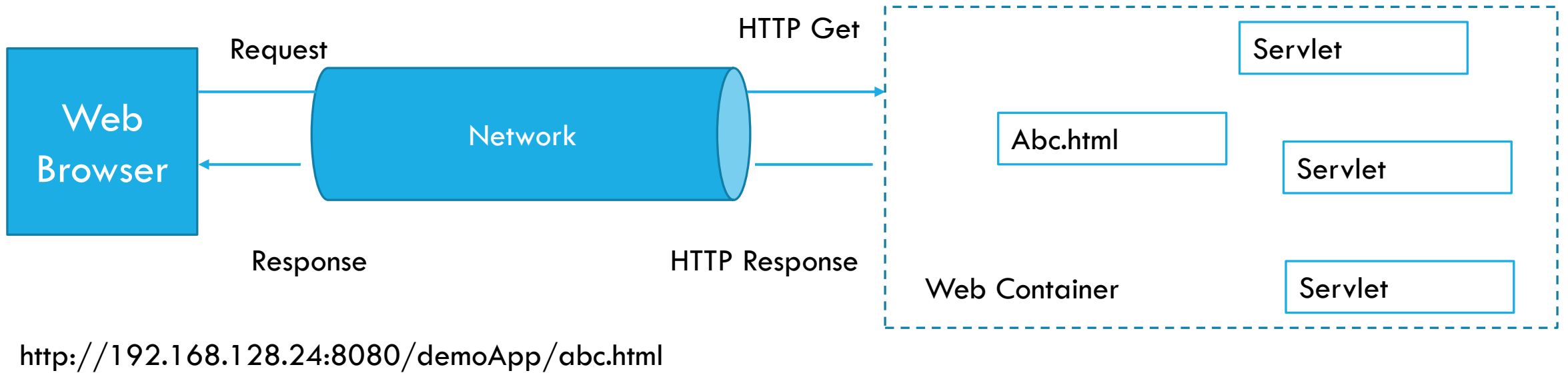




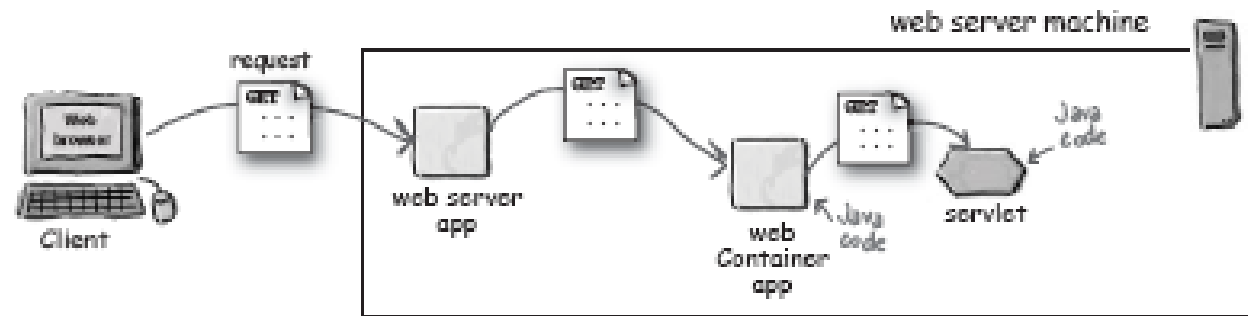
WEB APPLICATION DEVELOPMENT

Chandreyee Chowdhury

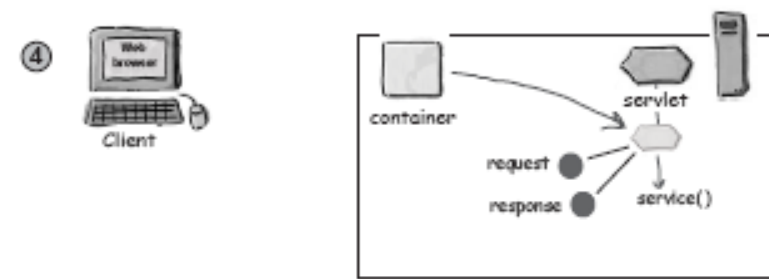
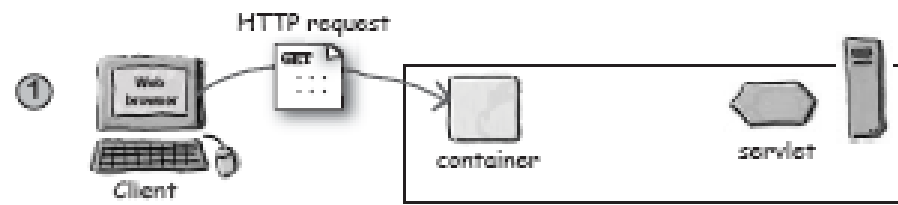
WEB CONTAINER



WEB SERVER VS WEB CONTAINER



<http://www.abc.com/home/index.html>



```
public class HelloServlet 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</TITLE></HEAD>\n" +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1>Hello World</H1>\n" +
            "</BODY></HTML>");
    }
}
```

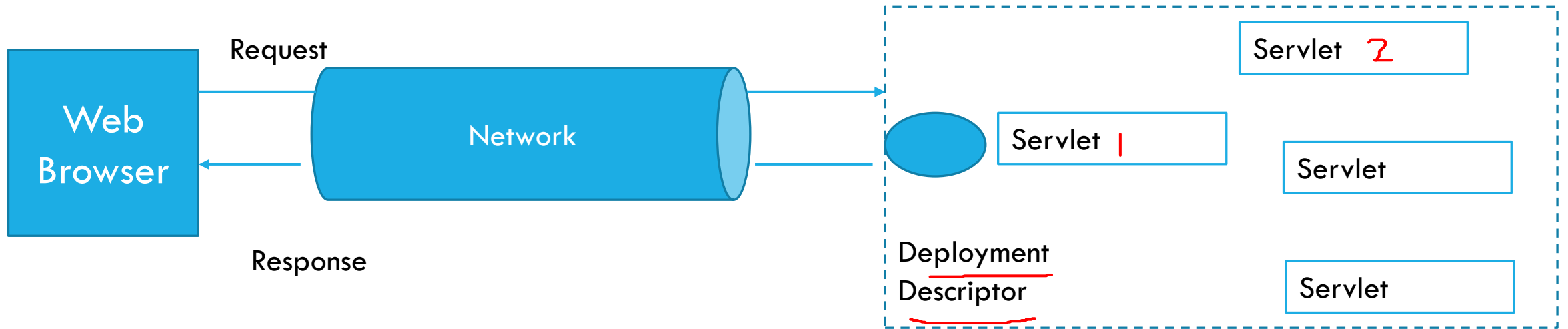
JSP

```
<HTML>  
<BODY>  
<%= new java.util.Date()%>  
</BODY>  
</HTML>
```

A MORE MEANINGFUL ONE

```
public class HelloServlet extends HttpServlet {  
    List<StudyMaterials> stList=new ArrayList<...> stList();  
    public void doGet(HttpServletRequest request, HttpServletResponse  
        response) throws ServletException, IOException {  
        String value=request.getParameter("key");  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        for(StudyMaterials st: this.stList)  
            if(value.equals...(....){  
                out.println("Title " + stList.getTitle() + " URL: " +  
                    stList.getURL());  
            }  
        }  
    }  
}
```

WEB CONTAINER



DEPLOYMENT DESCRIPTOR

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd" version="2.5">
```

```
<servlet>
  <servlet-name>Form1</servlet-name>
  <servlet-class>StudyMat.HelloServlet</servlet-class>
</servlet>
```

The <servlet> element tells the Container which class files belong to a particular web application.

```
<servlet-mapping>
  <servlet-name>Form1</servlet-name>
  <url-pattern>/store/home.do</url-pattern>
</servlet-mapping>
</web-app>
```

Think of the <servlet-mapping> element as what the Container uses at runtime when a request comes in, to ask, "which servlet should I invoke for this requested URL?"

Resultant URL

– http://hostname/webappName/MyAddress

Tomcat-specific

This directory name also represents the "context root" which Tomcat uses when resolving URLs.

tomcat

webapps

This part of the directory structure is required by Tomcat, and it must be directly inside the Tomcat home directory.

The name of the web app.

Part of the Servlets specification

WEB-INF

form.html

result.jsp

classes

lib

web.xml

This web.xml file MUST be in WEB-INF

Application-specific

com

example

web

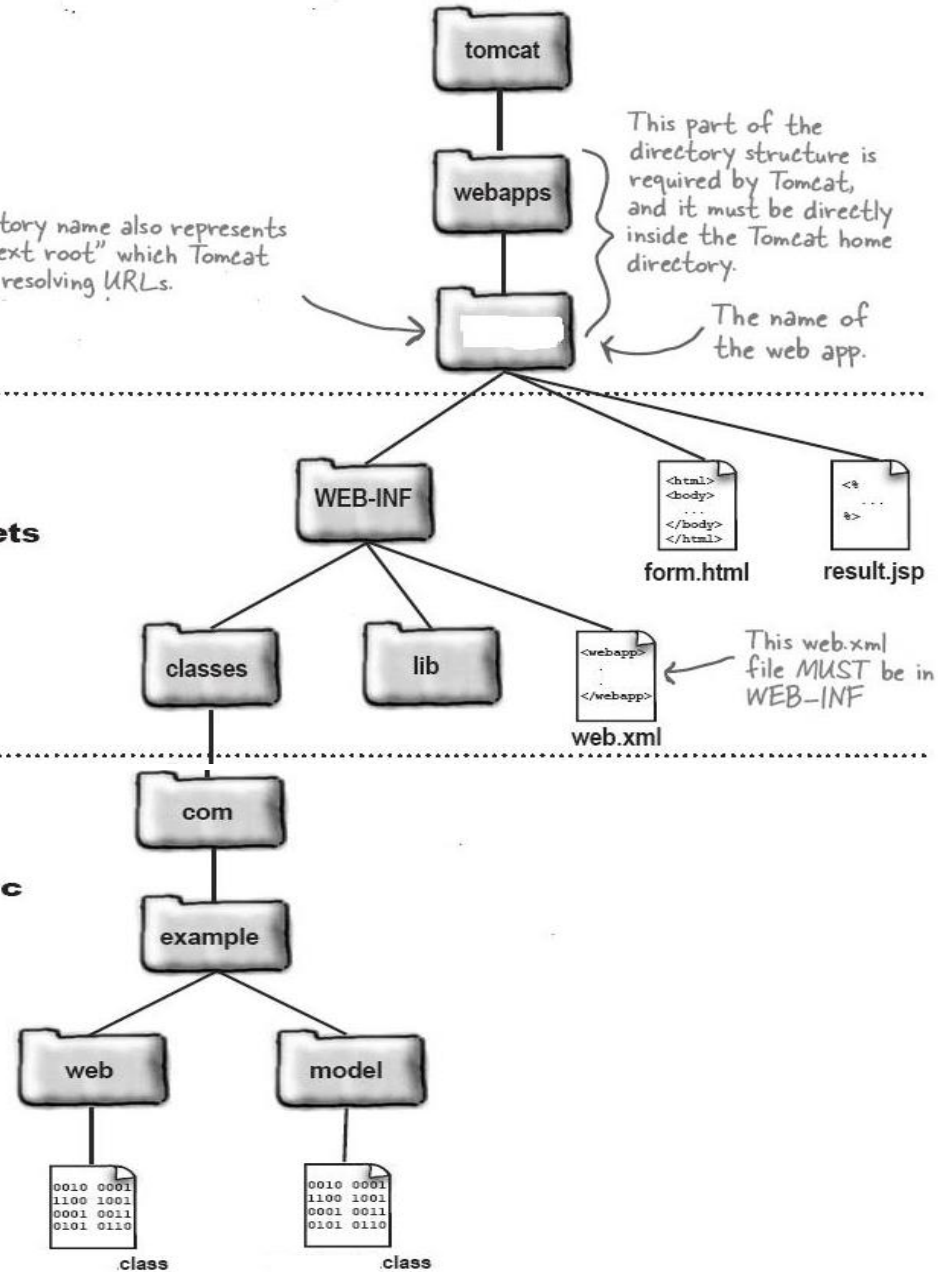
model

```
0010 0001
1100 1001
0001 0011
0101 0110
```

.class

```
0010 0001
1100 1001
0001 0011
0101 0110
```

.class

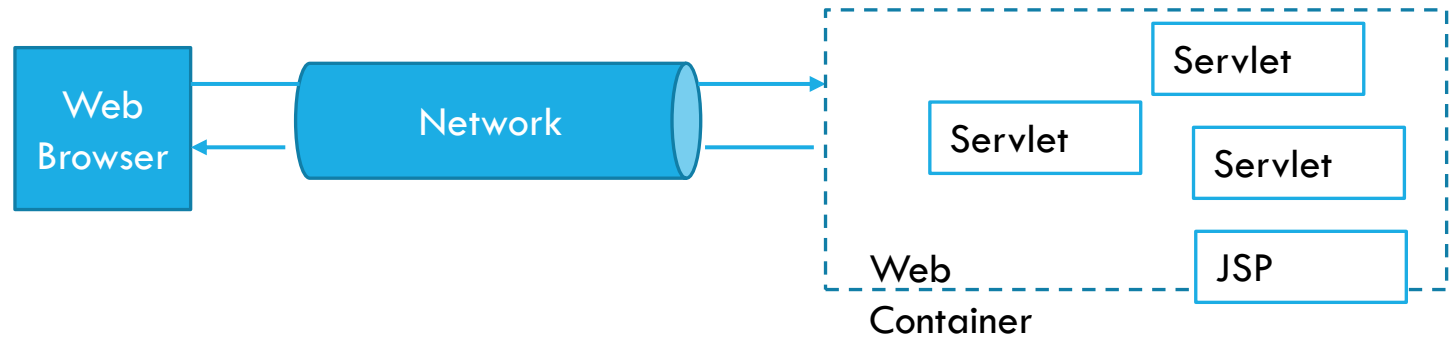


DEPLOYMENT DESCRIPTOR

The deployment descriptor (DD), provides a “declarative” mechanism for customizing your web applications without touching source code!

- Minimizes touching source code that has already been tested.
- Lets you fine-tune your app's capabilities, even if you don't *have the source code*.
- Lets you adapt your application to different resources (like databases), without having to recompile and test any code.
- Makes it easier for you to maintain dynamic security info like access control lists and security roles.
- Lets non-programmers modify and deploy your web applications

WEB CONTAINER



Communications support

- The container provides an easy way for your servlets to talk to web server. You don't have to build a `ServerSocket`, listen on a port, create streams, etc.
- The Container knows the protocol between the web server and itself,

Lifecycle Management

- It takes care of loading the classes, instantiating and initializing the servlets, invoking the servlet methods, and making servlet instances eligible for garbage collection

Multithreading Support

- The Container automatically creates a new Java thread for every servlet request it receives

Declarative Security

- With a Container, you get to use an XML deployment descriptor to configure (and modify) security without having to hard-code it into your servlet (or any other) class code
- You can manage and change your security without touching and recompiling your Java source files.

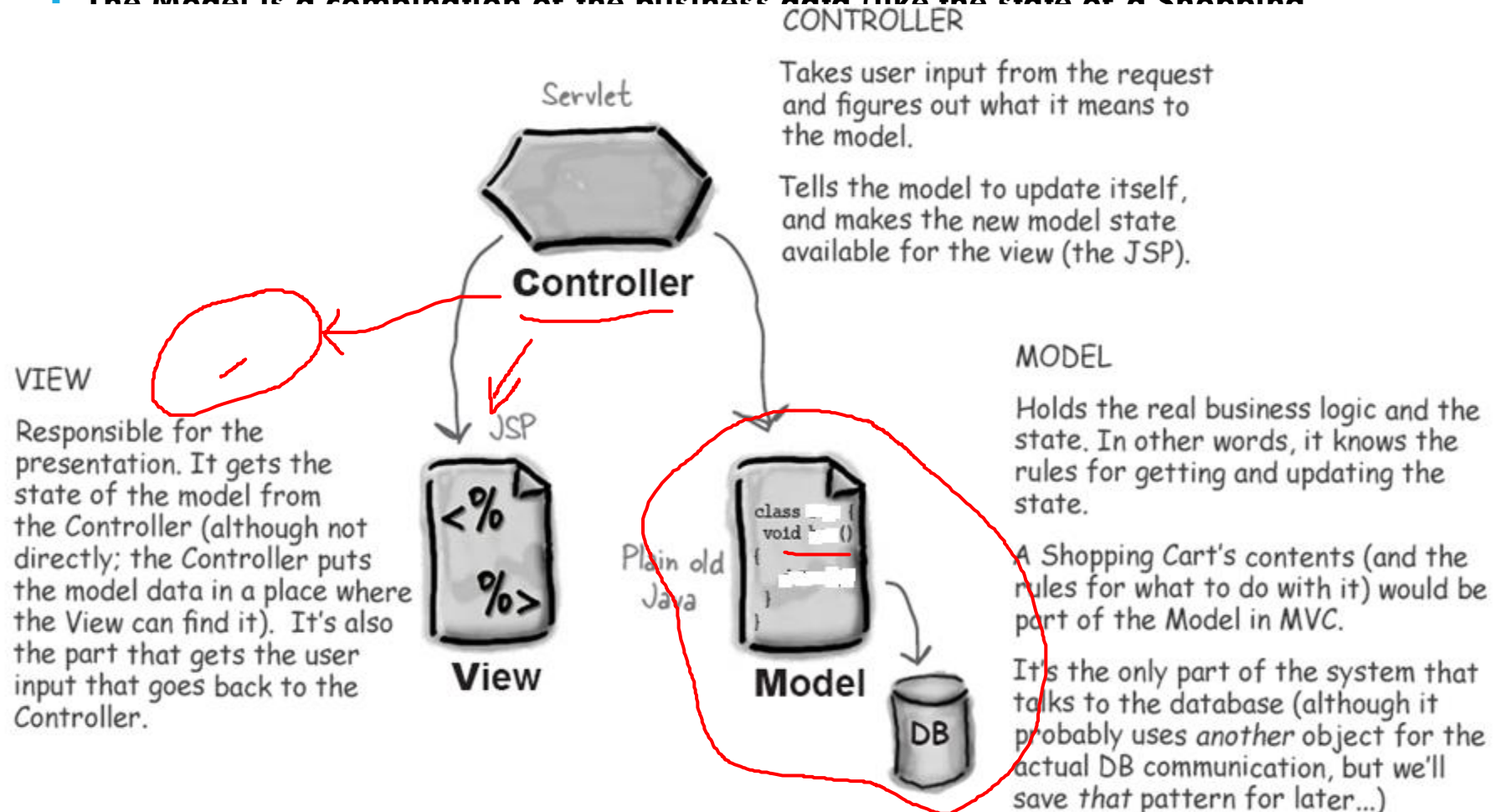
JSP Support

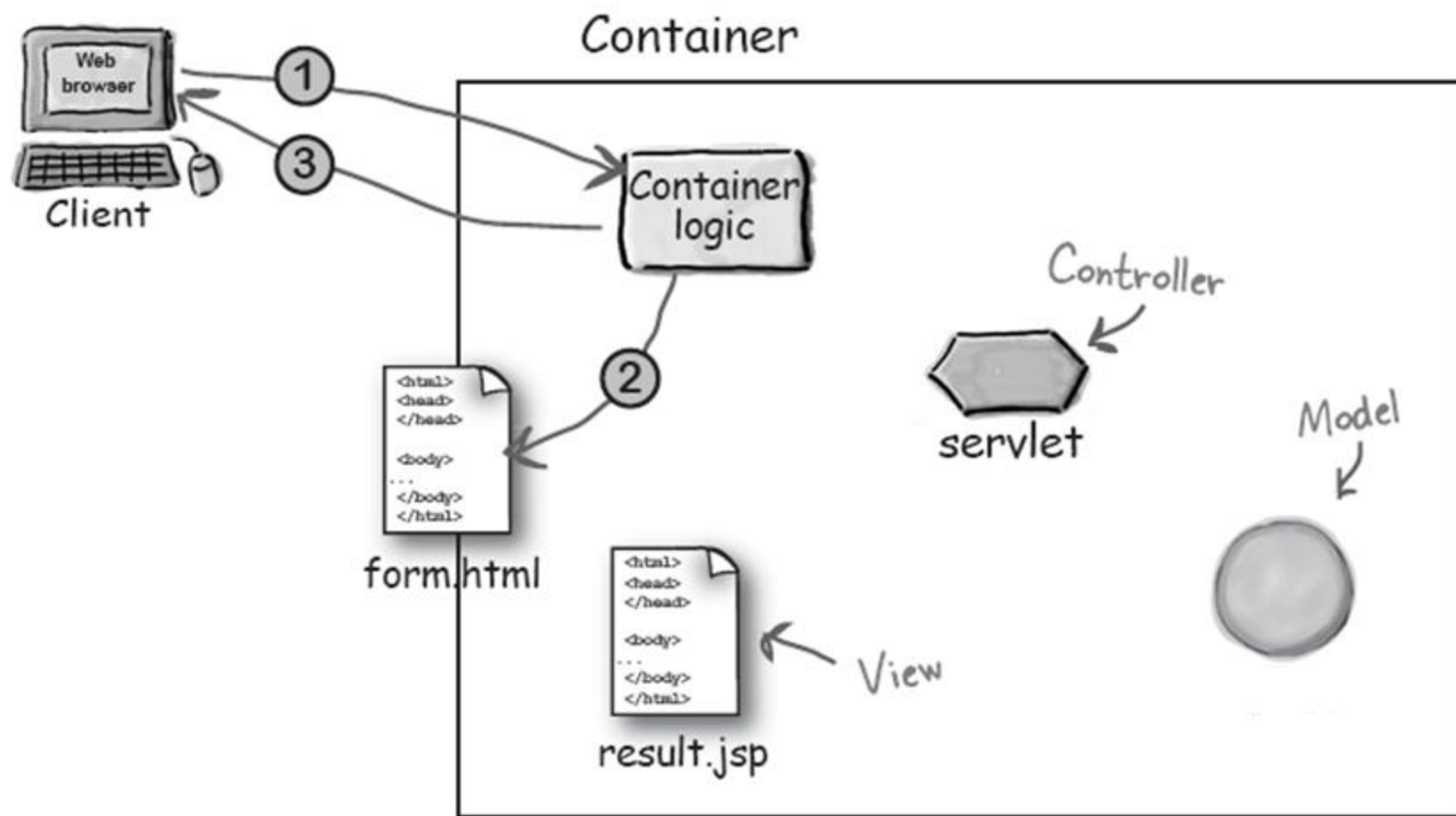
- The container takes care of translating a jsp file into java code

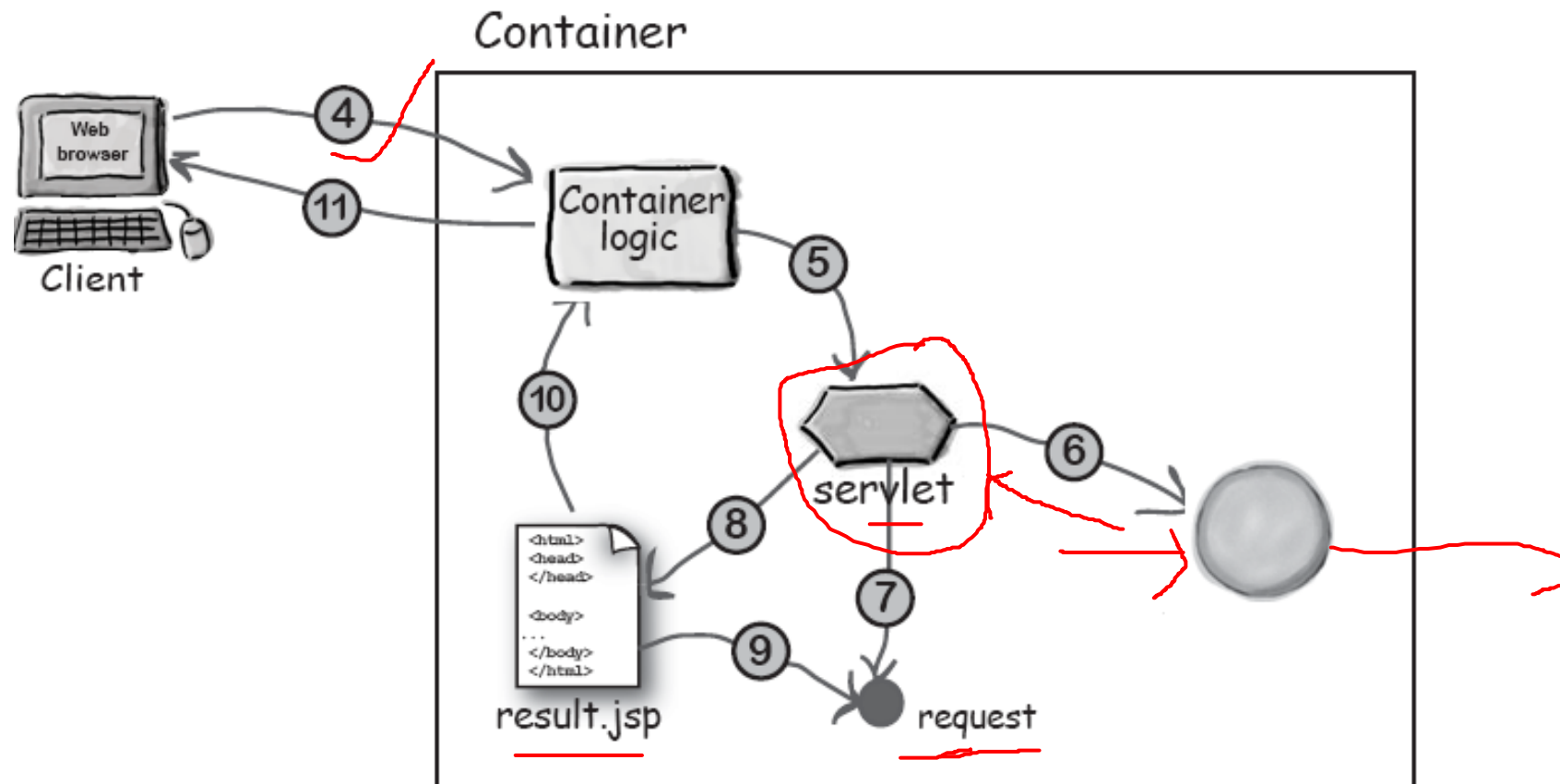
MVC

Model*View*Controller (MVC) takes the business logic out of the servlet, and puts it in a “Model”— a reusable plain old Java class.

- The Model is a combination of the business data (like the state of a Shopping

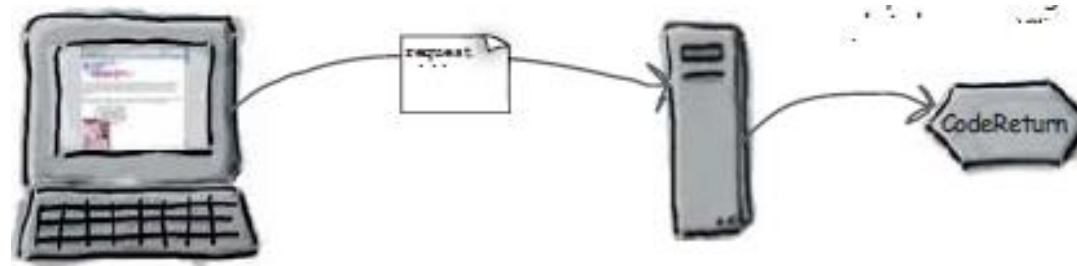




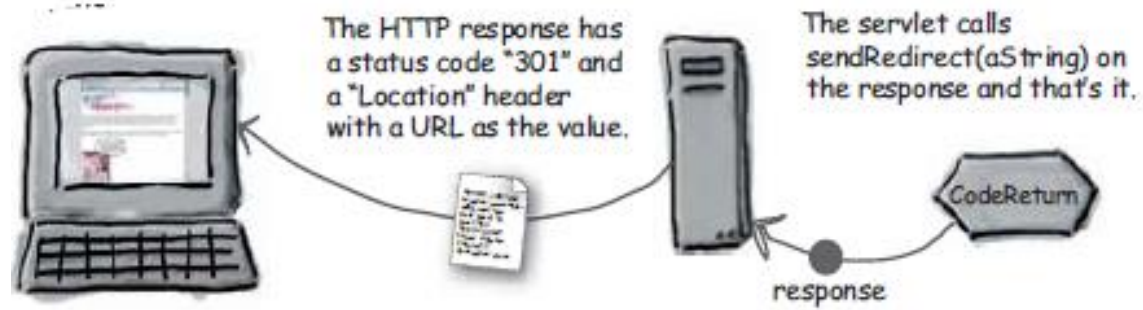


Servlet1 → servlet2 → jsp1 → jsp2

SEND REDIRECT



```
response.sendRedirect(http://www.abc.in);
```





Tomcat-specific

This directory name also represents the "context root" which Tomcat uses when resolving URLs.

This part of the directory structure is required by Tomcat, and it must be directly inside the Tomcat home directory.

The name of the web app.

Part of the Servlets specification

WEB-INF

```
<html>
<body>
...
</body>
</html>
```

form.html

```
<%
...
%>
```

result.jsp

classes

lib

```
<webapp>
...
</webapp>
```

web.xml

This web.xml file MUST be in WEB-INF

Application-specific

com

example

web

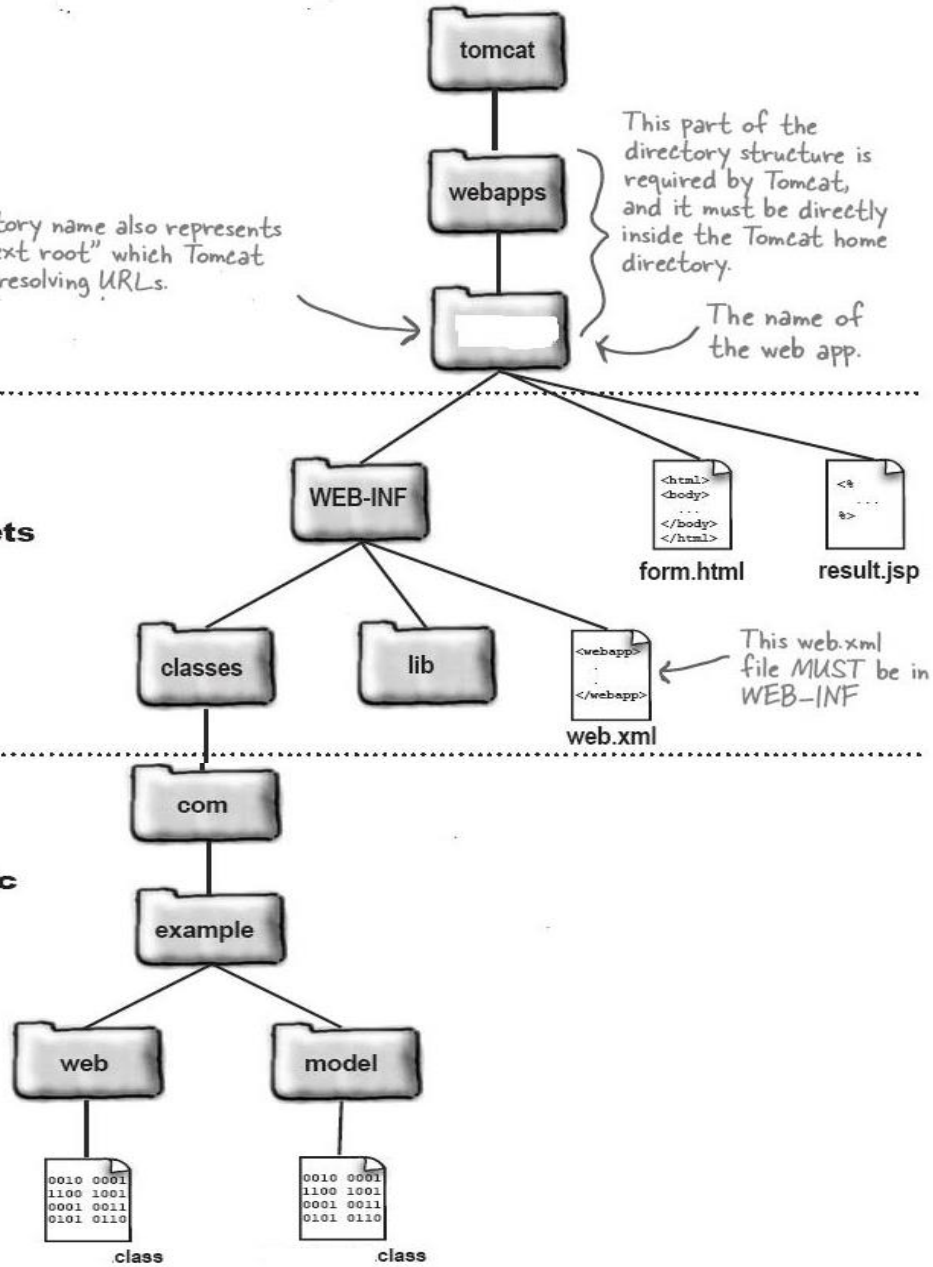
model

```
0010 0001
1100 1001
0001 0011
0101 0110
```

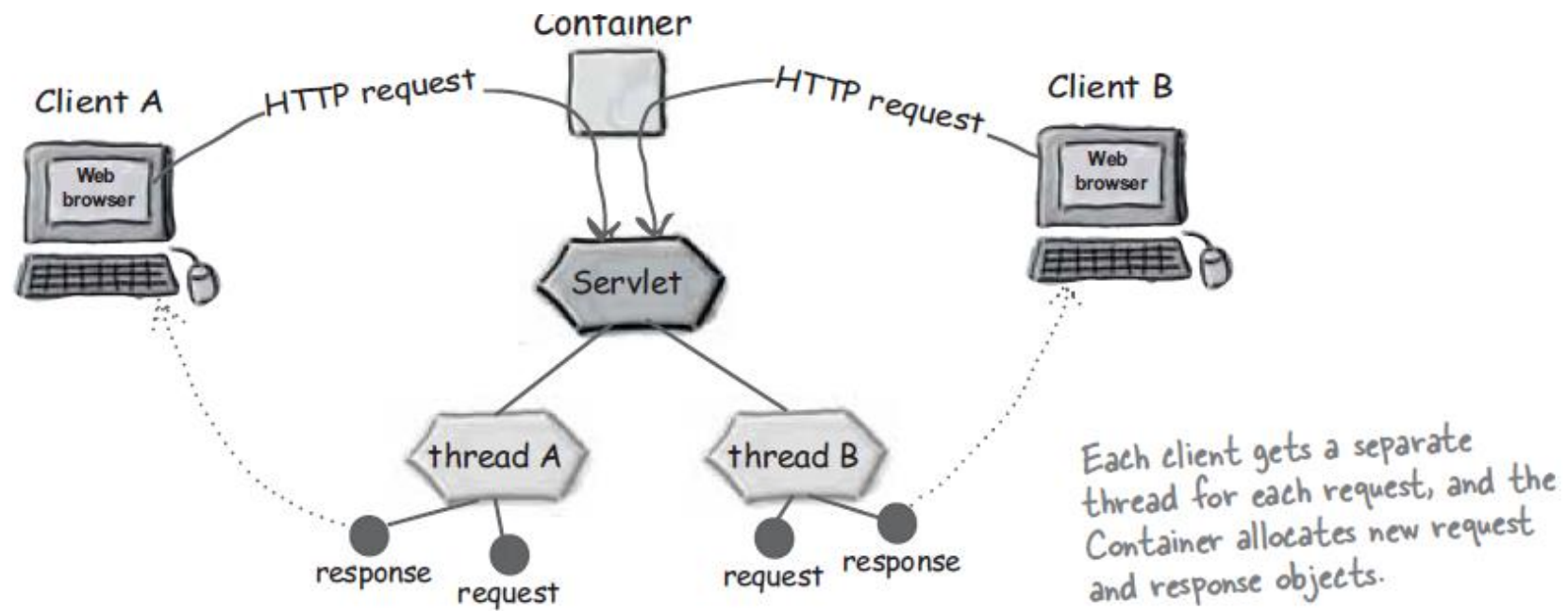
.class

```
0010 0001
1100 1001
0001 0011
0101 0110
```

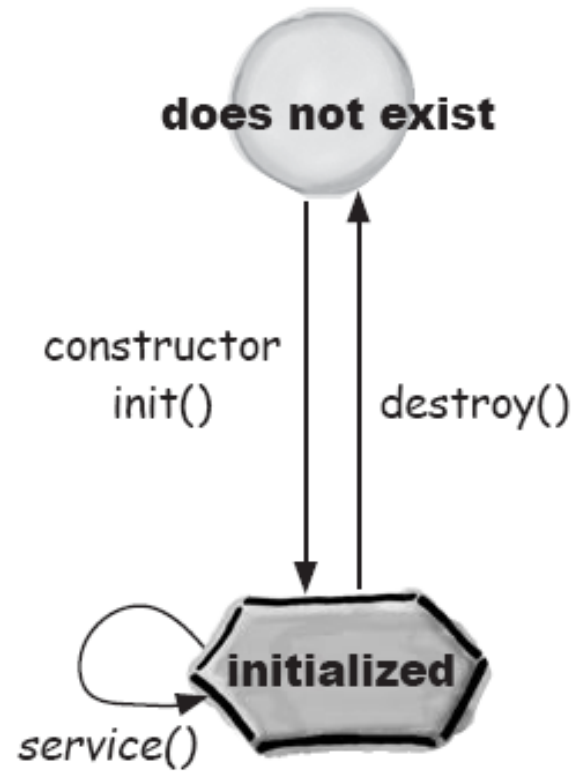
.class

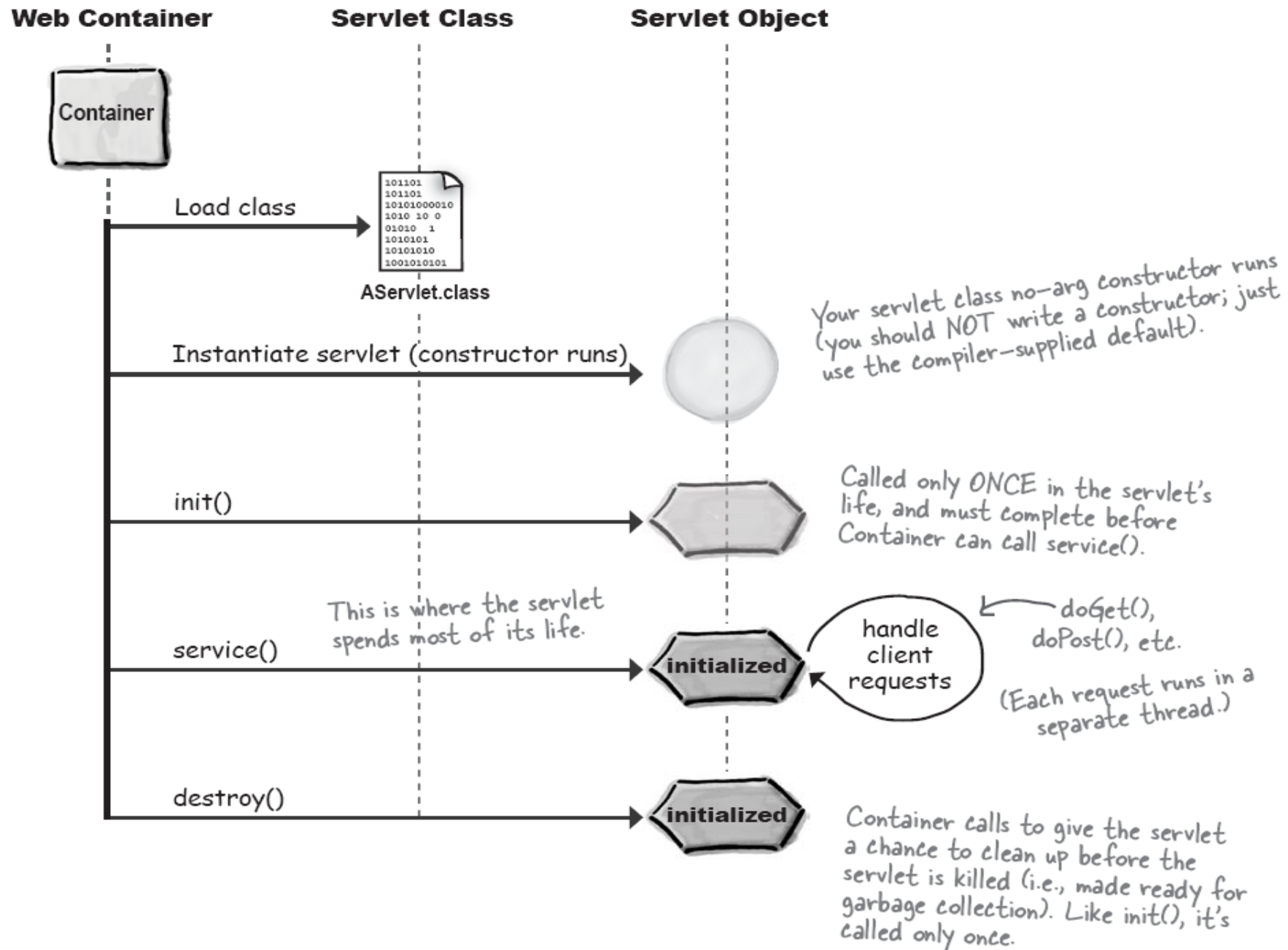


HANDLING MULTIPLE CLIENTS



SERVLET LIFECYCLE

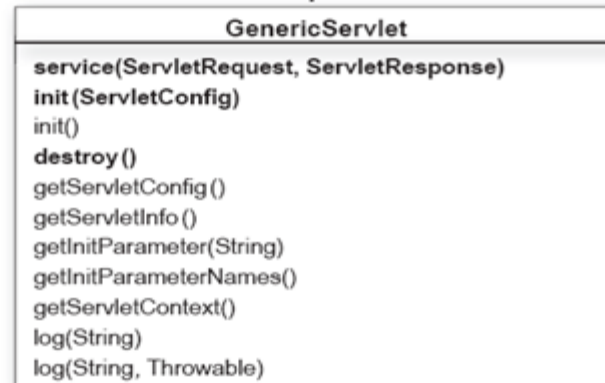






Servlet interface (javax.servlet.Servlet)

The Servlet interface says that all servlets have these five methods (the three in bold are lifecycle methods).



GenericServlet class (javax.servlet.GenericServlet)

GenericServlet is an abstract class that implements most of the basic servlet methods you'll need, including those from the Servlet interface. You will probably **NEVER** extend this class yourself. Most of your servlet's "servlet behavior" comes from this class.

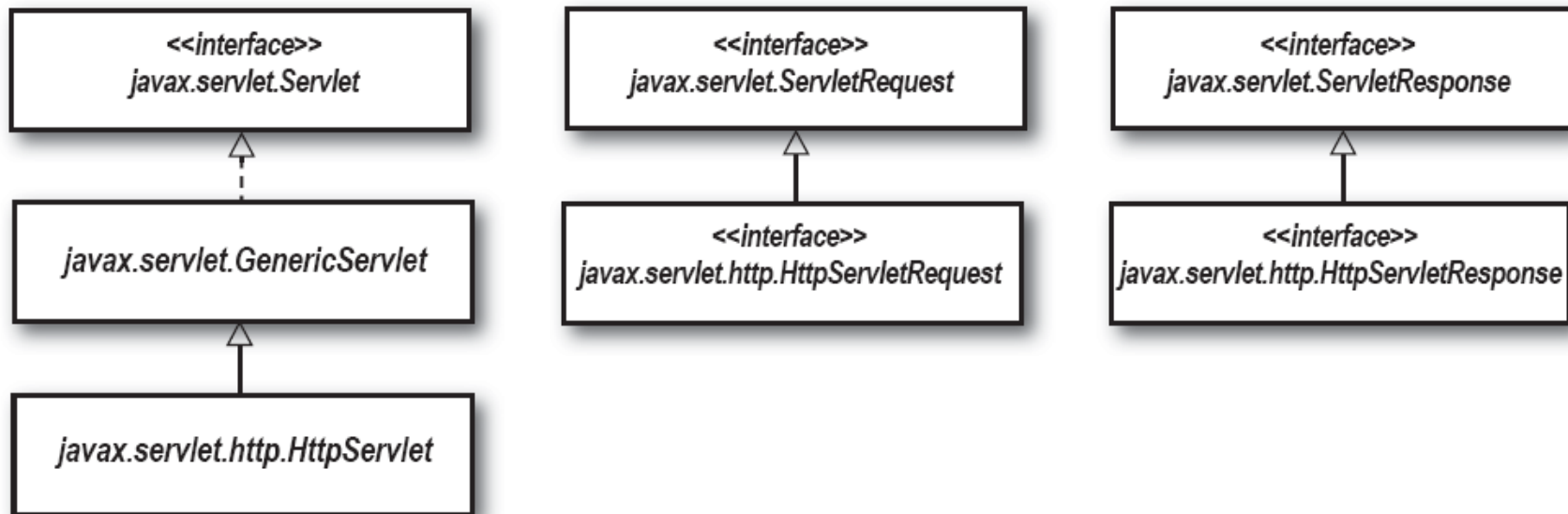


HttpServlet class (javax.servlet.http.HttpServlet)

HttpServlet (also an abstract class) implements the service() method to reflect the HTTPness of the servlet—the service() method doesn't take just ANY old servlet request and response, but an HTTP-specific request and response.

APIS

Key APIs



Compiling your servlet

- `Javac -classpath /.../tomcat/common/lib/servlet-api.jar; <servlet name>`

THREE BIG LIFECYCLE MOMENTS

Init()

- If you have initialization code (like getting a database connection or registering yourself with other objects), then you'll override the `init()` method in your servlet class

Service()

- You should NOT override the `service()` method. Your job is to override the `doGet()` and/or `doPost()` methods and let the `service()` implementation from `HTTPServlet` worry about calling the right one.

`doGet()` or `doPost()`

- Whichever one(s) you override tells the Container what you support

WHAT MAKES AN OBJECT A SERVLET

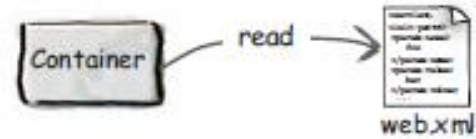
ServletConfig Object

- One ServletConfig object per servlet
- Use it to pass deploy-time information to the servlet (a database for example) that you don't want to hard-code into the servlet (servlet init parameters)
- Use it to access the ServletContext.
- Parameters are configured in the Deployment Descriptor
- Can also be annotated

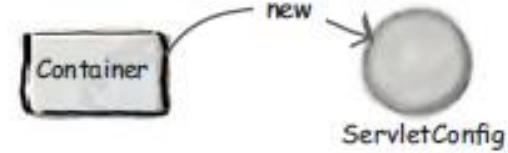
WHAT MAKES AN OBJECT A SERVLET

- ❑ One ServletContext per web app
- ❑ Use it to access web app *parameters* (also configured in the Deployment Descriptor).
- ❑ Use it as a kind of *application bulletin-board*, where you can put up messages (called *attributes*) that other parts of the application can access.
- ❑ Use it to get *server info*, including the name and version of the Container, and the version of the API that's supported.

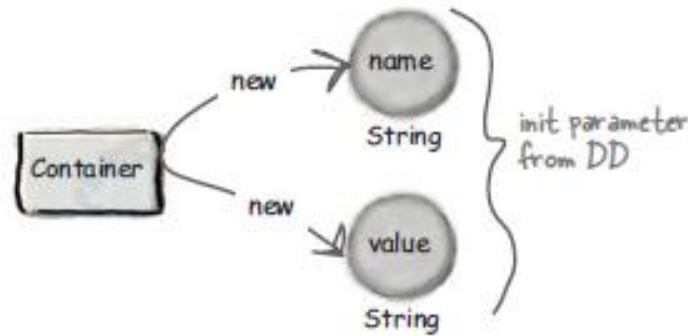
- ① Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



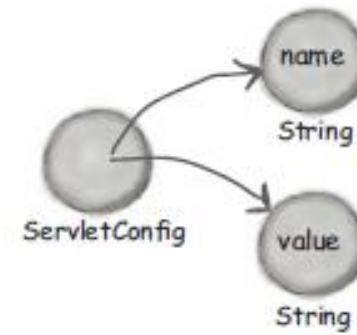
- ② Container creates a new ServletConfig instance for this servlet.



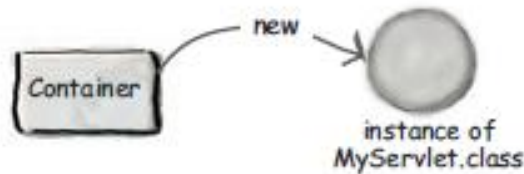
- ③ Container creates a name/value pair of Strings for each servlet init parameter. Assume we have only one.



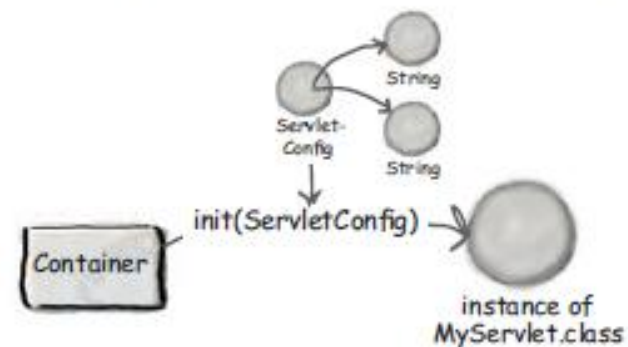
- ④ Container gives the ServletConfig references to the name/value init parameters.



- ⑤ Container creates a new instance of the servlet class.



- ⑥ Container calls the servlet's init() method, passing in the reference to the ServletConfig.



SERVLET CONFIG PARAMETERS

<servlet>

- **<servlet-name>InitTest</servlet-name>**
- **<servlet-class>moreservlets.InitServlet</servlet-class>**
- **<init-param>**
 - **<param-name>firstName</param-name>**
 - **<param-value>BCSE4</param-value>**
- **</init-param>**
- **<init-param>**
 - **<param-name>emailAddress</param-name>**
 - **<param-value>abc@jdvu.ac.in</param-value>**
- **</init-param>**

</servlet>

<servlet-mapping>

- **<servlet-name>InitTest</servlet-name>**
- **<url-pattern>/showInitValues</url-pattern>**

</servlet-mapping>

<context-param>

<param-name>firstName</param-name>
<param-value>BCSE4</param-value>

</context-param>

<context-param>

<param-name>emailAddress</param-name>
<param-value>abc@jdvu.ac.in</param-value>

</context-param>

READ CONFIG PARAMETER

```
public class InitServlet extends HttpServlet {
```

```
▪ private String firstName, emailAddress;  
▪ public void init() {  
  ▪ ServletConfig config = getServletConfig();  
  ▪ firstName = config.getInitParameter("firstName");  
  ▪ if (firstName == null) {  
    ▪ firstName = "Missing first name";  
  ▪ }  
  ▪ emailAddress = config.getInitParameter("emailAddress");  
  ▪ if (emailAddress == null) {  
    ▪ emailAddress = "Missing email address";  
  ▪ }  
}
```

ATTRIBUTES

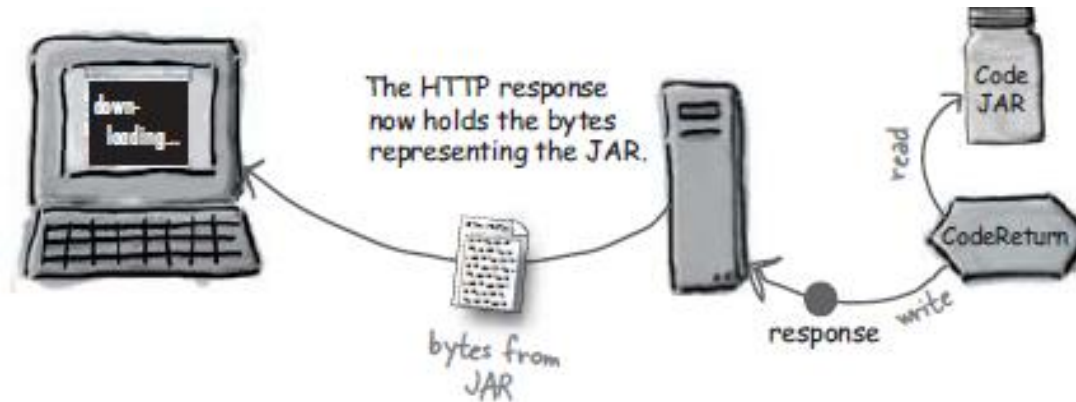
An attribute is a name/value pair in a map instance variable

An attribute is either bound to a ServletContext, HttpServletRequest or an HttpSession object

There is no servlet specific attribute

Return type of an attribute is an object

DOWNLOAD A JAR



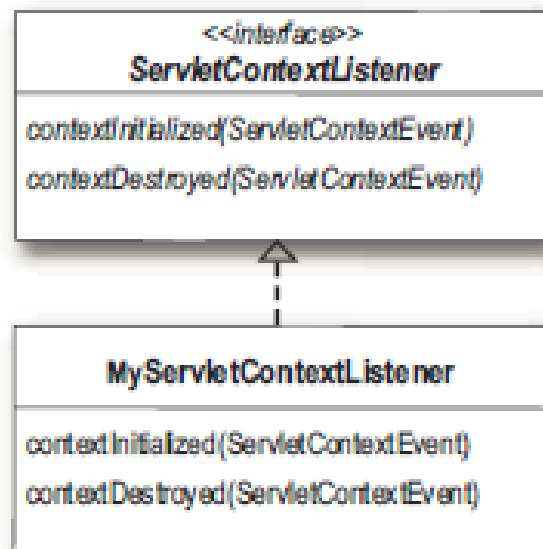
```
public class {  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws IOException, ServletException {  
  
        response.setContentType("application/jar");  
  
        ServletContext ctx = getServletContext();  
        InputStream is = ctx.getResourceAsStream("/bookCode.jar");  
  
        int read = 0;  
        byte[] bytes = new byte[1024];  
  
        OutputStream os = response.getOutputStream();  
        while ((read = is.read(bytes)) != -1) {  
            os.write(bytes, 0, read);  
        }  
        os.flush();  
        os.close();  
    }  
}
```

We want the browser to recognize that this is a JAR, not HTML, so we set the content type to "application/jar".

This just says, "give me an input stream for the resource named bookCode.jar".

Here's the key part, but it's just plain old I/O!! Nothing special, just read the JAR bytes, then write the bytes to the output stream that we get from the response object.

① Create a listener class



To listen for `ServletContext` events, write a listener class that implements `ServletContextListener`, put it in your `WEB-INF/classes` directory, and tell the Container by putting a `<listener>` element in the `Deployment Descriptor`.

② Put the class in WEB-INF/classes



(This isn't the *ONLY* place it can go... `WEB-INF/classes` is one of several places the Container can look for classes. We'll cover the others in the `Deployment` chapter.)

CONTEXT LISTENERS

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app ...>
  <context-param>
    <param-name>connectionString</param-name>
    <param-value>jdbc:myDriver:myDatabase</param-value>
  </context-param>
  <listener>
    <listener-class> /ContextListenerJDBC</ listener-class>
  </listener>
</web-app>
```

SETTING CONTEXT ATTRIBUTE

A ServletContextListener class:

```
import javax.servlet.*;

public class MyServletContextListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent event) {
        //code to initialize the database connection
        //and store it as a context attribute
    }

    public void contextDestroyed(ServletContextEvent event) {
        //code to close the database connection
    }
}
```

ServletContextListener is in
javax.servlet package.

A context listener
is simple: implement
ServletContextListener.

These are the two notifications
you get. Both give you a
ServletContextEvent

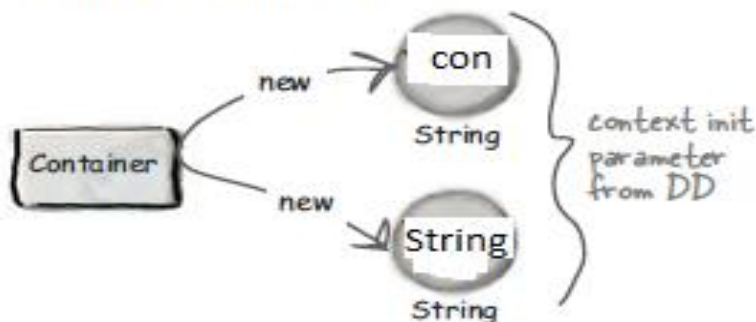
- ① Container reads the Deployment Descriptor for this app, including the `<listener>` and `<context-param>` elements.



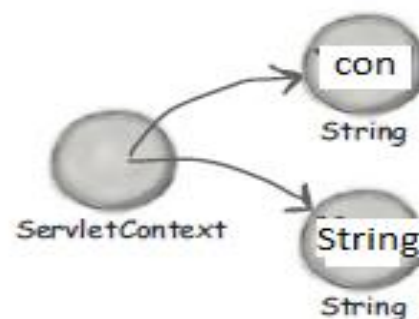
- ② Container creates a new `ServletContext` for this application, that all parts of the app will share.



- ③ Container creates a name/value pair of Strings for each context init parameter. Assume we have only one.



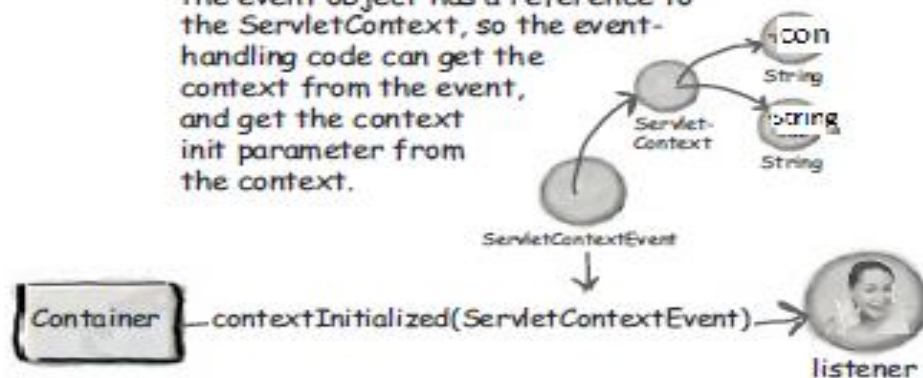
- ④ Container gives the `ServletContext` references to the name/value parameters.



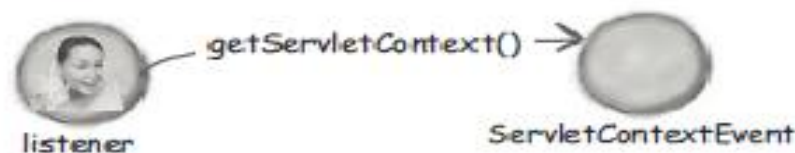
- ⑤ Container creates a new instance of the `MyServletContextListener` class.



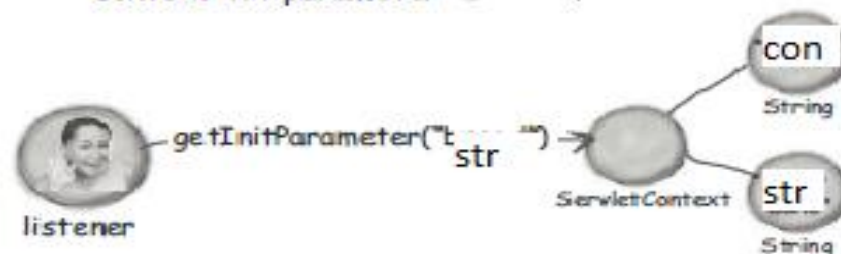
- ⑥ Container calls the listener's `contextInitialized()` method, passing in a new `ServletContextEvent`. The event object has a reference to the `ServletContext`, so the event-handling code can get the context from the event, and get the context init parameter from the context.



- 7 Listener asks `ServletContextEvent` for a reference to the `ServletContext`.



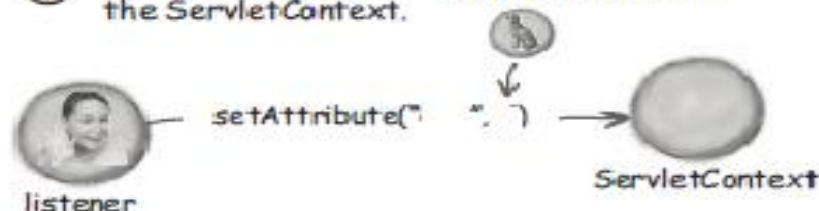
- 8 Listener asks `ServletContext` for the `context` init parameter



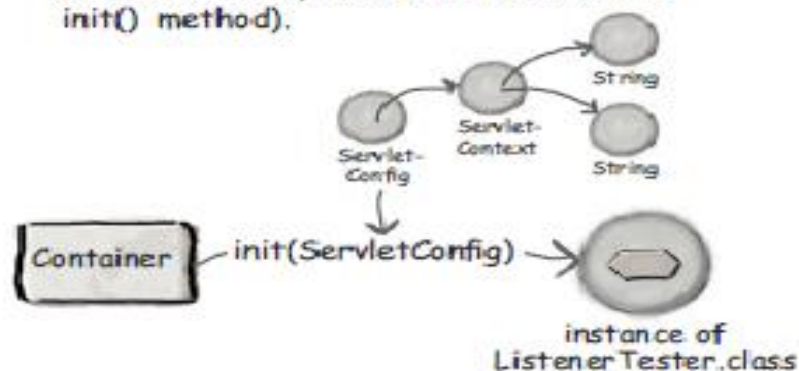
- 9 Listener uses the `init` parameter to construct a new `...` object.



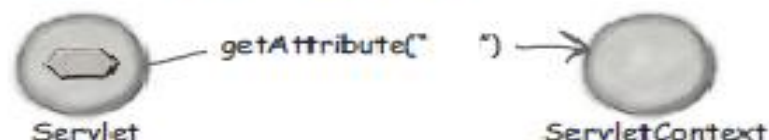
- 10 Listener sets the `...` as an attribute in the `ServletContext`.



- 11 Container makes a new `Servlet` (i.e., makes a new `ServletConfig` with init parameters, gives the `ServletConfig` a reference to the `ServletContext`, then calls the `Servlet`'s `init()` method).



- 12 `Servlet` gets a request, and asks the `ServletContext` for the attribute



- 13 `Servlet` calls `get...` on the `...` (and prints that to the `HttpServletResponse`).



CONTEXT LISTENER AND SERVLET

```
public class HelloServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html");  
  
        PrintWriter out = response.getWriter();  
  
        Connection con=(Connection)  
        getServletContext\(\).getAttribute\("connection1"\);  
  
        out.println("<HTML>\n" +  
            "  
            "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +  
            "  
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +  
            "  
            "<H1>"+ con.getSchema\(\) + "</H1></BODY></HTML>");  
  
    }  
}
```

```
public final class ContextListener implements ServletContextListener {

    private ServletContext context = null;

    public void contextDestroyed(ServletContextEvent event) {

        log("contextDestroyed()");

        this.context = null;

    }

    public void contextInitialized(ServletContextEvent event) {

        this.context = event.getServletContext();

        .....

        Connection con = DriverManager.getConnection( context.getInitParameter(" connectionString"), username, password);

        context.setAttribute("connection1", con);

        log("contextInitialized()");

    }

    private void log(String message) {

        if (context != null)

            context.log("ContextListener: " + message);

        else

            System.out.println("ContextListener: " + message);

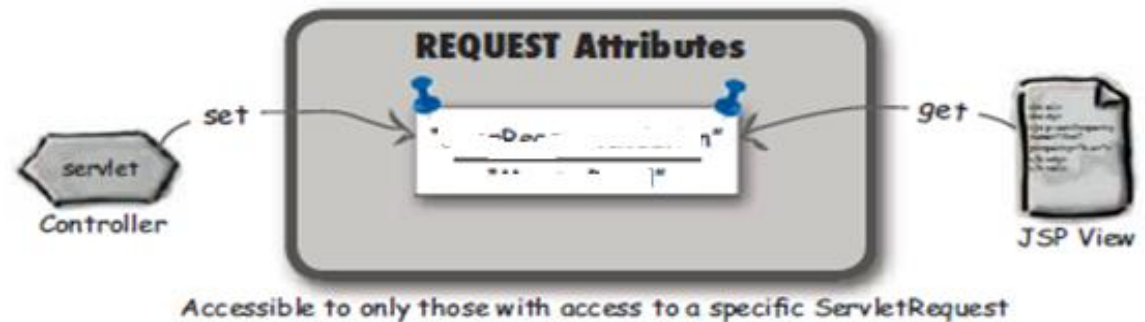
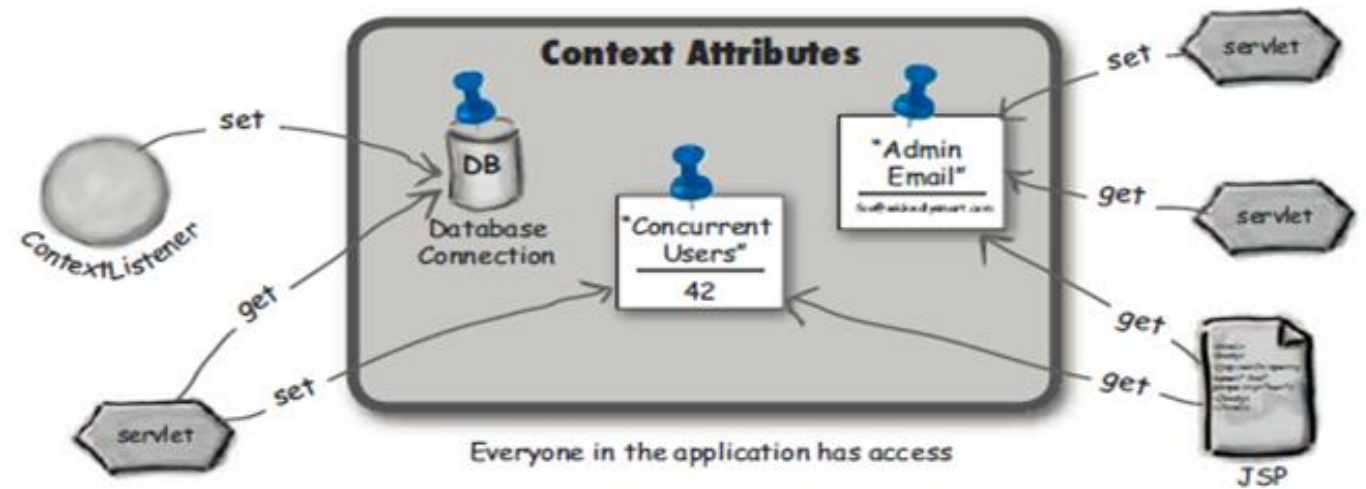
    }

}
```


Eight Listeners

Scenario	Listener interface	Event type
You want to know if an attribute in a web app context has been added, removed, or replaced.	<code>javax.servlet.ServletContextAttributeListener</code> <i>attributeAdded</i> <i>attributeRemoved</i> <i>attributeReplaced</i>	<code>ServletContextAttributeEvent</code>
You want to know how many concurrent users there are. In other words, you want to track the active sessions. We cover sessions in	<code>javax.servlet.http.HttpSessionListener</code> <i>sessionCreated</i> <i>sessionDestroyed</i>	<code>HttpSessionEvent</code>
You want to know each time a request comes in, so that you can log it.	<code>javax.servlet.ServletRequestListener</code> <i>requestInitialized</i> <i>requestDestroyed</i>	<code>ServletRequestEvent</code>
You want to know when a request attribute has been added, removed, or replaced.	<code>javax.servlet.ServletRequestAttributeListener</code> <i>attributeAdded</i> <i>attributeRemoved</i> <i>attributeReplaced</i>	<code>ServletRequestAttributeEvent</code>

SCOPE OF ATTRIBUTES



PROBLEM WITH CONTEXT ATTRIBUTES

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("test context attributes<br>");

    getServletContext().setAttribute("foo", "22");
    getServletContext().setAttribute("bar", "42");

    out.println(getServletContext().getAttribute("foo"));
    out.println(getServletContext().getAttribute("bar"));
}
```



```

public synchronized void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("test context attributes<br>");

    getServletContext().setAttribute("foo", "22");
    getServletContext().setAttribute("bar", "42");

    out.println(getServletContext().getAttribute("foo"));
    out.println(getServletContext().getAttribute("bar"));
}

```

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("test context attributes<br>");

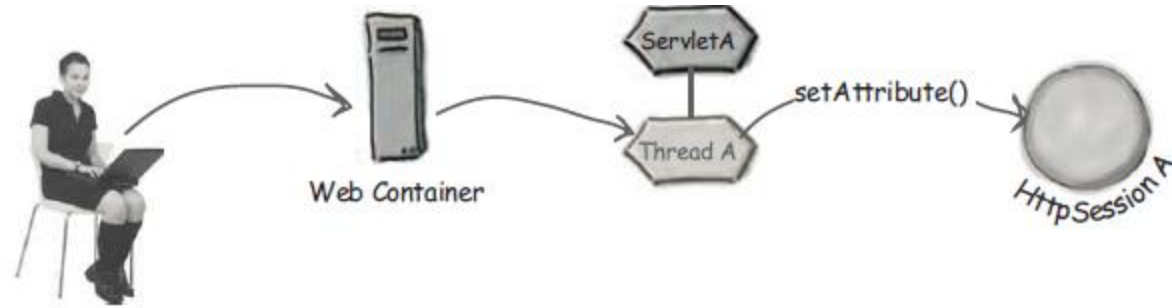
    synchronized(getServletContext()) {
        getServletContext().setAttribute("foo", "22");
        getServletContext().setAttribute("bar", "42");

        out.println(getServletContext().getAttribute("foo"));
        out.println(getServletContext().getAttribute("bar"));
    }
}

```

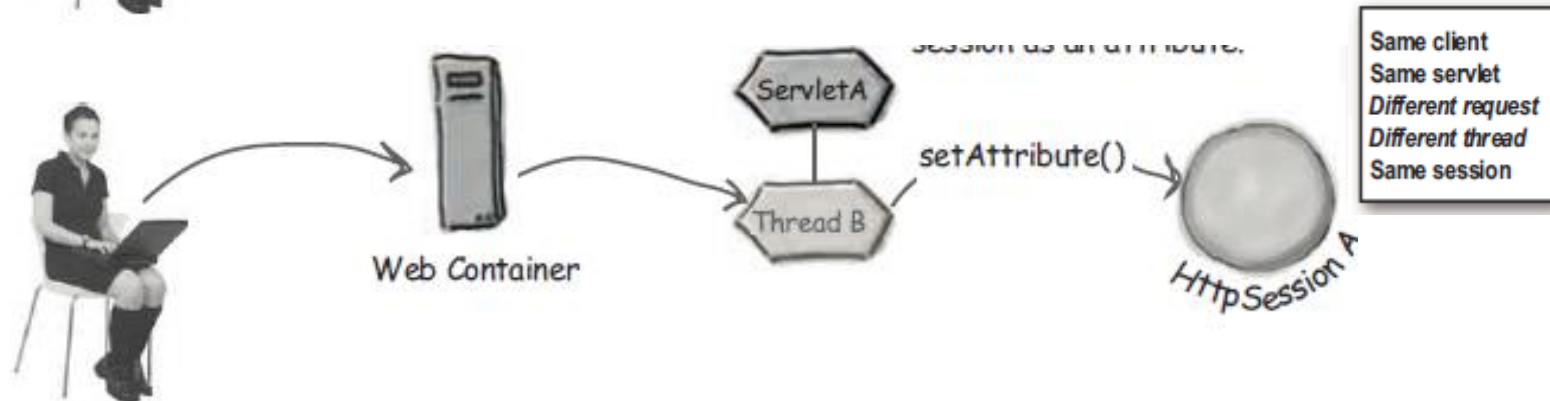
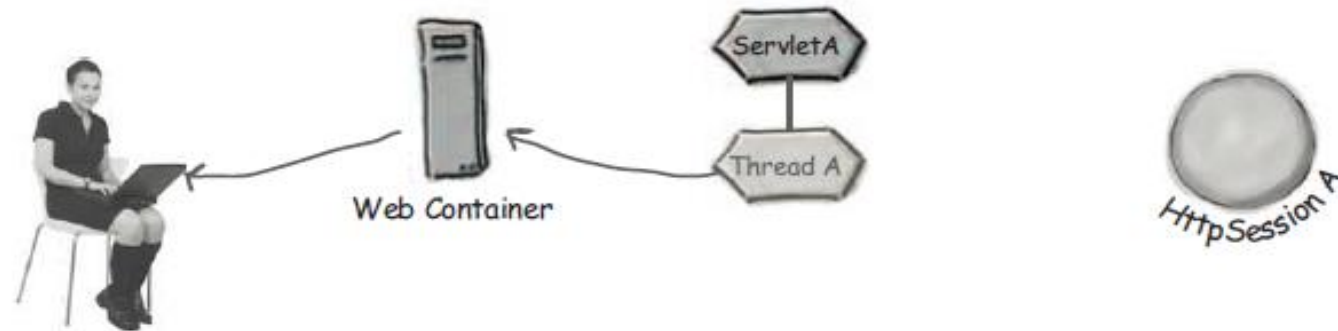
Now we're getting the lock on the context itself!! This is the way to protect context attribute state. (You don't want synchronized(this).)

Only request attributes and local variables are threadsafe



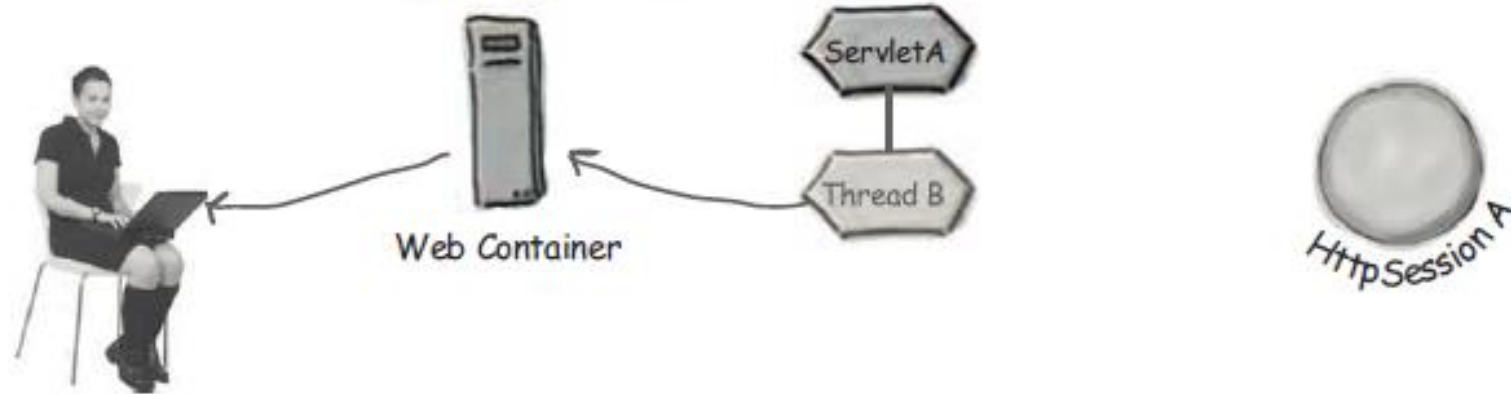
Client asks for a book on web technology

this case another question, "What price range?"



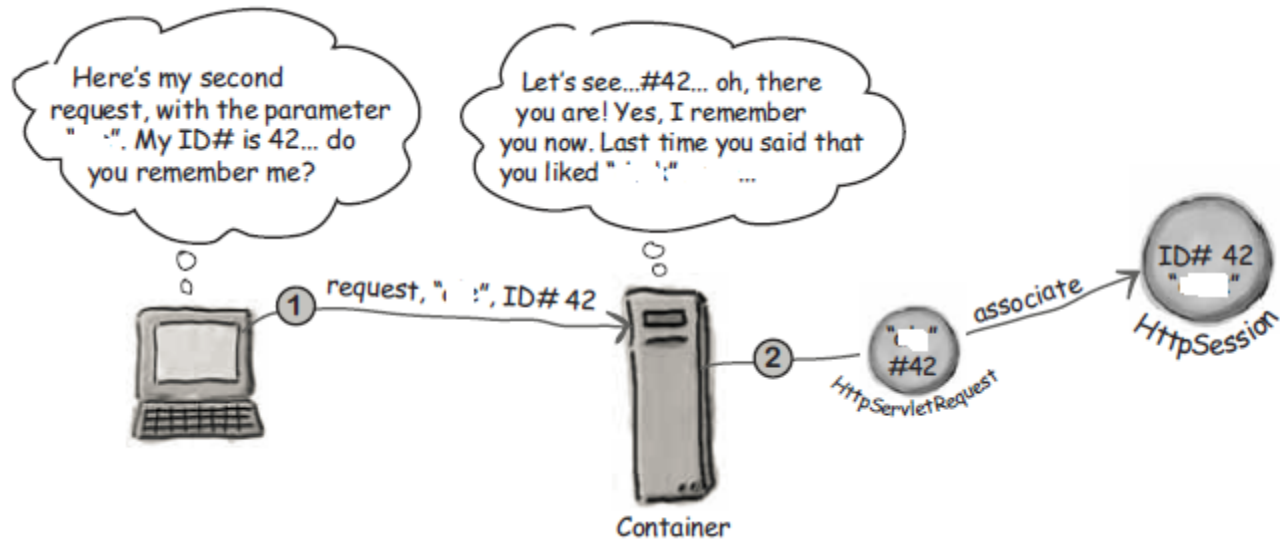
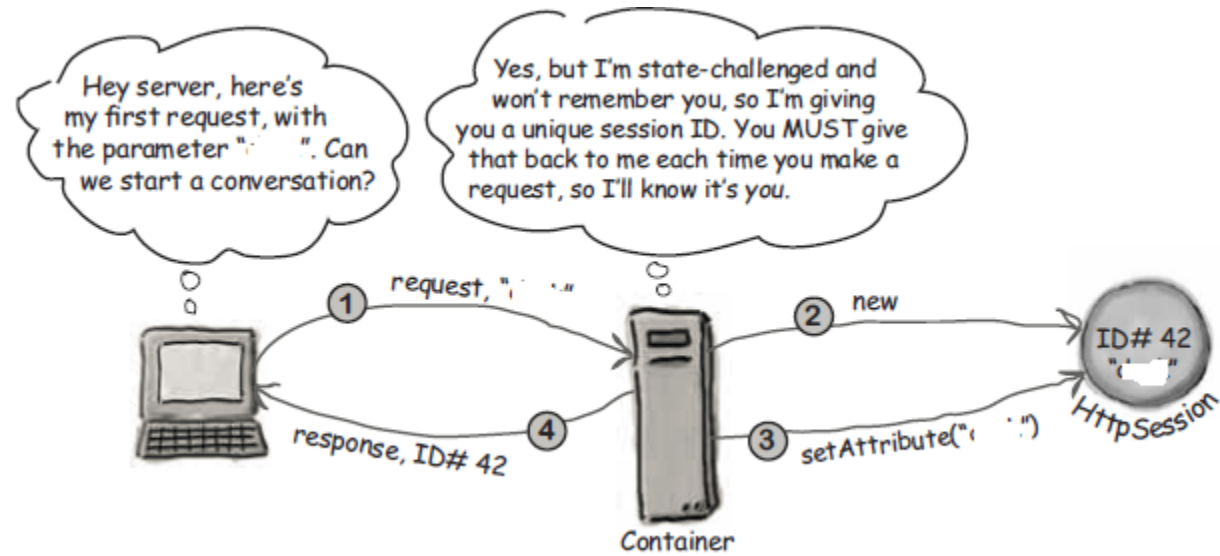
4

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



What if another client joins in?

WHO IS THE CLIENT



HOW TO EXCHANGE SESSION ID

Cookies



"Set-Cookie" is just another header sent in the response.

Here's your cookie with the session ID inside...



```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=0AAB6C8DE415
Content-Type: text/html
Content-Length: 397
Date: Wed, 19 Nov 2003 03:25:40 GMT
Server: Apache-Coyote/1.1
Connection: close

<html>
...
</html>
```

HTTP Response



OK, here's the cookie with my request

"Cookie" is another header sent in the request.



```
POST /select/selectBeerTaste2.do HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0
Cookie: JSESSIONID=0AAB6C8DE415
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
```

HTTP Request



ROLE OF THE CONTAINER

You *do* have to tell the Container that you want to create or use a session, but the Container takes care of generating the session ID, creating a new Cookie object, stuffing the session ID into the cookie, and setting the cookie as part of the response. And on subsequent requests, the Container gets the session ID from a cookie in the request, matches the session ID with an existing session, and associates that session with the current request.

Sending a session cookie in the RESPONSE:

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

IF (the request includes a session ID cookie)

find the session matching that ID

ELSE IF (there's no session ID cookie OR there's no current session matching the session ID)

create a *new* session.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("test session attributes<br>");

    HttpSession session = request.getSession();

    if (session.isNew()) {
        out.println("This is a new session.");
    } else {
        out.println("Welcome back!");
    }
}
```

getSession() returns a session no matter what... but you can't tell if it's a new session unless you ask the session.

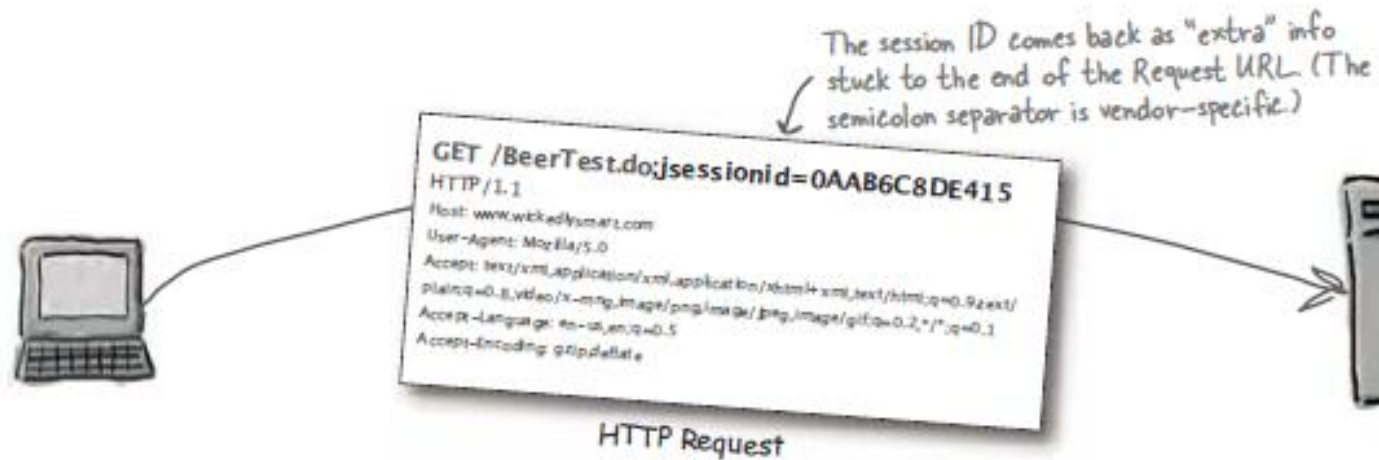
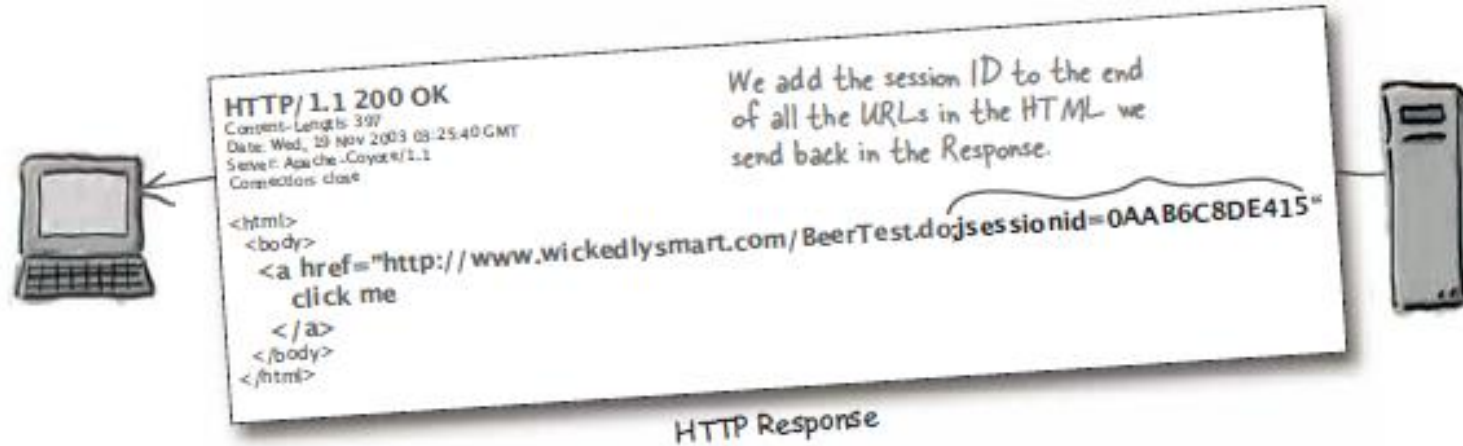
isNew() returns true if the client has not yet responded with this session ID.

Request.getSession(false);---if an existing session is not found it returns null

URL REWRITING

response.encodeURL("...")

Response.encodeRedirectURL(" ")



TERMINATING SESSIONS

Sessions are mostly used to get and set session scoped attributes

Ways to terminate a session

- It times out
- Calling `session.invalidate()`
- The application goes down (crashes or is undeployed)

SESSION TIMES OUT

```
<web-app ...>
  <servlet>
    ...
  </servlet>
  <session-config>
    <session-timeout>15</session-timeout>
  </session-config>
</web-app>
```

The "15" is in minutes. This says if the client doesn't make any requests on this session for 15 minutes kill it.*

```
session.setMaxInactiveInterval(20*60);
```

Only the session on which you call the method is affected.

The argument to the method is in seconds, so this says if the client doesn't make any requests on the session for 20 minutes, kill it.*

COOKIES

```
HTTP/1.1 200 OK
Set-Cookie: username=TomasHirsch
Content-Type: text/html
Content-Length: 397
Date: Wed, 19 Nov 2003 03:25:40 GMT
Server: Apache-Coyote/1.1
Connection: close

<html>
...
</html>
```

← Server sends
this first.

Session cookies vanish when the client's browser quits, but you **CAN** tell a cookie to persist on the client even after the browser shuts down.

ADDING AND GETTING COOKIES

Creating a new Cookie

```
Cookie cookie = new Cookie("username", name);
```

← The Cookie constructor takes a name/value String pair.

Setting how long a cookie will live on the client

```
cookie.setMaxAge(30*60);
```

← setMaxAge is defined in SECONDS. This code says "stay alive on the client for 30*60 seconds" (30 minutes). Setting max age to -1 makes the cookie disappear when the browser exits.

Sending the cookie to the client

```
response.addCookie(cookie);
```

ATTRIBUTES

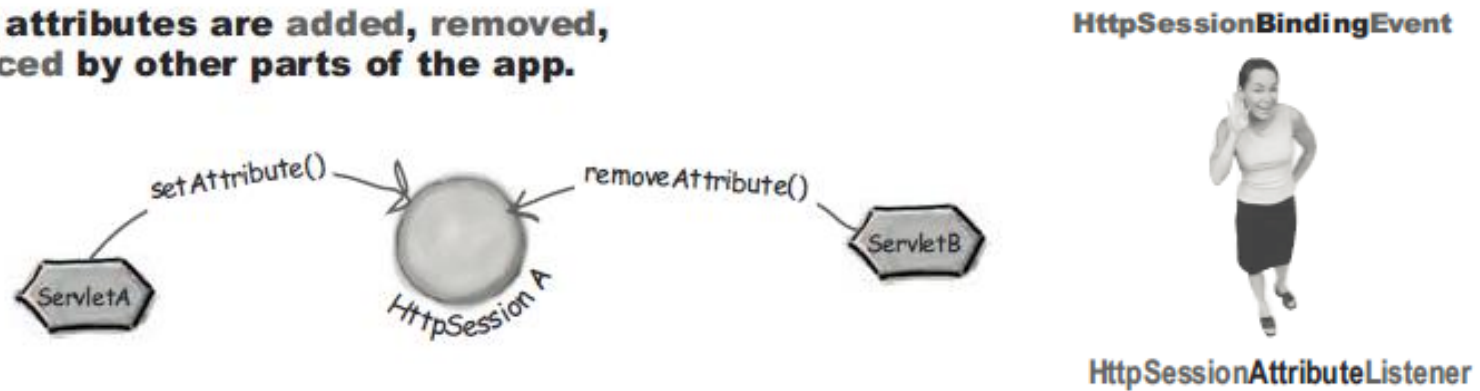
Scope	In a servlet	In a jsp Implicit obj
Application	<code>getServletContext.setAttribute("name",obj)</code>	<code>application.setAttribute("name",obj)</code>
Request	<code>request. .setAttribute("name",obj)</code>	<code>request. .setAttribute("name",obj)</code>
Session	<code>Request.getSession() .setAttribute("name",obj)</code>	<code>session.setAttribute("name",obj)</code>
Page	NA	<code>pageContext.setAttribute("name",obj)</code>

SESSION LIFECYCLE EVENTS

The session is created or destroyed.



Session attributes are added, removed, or replaced by other parts of the app.



HTTPSESSIONLISTENER

```
package com.example;  
import javax.servlet.http.*;
```

```
public class SessionCounter implements HttpSessionListener {
```

```
    static private int activeSessions;
```

```
    public static int getActiveSessions() {  
        return activeSessions;  
    }
```

← This class will be deployed in WEB-INF/classes like all the other web-app classes, so all servlets and other helper classes can access this method.

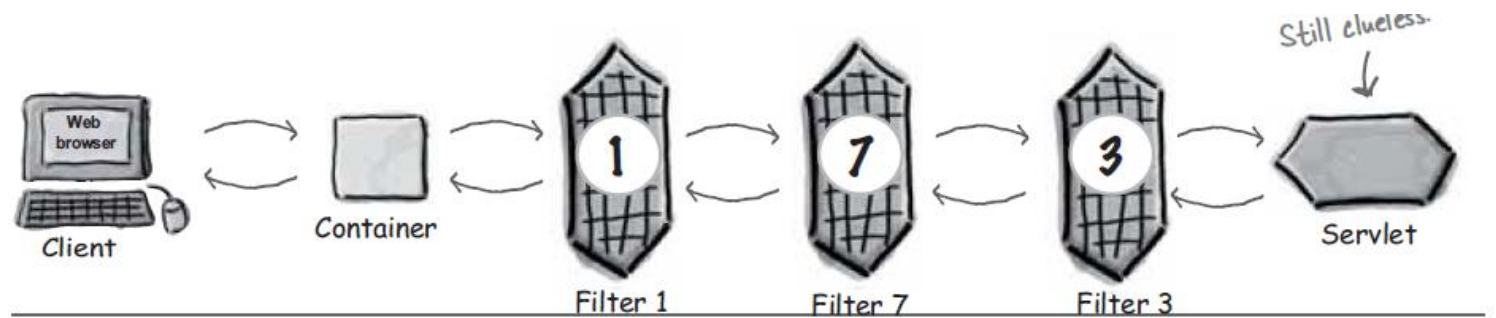
```
    public void sessionCreated(HttpSessionEvent event) {  
        activeSessions++;  
    }
```

← These methods take an HttpSessionEvent

```
    public void sessionDestroyed(HttpSessionEvent event) {  
        activeSessions--;  
    }  
}
```

```
<listener><listener-class>com.example.SessionCounter</listener-class></listener>
```


FILTERS



Request filters can:

- perform security checks
- reformat request headers or bodies
- audit or log requests

Response filters can:

- compress the response stream
- append or alter the response stream
- create a different response altogether

CODE FOR FILTER

```
@WebFilter("/SelectCoffeeMVC.do")
public class MyFilter implements Filter
{
    private FilterConfig filterConfig = null;

    public void doFilter(ServletRequest request, ServletResponse
response,   FilterChain chain)
        throws IOException, ServletException
    {
        String remark = ((HttpServletRequest)
request).getParameter("remark");

        if(remark.length()>0) {
            chain.doFilter(request, response);
        } else {
            response.setContentType("text/html");
        }
    }
}
```

SIMILARITY BETWEEN SERTVLETS AND FILTERS

- ❑ The container knows the APIs for filter
- ❑ The container manages their life cycle
- ❑ They are declared in the DD
 - ❑ Filter mapping
 - ❑ servletNames
 - ❑ url patterns

```
<filter-mapping>  
  <filter-name>MonitorFilter</filter-name>  
  <url-pattern>*.do</url-pattern>  
  <dispatcher>REQUEST</dispatcher>
```

- and / or -

```
<dispatcher>INCLUDE</dispatcher>
```

- and / or -

```
<dispatcher>FORWARD</dispatcher>
```

- and / or -

```
<dispatcher>ERROR</dispatcher>  
</filter>
```

CONVENTIONAL VIEW

