

Weeks 6 and 7

Previous week:

- * Concluded XML

Today's topics:

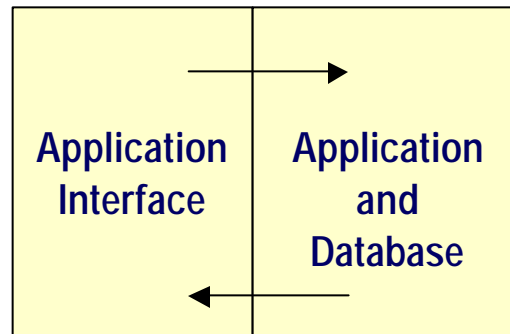
- * Introduction to Server-Side Development
- * Java Servlets, Form Processing, Session Tracking

Server-Side Application Development on Web

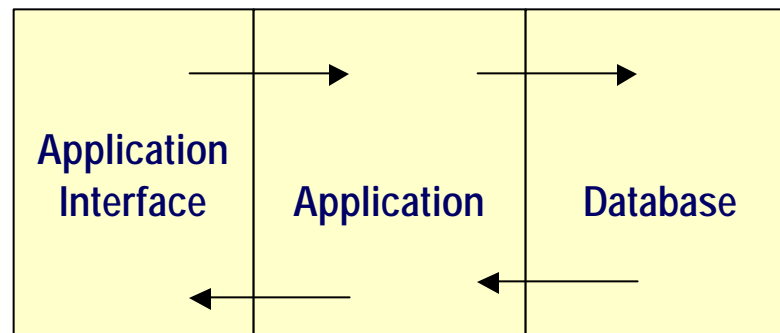
- Web servers were originally designed to serve *static* HTML pages
- Server-side application development has become widely popular on the Web due to
 - Web browsers, which present a recognisable universal interface on client side
 - Internet, which allows to connect servers with clients distributed across the network through TCP/IP protocol
 - Web servers, which run server side programs to generate HTML pages *dynamically*
- Server applications receive requests from a client interacting through a HTML page, service the request and return results in another HTML document

Traditional Application

- In a traditional application, user interface is closely coupled with the application. The database is also usually designed to serve application needs.

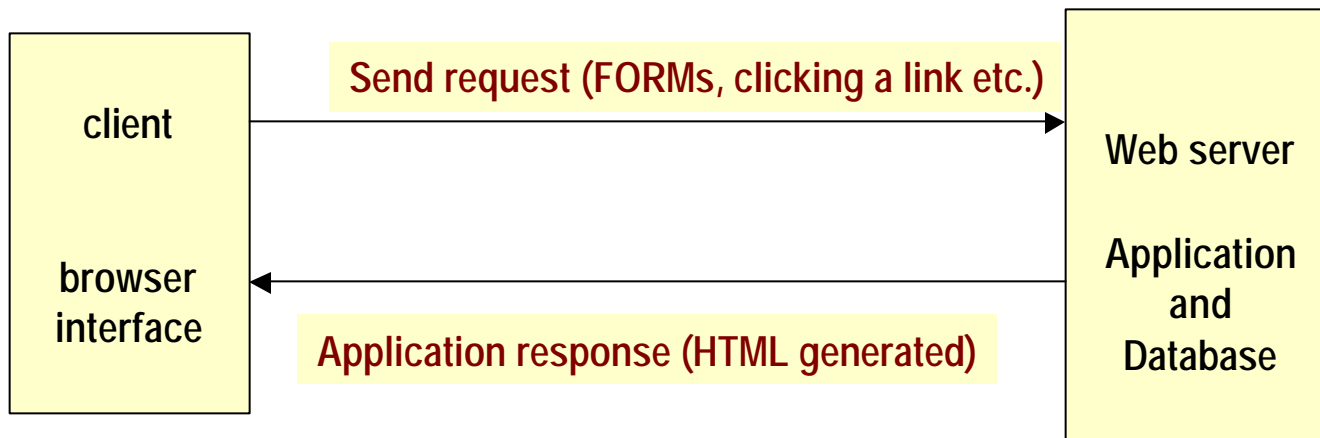


- It is possible to achieve a high level of modularity in these applications, however, user interface, database and application code are usually integrated in a single application development



Web Application

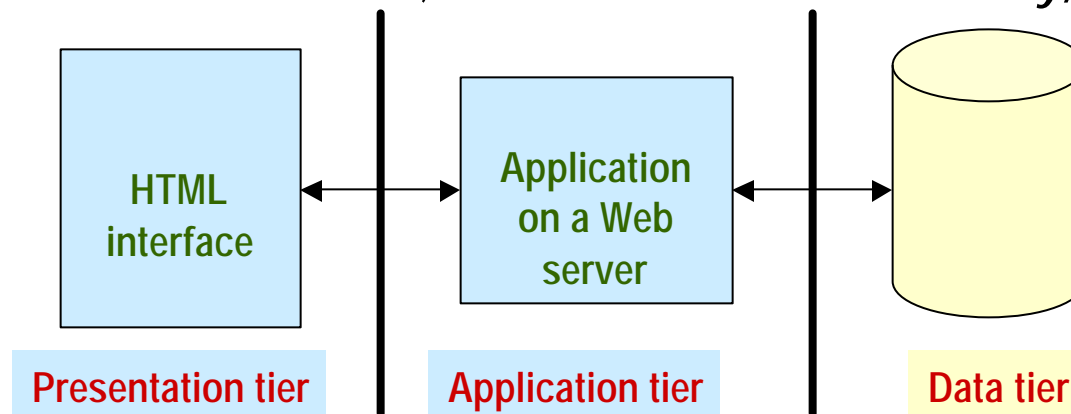
- In Web application development scenario, browser provides a universal user interface through client-side technology (HTML, JavaScript)
- The application runs on the server side to handle client requests



- Application development architectures: two-tier (above figure) and three-tier

Three-Tier Architecture (Week2 - Revisit)

- In a three-tier architecture, application may be viewed as three logical modules
 - **Top-tier**: Presentation tier on client side:
 - interacts with the user directly by accepting and displaying data using a browser (Netscape, IE)
 - **Middle-tier**: Application tier contains so called *business logic*
 - handles requests from clients, interacts with database and services requests
 - technologies: Java servlets, ASP, JSP, CGI
 - **Bottom-tier**: Data tier
 - application data (e.g. XML, Oracle, Access)
- *slow due to an additional level; resources are used efficiently, highly scalable*



Application tier

- The middle tier incorporates rules governing interpretation of data which is application dependant.

processing logic for handling user requests and extracting data from a database

- A good application should be able to
 - handle multiple user requests efficiently
 - connect various data sources if needed
 - track user sessions and store client status
 - handle security issues if needed
- Some server technologies:
 - CGI scripts
 - one of the earliest server-side options
 - resource intensive: a process created for every request (*heavy-weight processes*)
 - ASP (HTML embedding VBScript or JavaScript)
 - currently available only on IIS, PWS (Microsoft Technologies)
 - Proprietary API's from Netscape (NSAPI), Microsoft Internet Server API
 - normally written in C/C++; vendor specific
 - Java Servlets, Java Server pages (JSP) Refer to J2EE Solution
 - Wide support: from Netscape, Sun, Apache, Oracle, IBM
 - third party support (Allaire's JRun)

Client-Server Interaction (Revisit)

HTTP Request (Sending request message to Server):

- Refer to Weeks 2 and 3 lecture material on HTTP protocol; HTTP is a request-response protocol
- When a client sends a request, it consists of three parts:
 - **request line:**
 - HTTP method type (GET or POST)
 - resource name (URL)
 - protocol/version e.g. **POST /im51p/w7.html HTTP/1/1**
 - **header variables:** contains browser information (optional)
 - **message body:** in POST method request information is stored here (optional)

HTTP Response (Sending response message to client):

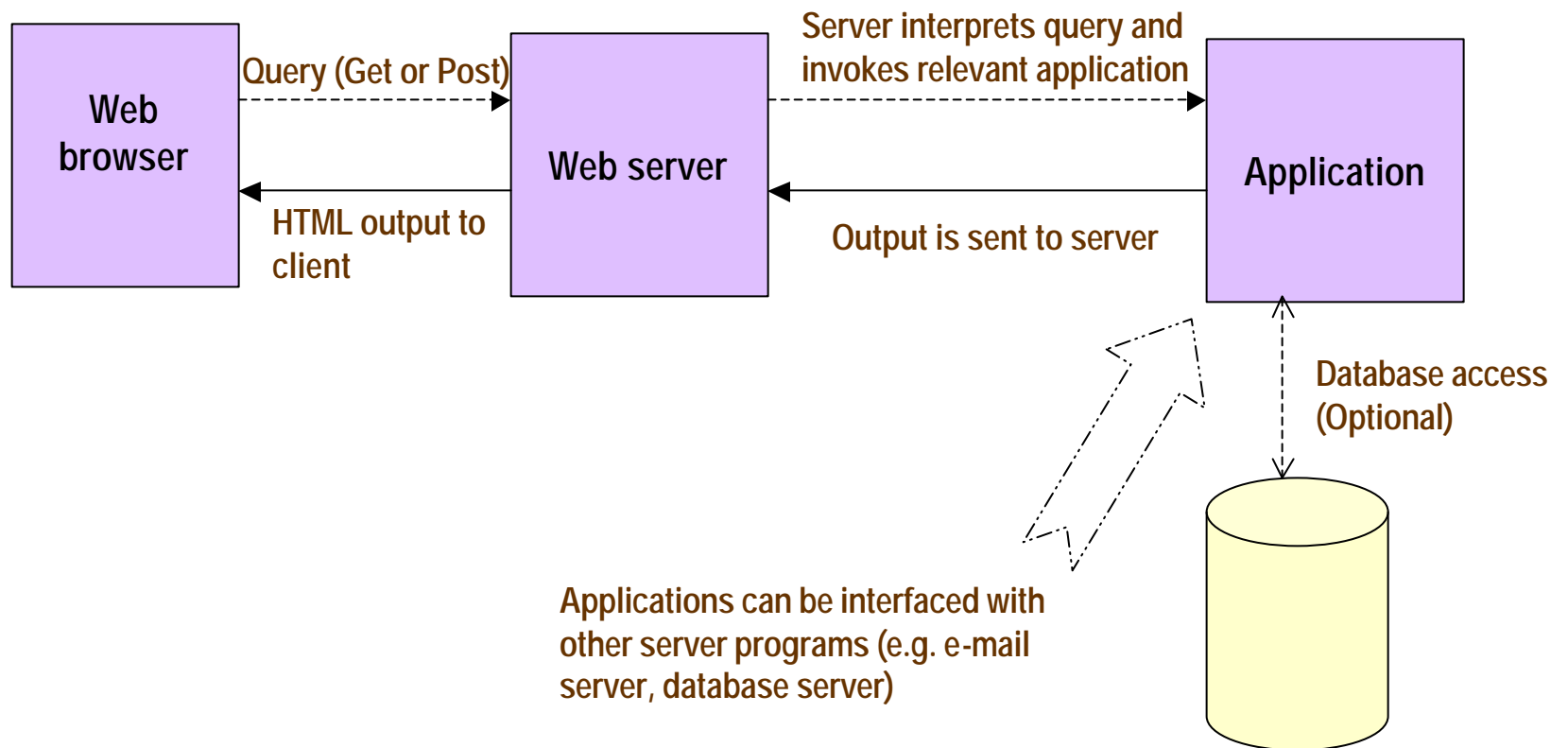
- The response sent by the server also consists of three parts:
 - response line (server protocol and status code)
 - header variables (server and response information)
 - message body (response, such as HTML)

GET and POST methods (Revisit)

- During client-server interaction, data is sent to the server primarily in two ways: GET or POST method
 - * GET: data is sent as a query string
client query is completely embedded in the URL sent to the server. The amount of query information is usually limited to 1KB. This information is stored in the request line.
 - * POST: data is sent as part of the message body
data is not visible in the URL. Large amounts of data can be sent to the server.

Server side application should handle both the methods appropriately.
Note that the client need not send data always to generate a response.

Server-Side processing



Java 2 Enterprise Edition (J2EE)

- J2EE is a collection of enterprise APIs
- Designed to aid J2EE developers in building 2/3-tier applications
- Facilitates resource pooling, transaction management etc.

J2EE suite of APIs include:

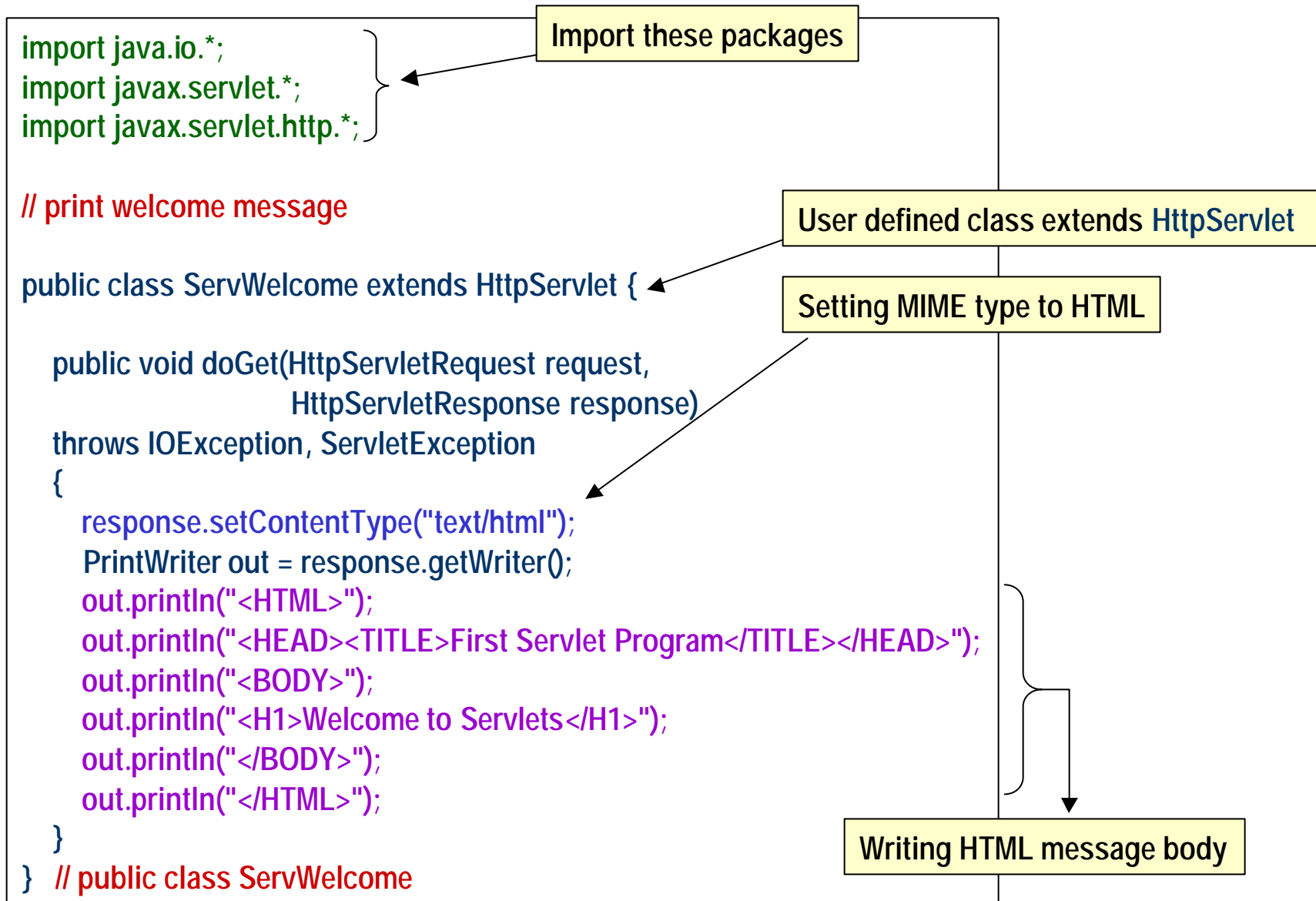
JSP	Java Sevlets	JDBC	EJB	XML	JAF
RMI	JNDI	JTA	JMS	JavaMail	

- This module mainly covers Java Servlets and JSP.

servlet development and testing:
Java Server Web Development Kit (JSWDK)

- j2ee
- WebLogic

First Servlet Example



First Servlet in Action



Java Servlets

- Server-side components
 - written in Java language
 - hosted on a Web server, run in a servlet container
 - platform and server independent; equivalents of Java **applets**
- used to
 - perform all CGI related tasks
 - dynamically create HTML pages
 - access databases through JDBC
 - email service using JavaMail
 - access distributed objects using RMI, CORBA or EJB
 - many others

Common Object Request Broker architecture



Java Servlets

- In this module, early example programs will use servlets for handling GET/POST methods of FORM based requests

- Servlet classes and interfaces are not part of core Java standard. They are defined in two packages:

- javax.servlet (generic servlet package)

- javax.servlet.http

- * extends generic servlet classes/interfaces for Web server to capture HTTP communication and servlet management

- servlet software: java.sun.com/products/servlets

- servlet development and testing (Sun's J2EE)

- java.sun.com/products/download.html

- [simt/unl.ac.uk/im51p/](http://simt.unl.ac.uk/im51p/) (Download Java Server Web Development Kit - JSWDK; simple kit.. Now obsolete)

- hosting servlets (also JSP, XML):

- simt-dev.unl.ac.uk (SIMT server)

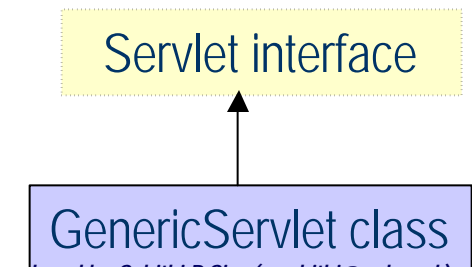
- www.mycgiserver.com (Free server accounts available)

Where to use Servlets?

- can be used with several serverlet-aware Web servers
 - Sun's Java web Server (JWS) www.sun.com
 - Apache Tomcat jakartha.apache.org
- commercial servers provide servlet support
 - IBM WebSphere, iPlanet Enterprise etc.
 - check java.sun.com/products/servlet for support currently available
- Other servers such as IIS use third-party support provided by
 - Allaire's Jrun www.allaire.com
 - New Atlanta's ServletExec www.newatlanta.com
- Servlets are used at several sites (e.g. Buy On Net www.buyonet.com)
- case studies available at Sun's java site

Servlet interface, GenericServlet

- javax.servlet package defines **Servlet** interface which must be implemented for running servlets
- **javax.servlet.GenericServlet** abstract class defines a platform-independent servlet
 - provides servlet configuration and a basic implementation of Servlet interface consisting of following three methods:
 - init** : initialising servlet, allocating resources, setting up any databases
 - service**: abstract method servicing zero or more requests from the client
 - main entry point for servlets
 - (servlet engine calls this method for each client request)
 - destroy**: releasing resources, cleanup
- service() method accepts two parameters *to communicate with the client*
 - **javax.servlet.ServletRequest**
encapsulates client's request using input stream
 - **javax.servlet.ServletResponse**
used to respond to client using an output stream



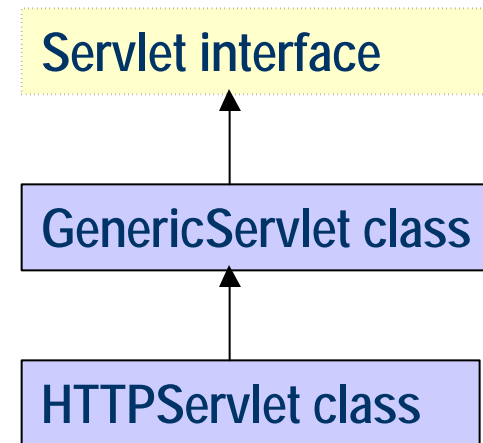
Developing HTTPServlets

- `javax.servlet.http.HttpServlet` further extends `GenericServlet` class for HTTP communication.
- Sub-classed for Web application servlets [implements `service()` method]

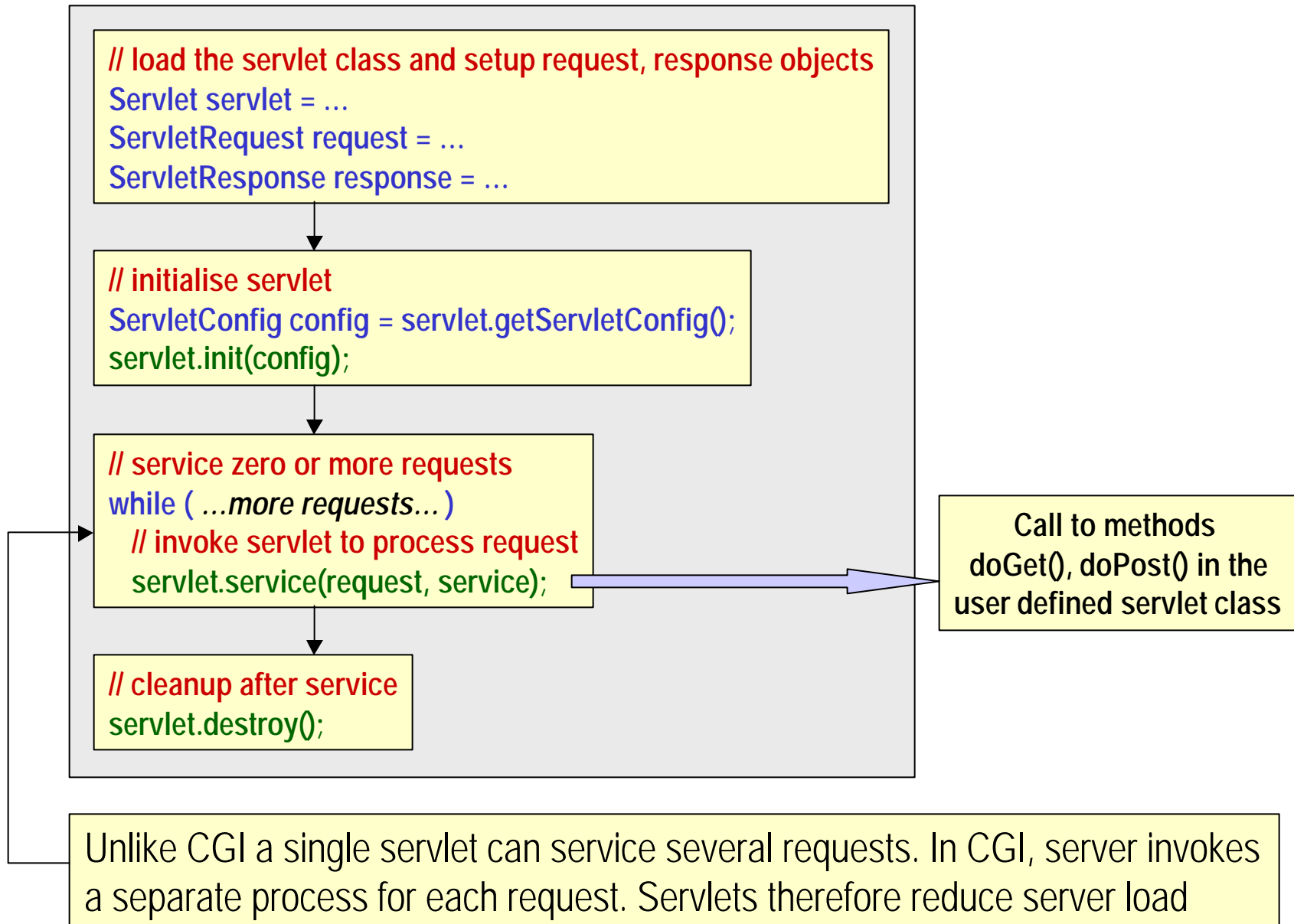
i.e. `HttpServlet` overrides the `service()` method; when called reads the HTTP method from the request and dispatches the request to relevant handling method

GET - `doGet` handling method and
POST - `doPost` handling method
xx - `doXx` handling method

- all `doXX()` methods have the same signature
- one need to implement `doGet` or `doPost`
- default implementation returns error



Servlet Life Cycle Managed by the Server



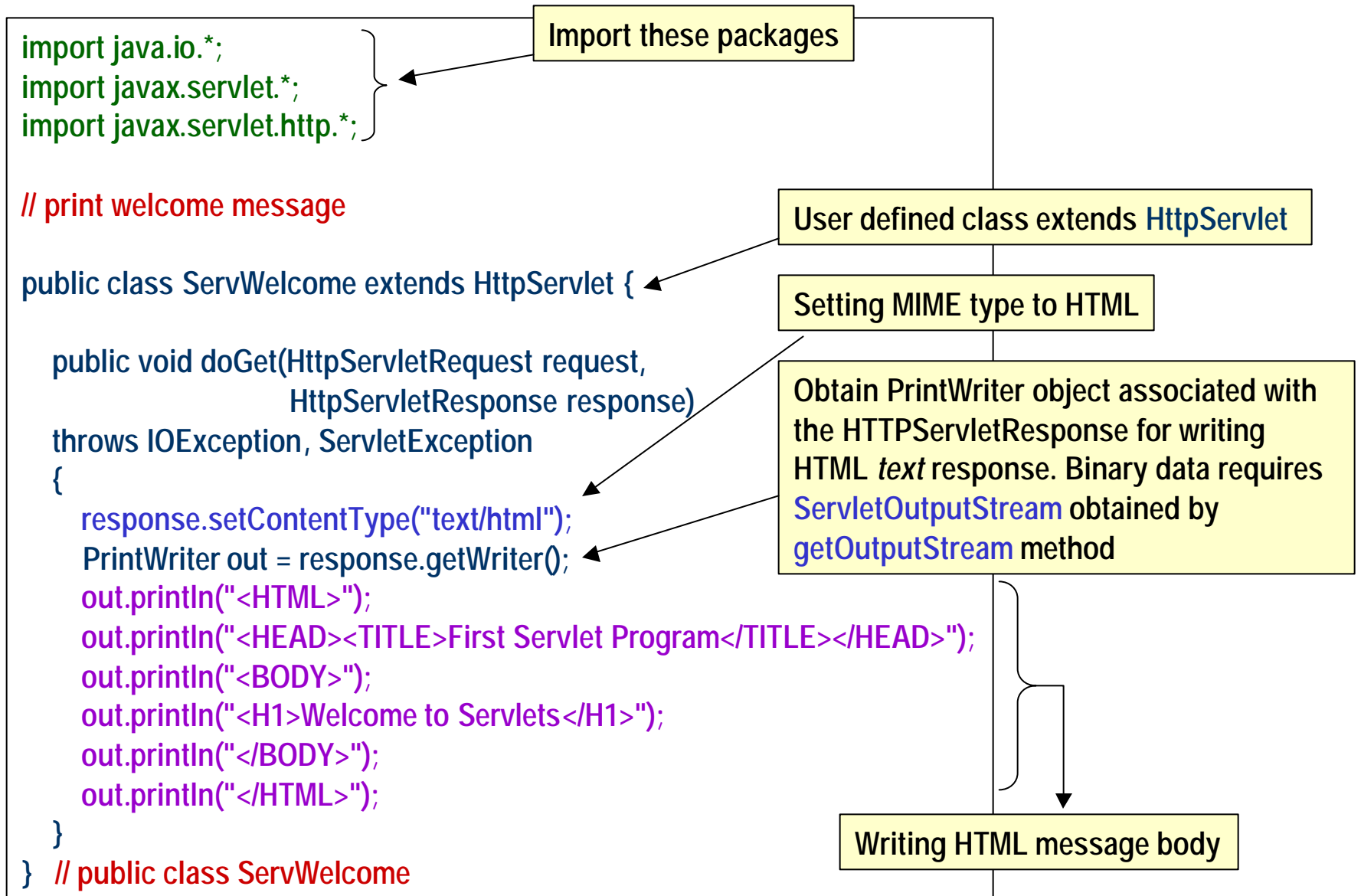
doGet() and *doPost()* methods

- Both the methods in **HTTPServlet** class define only default behaviour. The default behaviour is to send an error message as a response.
- The class implementing servlets must extend **HTTPServlet** class to set non-default response to client requests
- **doGet** has following signature:

```
public void doGet(HttpServletRequest request,  
                  HttpServletResponse response);
```
- **HttpServletRequest** represents the object containing request information from the client. **HttpServletResponse** object is used to send response to the client.
- **doPost** has identical signature

```
public void doPost(HttpServletRequest request,  
                   HttpServletResponse response);
```
- both the methods can generate **IOException** and **ServletException** which are *thrown* to the server engine.

First Servlet Example



- architecture of the javax.servlet package:

- provides interfaces and classes

- servlet interface only declares methods that manage the servlet and its communication with clients

- all servlets implement servlet interface directly or by extending a class that implements it, for example HttpServlet class.

developers (i.e. you) implement some or all of these methods

- GenericServlet defines protocol-independent servlet. HttpServlet extends it and provides HTTP-specific implementation of servlet interface

- some interfaces:

encapsulate communication



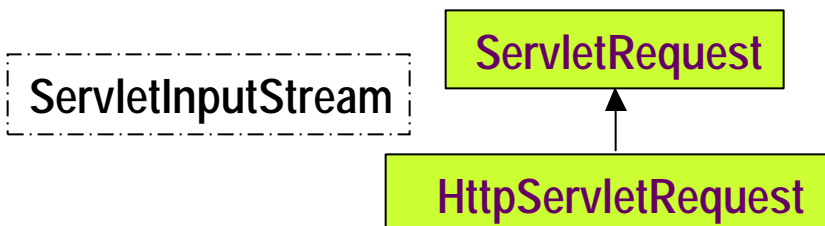
Servlet interface

GenericServlet class

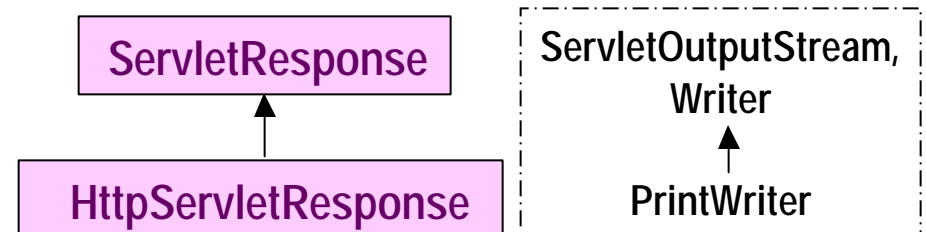
HttpServlet class

ServWelcome class

from client to server



from server to client



Recap



```
public class ServWelcome extends HttpServlet {  
  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws IOException, ServletException  
    {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>First Servlet  
                    Program</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("<H1>Welcome to Servlets</H1>");  
        out.println("</BODY>");  
        out.println("</HTML>");  
    }  
} // public class servWelcome
```

- *ServWelcome* extends *HttpServlet* class, which implements *Servlet* interface

GenericServlet implements *init* method *HttpServlet*
implements *Service* method

- *ServWelcome* overrides the *doGet* method in the *HttpServlet* class
- *doGet* is called when a client makes a GET request; simple HTML page is returned to the client as the response (GET is the HTTP default request method)
- within *doGet()*
 - user request is represented by *HttpServletRequest* object (request)
 - response to the user is represented by *HttpServletResponse* object (response)
 - text/html data is returned to the client using the *Writer* object obtained from the *HttpServletResponse* object

Servlet Program Compilation

- The file name is *ServWelcome.java* (name of the public class)
- The file is compiled like any other java program using *javac* command. If there are no errors, the compilation produces *ServWelcome.class*

The packages imported by the program *javax.servlet.** and *javax.servlet.http.** should be accessible during compilation.

Where to place servlets?

- Since servlets are run on the server, there are restrictions on where servlet class files are placed. One needs to have administrative access to install servlets on the server. (Similar to placing html files at special locations)
- If JSWDK has been installed on the machine, one can place servlets in a predefined **examples** directory or create a customised directory by following instructions
- In case of JRun, place them in a directory called servlets

Invoking Servlets

- Servlet class is invoked as below:

<http://<server-machine>:<port>/servlet/<class-name>>

- If there is ServWelcome.class in *examples* directory of JSWDK, it can be invoked as:

<http://localhost:8080/examples/servlet/ServWelcome>

- If there is servWelcome.class in *servlets* directory of JRun, it can be invoked as:

<http://simt-dev.unl.ac.uk/servlet/ServWelcome>

Servlet Output

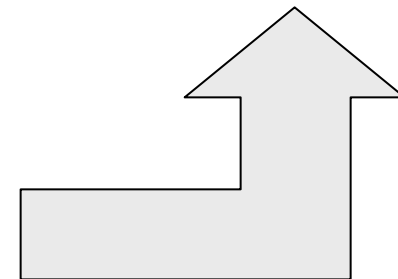
- The servlet generates a welcome message:

Welcome to Servlets

Output from HTML

```
public class ServWelcome extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws IOException, ServletException  
    {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>First Servlet Program</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("<H1>Welcome to Servlets</H1>");  
        out.println("</BODY>");  
        out.println("</HTML>");  
    }  
} // public class servWelcome
```

<HTML>
<HEAD><TITLE>First Servlet Program</TITLE></HEAD>
<BODY>
<H1>Welcome to Servlets</H1>
</BODY>
</HTML>

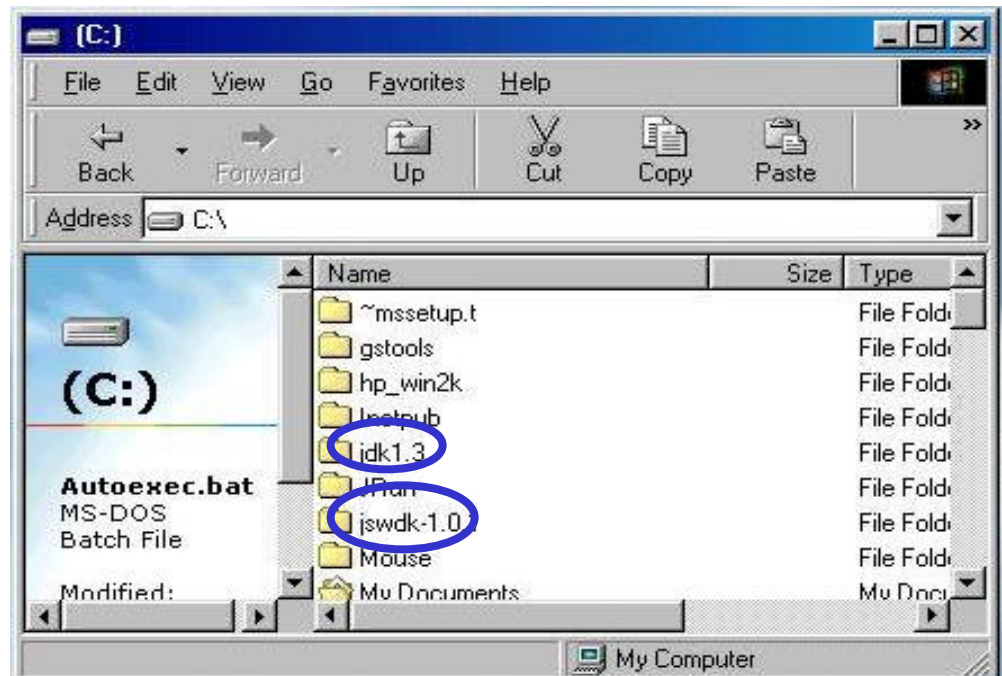


HTML generated by the
servlet (in the browser select
[view->source](#) to see this)

Testing servlets on home machine (1)

- download iswdk (from Sun's Web site or ask me for CD), unzip to install
- download jdk (latest version) and unzip to install

- check for autoexec.bat in c:\
- add the following three lines (change depending upon your set up)



`PATH="%PATH%;C:\jdk1.3\bin"`

`set CLASSPATH=C:.\;C:\jswdk-1.0.1\lib\servlet.jar;C:\jswdk-1.0.1\lib\jspengine.jar;`

`set JAVA_HOME=C:\jdk1.3`

- restart the machine if required or run autoexec.bat by typing its name at the MSDOS prompt **i.e. c:\>autoexec**

Testing servlets on home machine (2)

- Open MSDOS window; change directory to JSWDK
`c:\>cd jswdk-1.0.1`
- start the web server by typing
`c:\jswdk-1.0.1>startserver`
- If installed correctly, a new MSDOS window opens and displays appropriate messages (do not close this window)
- Open Netscape browser; type `http://localhost:8080`
- You will see the introductory page of JSWDK kit.
- Follow servlets, JSP links and explore examples
- Read the instruction manual
- Now, save your first servlet as `ServWelcome.java` and compile it using `javac`
i.e. `c:\im51p\servlets>javac ServWelcome.java`
- Following successful compilation, copy class file to
`c:\jswdk-1.0.1\examples\web-inf\servlets>` directory
- To run the servlet, type `http://localhost:8080/examples/servlet/ServWelcome`
- To stop the server, `c:\jswdk-1.0.1>stopserver`

Testing servlets in SIMT Labs (1)

ISS has kindly helped me in delivering a Java Console on every student machine. The console menu will allow the students to compile Java programs, start the JSWDK web server, run your Web application (e.g. servlet) and stop the jJSWDK server. Please follow the instructions to perform these operations.

1. Follow START --> PROGRAMS --> COMPILERS --> JAVA DEVELOPMENT KIT --> JAVA CONSOLE

2. A window pops up with a top menu bar. Look for menu item "Java"

3. Look for the following menu options (sequence may be different)

Java

Compile

Run

Start WebServer

Run Web Application

Stop WebServer

Testing servlets in SIMT Labs (2)

4. Open a java file

Follow: File --> Open -->

File will be loaded into the editor

5. Compile the file (classpath is set automatically)

Follow: Java --> Compile

If compiled successfully, a class file is created. Follow instructions in the compile window.

6. Start the Web server

Follow: Java --> Start WebServer

JSWDK Web server will be started (you will find a DOS window minimised in your toolbar. Do not close this window. Keep it minimised).

7. Start the Web application

Follow: Java --> Start Web Application

You will have to choose a class file to open (e.g. ServWelcome.class) from `jswdk\examples\web-inf\servlets\` directory. Please note that you should place all your servlet class files in this directory only.

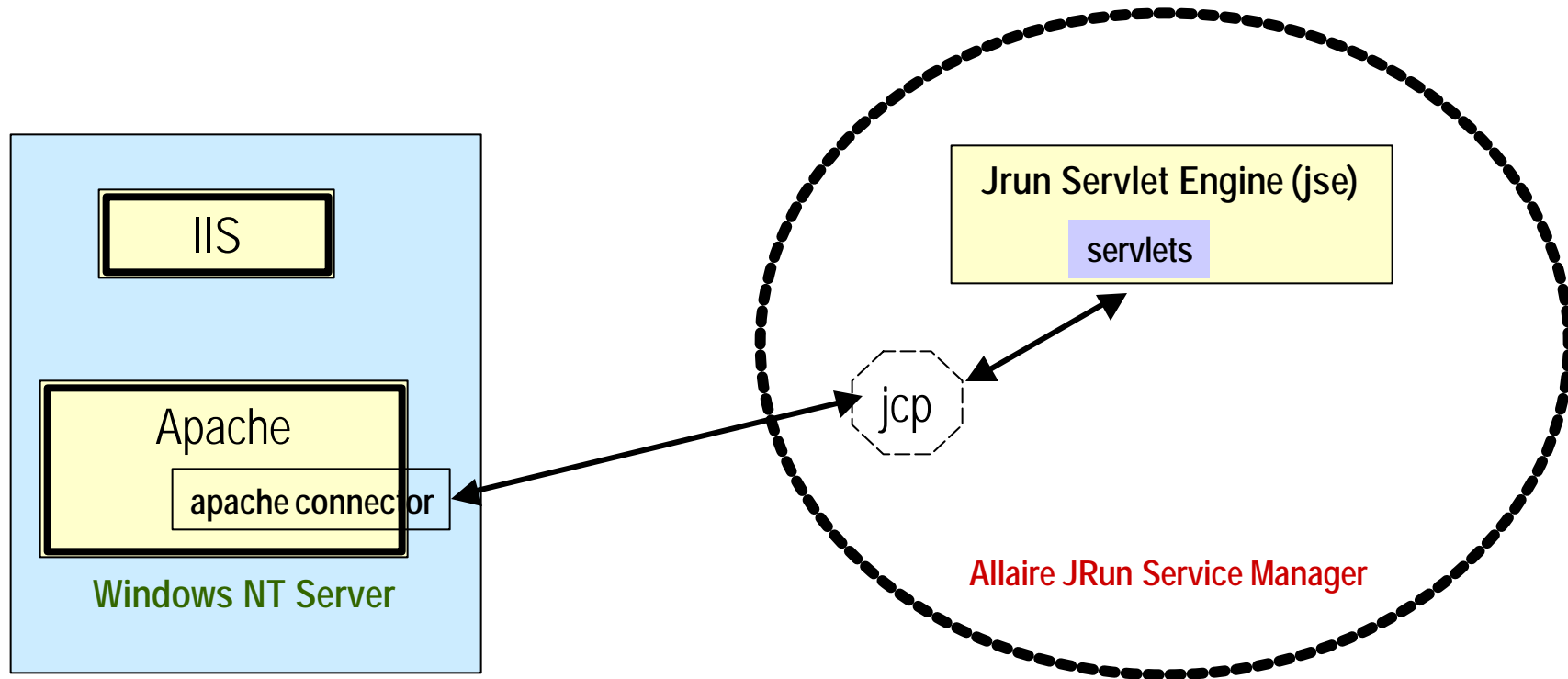
Testing servlets in SIMT Labs (3)

8. You will see the Web application running at the localhost
 9. Before you logout, make sure to stop the server from Java --> Stop WebServer
Also save all your work. The saved class files are automatically placed in the right directory for the JSWDK server to run the application.
 10. NOTE: If you recompile your java program, to see the changes you have to stop and restart the JSWDK server.
 11. Following successful compilation and testing, you may upload the class file to simt server
(servlets@ftp:simt-dev.unl.ac.uk)
 12. In case you are unsuccessful, contact Serkan or Chris or myself.
- DO NOT APPROACH ISS HELPDESK ON THIS ISSUE**
13. More space for im51p students (check your space)

When you first invoke start WebServer menu option, JSWDK installed on n: drive, is copied to your location and is installed. To allow the complete installation of the server, you require sufficient space on h: drive. ISS will allocate 10MB more per student for this purpose. Try not to overuse this space. If you are unsuccessful in running the server, first check the available space. Try to free some space and restart the server.

Deploying servlets on simt - dev

Server Environment - FTP your servlets using simt-dev account and place them under *servlets* directory



Jcp -Jrun connector proxy service: receives servlet requests and sends them to jse

jse - responsible for handling servlet requests

Jrun supports servlets, JSP and XML technology

servlet development and testing:
Use J2EE (or JSWDK), WebLogic or Tomcat

Step-Wise

1) write the Servlet

- import necessary java classes
- inherit from GenericServlet or HttpServlet class
- override the service method; doGet or doPost methods
- save the file with .java extension

2) compile the servlet (you can create a batch file; see im51p Web page)

- set the classpath; make sure servlet.jar is included in the class path (also tools.jar)
- invoke javac (e.g. `javac ServWelcome.java`; `javac -cp xx/servlet.jar ServWelcome.java`)

3) install the servlet

- JSWDK: place it under examples\web-inf\servlets directory or
- SIMT server: `ftp://servlets@simt-dev.unl.ac.uk`

4) test the servlet (first start the server; in JSWDK - run startserver.bat file)

- `http://localhost:8080/examples/servlet/ServWelcome` or
- `http://simt-dev.unl.ac.uk/servlet/ServWelcome`

Recap



```
public class ServWelcome extends HttpServlet {  
  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws IOException, ServletException  
    {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>First Servlet  
                    Program</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("<H1>Welcome to Servlets</H1>");  
        out.println("</BODY>");  
        out.println("</HTML>");  
        out.close();  
    }  
} // public class servWelcome
```

- *ServWelcome* extends *HttpServlet* class, which implements *Servlet* interface

GenericServlet implements *init* method *HttpServlet* implements *Service* method

- *ServWelcome* overrides the *doGet* method in the *HttpServlet* class
- *doGet* is called when a client makes a GET request; simple HTML page is returned to the client as the response (GET is the HTTP default request method)
- within *doGet()*
 - user request is represented by *HttpServletRequest* object (request)
 - response to the user is represented by *HttpServletResponse* object (response)
 - text/html data is returned to the client using the *Writer* object obtained from the *HttpServletResponse* object

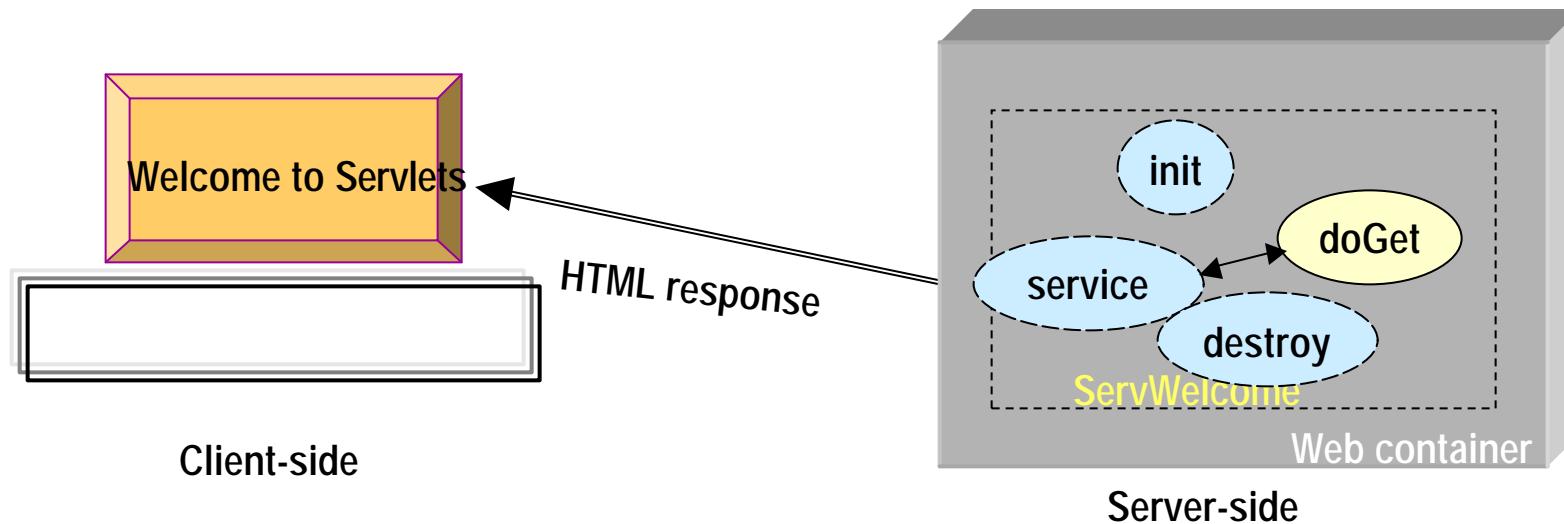
```

public class ServWelcome extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>First Servlet
                    Program</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Welcome to Servlets</H1>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
} // public class servWelcome

```

Only HTML Response



HttpServletRequest class

- **request**: instance of `HttpServletRequest` class which contains methods for accessing HTTP-specific information
- also information about the request and server environment: names of parameters passed by the client, the protocol used by the client etc.

`String request .getMethod()` - obtain method name GET or POST
`String request .getParameter(String name)` - get a parameter with the given name in the request (form data) and return its value
`String request .getRemoteHost()` - client's host name
`String request .getRemoteAddr()` - IP address of client
`String request .getServerName()` - server name
`int request .getServerPort()` - port on which server is listening
`String request .getHeader("User-Agent")` - name of client's web browser
`String request .getHeader("Referer")` - URL of page that called the servlet

some more methods ...

HttpServletResponse class

- **response**: instance of `HttpServletResponse` class which contains methods for manipulating HTTP-specific information
- allows the servlet to set the content length, status codes and MIME types of the reply

`void response.addCookie(Cookie cookie)`

`void response.sendError(int status)` - standard HTTP status codes

`void response.setStatus(int status)` - send HTTP status codes that are not errors

`void response.sendRedirect(String location)` - sends a redirect response to the client

some more methods ...

```

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

// print environment data of client/server
public class ServEnvData extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        response.setContentType("text/html"); // should be done before you use getWriter()
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Environment Data</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Servlet Environment</H1>");
        out.println("<B>Method:</B>" + request.getMethod() + "<BR>");
        out.println("<B>Client Host:</B>" + request.getRemoteHost() + "<BR>");
        out.println("<B>Client's IP Address:</B>" + request.getRemoteAddr() + "<BR>");
        out.println("<B>Server Name:</B>" + request.getServerName() + "<BR>");
        out.println("<B>Server Port:</B>" + request.getServerPort() + "<BR>");
        out.println("<B>Client's Browser:</B>" + request.getHeader("User-Agent") + "<BR>");
        out.println("<B>Client's URL:</B>" + request.getHeader("Referer") + "<BR>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    } } // public class ServEnvData

```

ServEnvData.java

Servlet Environment

Method:GET

Client Host:localhost

Client's IP Address:127.0.0.1

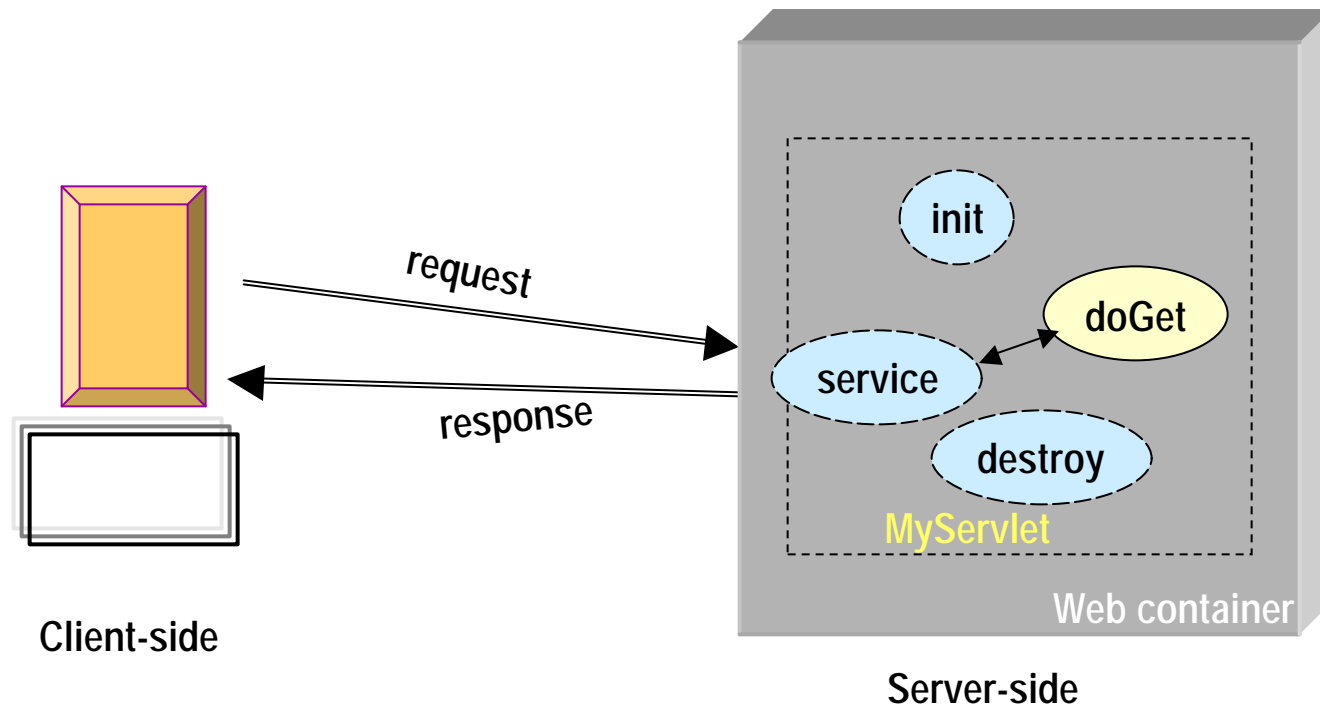
Server Name:localhost

Server Port:8080

Client's Browser:Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)

Client's URL:null

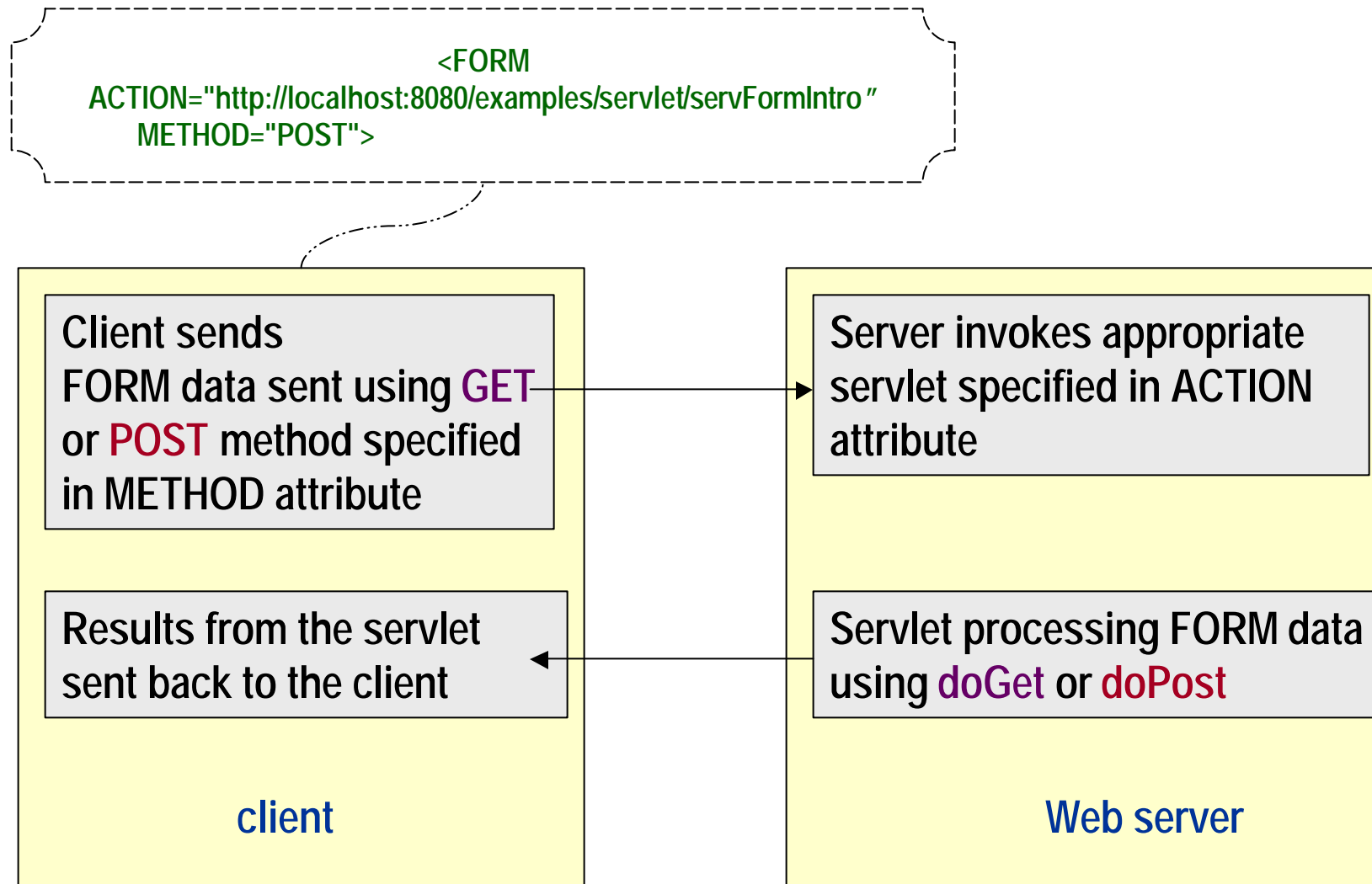
Sample HTML output



Receives Request and generates Response

Processing Forms Using Servlets

- Java technology has offered several client-side benefits such as applets
 - dynamically extend client functionality
- servers are now taking the advantage of Java technology
 - dynamically extend network server functionality
- servlet code can be downloaded into a currently operating server to extend its behaviour in order to provide new or temporary services to the clients
- servlets provide effective **server-side form processing:**
- forms provide a user interface for a Web application; three steps in form processing:
 - specify the form with input fields and widgets to be presented to the users (text fields, drop-down menus, checkboxes etc.)
 - perform client-side data validation
 - submit data to the server-side component, that is servlet, for processing



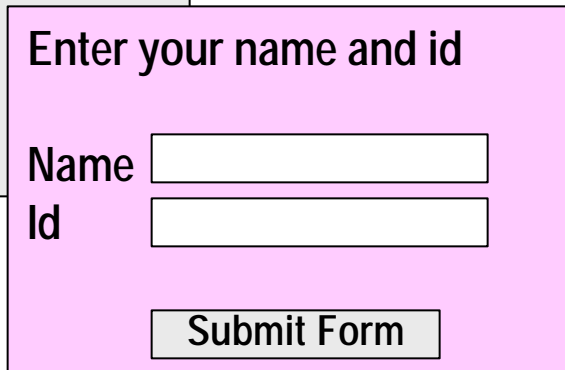
Sending Data Using FORMS

FORM Example 1

```
<HTML>
<BODY>
<FORM ACTION="http://localhost:8080/examples/servlet/servFormIntro"
      METHOD="POST">
<B>Enter your name and id</B><BR>
Name </B><INPUT TYPE="text" NAME="User Name" SIZE=30> <BR>
Id   </B><INPUT TYPE="text" NAME="User Id" SIZE=30> <BR><BR>

<INPUT TYPE="submit" VALUE="Submit Form">
</FORM>
</BODY>
</HTML>
```

FORM data



Enter your name and id

Name

Id

- The FORM invokes the servlet **servFormIntro** through the ACTION attribute
- The method used is **POST**
- On clicking **Submit Form** button the FORM data is sent
- Each of the FORM entities should be given name to be used by the servlet

Extracting FORM data

- If GET method, FORM data is sent to the server as a part of URL: if the Name is T.Johnson and Id is 346789, the URL looks like
<http://localhost:8080/examples/servlet/servFormIntro?User+Name=T.Johnson&User+Id=346789>

- `HttpServletRequest` class provides methods for extracting name-value pairs

`Enumeration getParameterNames()` - returns all parameter names in the request as an Enumeration variable

`String getParameter(String name)` - returns value of the parameter of a given name in the request

- Enumeration class provides methods for obtaining elements one by one

`boolean hasMoreElements()` - returns true if there are more elements to enumerate
`nextElement()` - returns next element.

FORM Extraction Example 1

```
public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException
{
    response.setContentType("text/html");
    StringBuffer buf = new StringBuffer();
    Enumeration e = request.getParameterNames();
    PrintWriter out = response.getWriter();
    buf.append("<HTML>");
    buf.append("<BODY>");
    buf.append("<H1>Student Selection</H1>");
    while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        String value = request.getParameter(name);
        buf.append(name + " = <B>" + value + "</B><BR>");
    }
    buf.append("</BODY>");
    buf.append("</HTML>");
    out.println(buf.toString());
    out.close();
} // doPost
```

StringBuffer used to append output data.

nextElement returns an object. cast it as required

Enumeration of elements

StringBuffer written into output

The servlet prints name value pairs in the output

FORM Example 2

```
<HTML><HEAD><TITLE>EXTRACTING FORM VALUES: SELECT MENU</TITLE></HEAD>
<BODY><FONT SIZE="4">
<CENTER>PLEASE SELECT TOPICS OF INTEREST<BR>
<FORM ACTION="http://localhost:8080/examples/servlet/FormMultiValuesDemo"
      METHOD="POST">
<SELECT NAME="topic" MULTIPLE>
<OPTION VALUE="a">APL</OPTION>
<OPTION VALUE="b">BASIC</OPTION>
<OPTION VALUE="c">C++</OPTION>
<OPTION VALUE="e">Eiffel</OPTION>
<OPTION VALUE="f">FORTRAN</OPTION>
<OPTION VALUE="j">Java</OPTION>
<OPTION VALUE="p">Pascal</OPTION>
</SELECT><BR>
<INPUT TYPE="submit" VALUE="Submit Selection">
<INPUT TYPE="submit" VALUE="Submit Selection"><BR>
</FORM></CENTER>
<HR></BODY></HTML>
```

If the user selected APL, BASIC and C++ in the list box and submitted the request, the transmitted request would include the following text:

topic=a&topic=b&topic=c

FORM Example 2

```
topic=a&topic=e&topic=j
```

Because there are three name-value pairs with the same name, *getParameter("topic")* can not be used. Instead, use *getParameterValues* method provided by Servlet API that returns a String array that preserves the order of the selections as presented on the page.

In this example, the array will have three String objects with the values "a", "b" and "c".

FormMultiValuesDemo

- 1) import necessary packages
- 2) Start building *FormMultiValuesDemo* servlet class and the **post** method
- 3) Set the content-type
- 4) Get the *PrintWriter* object using *getWriter()* method provided by the response object.
- 5) Get the *String* array of values using *getParameterValues("topic")* of request object.
- 6) Start writing the HTML page to a String buffer
- 7) Print the values in the array
 Use a for loop; get the length of the array and print its elements
- 8) Complete the HTML page
- 9) Close **post** method and *FormMultiValuesDemo* servlet class

Session Tracking

Multiple Requests from a Single User or Session

In the servlet model discussed so far:

- a servlet receives a request, extracts parameters and generates a response
- each request is treated independently; the response to each request depends on its parameter values only (i.e. not any earlier request)
- servlet does not know how to relate requests (a series of requests may originate from the same user over a time period)

Session: defined as interaction between a client and a web server that may span a series of requests over some period of time

however

- sessions are common in many business applications: on-line shopping cart, web based e-mail (e.g. hotmail), on-line banking
- a user generally sends some identification and the business application at the server side is expected to 'remember' this when the user sends subsequent requests

Need for Session Tracking

Servlet response should depend not only the current request, but also on the information gathered from the previous requests

- Since a servlet can handle multiple requests at the same time it must also distinguish between different users: for example, in a web mail application, the mailbox of a user should not be sent to some other user

bad news: **Remember HTTP is stateless**

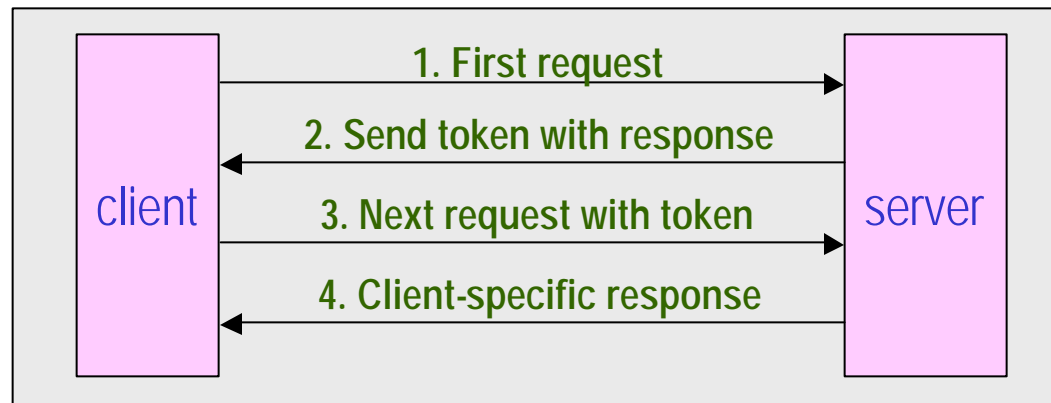
The request and response process uses HTTP protocol which does not support session tracking

- It is necessary to keep track of actions of a user, that is, one need to implement techniques to track user sessions

Principle of Session Tracking

In order to allow the server application to recognise a client's multiple requests:

1. client sends an initial request
2. server (i.e. servlet, in this case) sends a unique token (for example a string value) to the client
3. when the client sends another request, the token is sent back to the server
4. the token value is used by the servlet to identify the client and the details of the session. A response specific to the client is sent back



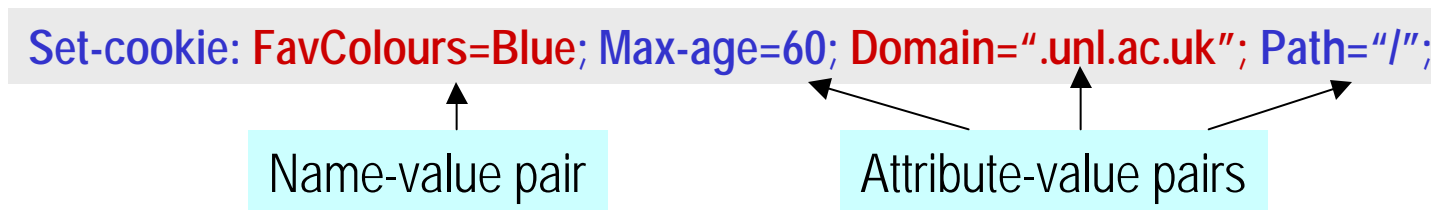
There are several approaches to implement token exchange mechanism

Session Tracking Techniques

- **URL rewriting**: the token is sent as part of the URL. When the server dynamically generates a page for the client, the token is embedded in the every URL in the page as an extra parameter. When the client clicks on any of the URLs, the token is retransmitted to the server
- **Hidden Form Fields**: Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server, the hidden fields act as tokens
- **cookies**: cookies (originally invented by Netscape) are exchanged in header fields of request and response information. One advantage with cookies is that it does not involve modifying dynamically generated URLs or forms. It is possible that some users may choose to disable cookies; in this case token exchange strategy fails.
- **Secure Socket Layer Sessions**: In the process of establishing encrypted connection between the client and the server, session keys are generated by the server and are used based on HTTPS protocol.

Cookies

- Web servers send cookies to the client in response header
- A cookie in the header is of following form:



- A cookie has name-value pair along with additional attribute-value pairs.
- When client's browser receives a cookie, it can either accept and store it or reject it.
- In case the browser accepts the cookie: If there is a request from the browser within next 60 units of time to `www.unl.ac.uk` server, the cookie information is included in the request header. The server can read cookie and identify the client using the cookie information

Extracting Cookies in a Servlet

A servlet can extract cookies from the request header.

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
{
    ....
    // extract cookies
    Cookie cookie, cookieFound = null;
    Cookie[] cookieArr = request.getCookies();
    if (cookieArr != null) {
        for (int i = 0; i < cookieArr.length; i++) {
            cookie = cookieArr[i];
            if (cookie.getName().equals("FavColours")) {
                cookieFound = cookie;
            }
        }
    }
    ....
} // doGet()
```

Retrieve cookies from the request header using `getCookies()` method (returns an array) of `HttpServletRequest`.

Search for a cookie by the name "FavColours". Cookie name is obtained by Cookie method `getName()`.

Setting Cookies in a Servlet

Cookies can be set in response header and sent to the client browser. Following code fragment creates a new cookie for the client if no cookie found in the client request header.

Create a new cookie with a name ("FavColours"), and a value ("Blue")

```
if (cookieFound == null) {  
    cookie = new Cookie("FavColours", "Blue")  
    cookie.setMaxAge(300);  
    response.addCookie(cookie);  
}
```

Setting cookie age to 5 minutes using Cookie method `setMaxAge()`

Adding cookie to response header using `addCookie()` method of `HttpServletResponse`. When client receives the response, the cookie will be stored at the client's site provided that cookies are enabled by the browser

Sending Client Specific Response

If a cookie is found in a request header, based on cookie's name and value, which are assumed to be uniquely set for the client, a customised response can be sent

```
StringBuffer buf = new StringBuffer();
```

```
.....
```

```
if (cookieFound != null) {
```

```
    buf.append("<B>Welcome to Favourite colour cookie: " + cookieFound.getValue() + "! </B><BR>");
```

```
    buf.append("<B>Cookie Name: " + cookieFound.getName() + "</B><BR>");
```

```
    buf.append("<B>Value: " + cookieFound.getValue() + "</B><BR>");
```

```
}
```

```
response.setContentType("text/html");
```

```
PrintWriter out = response.getWriter();
```

```
out.println(buf.toString());
```

} Browser
display

Welcome to favourite colour cookie: Blue!
Cookie Name: FavColours
Value: Blue

The code fragment writes cookie's name and value in the response. When the servlet is run first time, cookie is set. For subsequent requests within 5 minutes, cookie's name and value are printed

```

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

public class ServCookie extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException,
        ServletException
    {
        // extract cookies
        Cookie[] cookieArr = request.getCookies();
        Cookie cookie;
        Cookie cookieFound = null;
        String cookieName = "FavColours";
        String cookieVal1 = "Blue";

        StringBuffer buf = new StringBuffer();
        buf.append("<HTML>");
        buf.append("<BODY>");
        buf.append("<H1>Cookie List</H1>");
        // search for cookie by the name cookieName
        if (cookieArr != null) {
            for (int i = 0; i < cookieArr.length; i++) {
                cookie = cookieArr[i];
                if (cookie.getName().equals(cookieName))
                {
                    cookieFound = cookie;
                }
            }
        }
    }
}

```

```

if (cookieFound == null) {
    buf.append("<B>No cookie, a new cookie will be
        created</B><BR>");
    cookie = new Cookie(cookieName, cookieVal1);
    cookie.setMaxAge(300); // set age to 5 minutes
    response.addCookie(cookie); // adds cookie to HTTP
        header
}
else {
    buf.append("<B>Welcome to Favourite colour cookie: " +
        cookieFound.getValue() + "! </B><BR>");
    }
    buf.append("</BODY>");
    buf.append("</HTML>");

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(buf.toString());
} // public class servCookie

```

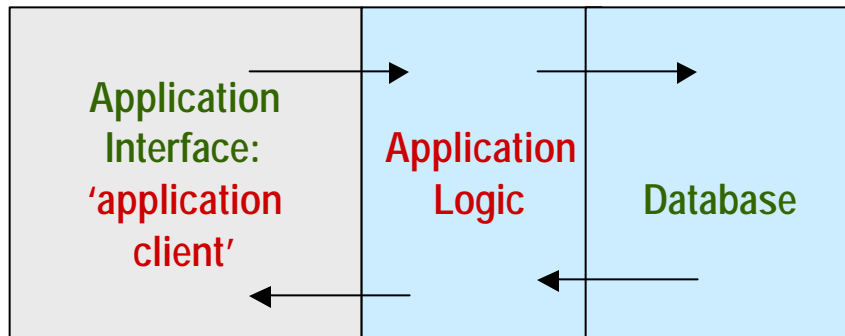
ServCookie.java

Cookies in Sessions

- session tracking can be done with cookies (as token mechanism described earlier)
- however cookies can be disabled by the browser. This results in the failure of tracking sessions, meaning each request from the client is treated independently
- URL rewriting is a viable alternative if cookies are not accepted by the browser
- [javax.servlet.http.HttpSession](#) interface provides token mechanism in abstract manner independently of whether a browser accepts or rejects cookies. J2EE web container provides an implementation of this API:
 - If the cookies are accepted, sessions are implemented using cookies
 - otherwise, URL rewriting is used
 - a session id is used as a token in either case

Web Containers

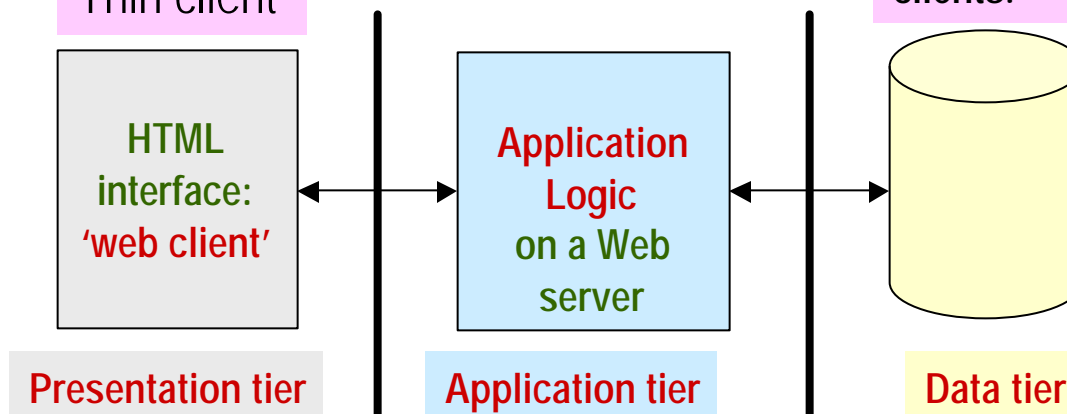
'Fat' and 'Thin' Clients in 3-tier Architecture



Fat client

In traditional applications, client layer drives not only the user interaction (usually through a GUI), but also bulk of the application logic. They are called application clients and are also characterised as 'fat' clients.

Thin client



In Web based applications, client layer is the browser which manages the user interaction (e.g. via HTML) and delegates the application logic completely to the server. They are 'web clients' and are characterised as 'thin' clients.

One can develop applications based on either type of clients using J2EE platform

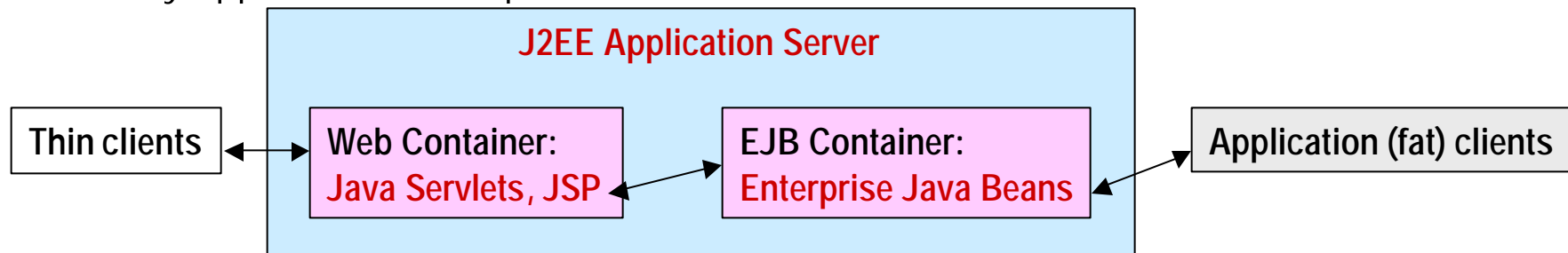
Features of Web Clients

- User interaction is managed by a browser or a similar 'general-purpose' application. For an end user, this is the client layer
- User interface is defined by HTML, Dynamic HTML (CSS with JavaScript), XML (with XSL)
- HTTP/HTTPS as the information exchange protocol between the web clients and the web applications on the server. Web applications (such as servlets) run application or business logic on the server on behalf of the web-clients
- All Web applications are hosted in Web containers

Web clients requests are handled by **Web Containers** in J2EE platform

J2EE Architecture

- J2EE platform includes one or more containers to service client requests; most important of them are Web container and EJB container
- containers host **application components** - servlets, JSP and EJBs developed by application developers



- Web container interacts with web clients (thin clients) and contains servlets and JSP
- EJB container has two types of clients:
 - application clients: stand alone applications (fat clients) accessing EJB components
 - application components from the web container, i.e. servlets and JSP from the web container can access EJP components in the same way as the application clients.

Container Architecture

A container

- implements container-specific API (Java Servlet API for Web container and EJP API for EJB container) needed for developing application components
- provides infrastructure for running the components and also providing additional services like database connectivity, mail etc.
- manages the life cycle of application component

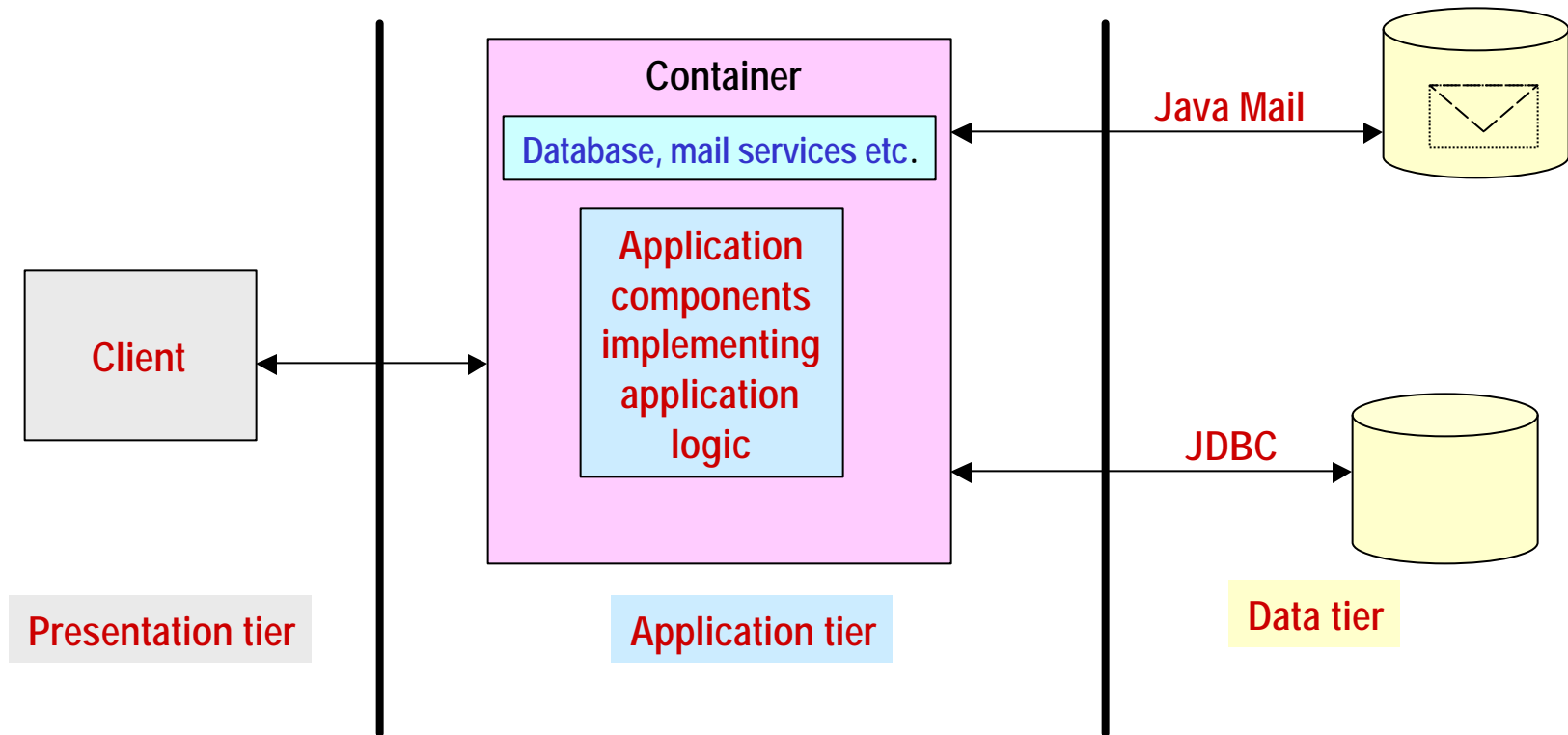
A container therefore can also be termed Java 1.2 run time environment for the application components

Application developer should

- implement the application components (servlets, JSP, EJB etc.)
- deploy the component using [deployment descriptors](#):
 - deployment descriptor is generally an XML file (e.g. webserver.xml); specifies how the component is installed and accessed.

In this course, we concentrate mainly on the Web Containers

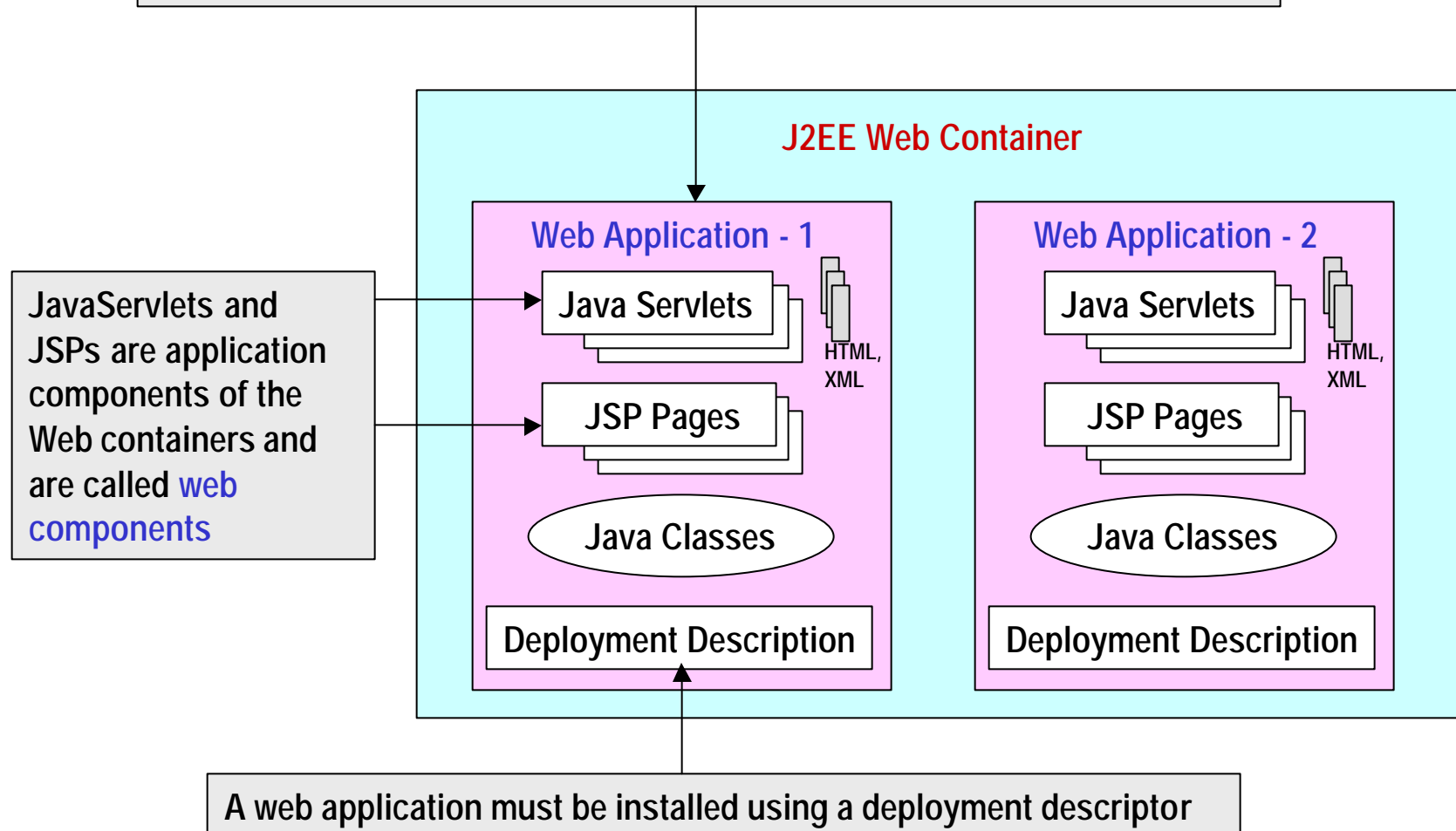
Container and Application Components



- container provides run time support and services to connect to databases
- this allows application components to concentrate only on application logic by implementing a set of APIs specified by the container

Web Containers

Web components are building blocks of a **web application**. A web application can also include other helper classes and static resources such as HTML and XML documents.



HTTP Request and Response Process

