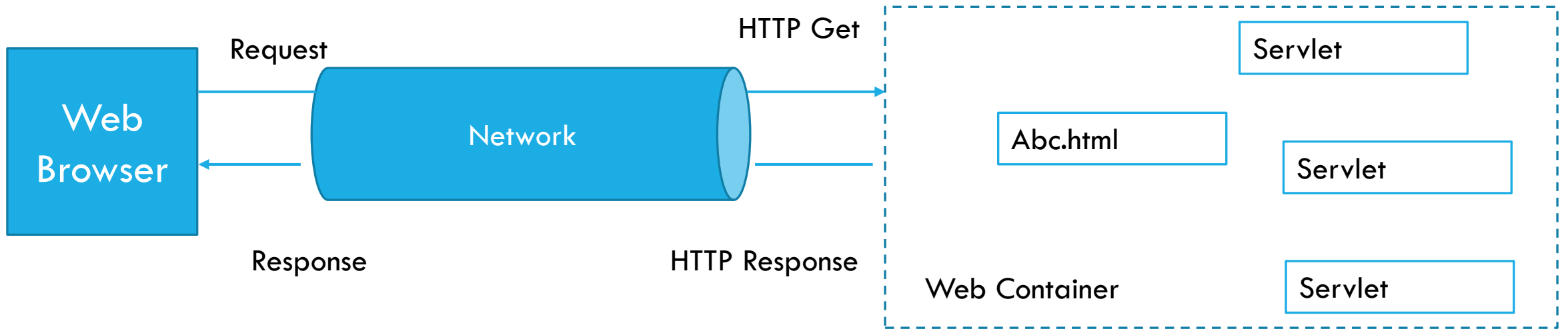




# WEB APPLICATION DEVELOPMENT

Chandreyee Chowdhury

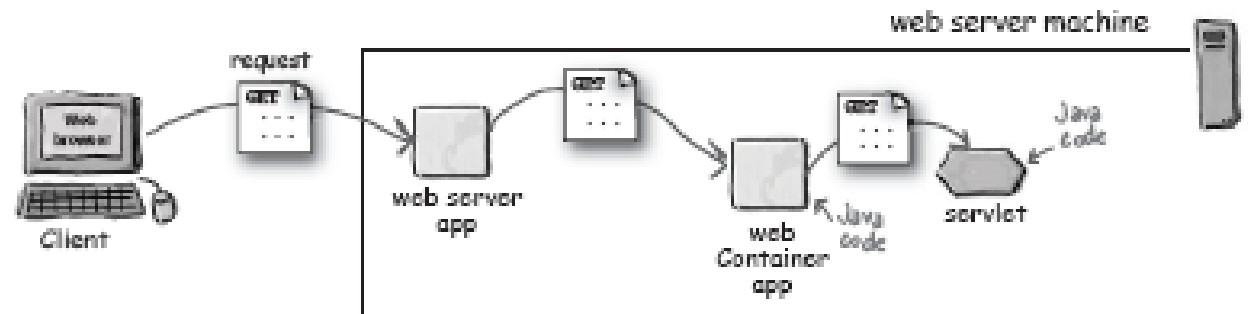
# WEB CONTAINER



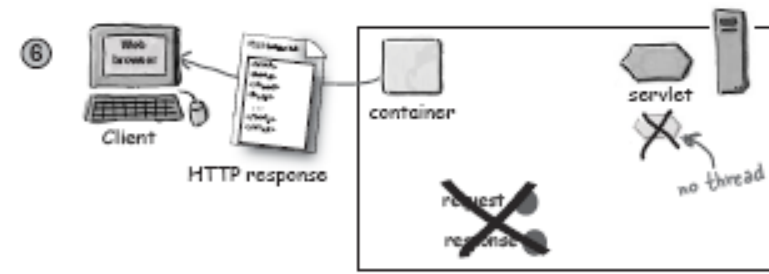
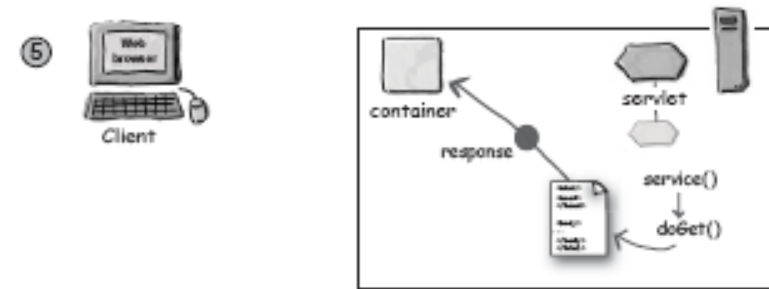
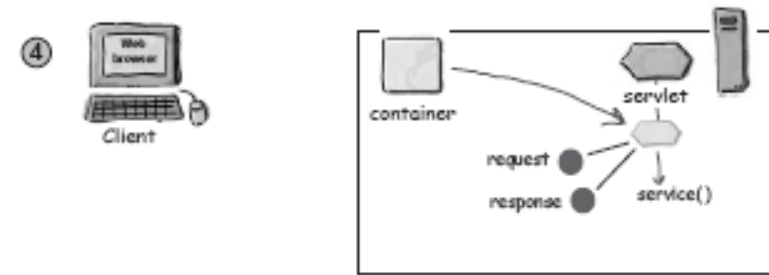
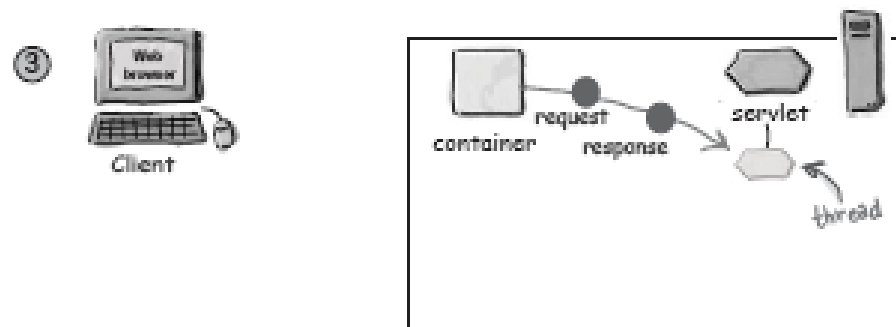
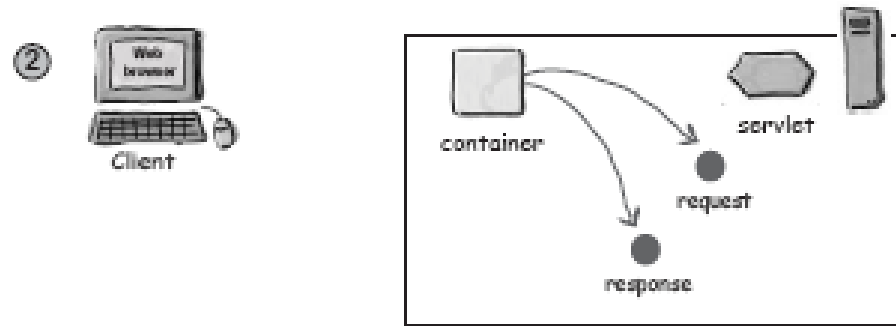
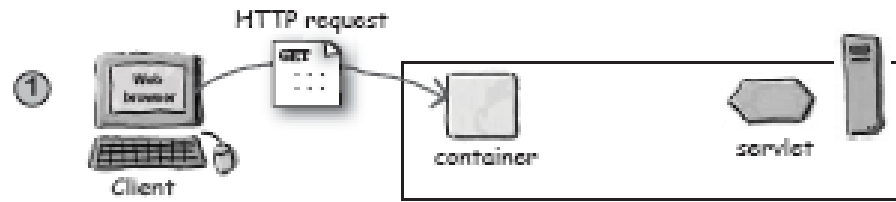
<http://192.168.128.24:8080/demoApp/abc.html>

<http://192.168.128.24:8080/demoApp/def/>

# WEB SERVER VS WEB CONTAINER



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



# A “HELLO WORLD” SERVLET

(FROM THE TOMCAT INSTALLATION DOCUMENTATION)

```
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>");
    }
}
```

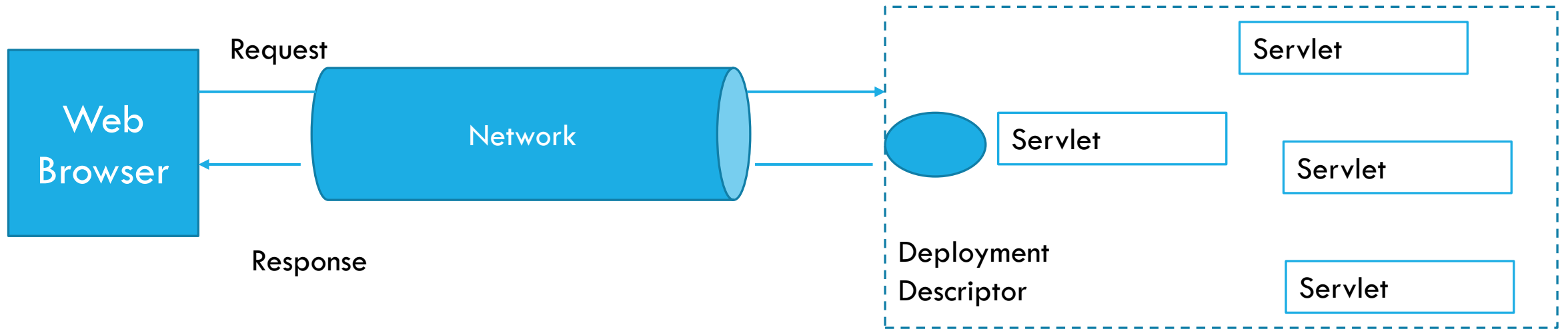
# JSP

```
<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

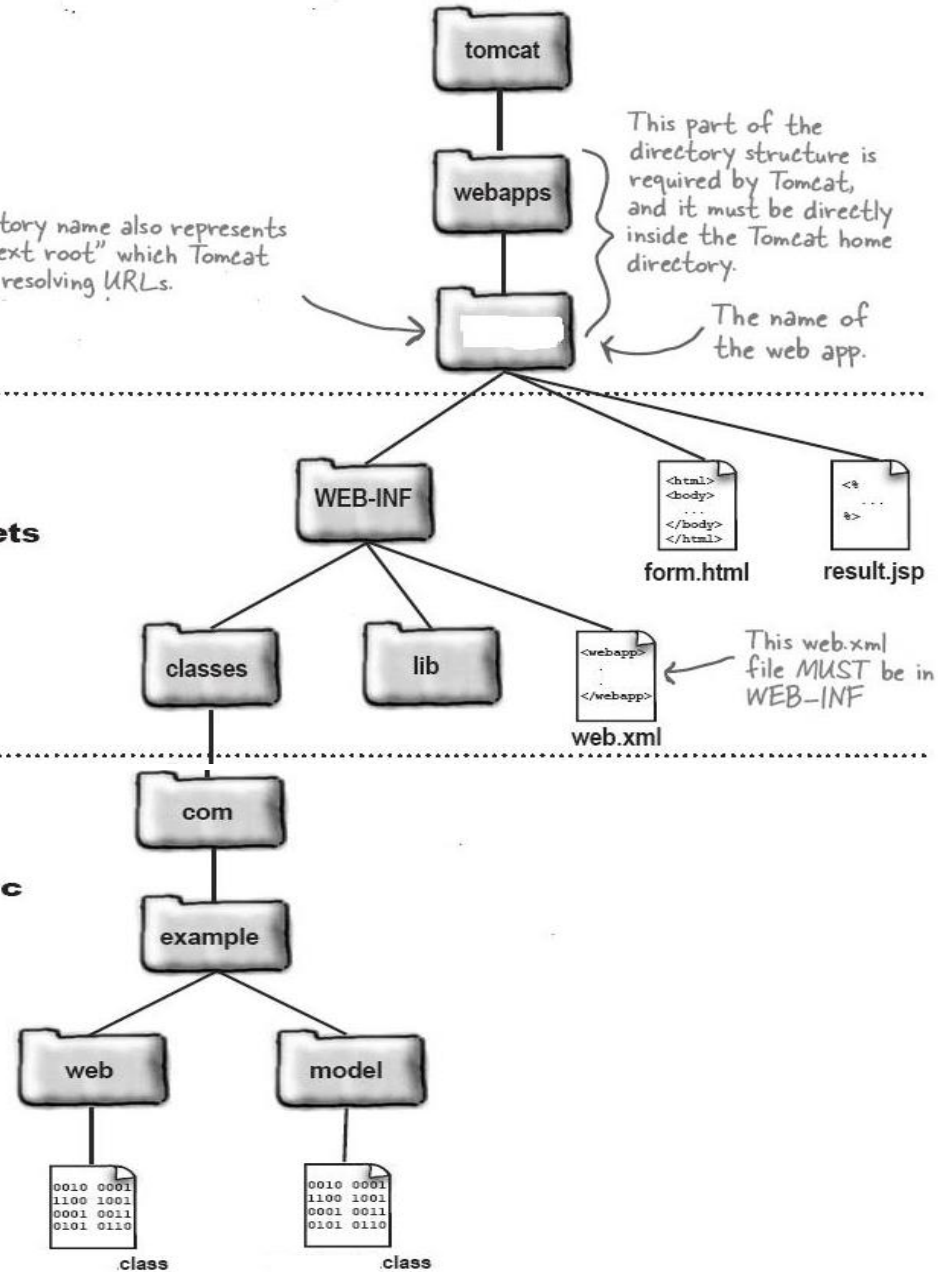
model

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

.class

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

.class

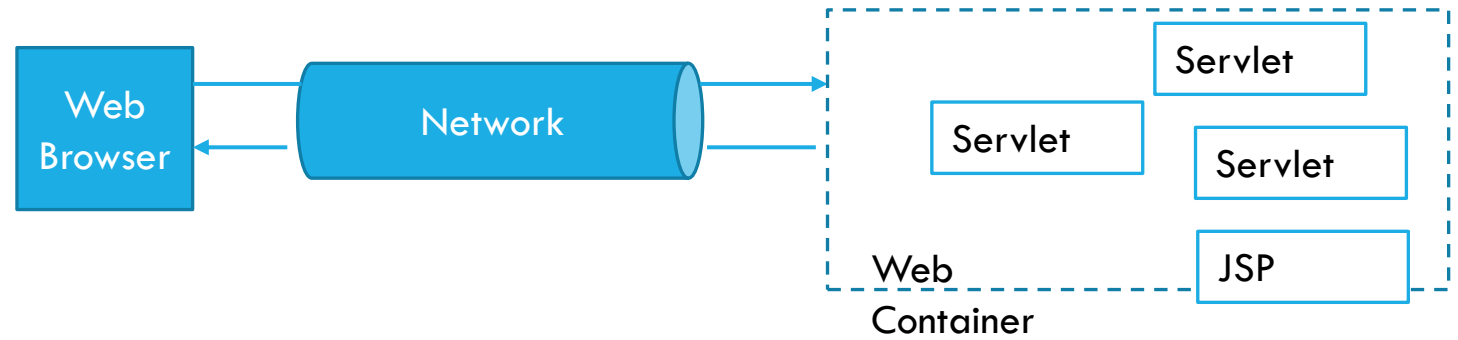


# DEPLOYMENT DESCRIPTOR

**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.

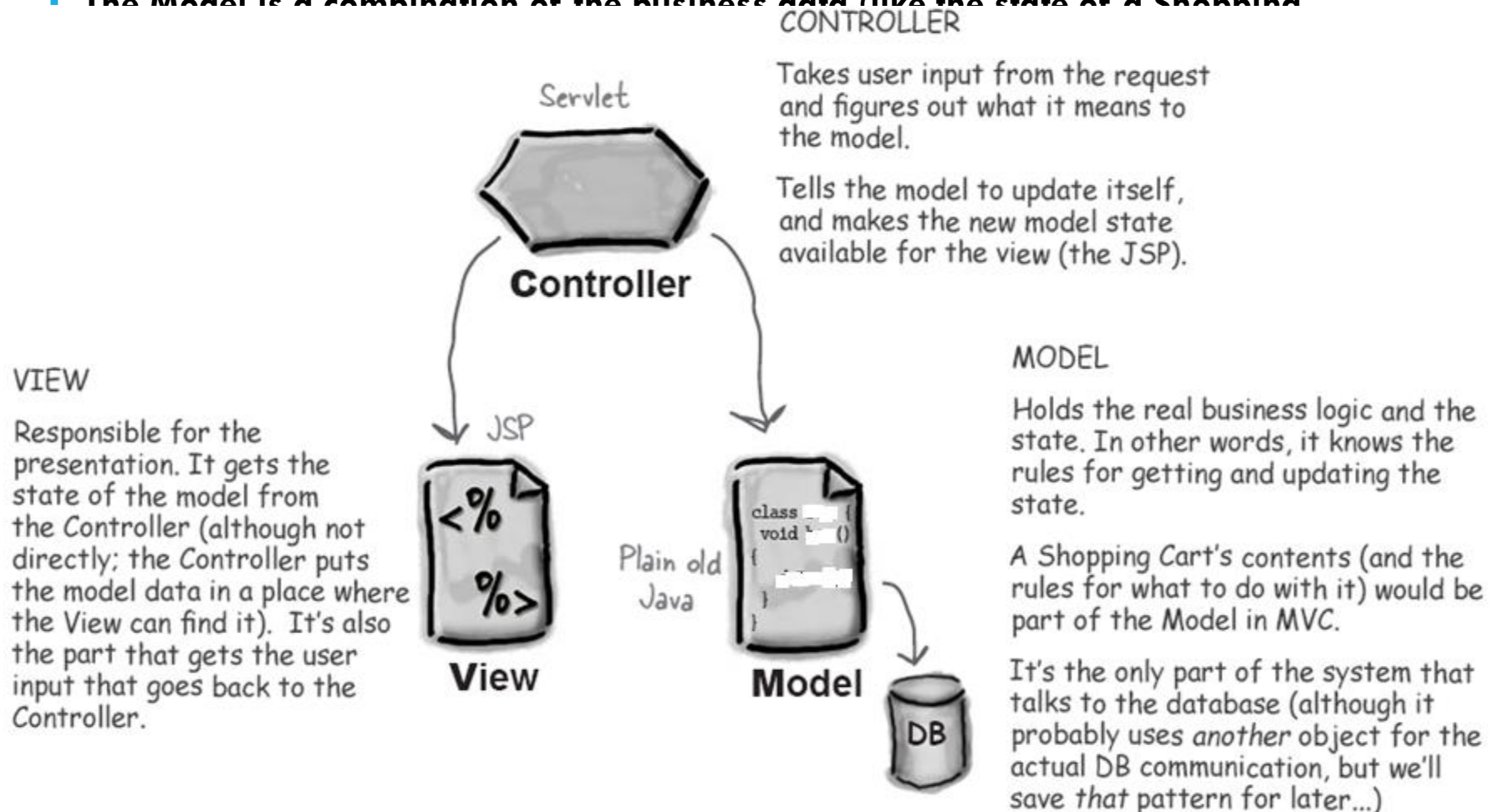
## JSP Support

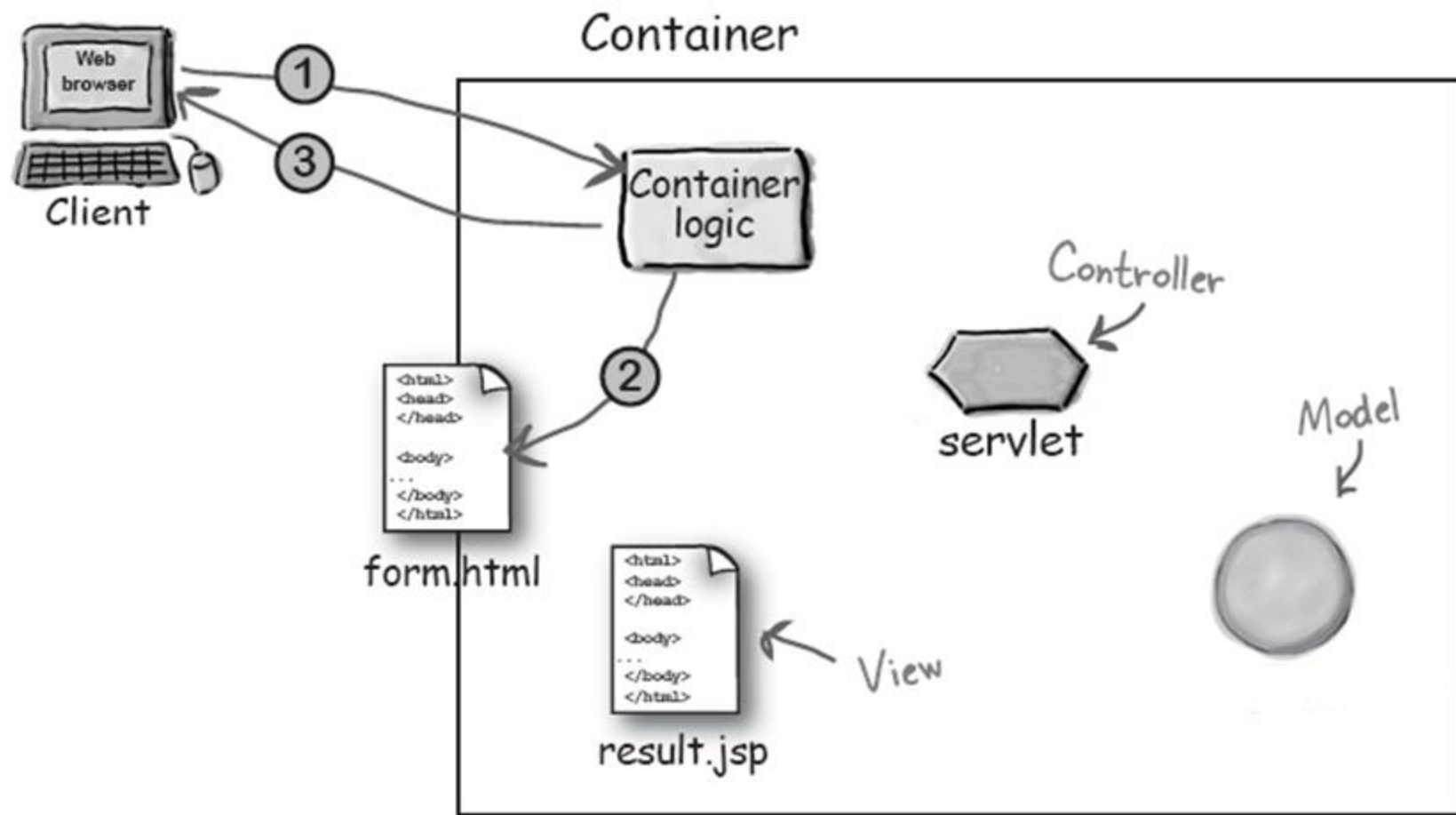
- The container takes care of translating a jsp file into java code

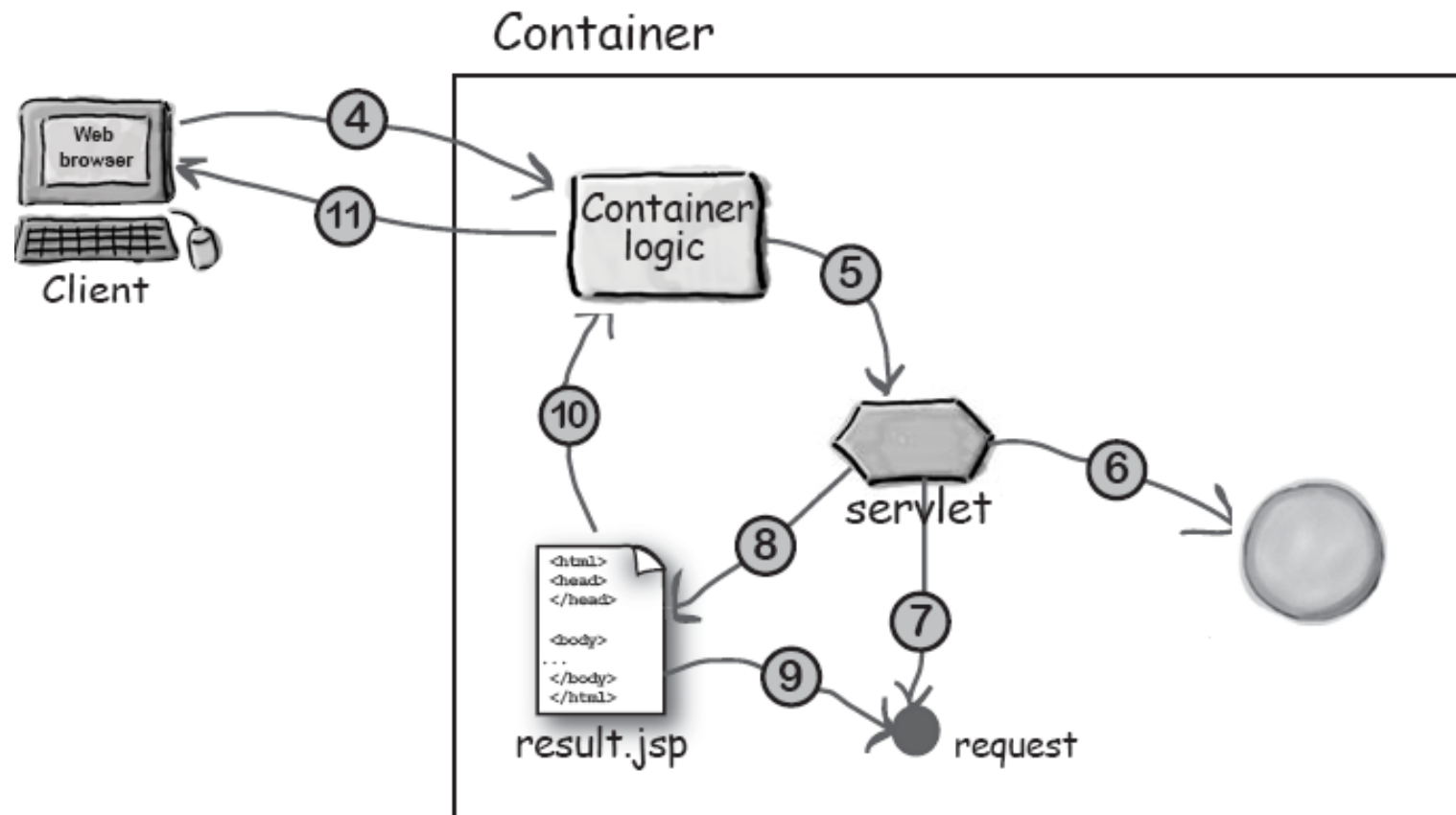
# MVC

**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

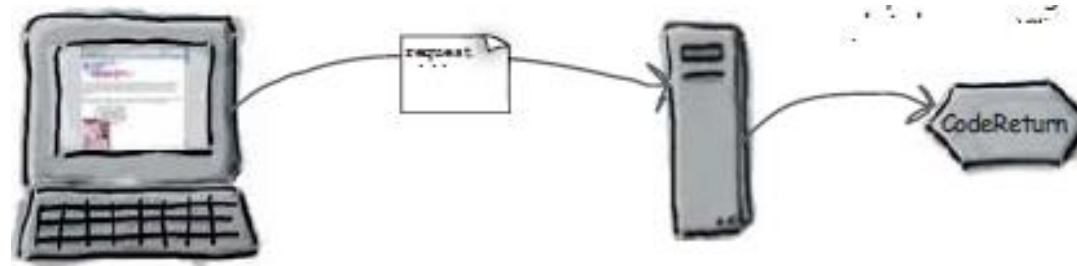






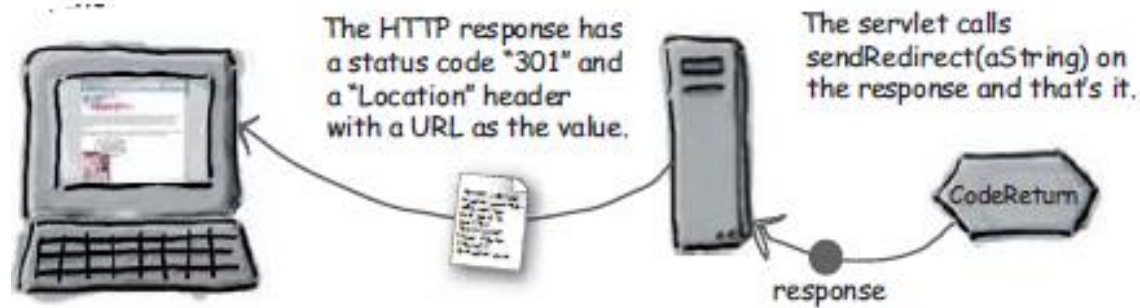
Servlet1 → servlet2 → jsp1 → jsp2

# SEND REDIRECT



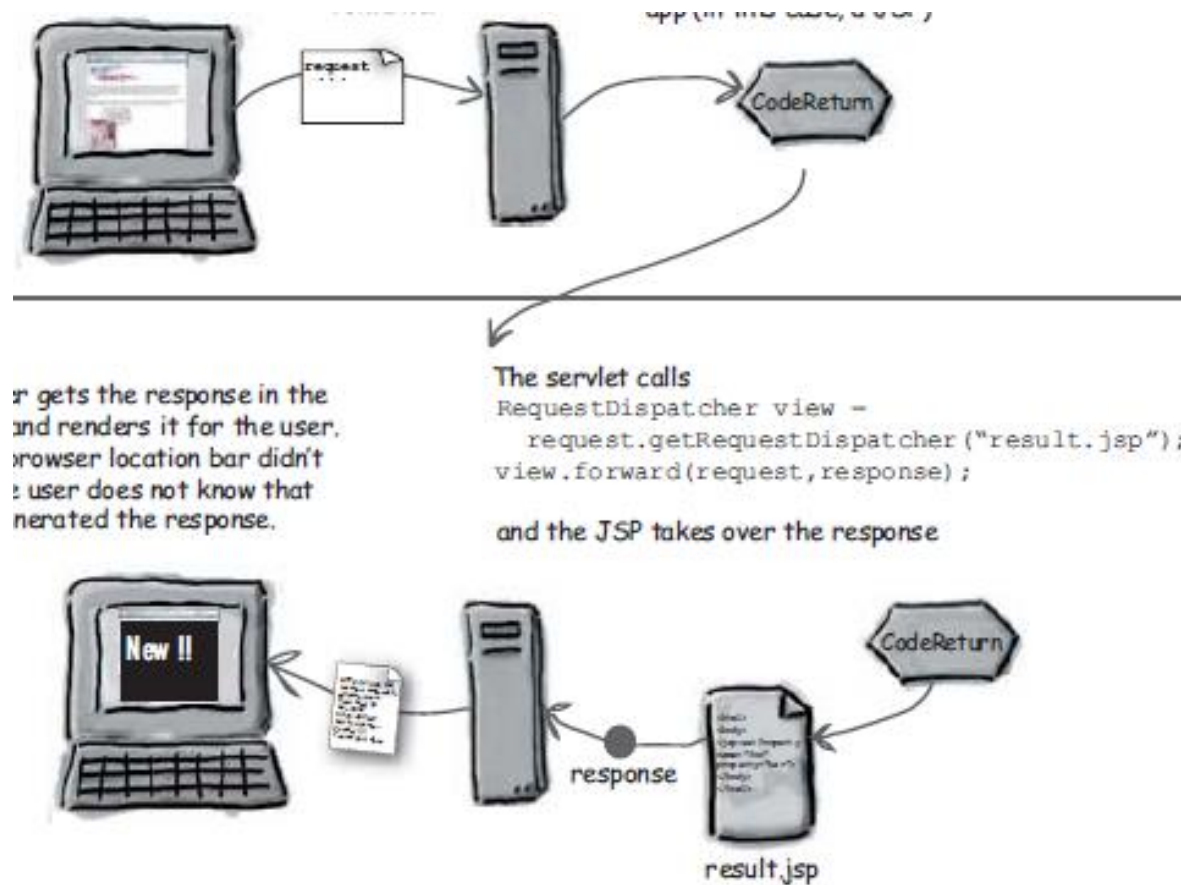
---

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





# REQUEST DISPATCHER



### 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

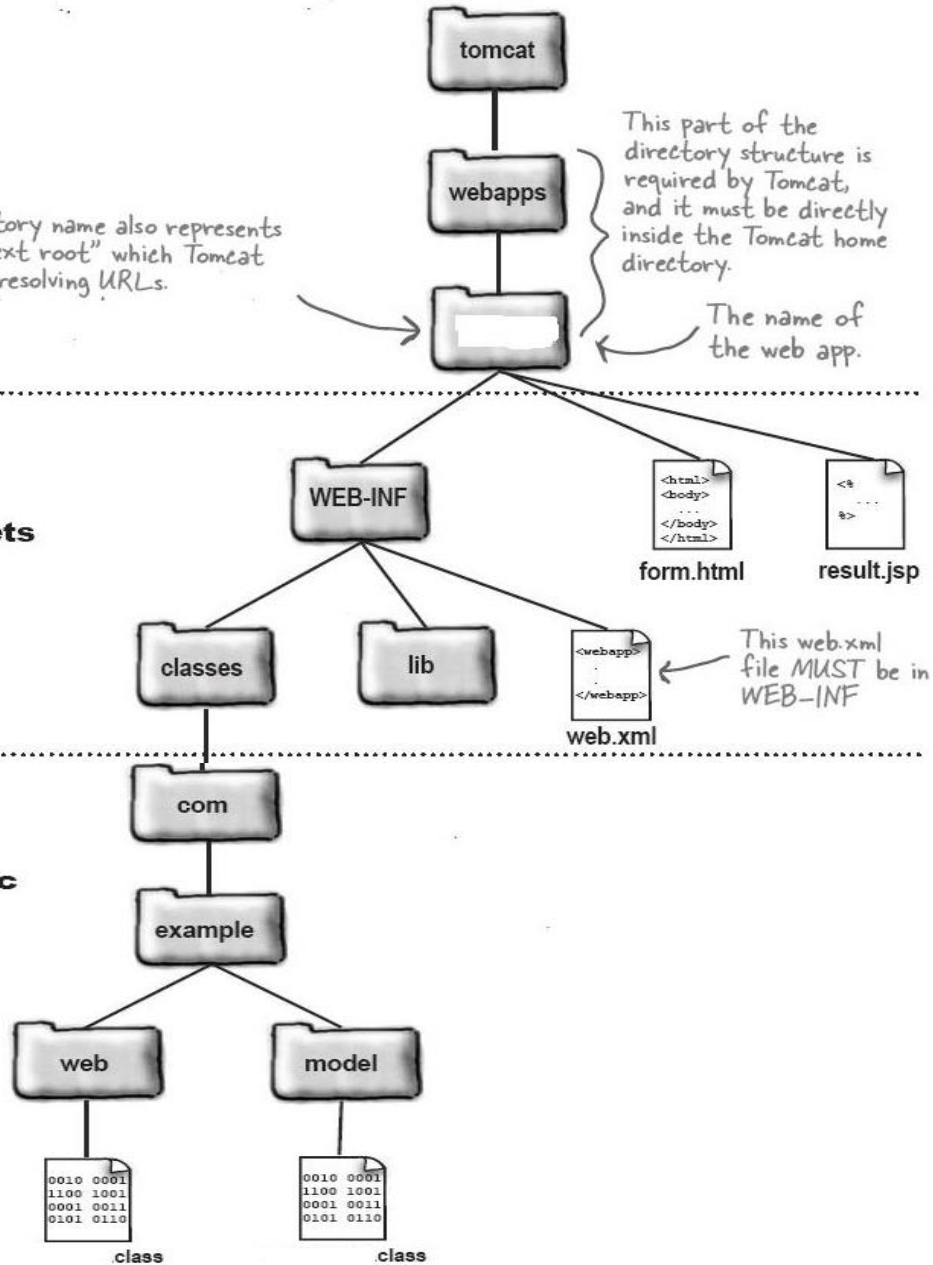
model

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

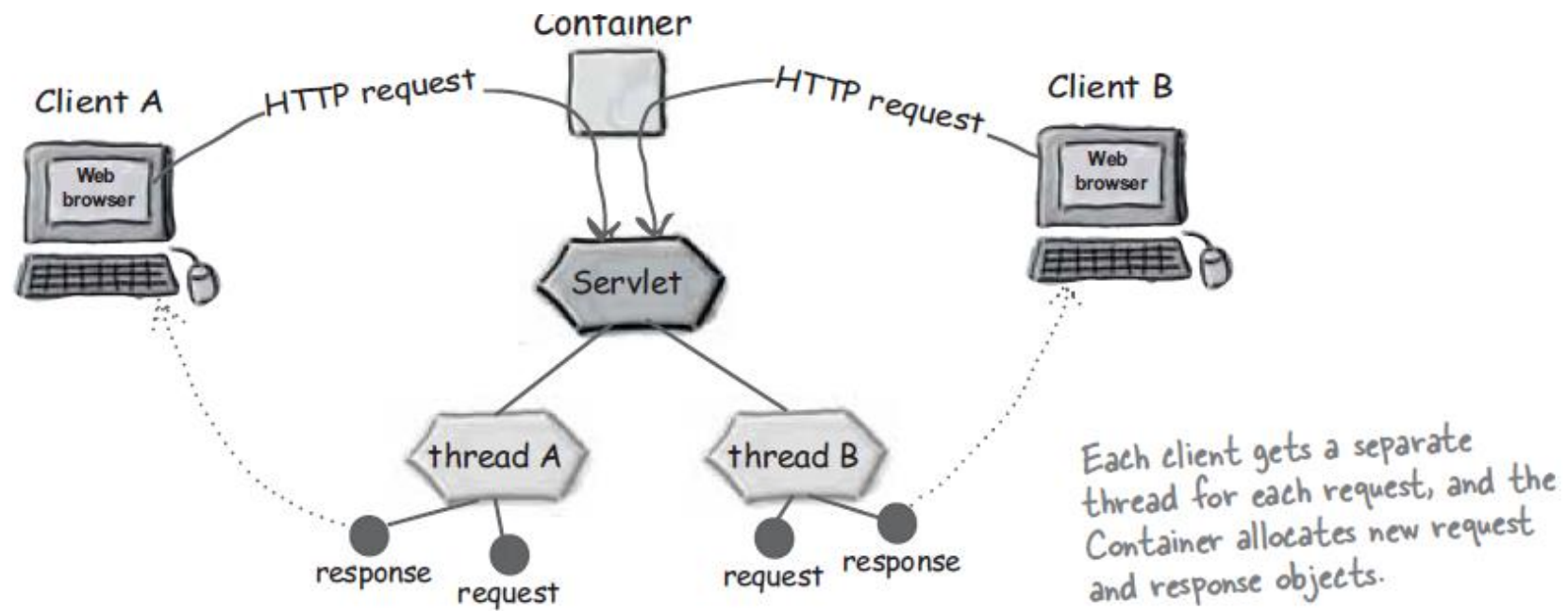
.class

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

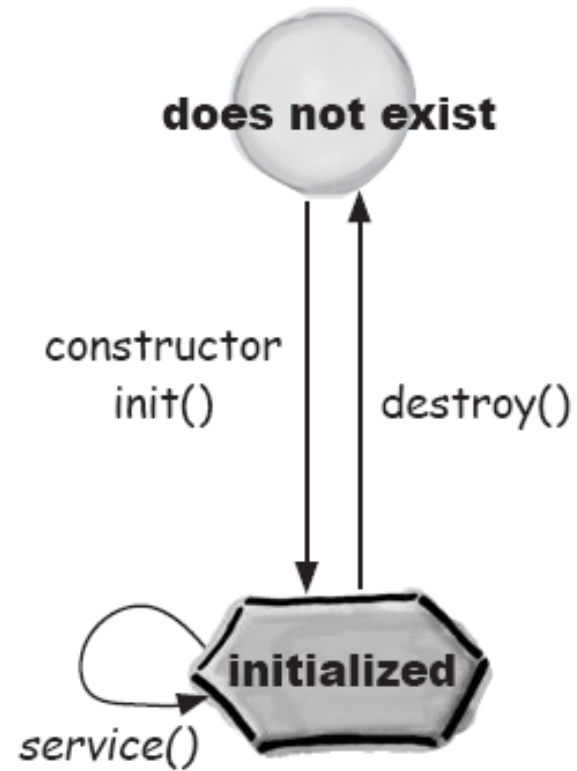
.class

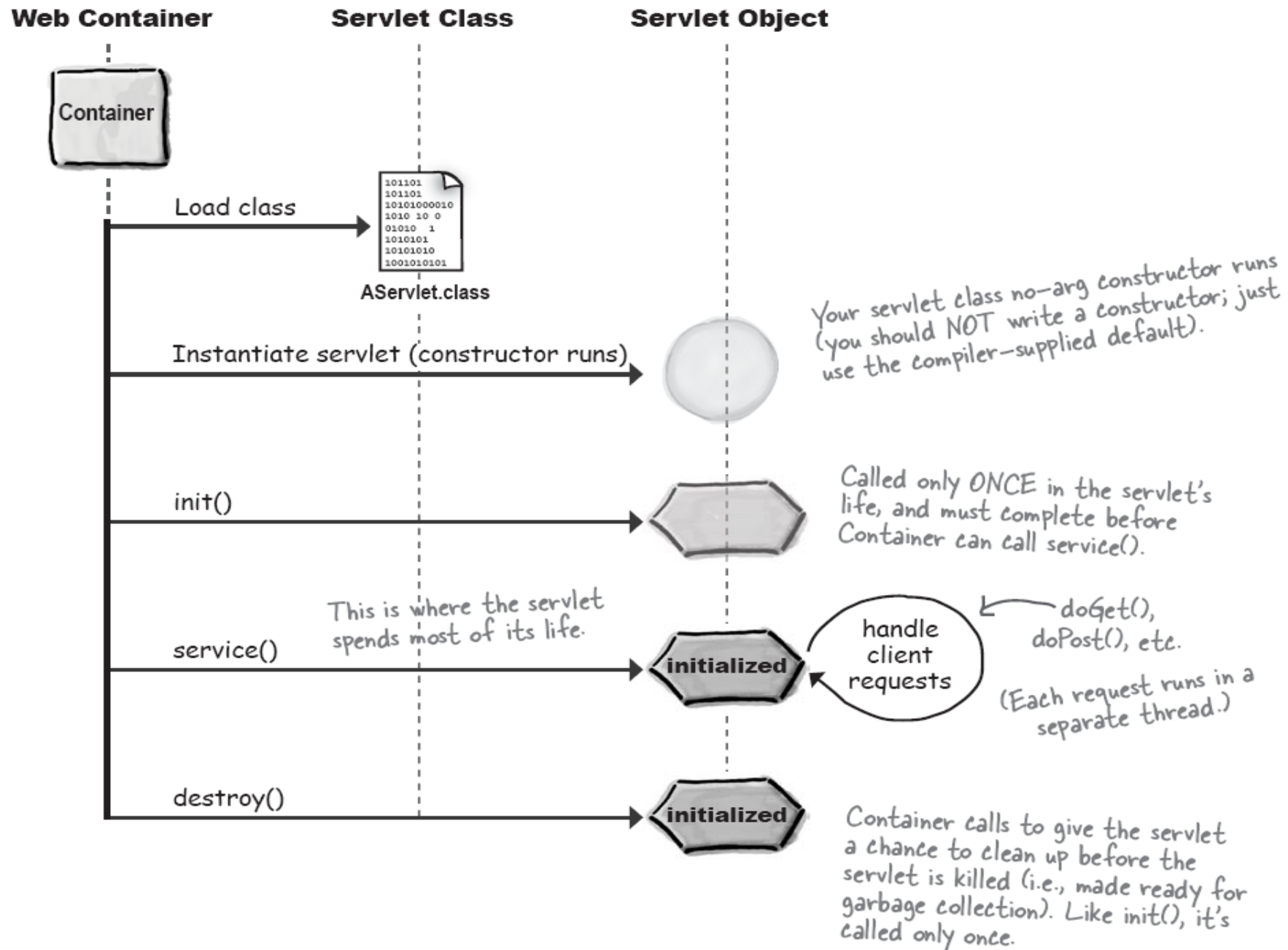


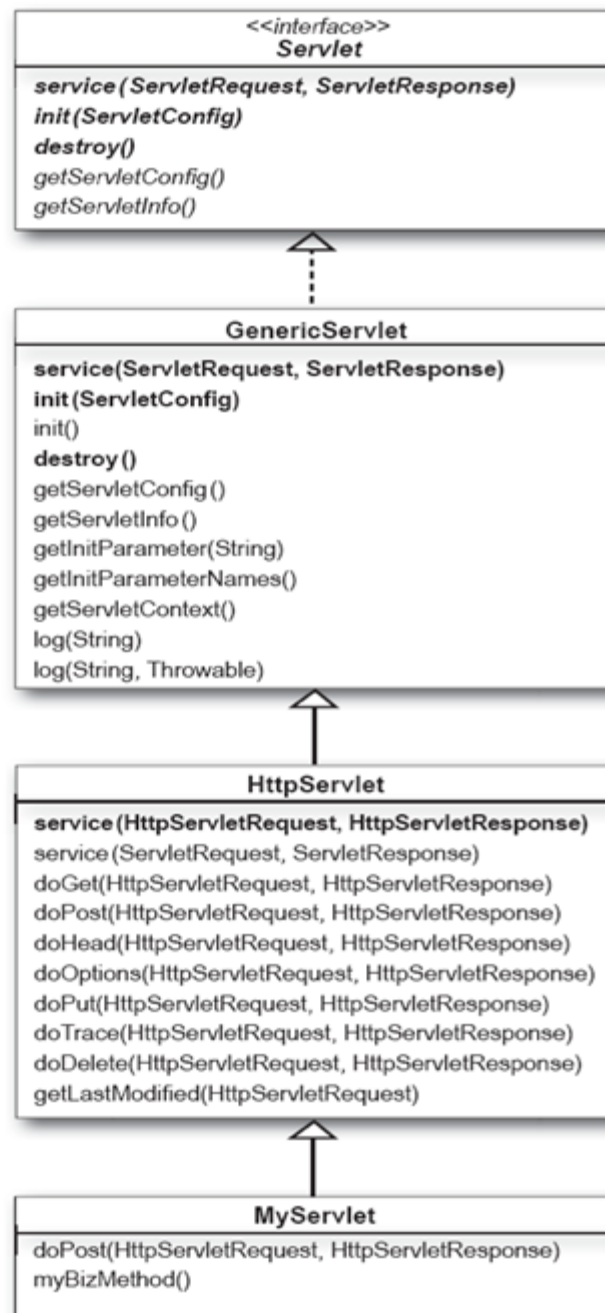
# HANDLING MULTIPLE CLIENTS



# 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)

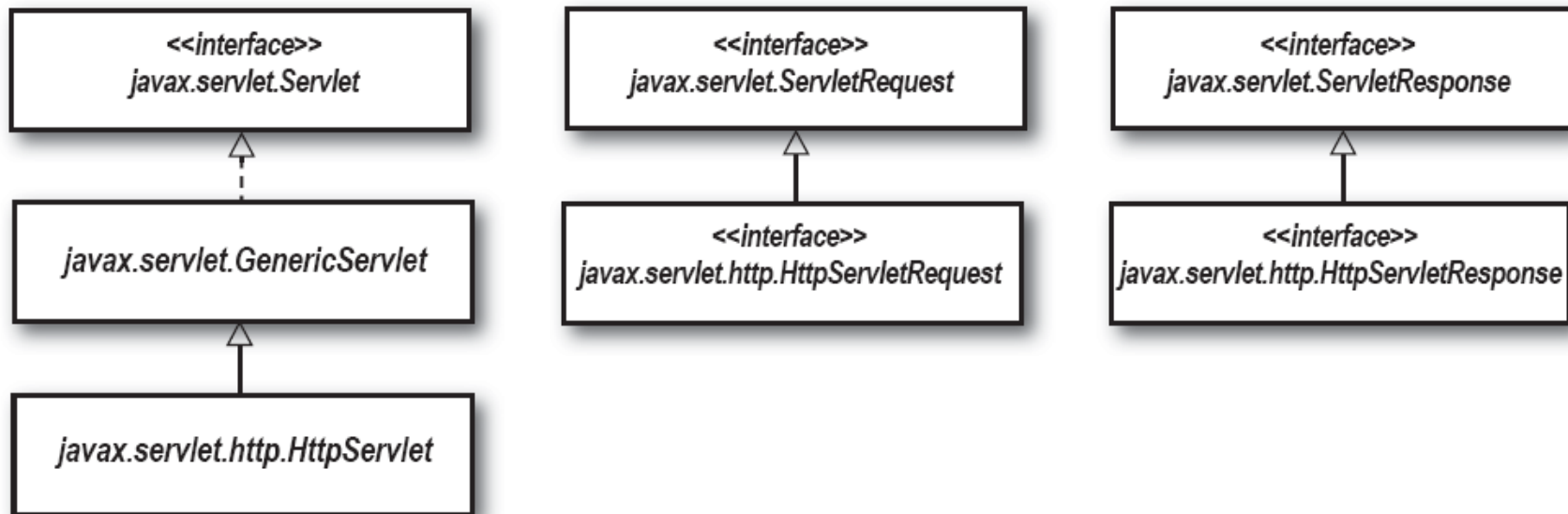
HttpServlet (also an abstract class) implements the service() method to reflect the HTTPness of the servlet—the service() method doesn't take just ANY old servlet request and response, but an HTTP-specific request and response.

## MyServlet class (com.wickedlysmart.foo)

Most of your servletness is handled by superclass met. All you do is override the HTTP methods you need.

# APIS

## Key APIs



## Compiling your servlet

- `Javac -classpath \....\tomcat\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

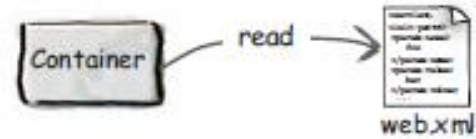
## ServletConfig Object

- 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

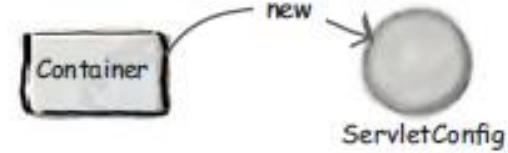
# WHAT MAKES AN OBJECT A SERVLET

- ❑ 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.

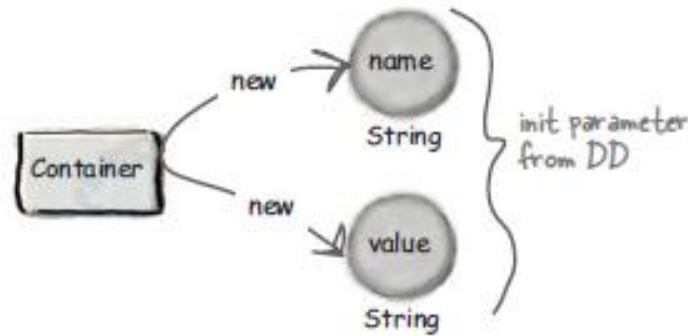
- ① Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



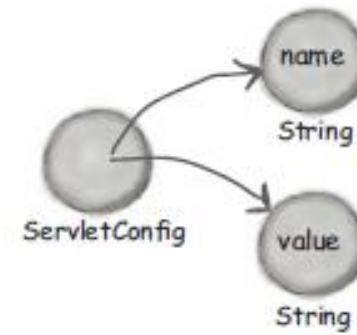
- ② Container creates a new ServletConfig instance for this servlet.



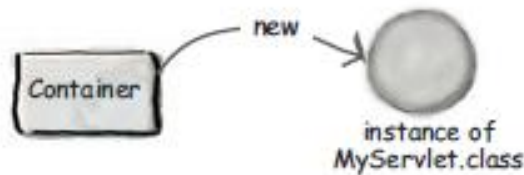
- ③ Container creates a name/value pair of Strings for each servlet init parameter. Assume we have only one.



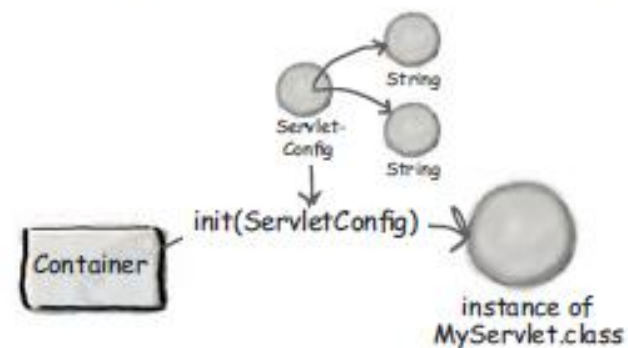
- ④ Container gives the ServletConfig references to the name/value init parameters.



- 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

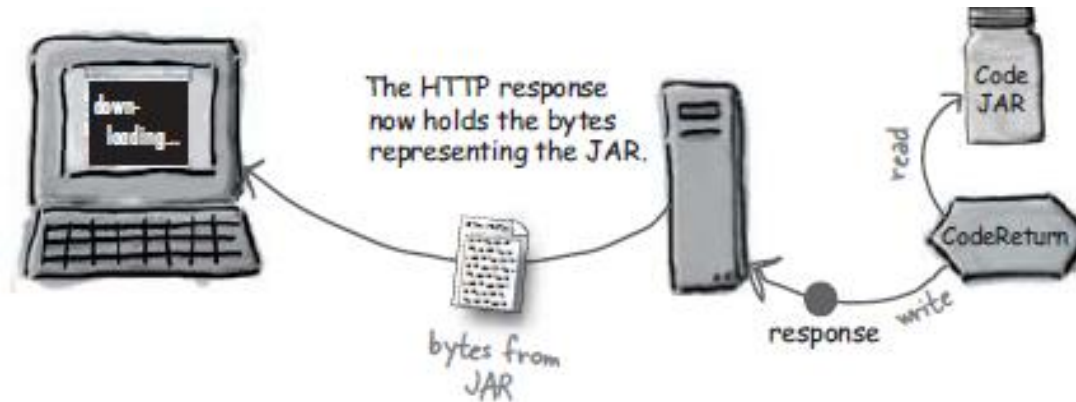
An attribute is a name/value pair in a map instance variable

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.