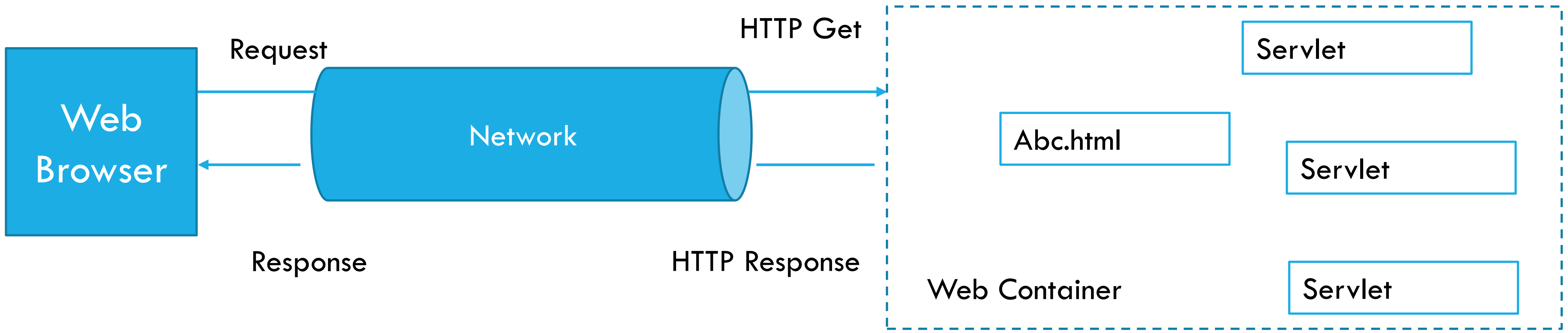
WEB APPLICATION DEVELOPMENT

Chandreyee Chowdhury

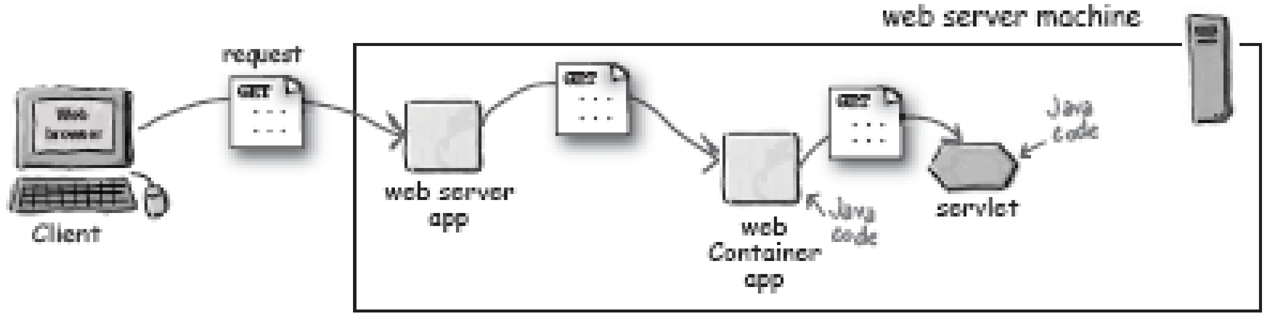
WEB CONTAINER



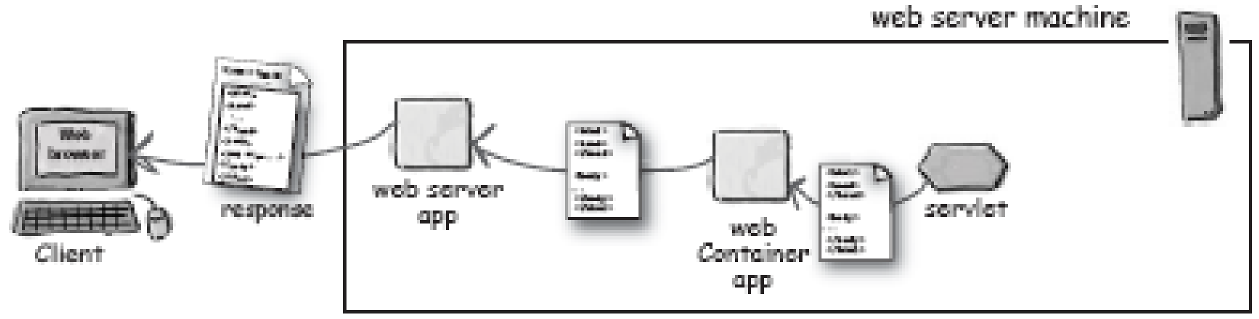
http://192.168.128.24:8080/demoApp/abc.html

http://192.168.128.24:8080/demoApp/def/

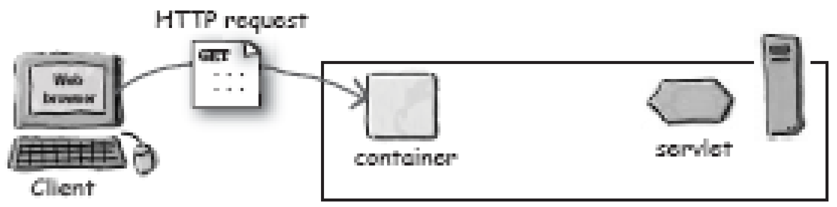
WEB SERVER VS WEB CONTAINER



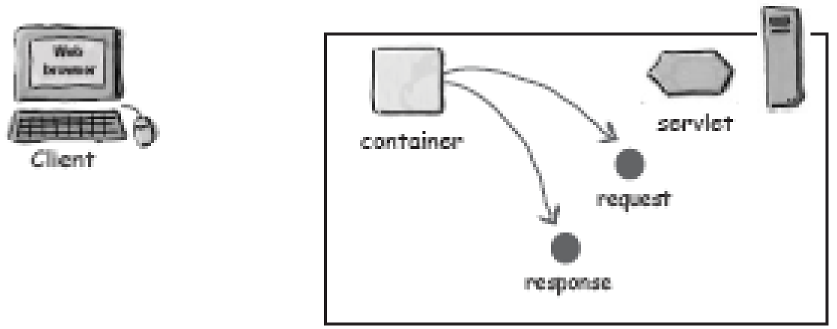
http://www.abc.com/home/index.html



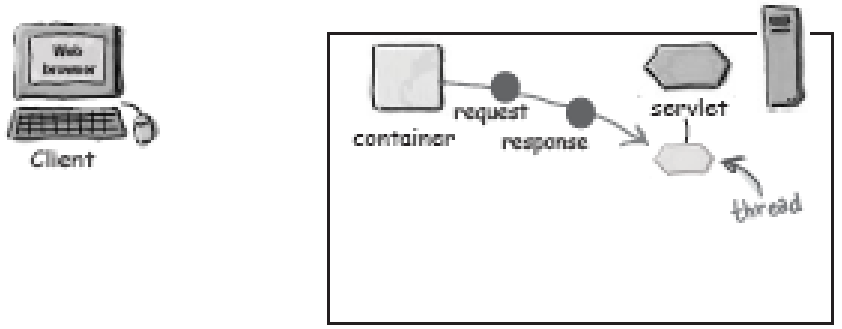
①


HTTP request
GET
Web browser
Client
container
servlet

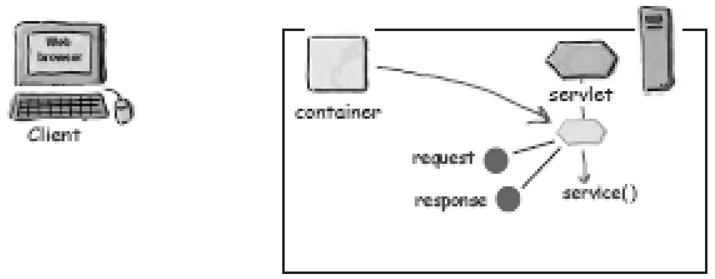
②


Web browser
Client
container
servlet
request
response

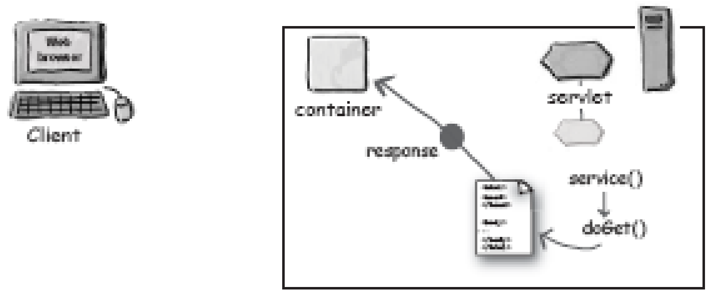
③


Web browser
Client
container
request
response
servlet
thread

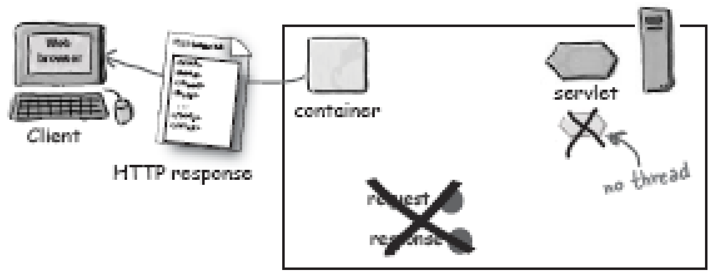
④


Web browser
Client
container
servlet
request
response
service()

⑤


Web browser
Client
container
servlet
response
service()
doGet()

⑥


Web browser
Client
HTTP response
container
servlet
no thread
request
response

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>");
  }
}
```

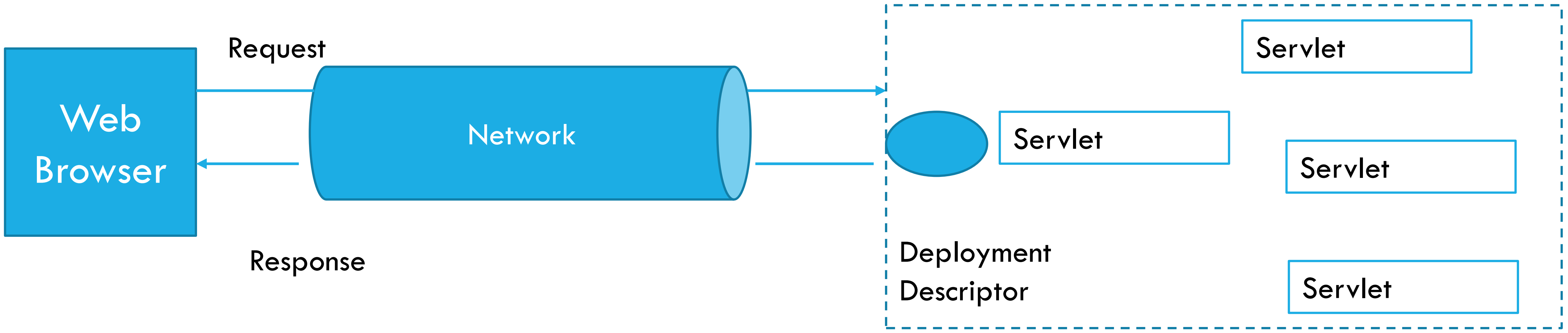
JSP

```
<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

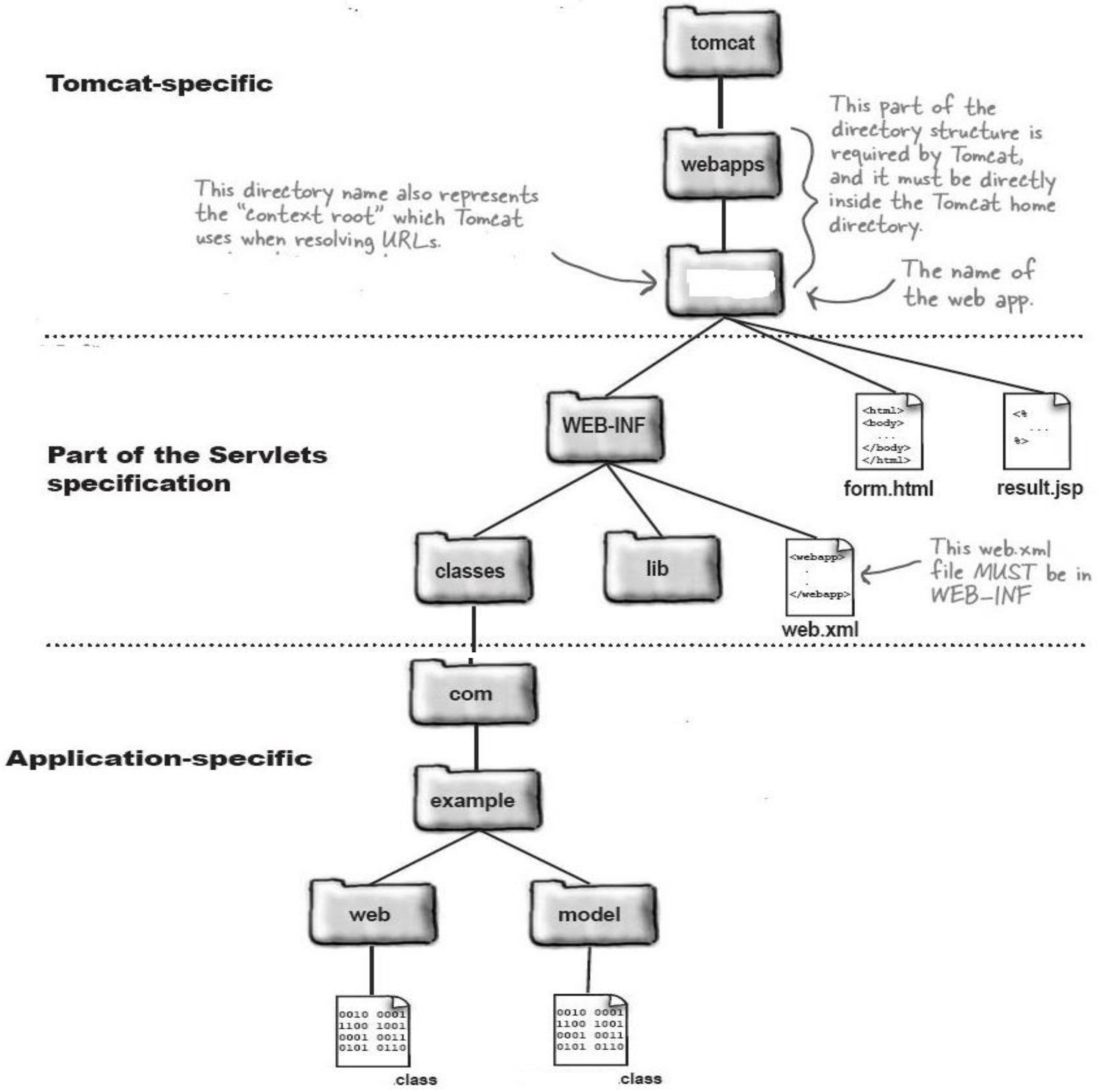
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd"    version="2.5“>

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

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

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

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*



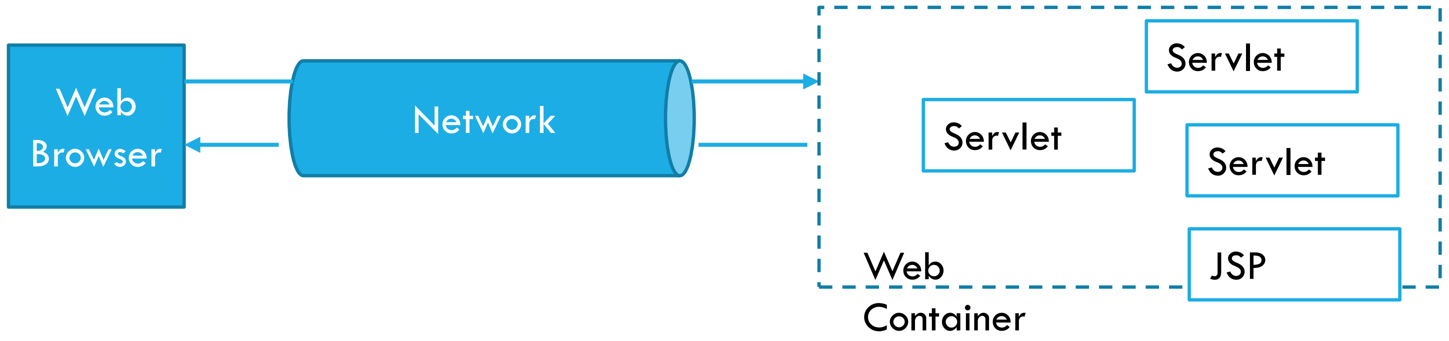
Tomcat-specific
tomcat
webapps
This part of the directory structure is required by Tomcat, and it must be directly inside the Tomcat home directory.
This directory name also represents the "context root" which Tomcat uses when resolving URLs.
The name of the web app.
Part of the Servlets specification
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
.class
.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.

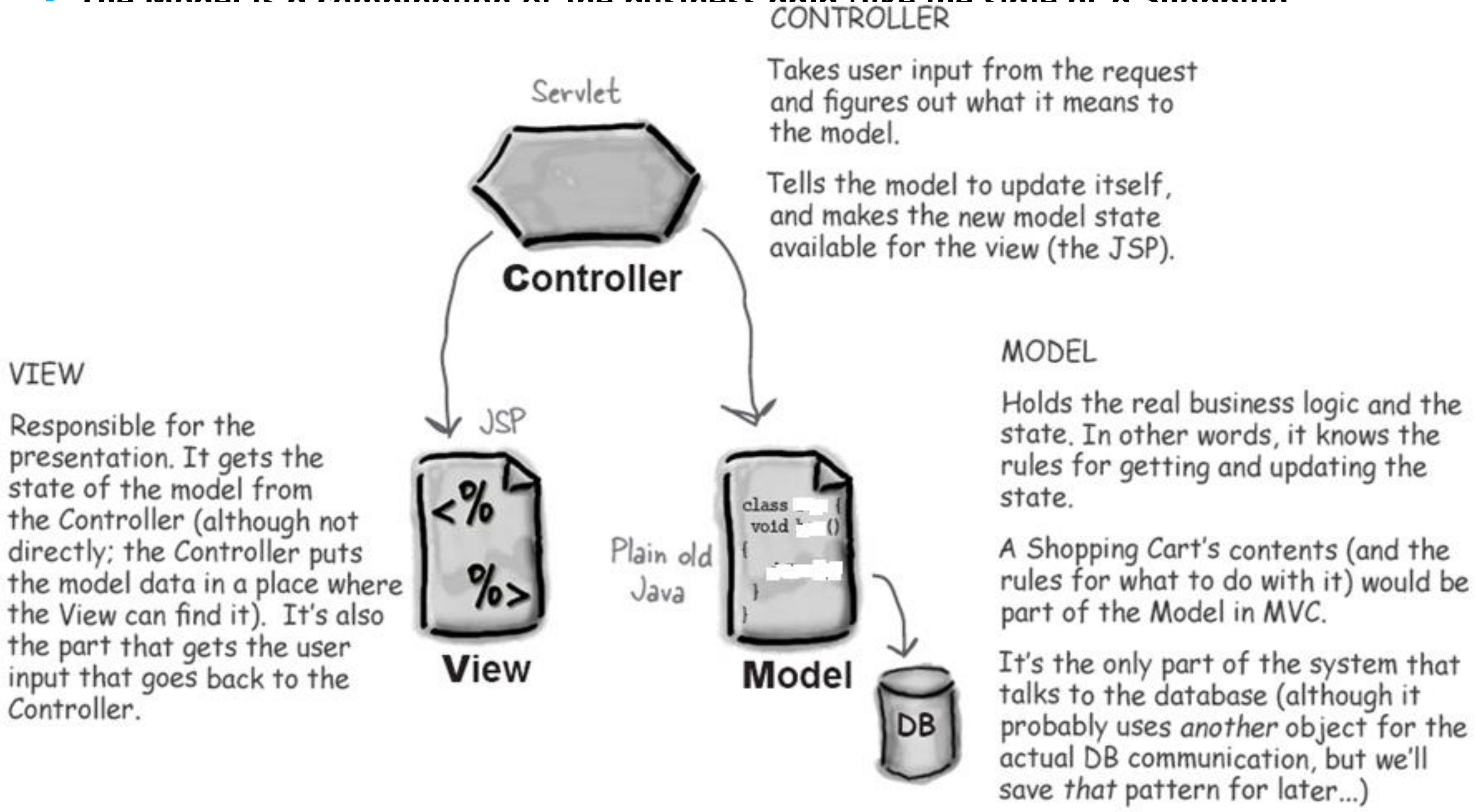
JSP Support

- 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**



CONTROLLER

Takes user input from the request and figures out what it means to the model.

Tells the model to update itself, and makes the new model state available for the view (the JSP).

VIEW

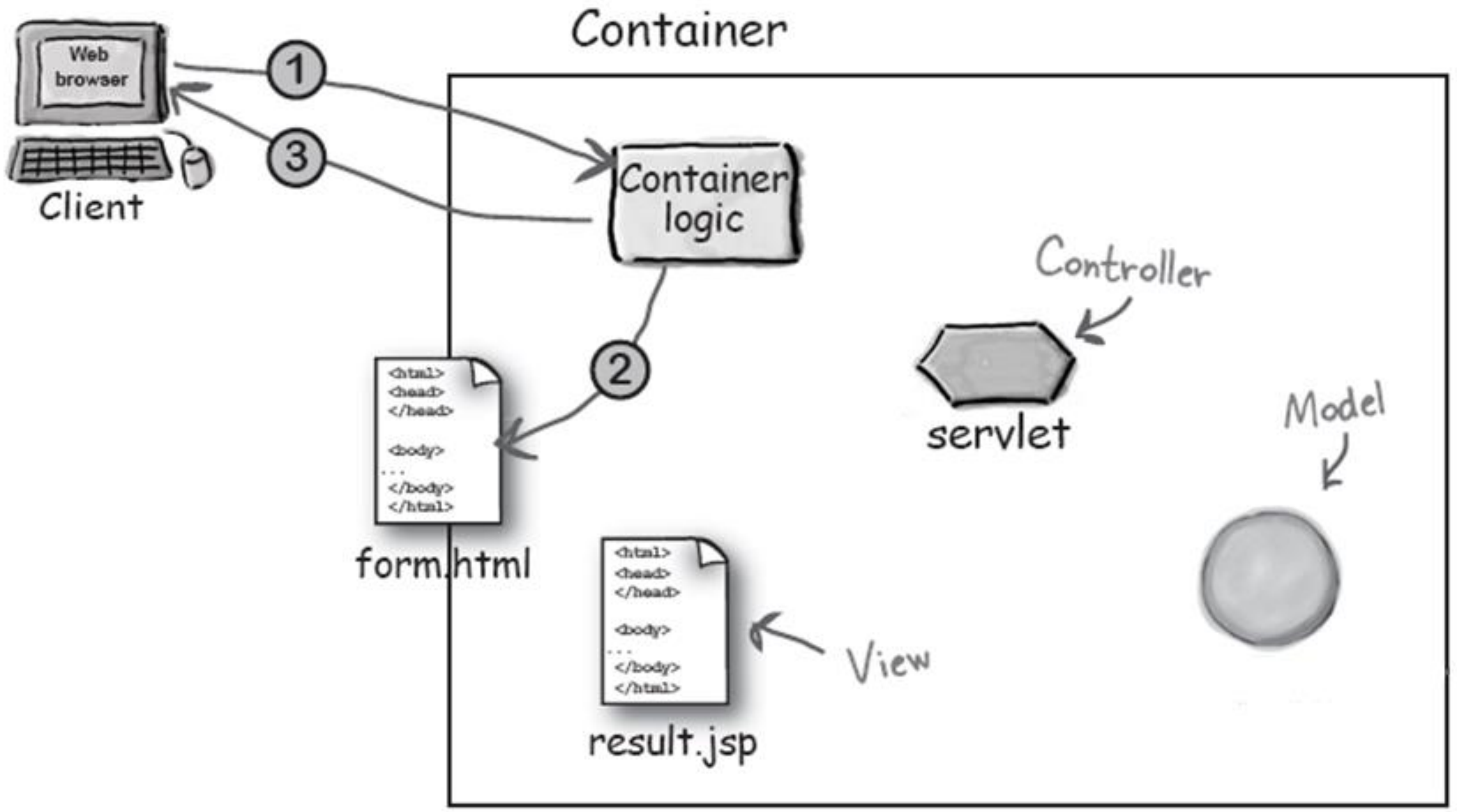
Responsible for the presentation. It gets the state of the model from the Controller (although not directly; the Controller puts the model data in a place where the View can find it). It's also the part that gets the user input that goes back to the Controller.

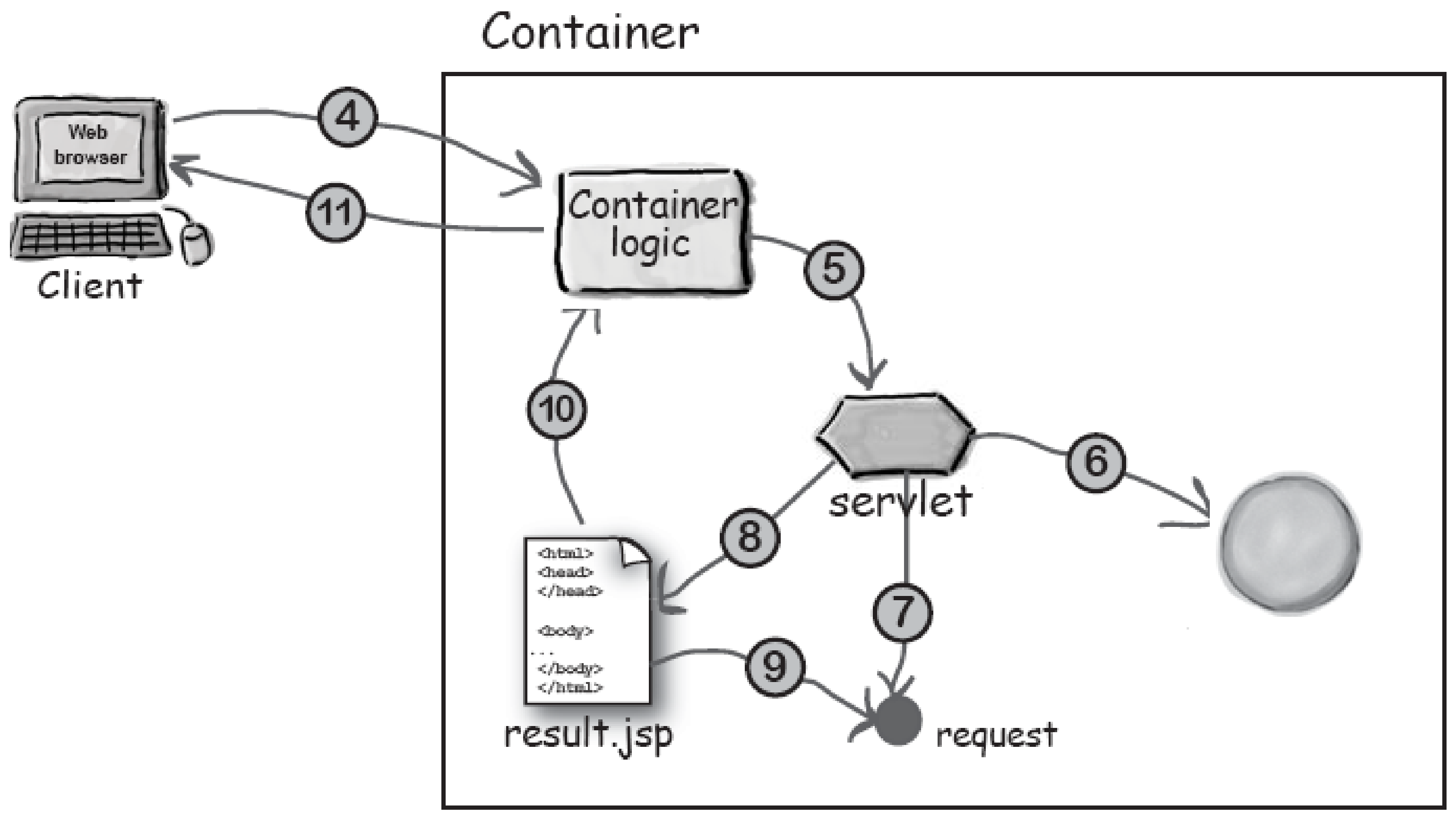
MODEL

Holds the real business logic and the state. In other words, it knows the rules for getting and updating the state.

A Shopping Cart's contents (and the rules for what to do with it) would be part of the Model in MVC.

It's the only part of the system that talks to the database (although it probably uses *another* object for the actual DB communication, but we'll save *that* pattern for later...)


Container
Web browser
Client
1
3
Container logic
2
Controller
servlet
Model
form.html
result.jsp
View
<html>
<head>
</head>
<body>
...
</body>
</html>



Servlet1→servlet2→jsp1→jsp2