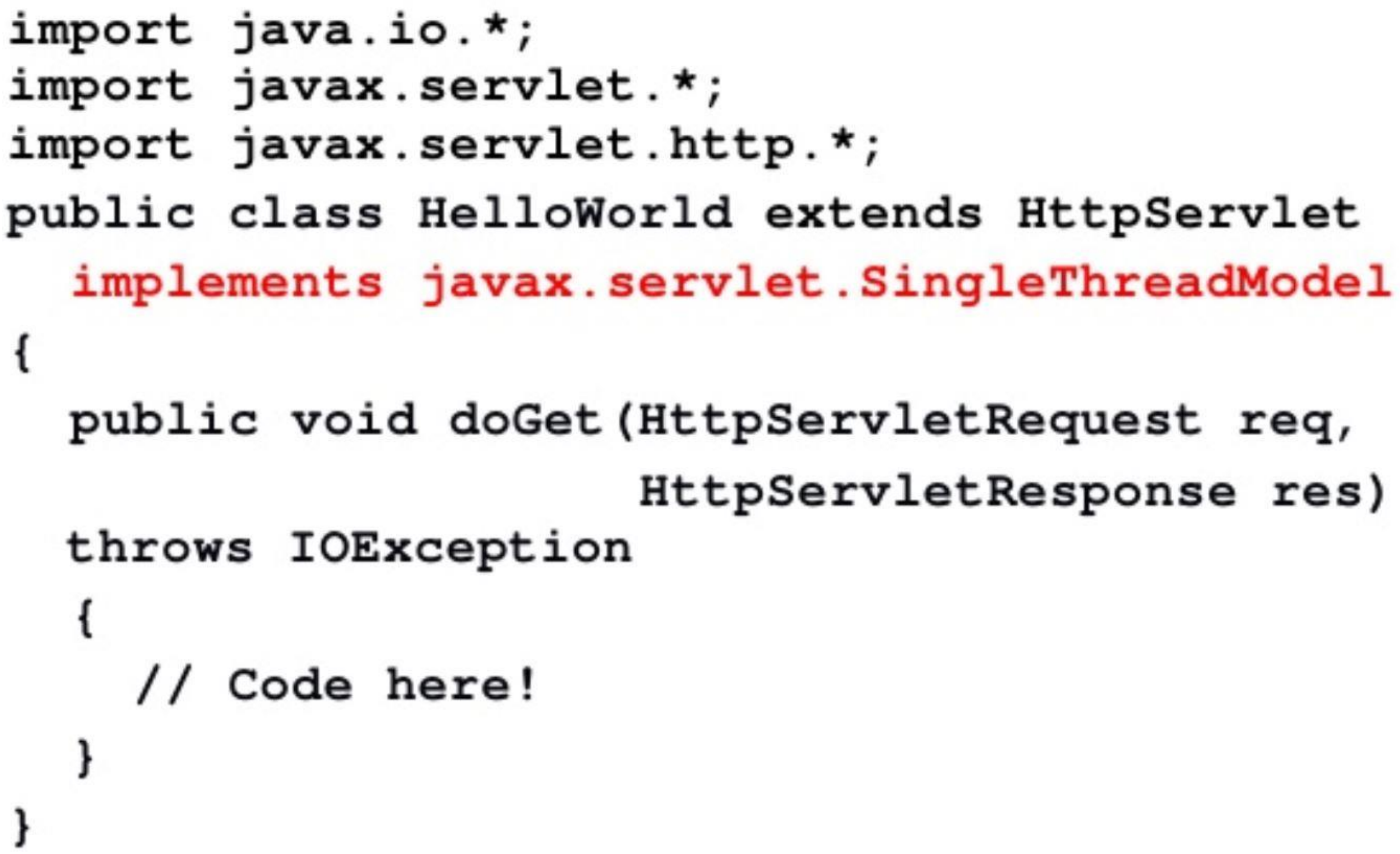
WRITING SERVLETS (CONT.)

❑ **Single threaded example (Cont.)**

- In some applications, single thread model is required:
  - New servlet for each request
  - Use of instance variables without synchronization

WRITING SERVLETS (CONT.)

Single threaded example (Cont.)



```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
  implements javax.servlet.SingleThreadModel
{
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
  throws IOException
  {
    // Code here!
  }
}
```

HTTPSERVLETREQUEST CLASS

- **Environment access (non-comprehensive list)**

getContentLength()	getServletPath()
getContentType()	getQueryString()
getProtocol()	getRemoteUser()

getServerName()
getServerPort()
	getPathInfo()
	getPathTranslated()

getRemoteHost()
getRemoteAddr()
	getHeader()

HTTPSERVLETREQUEST CLASS

❑ **Parameter access (non-comprehensive list)**

GetScheme	GetRequestedSessionId
GetInputStream	GetRequestURI
GetParameter GetParameterValues GetParameterNames	GetHeader GetIntHeader, GetDateHeader GetHeaderNames
GetReader	GetSession
GetCookies	GetContentType

HTTPRESPONSE CLASS

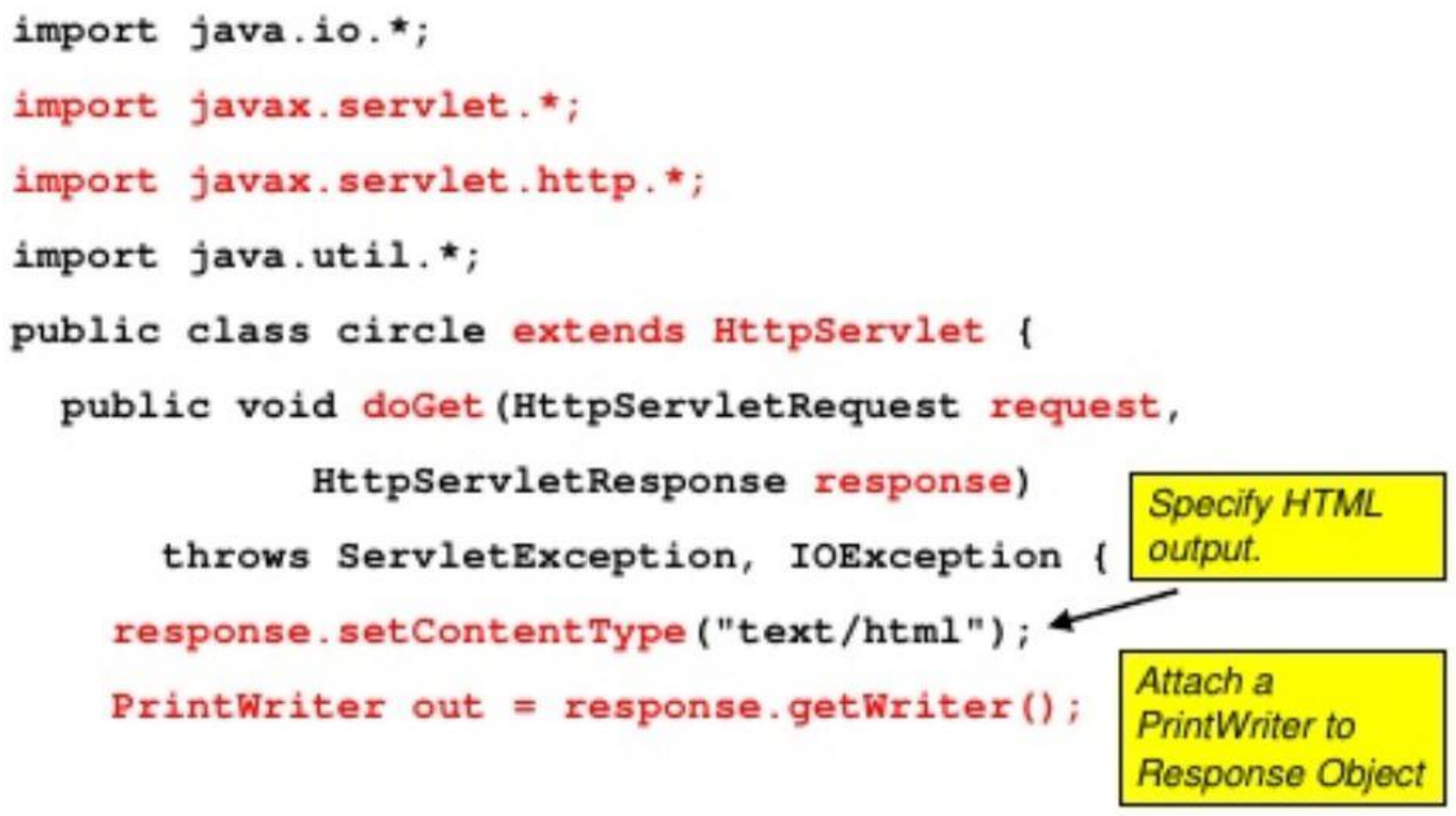
- **Methods (non-comprehensive list)**

getOutputStream()	sendError()
getWriter()	sendRedirect()
getCharacterEncoding()	setStatus()

setContentLength()
setContentType()
	setHeader()
	setIntHeader()

AddCookie()	encodeURL()

EXAMPLE - CIRCLE SERVLET

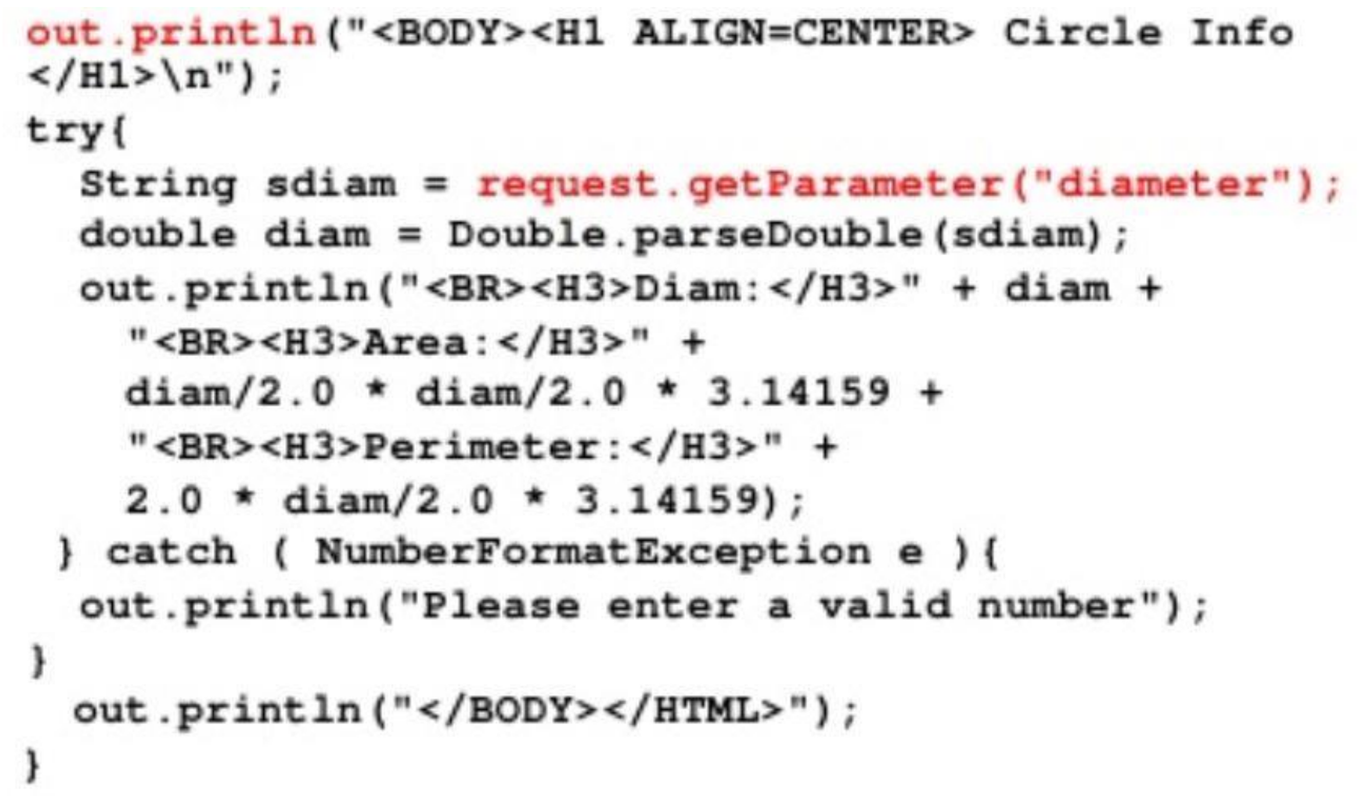


```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class circle extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
```

Specify HTML output.

Attach a PrintWriter to Response Object

EXAMPLE - CIRCLE SERVLET (CONT.)



```
out.println("<BODY><H1 ALIGN=CENTER> Circle Info
</H1>\n");
try{
   String sdiam = request.getParameter("diameter");
   double diam = Double.parseDouble(sdiam);
   out.println("<BR><H3>Diam:</H3>" + diam +
     "<BR><H3>Area:</H3>" +
     diam/2.0 * diam/2.0 * 3.14159 +
     "<BR><H3>Perimeter:</H3>" +
     2.0 * diam/2.0 * 3.14159);
 } catch ( NumberFormatException e ){
   out.println("Please enter a valid number");
}
  out.println("</BODY></HTML>");
}
```

SESSION TRACKING

❑ Context / Motivations

- Many applications need to maintain state accross a series of requests from the same user/browser
  - E.g. Clients at an on-line store and need to add items to their cart
  - E.g. Clients decide to proceed to checkout
- HTTP is a stateless protocol
  - Each time, a client talk to a browser, it opens a new connection
  - Server does not automatically maintains « conversational state » of a user

SESSION TRACKING (CONT.)

❑ **Existing Mechanisms**

- Cookies (the more used!)
- URL rewriting
- Hidden form fields

SESSION TRACKING (CONT.)

❑ Cookies

- A small amount of information sent by a servlet to a browser
  - Has a ***name***, a single ***value*** and optional ***attributes*** (name/value pairs)
- Saved by the browser, and later sent back to the server in subsequent requests
- Server uses cookie’s value to extract information about the session from some location on the server

SESSION TRACKING (CONT.)

❑ Cookies (Cont.)

- The ***HttpServletRequest*** class includes the ***getCookies()*** method
  - This returns an array of cookies , or null if there are not any
- Cookies can then be accessed using three methods
  - String ***getName()***
  - String ***getValue()***
  - String ***getVersion()***

SESSION TRACKING (CONT.)

❑ **Cookies (Cont.)**

- Cookies can be created using ***HttpServletResponse.addCookie()*** and the constructor
  - ***New Cookie(String name, String value);***
- Expiration can be set using
  - ***setMaxAge (int seconds)***

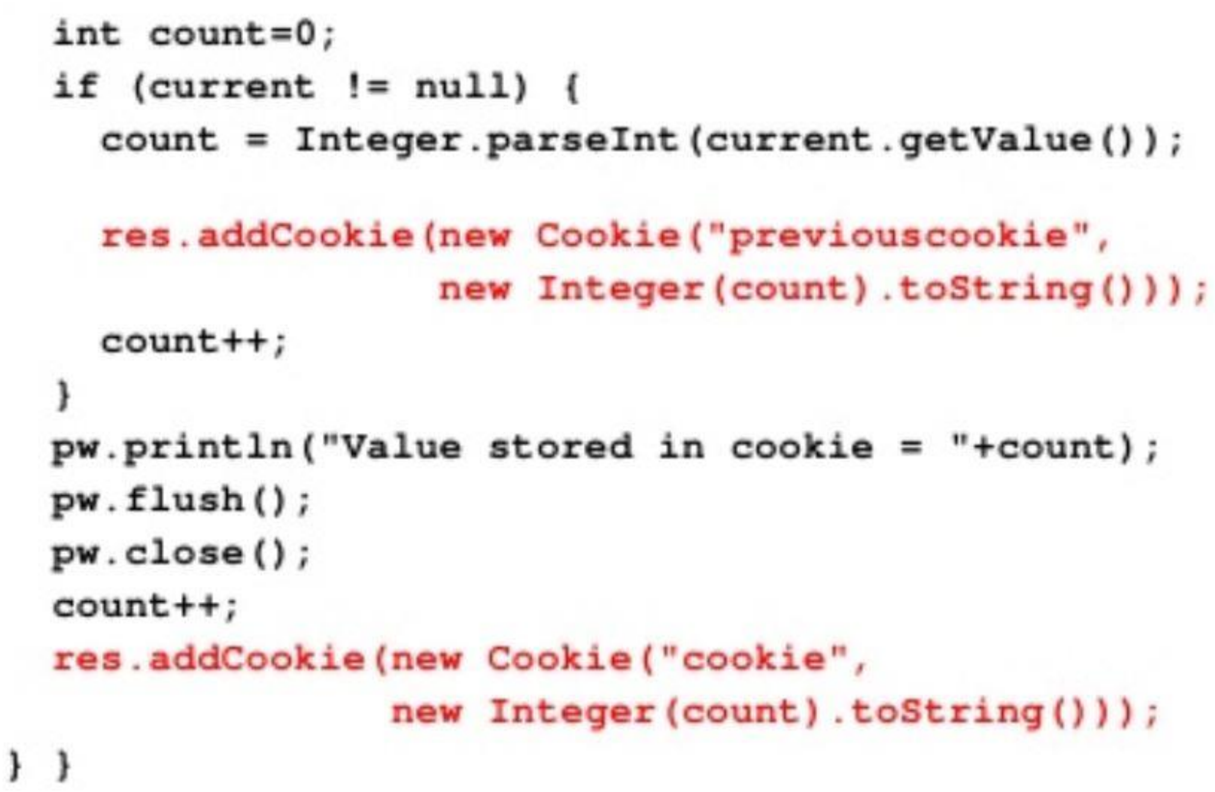
SESSION TRACKING (CONT.)

❑ Cookies (Cont.)

```
public class CookieTest extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res) throws IOException {
    OutputStream out = res.getOutputStream();
    PrintWriter pw = new PrintWriter(new BufferedWriter(new
                                   OutputStreamWriter(out)));
    Cookie[] cookies = req.getCookies();
    Cookie current = null;
    if (cookies != null) {
      for (int i=0; i < cookies.length; i++) {
        pw.println("name=" + cookies[i].getName());
        pw.println("value=" + cookies[i].getValue());
        pw.println("version=" + cookies[i].getVersion());
        if (cookies[i].getName().equals("cookie"))
        { current=cookies[i]; }
        pw.println();
      } }
```

SESSION TRACKING (CONT.)

- Cookies (Cont.)



```
  int count=0;
  if (current != null) {
    count = Integer.parseInt(current.getValue());

    res.addCookie(new Cookie("previouscookie",
                       new Integer(count).toString()));
    count++;
  }
  pw.println("Value stored in cookie = "+count);
  pw.flush();
  pw.close();
  count++;
  res.addCookie(new Cookie("cookie",
                     new Integer(count).toString()));
} }
```

SESSION TRACKING (CONT.)

Cookies (Cont.)

- Advantages
  - Very easy to implement
  - Highly customable
  - Persist across browser shut-downs
- Disadvantages
  - Users may turn off cookies for privacy or security reason
  - Not quite universal browser support (?)