JAVA SERVLETS

JOHN BRYCE
Leading in IT Education
a matrix company

Topics

- Straight the line regarding JEE
- Introduction to HTTP
- General look at web applications
- Servlets API
  - The general & HTTP packages
  - Coding servlets
  - Handling requests
  - Handling responses
  - Cookies
  - Session & session events
  - Servlet context
  - Filters
- Introduction to JSP

J2EE overview

The J2EE specification has the following components:

- The J2EE platform – standard of J2EE services
- The CTS – Compatibility Test Suite
  - For compliance verification
- J2ee Reference implementation
  - For application prototyping
- J2EE blueprints
  - Best practices for development

J2EE components

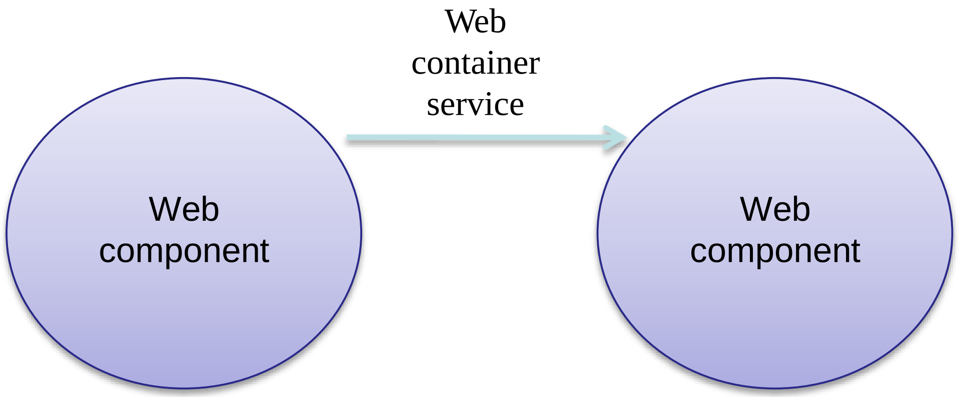
The J2EE components must be supported by any J2EE product

Components are deployed to J2EE application server

J2EE containers

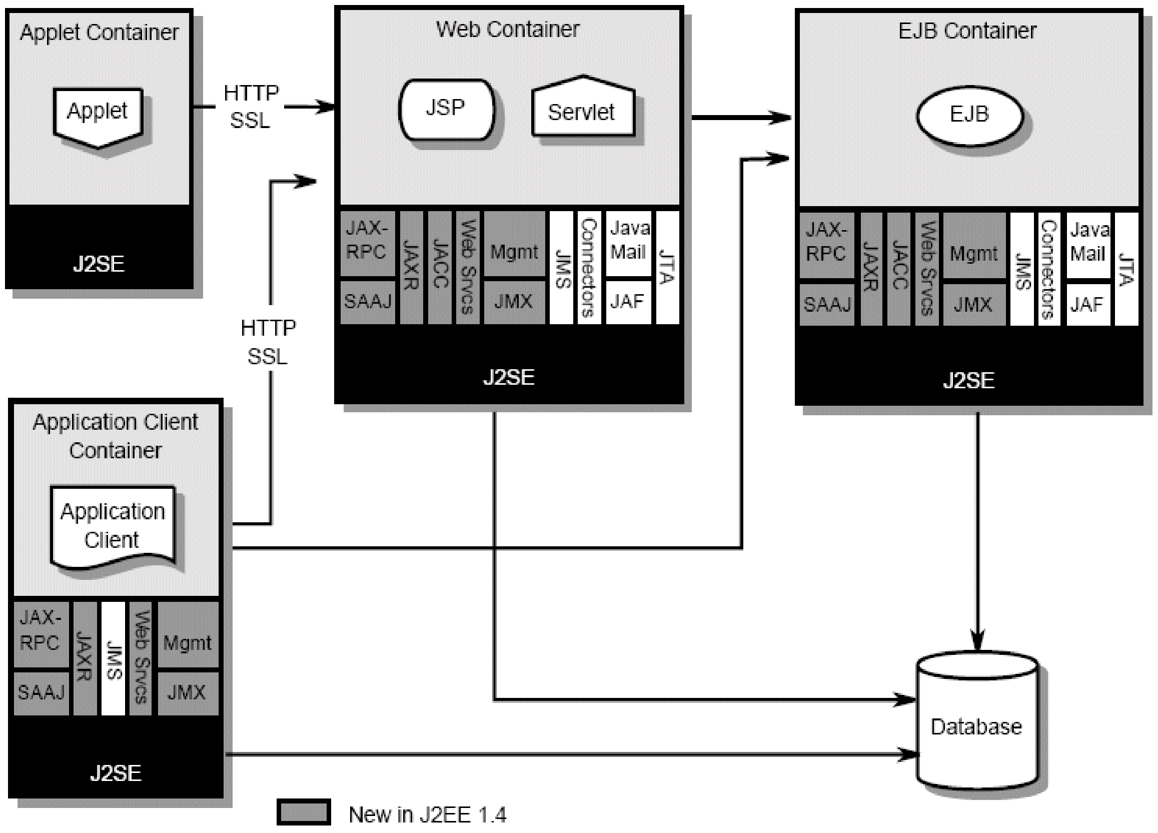
Containers manage the components they contain

J2EE components do not interact with other J2EE components – the services provided by the container do the interaction!



J2EE architecture

JOHN BRYCE
Leading in IT Education
a matrix company



J2EE standard services

Remote Method Invocation/Internet Inter-ORB Protocol (RMI-IIOP)

Java Interface Definition Language (Java IDL)

Java Transaction API (JTA)

Web Services (including JAX-RPC and SAAJ)

JavaMail

Hypertext Transfer Protocol (HTTP)

JavaBeans Activation Framework (JAF)

Java API for XML Parsing (JAXP)

J2EE Connector Architecture (JCA)

Java Message Service (JMS)

Java Authorization Service Provider Contract for Containers (JACC)

JDBC – Java Database connectivity

JNDI – Java Naming and Directory Interface

Java Authentication and Authorization Service (JAAS)

J2EE communications – RMI-IIOP and IDL

RMI-IIOP

- Java Remote Method Invocation ("Java RMI") technology run over Internet Inter-Orb Protocol ("RMI-IIOP") delivers Common Object Request Broker Architecture (CORBA) distributed computing capabilities to the Java 2 platform.
- Programming is RMI but independent of protocol

IDL

- J2EE components can invoke external CORBA objects

JTA – Transactions API

Interfaces between a transaction manager and the parties involved in a distributed transaction system.

Developers can also use EJB transaction management

Objects such as UserTransaction are used with methods such as:

- begin() – a new transaction on a thread
- commit() – End the transaction on the thread
- getStatus(), rollback(), setTransactionTimeout()

Messaging - JMS and JAF

- JMS
  - Standard vendor neutral service used to access enterprise messaging systems.
  - Same as JDBC is for DBs
  - Messages can be asynchronous.
- JAF
  - Part of the GlassFish opensource project
  - Takes a message as a bean of the JAF and knows how to send it as a message driven bean

JAXP

Java API for parsing XML

Detaches the compliance to a certain parser

Ability to respond to parsing events (SAX)

Fully operational and with additional functionality of the DOM interface

XML documents transformations

Works with interfaces

Web Services

J2EE specifications enable installing web services applications in a J2EE application server.

- Client side – access of a J2EE application to a web service as a traditional remote object
- Server side – Implementing WS in servlets and stateless session EJBs – managed by the container.
- Web services deployment – Deploying in a J2EE compliant web server.

Connectors - JCA

Resource adapters which enable access to Enterprise Information Systems.

This allows pooling of connections to EIS systems.

Transaction management contract between EIS and the transaction manager.

Enables secure access to EIS.

Authentication services - JAAS

Extension of the existing Java serurity model.

Used for user authentication for either executing a certain code or performing certain actions.

JACC - Provides an interface between the JAAS and the container.

HTTP

Hyper Text Transport Protocol

Getting Familiar With HTTP

HTTP - Hyper Text Transfer Protocol

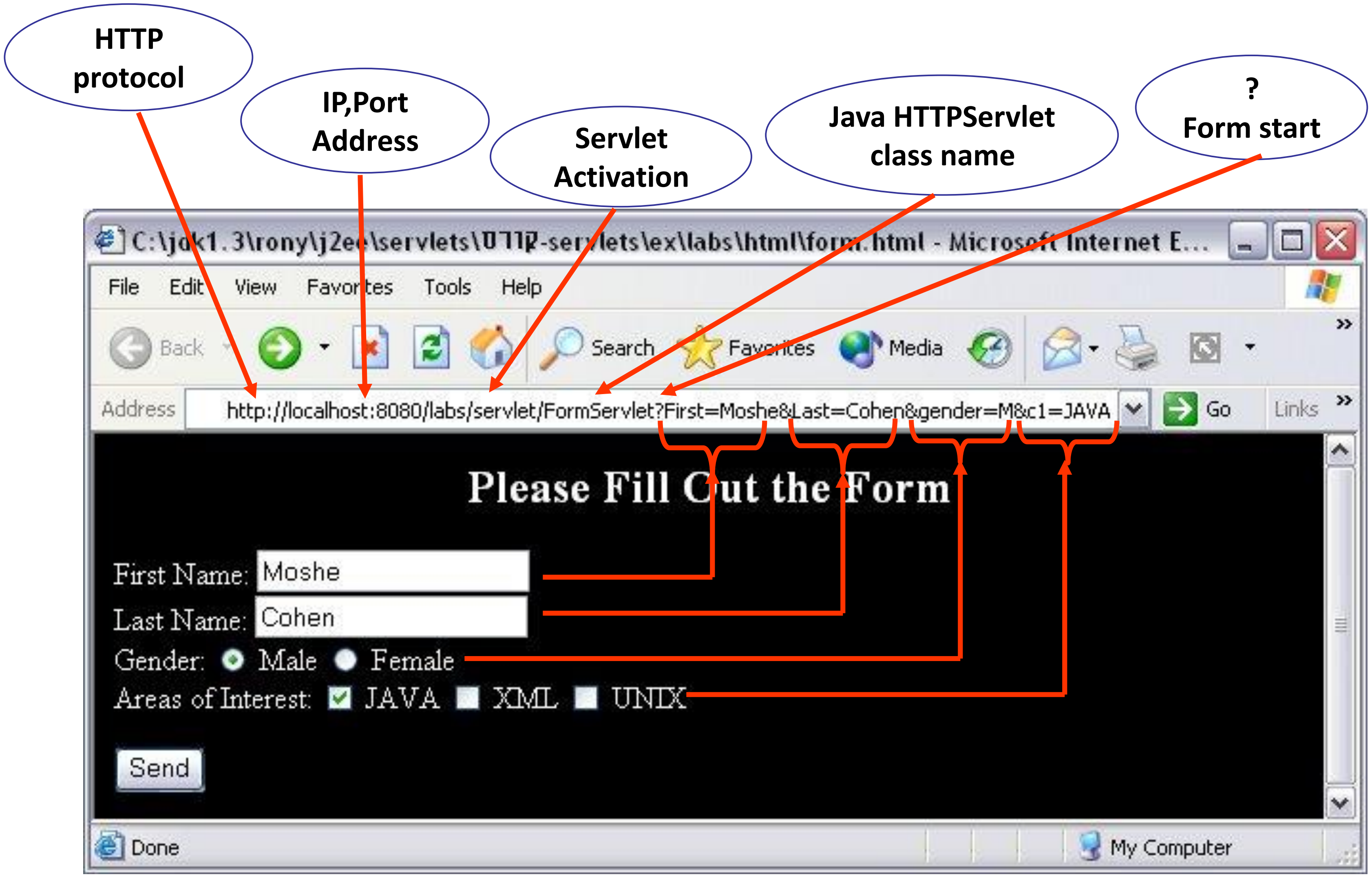
define the data transmission during the socket life cycle

works in a request-response environment

The way browsers and web servers talk

Is a W3C standard

Getting Familiar With HTTP



http://localhost:8080/labs/servlet/FormServlet?First=Moshe&Last=Cohen&gender=M&c1=JAVA

Please Fill Out the Form

First Name: Moshe

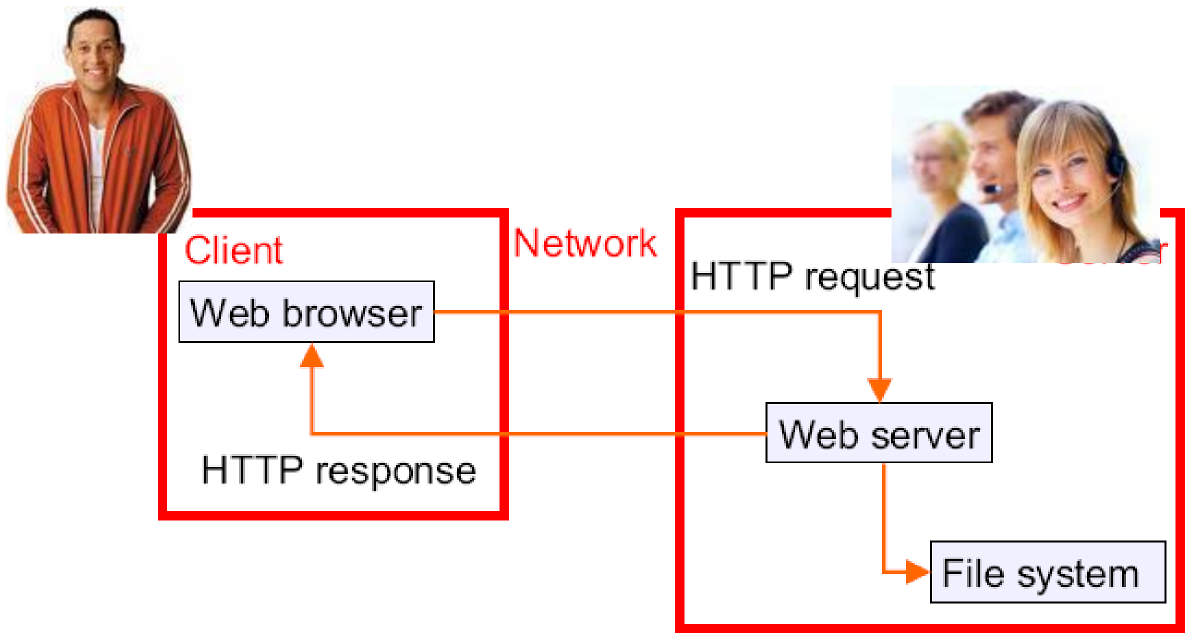
Last Name: Cohen

Gender: [✓ Male] Female

Areas of Interest: [✓ JAVA] XML UNIX

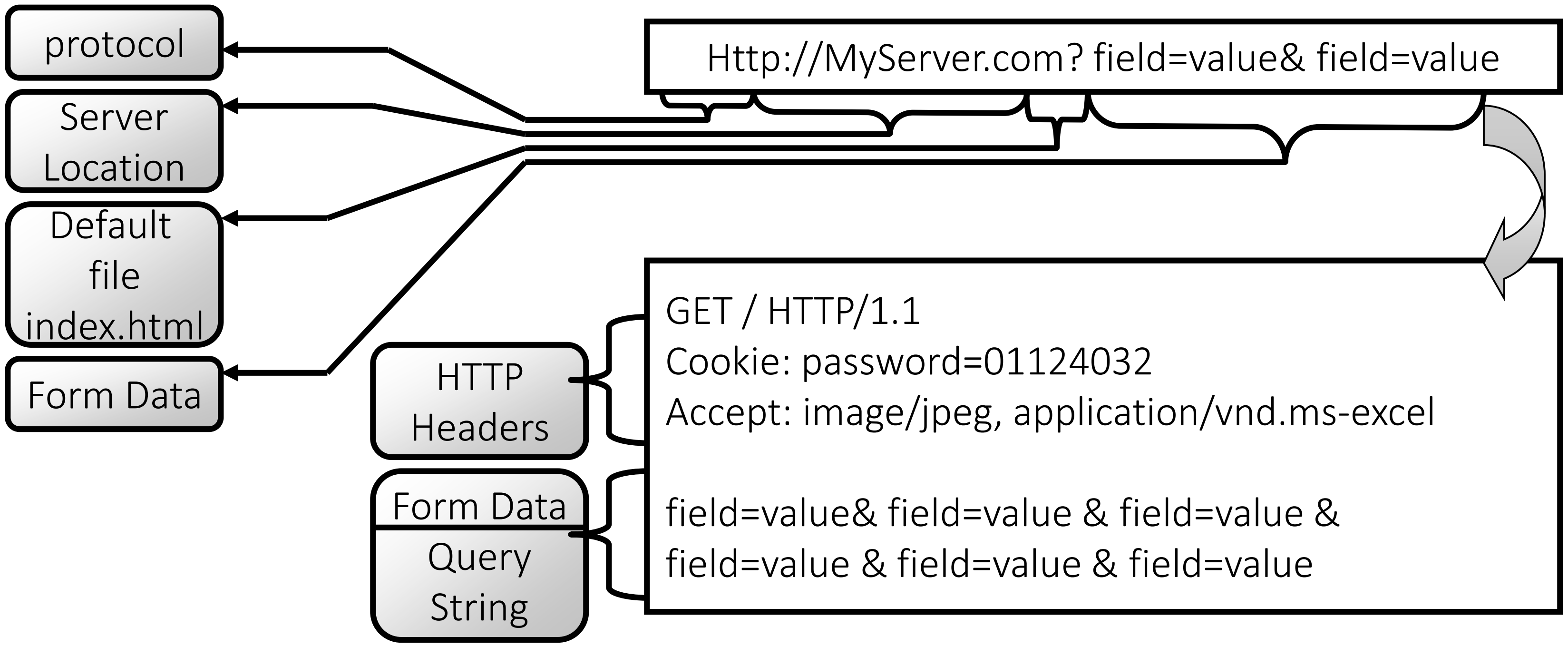
Send

Web application cycle



Getting Familiar With HTTP

HTTP **Request** structure:



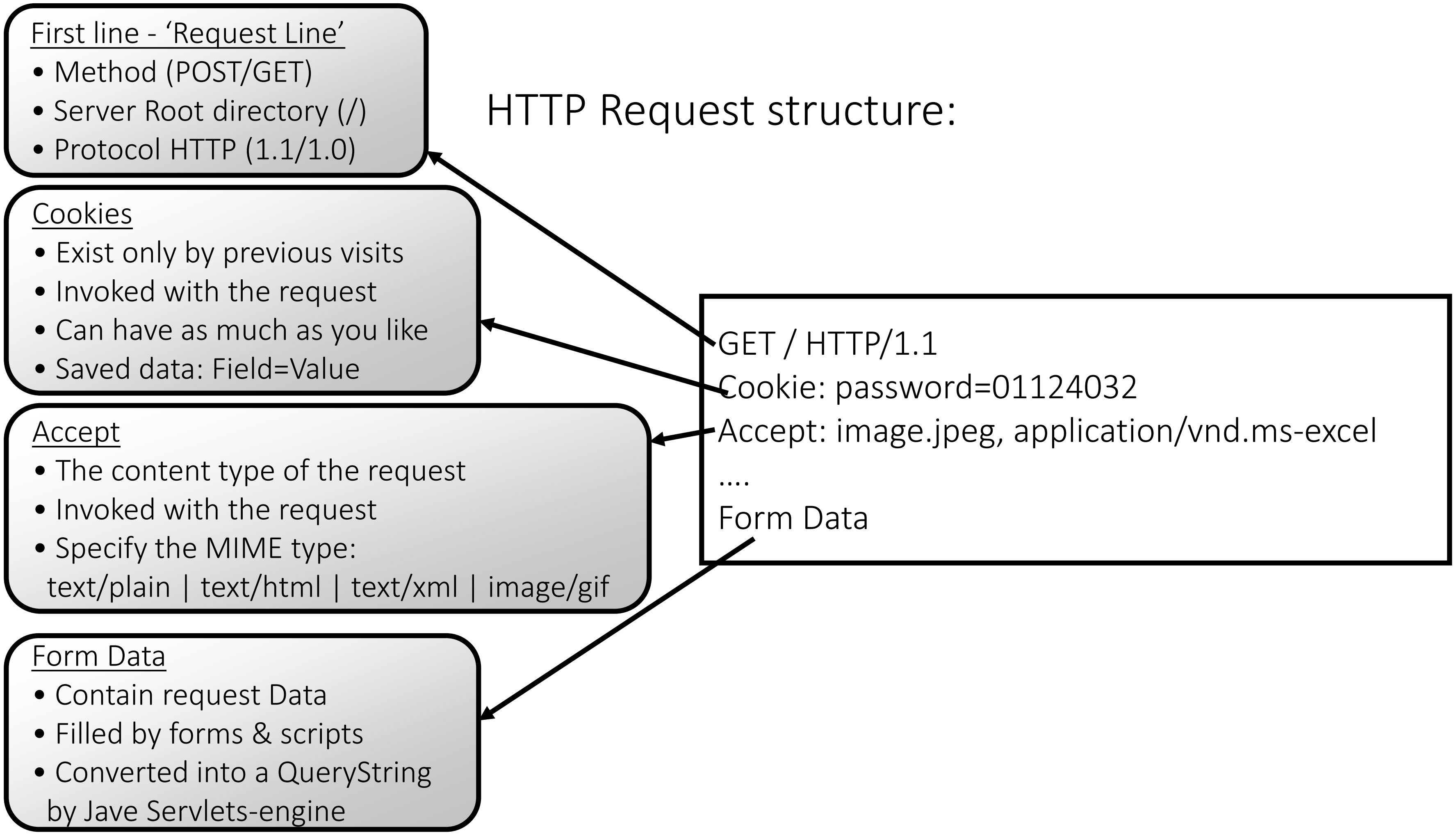
Http://MyServer.com? field=value& field=value

GET / HTTP/1.1
Cookie: password=01124032
Accept: image/jpeg, application/vnd.ms-excel

field=value& field=value & field=value &
field=value & field=value & field=value

Getting Familiar With HTTP

HTTP Request structure:



First line - 'Request Line'
- Method (POST/GET)
- Server Root directory (/)
- Protocol HTTP (1.1/1.0)

Cookies
- Exist only by previous visits
- Invoked with the request
- Can have as much as you like
- Saved data: Field=Value

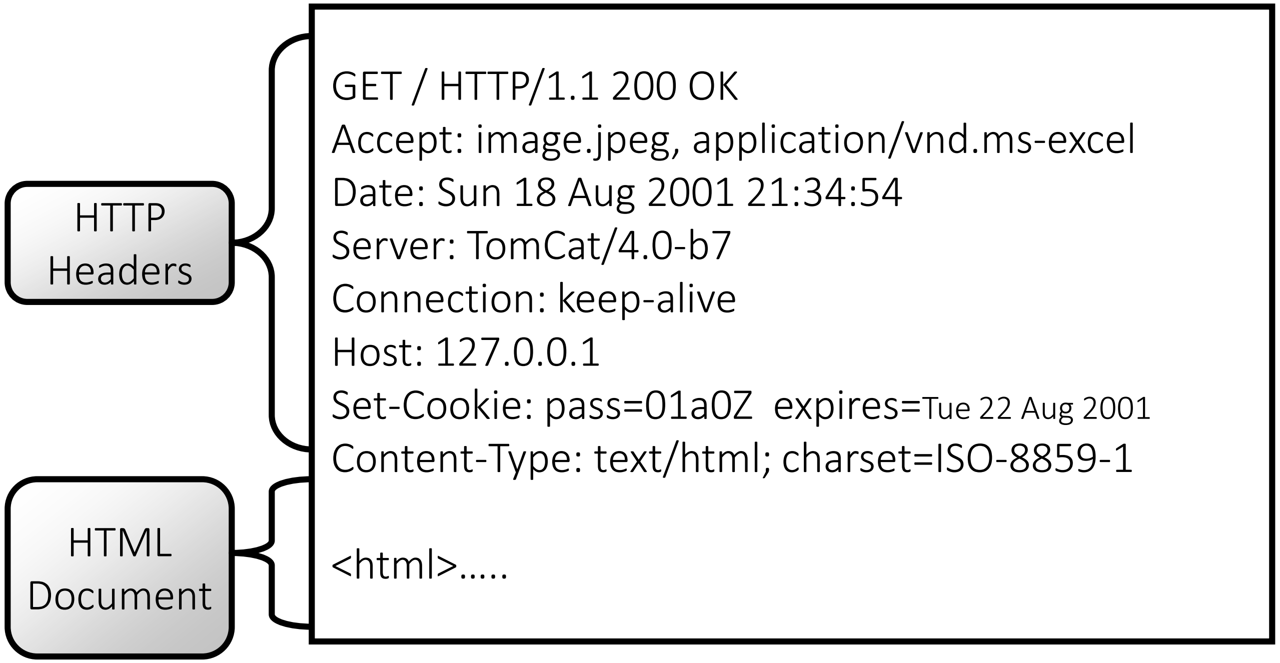
Accept
- The content type of the request
- Invoked with the request
- Specify the MIME type:
  text/plain | text/html | text/xml | image/gif

Form Data
- Contain request Data
- Filled by forms & scripts
- Converted into a QueryString
  by Jave Servlets-engine

```
GET / HTTP/1.1
Cookie: password=01124032
Accept: image.jpeg, application/vnd.ms-excel
....
Form Data
```

Getting Familiar With HTTP

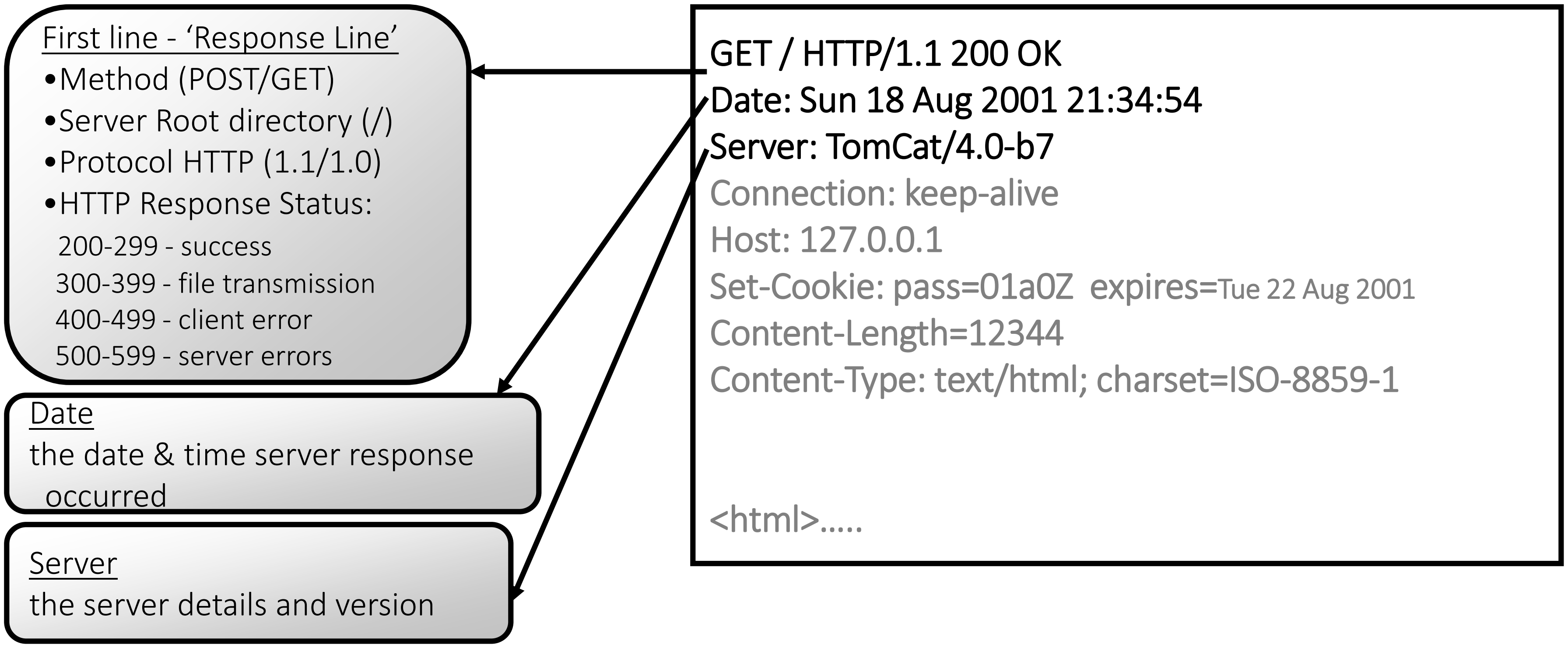
HTTP **Response** structure:



```
GET / HTTP/1.1 200 OK
Accept: image.jpeg, application/vnd.ms-excel
Date: Sun 18 Aug 2001 21:34:54
Server: TomCat/4.0-b7
Connection: keep-alive
Host: 127.0.0.1
Set-Cookie: pass=01a0Z  expires=Tue 22 Aug 2001
Content-Type: text/html; charset=ISO-8859-1

<html>.....
```

Getting Familiar With HTTP

HTTP Response structure:



First line - 'Response Line'
- Method (POST/GET)
- Server Root directory (/)
- Protocol HTTP (1.1/1.0)
- HTTP Response Status:
  - 200-299 - success
  - 300-399 - file transmission
  - 400-499 - client error
  - 500-599 - server errors

Date
the date & time server response occurred

Server
the server details and version

```
GET / HTTP/1.1 200 OK
Date: Sun 18 Aug 2001 21:34:54
Server: TomCat/4.0-b7
Connection: keep-alive
Host: 127.0.0.1
Set-Cookie: pass=01a0Z  expires=Tue 22 Aug 2001
Content-Length=12344
Content-Type: text/html; charset=ISO-8859-1


<html>.....
```

Getting Familiar With HTTP

HTTP Response structure:

Connection
the kind of connection:
close - when there is no content-length
keep-alive - default

Host
the IP address of the server

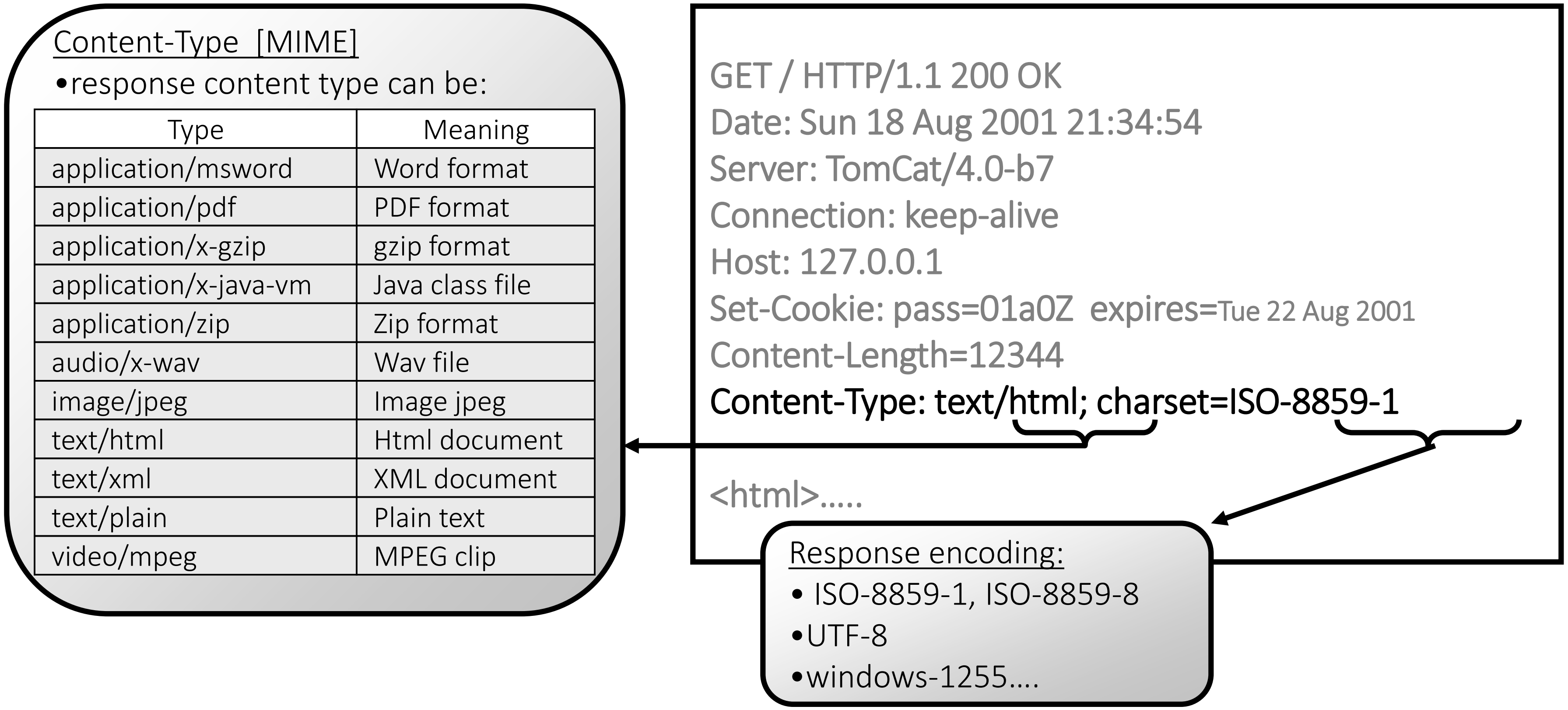
Set Cookie
•add cookie (field&value)
•can determine expire date

Content-Length
•the number of bytes being sent
•without Content-Length:
Connection=close

```
GET / HTTP/1.1 200 OK
Date: Sun 18 Aug 2001 21:34:54
Server: TomCat/4.0-b7
Connection: keep-alive
Host: 127.0.0.1
Set-Cookie: pass=01a0Z  expires=Tue 22 Aug 2001
Content-Length=12344
Content-Type: text/html; charset=ISO-8859-1

<html>.....
```

Getting Familiar With HTTP

HTTP Response structure:



Content-Type [MIME]

- response content type can be:

Type	Meaning
application/msword	Word format
application/pdf	PDF format
application/x-gzip	gzip format
application/x-java-vm	Java class file
application/zip	Zip format
audio/x-wav	Wav file
image/jpeg	Image jpeg
text/html	Html document
text/xml	XML document
text/plain	Plain text
video/mpeg	MPEG clip

```
GET / HTTP/1.1 200 OK
Date: Sun 18 Aug 2001 21:34:54
Server: TomCat/4.0-b7
Connection: keep-alive
Host: 127.0.0.1
Set-Cookie: pass=01a0Z  expires=Tue 22 Aug 2001
Content-Length=12344
Content-Type: text/html; charset=ISO-8859-1

<html>.....
```

Response encoding:

- ISO-8859-1, ISO-8859-8
- UTF-8
- windows-1255....

Web Application Structure

Web Application includes:

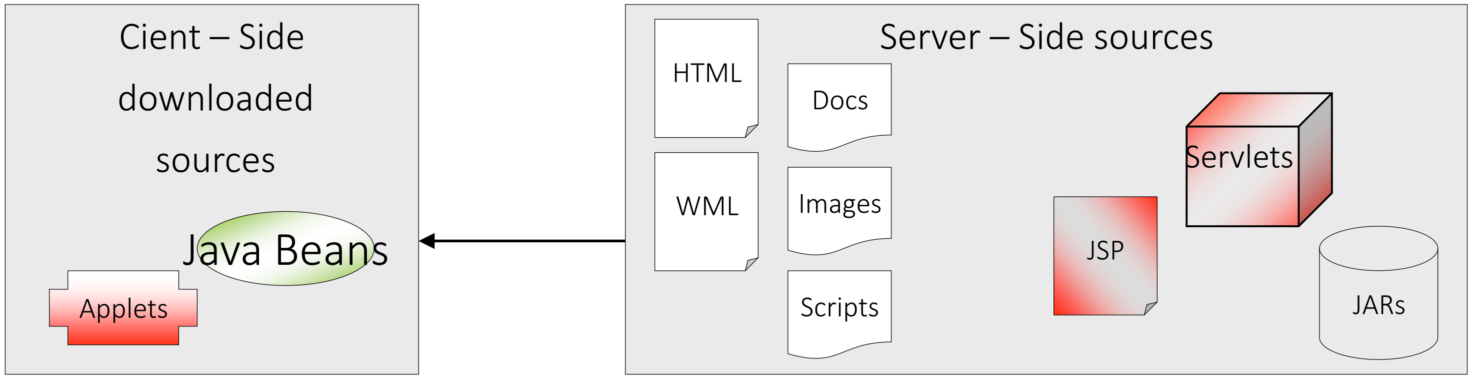
Dynamic content [Servlets and JSPs]

Static content [HTML, WML, documents, images etc.]

Client-side scripts [Applets, Java-scripts, flash apps.]

Value Objects [Java Beans]

Helper Classes [classes and jar files]

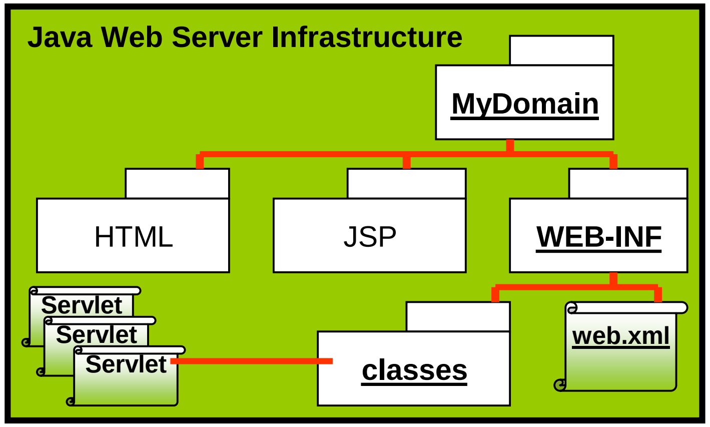


Web Application Structure

Must include:

Base directory [the directory name is the actual application name]

WEB-INF directory



May include:

web.xml configuration file [Deployment Descriptor]

Servlets [located in WEB-INF\classes dir]

JSP files

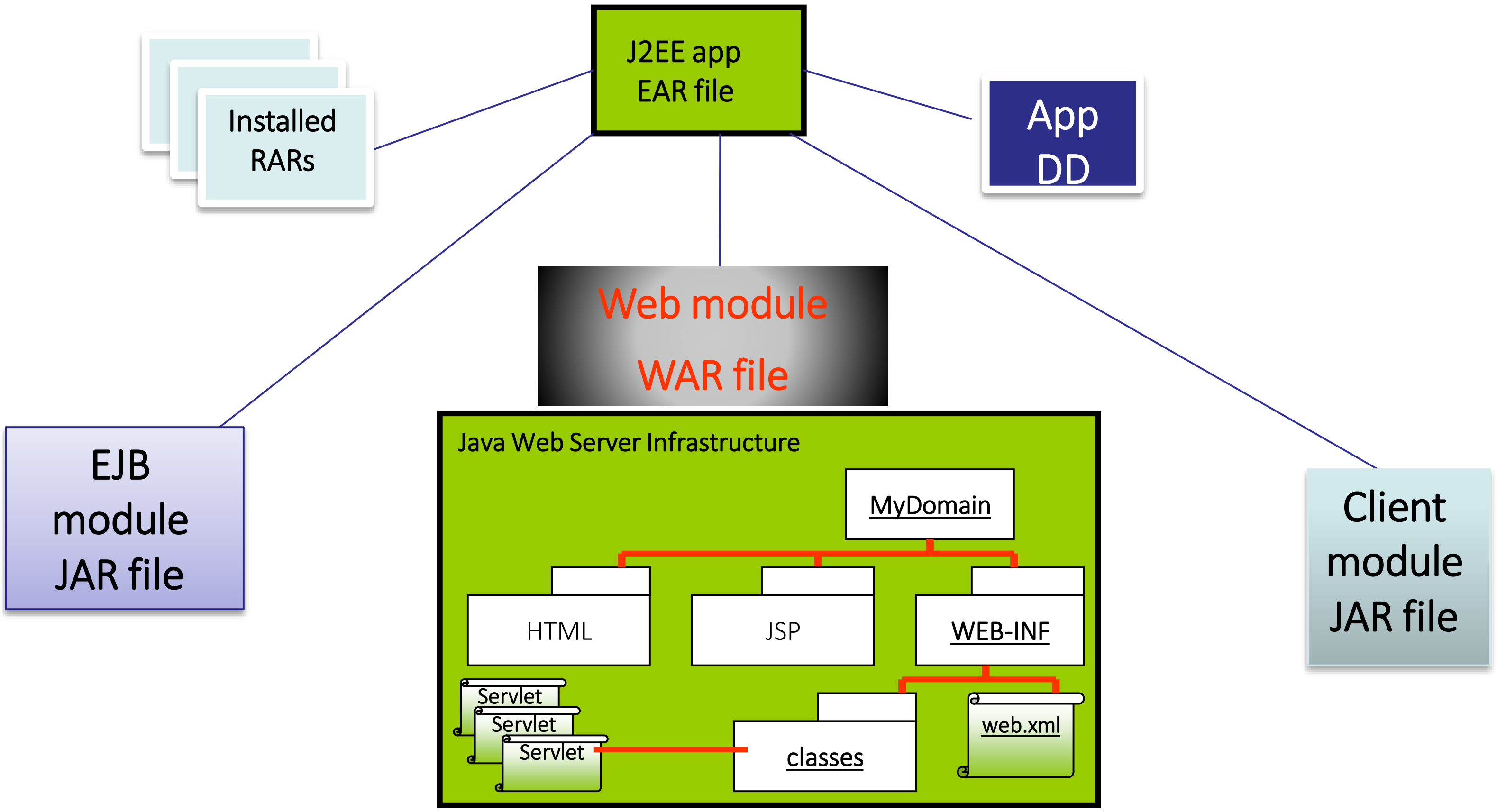
HTML files

Documents, Images

Jars and other classes [usually in a *lib* directory]

JOHN BRYCE
Leading in IT Education
a matrix company

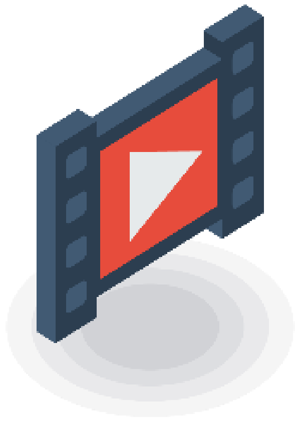
J2EE Application Structure



Lab 1

JOHN BRYCE
Leading in IT Education
a matrix company

Web project



SERVLETS

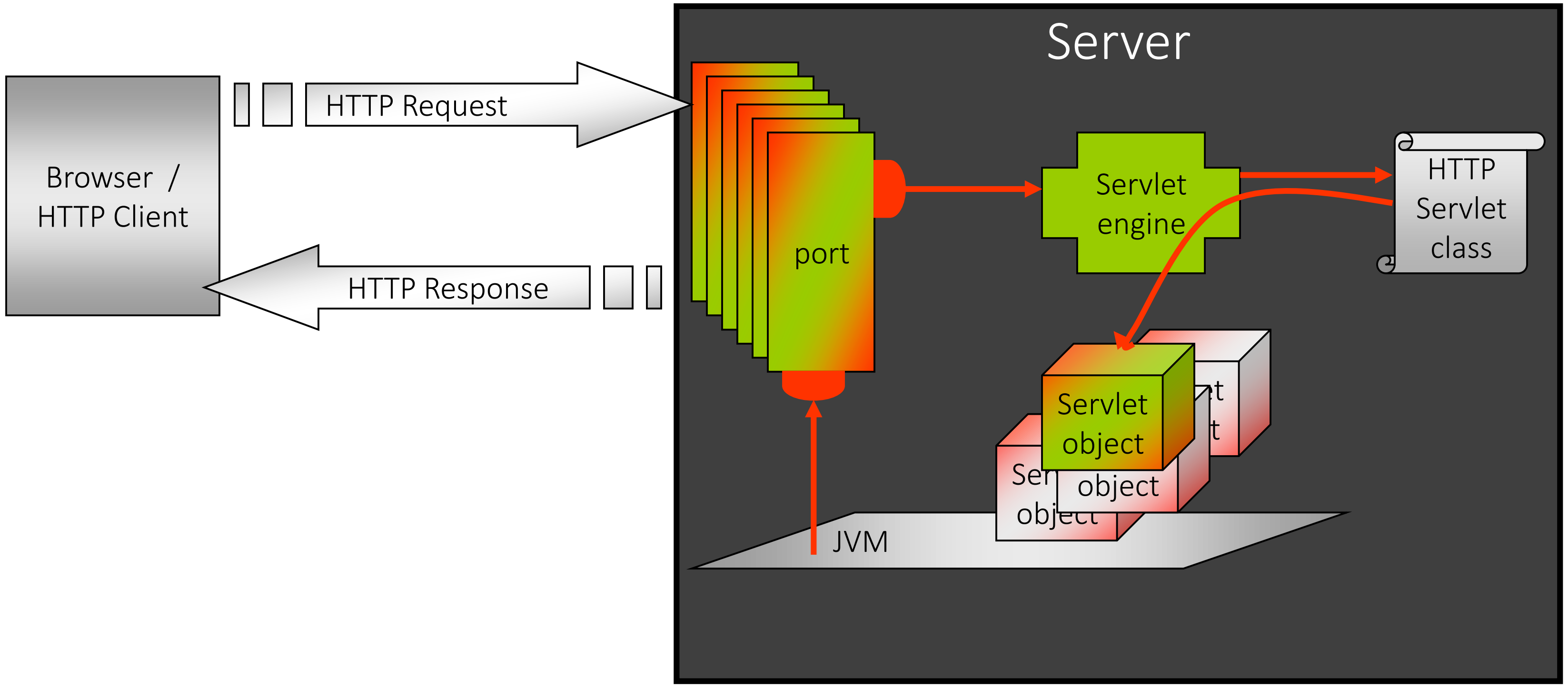
Servlets

Java solution for Common Gateway Interface (CGI)

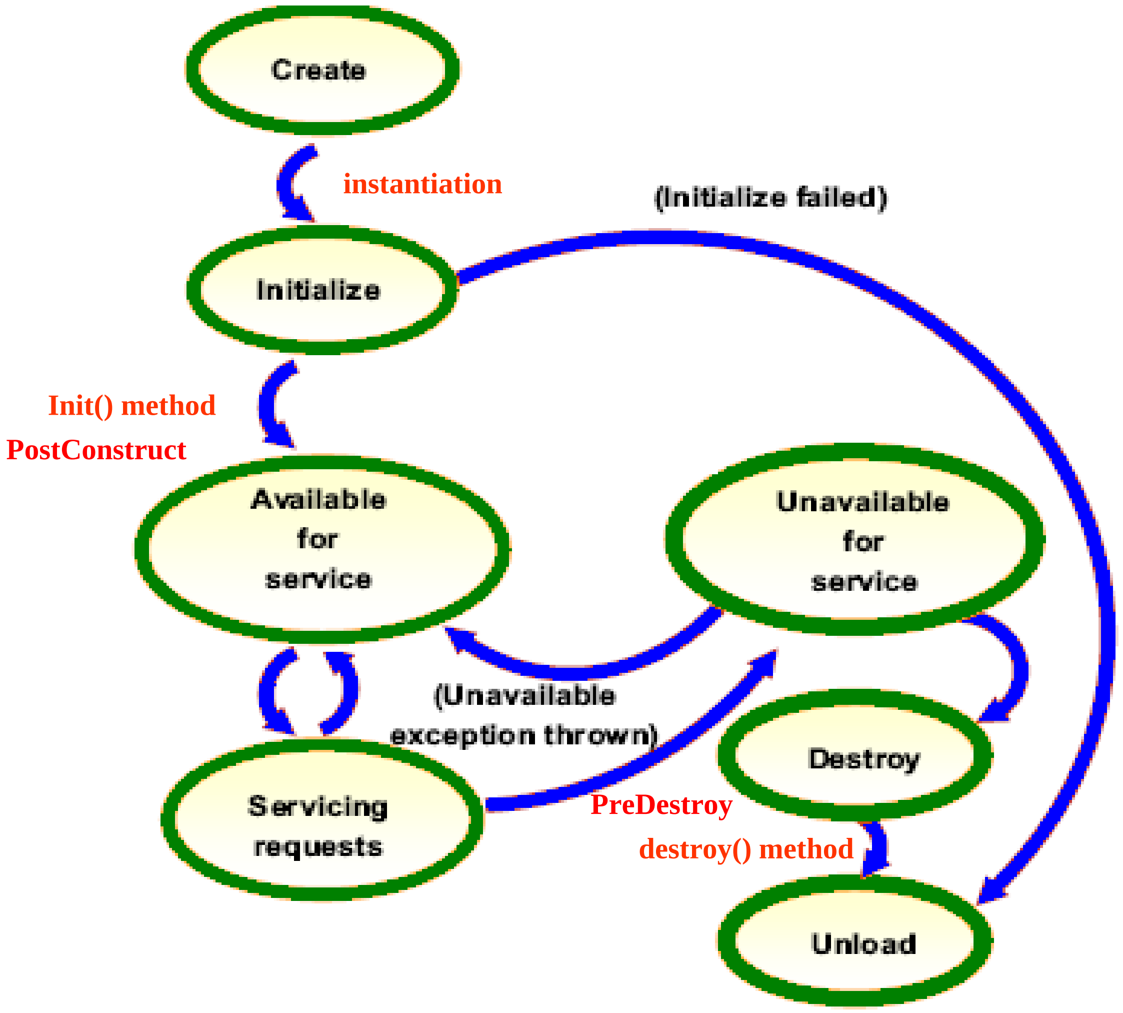
Servlet is an easy way of communicating with HTTP clients

HTTPServlet supplies methods of reading **HTTP requests**
& building **HTTP response**

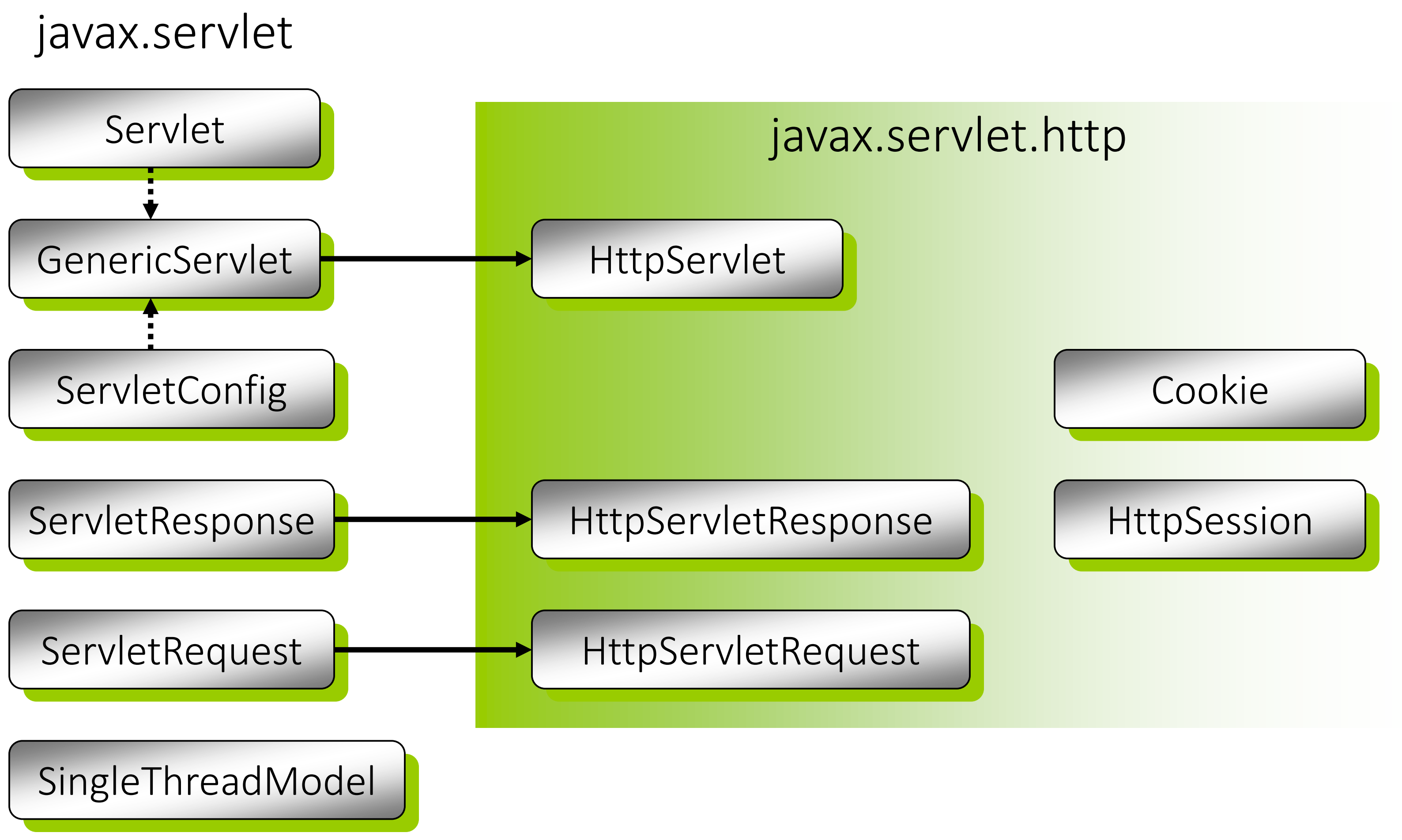
How does it work ?



Servlets Lifecycle



Servlet Package



Servlet Interface

Defines basic behavior of a servlet (Servlet Life Cycle):

init(ServletConfig config) - when servlet is first loaded

service() - executed for each client request & generates response

destroy() - executed on server shut down or when un-deploying application

GenericServlet Class

Implements Servlet & ServletConfig interfaces

Supplies advanced method for working with a servlet instance:

<u>Servlet Configuration:</u>

- getInitParameter (String param) - returns the value of the parameter as String
- getServletName () - return the name of the servlet instance as given by the server configuration
- getServletConfig () - return the ServletConfig that hold all server parameter & values configuration

<u>Servlet Life Cycle:</u>

- init (ServletConfig config)
- service(ServletRequest req, ServletResponse res)
- destroy()

<u>Servlet Life Cycle Wrappers:</u>

- @PostConstruct
- @PreDestroy

HttpServlet Classes

Extends GenericServlet

Uses advanced method for dealing with HTTP requests & responses:

- ServletRequest
- ServletResponse

Replaces the service method to match HTTP protocol:

HTTP GET/POST:

- GET doGet (HTTPServletRequest req, HTTPServletResponse res)
- POST doPost (HTTPServletRequest req, HTTPServletResponse res)
- DELETE doDelete (HTTPServletRequest req, HTTPServletResponse res)
- PUT doPut (HTTPServletRequest req, HTTPServletResponse res)
- HEAD doHead (HTTPServletRequest req, HTTPServletResponse res)

HttpServlet

Handler for HTTP-specific requests

Methods for HTTP specific actions:

  doGet(HttpServletRequest , HttpServletResponse)

  doPost(HttpServletRequest , HttpServletResponse)

service(HttpServletRequest , HttpServletResponse)

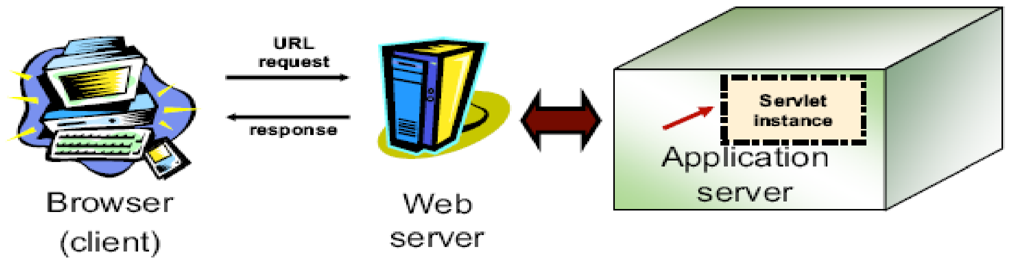
  is for any type of HTTP method

HttpServlet

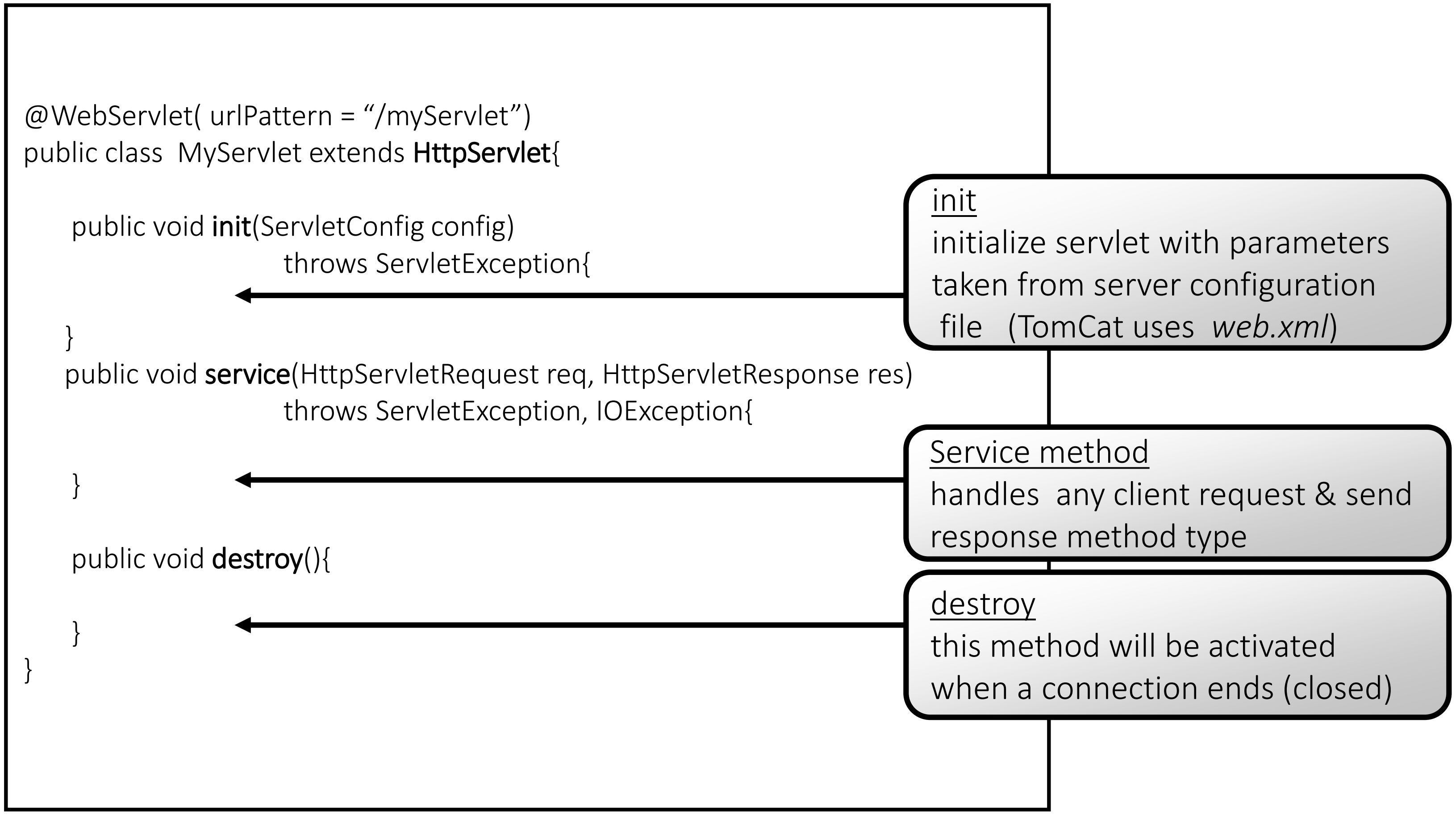
Methods doGet() and doPost() have two parameters

- HttpServletRequest – provides request parameters, HttpSession information
- HttpServletResponse – assists in supplying a response to the requesting client

Reading and writing a response is most of the activity



HttpServlet Structure



```
@WebServlet( urlPattern = “/myServlet”)
public class  MyServlet extends HttpServlet{

      public void init(ServletConfig config)
                              throws ServletException{

     }
     public void service(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException{

      }

      public void destroy(){

      }
}
```

init
initialize servlet with parameters taken from server configuration file (TomCat uses *web.xml*)

Service method
handles any client request & send response method type

destroy
this method will be activated when a connection ends (closed)

Lifecycle annotations

PostConstruct

- A method denoted with post construct will be fired right after the constructor and before init()

PreDestroy

- A method denoted with pre destroy will be fired right after destroy() and before the servlet instance is discarded

Denoted methods:

- Can have any name
- Must return void
- Must accept no arguments

HttpServlet Structure Lifecycle Wrappers

```
@WebServlet( urlPattern = “/myServlet”)
public class  MyServlet extends HttpServlet{

    public void init(ServletConfig config) throws ServletException{…}
    @PostConstruct
    public void doSomethingBeforeServicing() {…}

     public void doGet(HttpServletRequest req, HttpServletResponse res)
                          throws ServletException, IOException{

     }

     @PreDestroy
     public doSomethingBeforeDestroying(){…}
     public void destroy(){…}
}
```

<u>Post construct</u>
Is called after init() and before service()

<u>GET method</u>
handles client GET request & send response

<u>Pre destroy</u>
Is called before destroy() is invoked

GET or POST ?

When requests method handled in one way use service method instead:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
…
public class  MyServlet extends HttpServlet{
        ….
                public void service (HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException{
                //handle request & response
        }
}
```

GET or POST ?

Since the server cannot anticipate the HTTP method,

use this way to handle any kind of requests:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
…
public class  MyServlet extends HttpServlet{

      public void doPost (HttpServletRequest req, HttpServletResponse res) {
                  doGet (req, res);
      }

      public void doGet (HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException{
                  //handle request & response
      }
}
```

HTML forms can define the method used to generate the request

HTML links uses GET method

web.xml Structure

web.xml in the standard web application configuration file

Located under WEB-INF directory

Usually called – Deployment Descriptor

Is portable and supported in any J2EE complaint web server

Is optional since Servlet 3.0

Defines:

Servlets & JSPs*

Servlets & JSPs URL mapping*

Filters*

Initial parameters*

Session listeners*

- Security roles*
- Security constraints*
- Welcome pages
- Error pages

* can be set via annotations since Servlet version 2.5-3.0

Defining Servlets

- @WebServlet
  - Main Attributes:
    - name – servlet logical name
    - urlPattern - url pattern to call this servlet
    - description
    - loadOnStartup – number indicated the loading order

Defining Servlets

Example:

```
@WebServlet(urlPatterns = { "/Hello" },
                    loadOnStartup = 1)
public class HelloServlet extends HttpServlet {
   ....
```

Calling HelloServlet:

http://localhost:8080/labs/Hello

Other mapping patterns:

/   -   any JSP of Servlet*
/   -   any unsolved URL (JSP is not included)

Defining Welcome Pages in web.xml

Example:

Welcome file list will contain the URL mapping of the welcome resources and will Call the first one on client request.
In any case of 'file not found' the next resource on the list is returned

```
…
…
<welcome-file-list>
        <welcome-file>/Hello</welcome-file>
        <welcome-file>/NotAvailable</welcome-file>
</welcome-file-list>
```

Defining Error Pages in web.xml

Example:

Error pages can be bounded to a specific HTTP error status.
<error-page> element can be re-used

```
…
<error-page>
    <error-code>404</error-code>
    <location>/error.jsp</location>
</error-page>
```

Servlet Initializing Parameters

ServletConfig can be handed to each servlet

Each servlet has its own dedicated config instance

Is handed to the servlet in init() method – only once

Use annotations to populate config with params that later will be available for the servlet

- @WebServlet - initParams attribute
- Accepts multiple @WebInitParam
- Values are in java.lang.String format

Servlet Initializing Parameters

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
@WebServlet(  urlPatterns = { "/init" },
                    initParams = {
                               @WebInitParam(name = “text", value = “hello !!", description = “simple text"),
                                 @WebInitParam(name = “times", value = “10” , description = “times to print")
                                 })
public class  InitServlet extends HttpServlet{
    String text;
     int times;

  public void init(ServletConfig config)  throws ServletException{
     text=config.getInitParameter("text");
    String temp=config.getInitParameter("times");
    times=Integer.parseInt(temp);
    for(int i=0;i<times;i++)
        System.out.println((i+1)+" "+text);
  }
}
```

Lab 2

HttpServletRequest Class

Supplies methods for dealing with all parts of HTTP request:

I

HTTP Headers:

- getProtocol () - returns the request protocol [HTTP/1.1, HTTP/1.0….]
- getServerPort () - returns the request port [80, 88….]
- getContentType () - returns the MIME type contained in the request [text/html, image/gif…]
- getContentLength () - return int size of the POST method request body
- getServletPath () - returns the servlet path part of the request
- getPathInfo() - returns the local path of the requested servlet (starting from the default classes dir)
- getPathTranslated() - returns the full path of the servlets (starting from the server root directory)
- getQueryString () - returns the Form Data as a string [?val1=11&val2=22&val3=33&…]
- getRemoteAddr () - returns the remote IP address [207.115.1.176]
- getMethod () - returns the request method [GET / POST]
- getHeader (String name) - general method, returns the value of the specified header name
- getCookies () - will be discussed later

Ex: *labs\html\headers.html [ReqHeaders.java]*

HttpServletRequest Class

...parts of HTTP request:

II

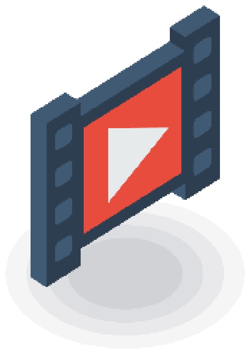
<u>**HTTP Form Data :**</u>

- getParameter (String param) - returns the value of the parameter as <u>String</u>
- getParameterNames () - returns <u>Enumeration</u> containing all parameter names
- getParameterMap () - return a <u>Map</u> containing all parameter names and their values

Ex: *labs\html\form.html [FormServlet.java]*

Lab 3

servlets



HttpServletRequest Class

External Objects can be carried within the Request

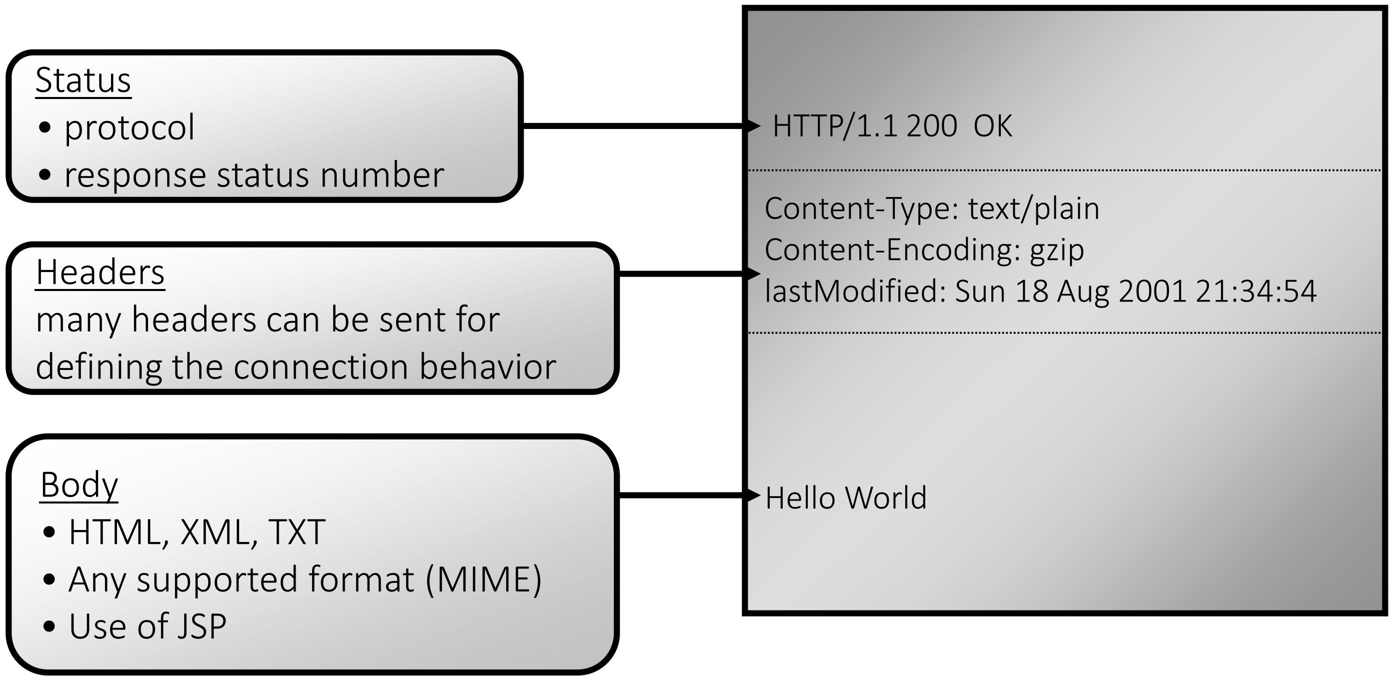
Those Objects might be Java-Beans

<u>Adding & Getting the external object by name :</u>

- req.getAttribute (String name) - returns the object instance by name or null if doesn’t exists
- req.setAttribute (String name, Object obj) – registers the object instance to the given name

HttpServletResponse Class

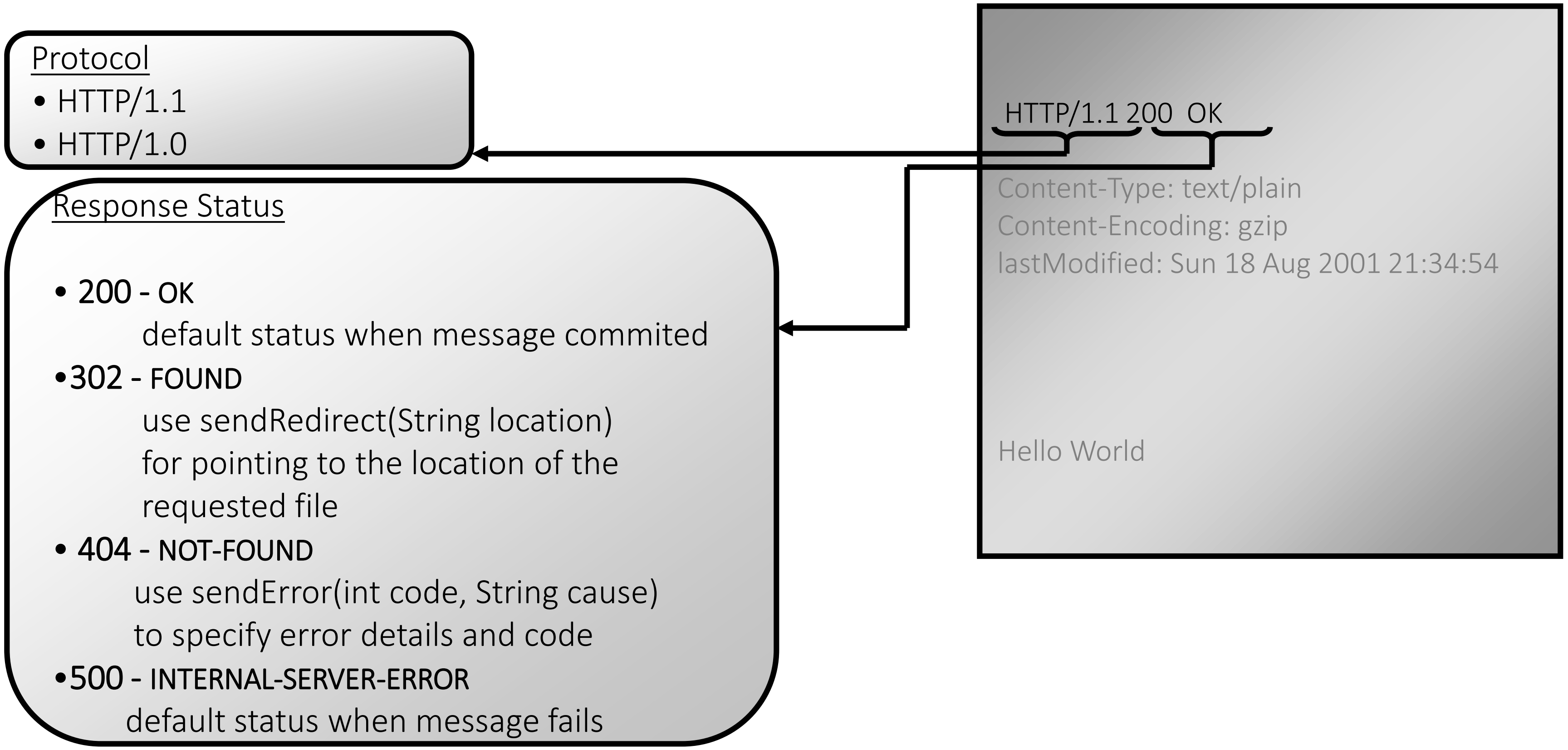
Response Structure:



HttpServletResponse Class

JOHN BRYCE
Leading in IT Education
a matrix company

Status:



Protocol
- HTTP/1.1
- HTTP/1.0

Response Status

- 200 - OK
  default status when message commited
- 302 - FOUND
  use sendRedirect(String location)
  for pointing to the location of the
  requested file
- 404 - NOT-FOUND
  use sendError(int code, String cause)
  to specify error details and code
- 500 - INTERNAL-SERVER-ERROR
  default status when message fails

```
HTTP/1.1 200  OK
Content-Type: text/plain
Content-Encoding: gzip
lastModified: Sun 18 Aug 2001 21:34:54


Hello World
```

HttpServletResponse Class

Methods for configuring response status:

```
public void setStatus(int status_code)
public void sendError(int err_status_code)
```

Use HttpServletResponse constants for status codes
For example:
- SC_OK (200)
- SC_FORBIDDEN (403)
- SC_INTERNAL_SERVER_ERROR (500)

Method for client redirecting:

```
public void sendRedirect(String url)
```

Ex: *labs\html\status.html [StatusServlet.java]*

Response redirection

Response redirect –

- Url may be absolute or relative
- The container changes the calling to an absolute url before sending the response to the client
- If a response has been flushed a IllegalStateException will be thrown
- After calling redirect writing to the response is not permitted since it has been committed already

Error sending

Goes to specific error page describing the error

Empties the buffer

Sets a response status code

If the response has already been committed IllegalStateException is thrown.

Does not touch cookies, headers

Changes the content type to

Response redirection example

```
private void processRequest(
         HttpServletRequest request,
         HttpServletResponse response) ... {
   // process request headers & query data
   ...
   // redirect to another URL
   String url = "/YourResults.html";
   if (test.equals("Error"))
     response.sendError(HttpServletResponse.SC_BAD_REQUEST);
   else
      response.sendRedirect
        (response.encodeRedirectURL(url));
   return;
}
```

Method called from doGet() or doPost() method

HttpServletResponse Class

Headers:

Headers

- JSDK contain 2 kind of methods for headers defining :

Original HTTP Headers

setContentType (String)	determine the MIME
setContentLength (int)	for long connections
addCookie (Cookie)	will be discussed later

Additional HTTP Headers

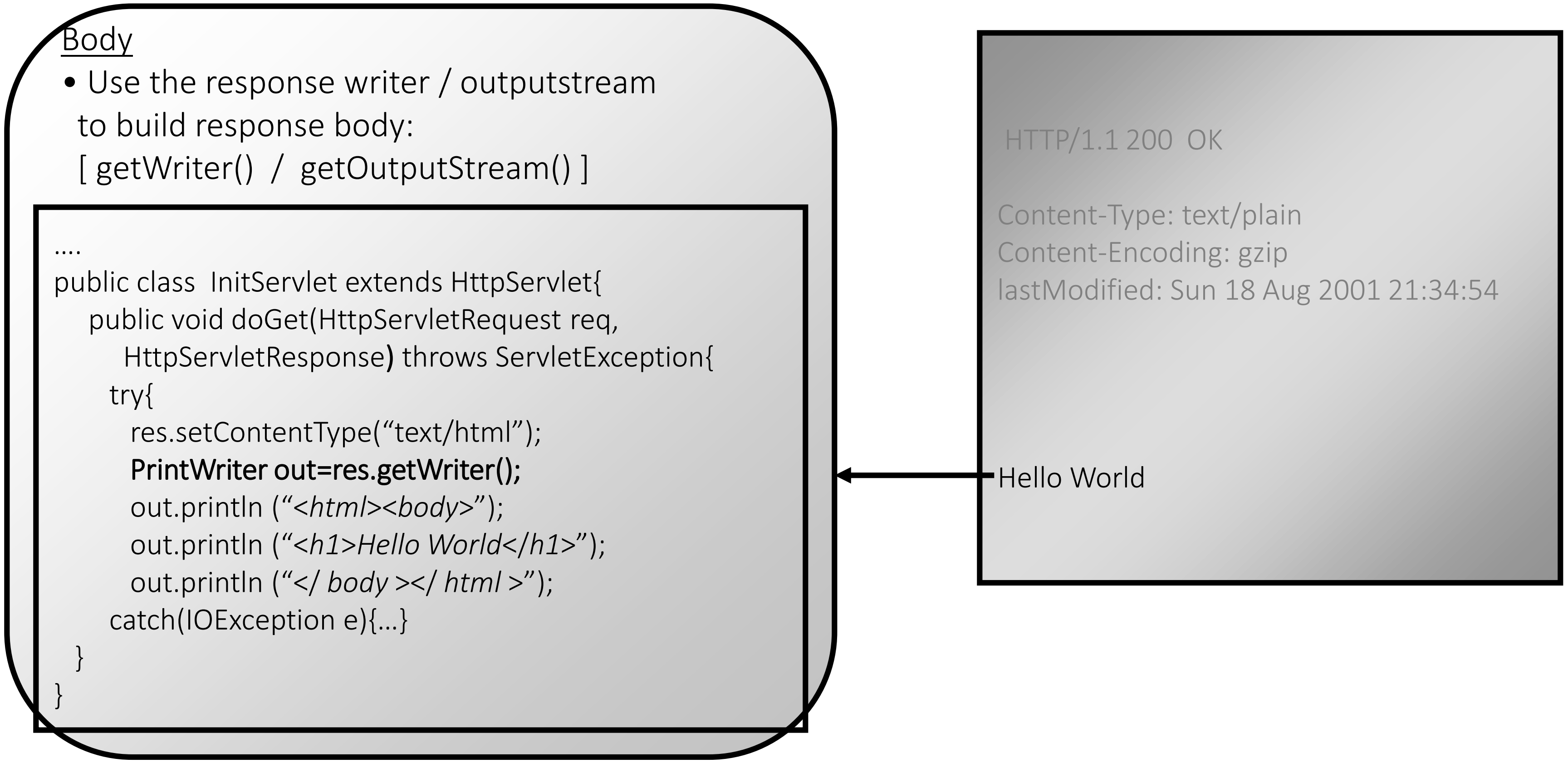
setDateHeader (String, long)	to send date string
setIntHeader***** (String, int)	to send an int value
setHeader*****(String, String)	all kind of headers

```
HTTP/1.1 200  OK

Content-Type: text/plain
Content-Encoding: gzip
lastModified: Sun 18 Aug 2001 21:34:54

Hello World
```

*see Other Response Headers

HttpServletResponse Class

Body:



Body

- Use the response writer / outputstream to build response body:
  [getWriter() / getOutputStream()]

```
….
public class  InitServlet extends HttpServlet{
    public void doGet(HttpServletRequest req,
        HttpServletResponse) throws ServletException{
      try{
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println ("<html><body>");
        out.println ("<h1>Hello World</h1>");
        out.println ("</ body ></ html >");
      catch(IOException e){…}
  }
}
```

```
HTTP/1.1 200  OK

Content-Type: text/plain
Content-Encoding: gzip
lastModified: Sun 18 Aug 2001 21:34:54



Hello World
```

Ex: *labs\html\hello.html [HelloServlet.java]*

Other Response Headers

Will be set using setHeader method:

```
res.setHeader  (String name, String value)
res.setIntHeader  (String name, int value)
```

May be:

Header Name	Values & Details
cache-control	public | private | no-cache
max-age	will expire after XXX sec [int]
connection	close | keep-alive [= setContentLength()]
content-incoding	ISO-8859-1 | UTF-8 [define before writing]
last-modified	"Sun 18 Aug 2001 21:34:54" [for future check]
refresh	refresh long connection every XXX sec [int]

Other Response Headers

Cache-control

public - stored data will be available to all
private - stored data will be available only for this user
no-cache - no storing data

Refresh

browser will send <u>same</u> request in/every period of time
will be automatically canceled when changing address
use : - to send long data streams
- when the requested data is constantly updated

Ex: *labs\html\refresh.html [RefreshServlet.java]*

Other Response Headers

Pragma

Similar to cache-control only used for backward compatibility with HTTP 1.0. Cache-control is HTTP 1.1 version.

The no-cache value is used for pragma

Some browsers might respond only to pragma and not to cache-control

Setting the response headers

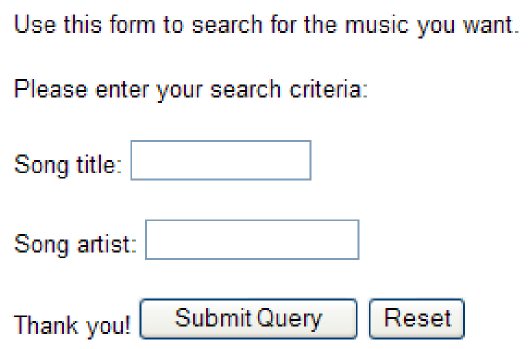
Can be set using the HttpServletResponse methods setHeader, setDateHeader and setIntHeader.

Some headers have their own methods:

- setContentType
- setContentLength
- addCookie
- sendRedirect

HTML form example

```
<P>Use this form to search for the music you want.
<FORM METHOD="POST" ACTION="/Music/SearchServlet">
<P>Please enter your search criteria:
<P>Song title:
<INPUT NAME="song_title" TYPE="TEXT" SIZE="12" MAXLENGTH="20">
<P>Song artist:
<INPUT NAME="song_artist" TYPE="TEXT" SIZE="15" MAXLENGTH="25">
<P>Thank you!
<INPUT TYPE="SUBMIT">
<INPUT TYPE="RESET">
</FORM>
```


Use this form to search for the music you want.

Please enter your search criteria:

Song title:

Song artist:

Thank you!

```
POST /Music/SearchServlet HTTP/1.1
Accept: */*
Referer:
http://www.music.ibm.com/Music/musicSearch.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; …)
Host: localhost:9080
Content-Length: 50
Connection: Keep-Alive
Cache-Control: no-cache
song_title=Hello&song_artist=Jones&limit_number=20
```

HTML form example

```
public class SearchServlet extends HttpServlet {
public void doPost(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {
…
Enumeration enum = req.getParameterNames();
while (enum.hasMoreElements()) {
String name = (String) enum.nextElement();
String value = req.getParameter(name);
//… do something with each pair…
{
…
{
{
```

- This example reads a post – when posting the form from the last slide the request goes to doPost() method
- Enumerate on all parameters and extract values

Lab 4

Dealing With Files

Files uploading

Files will be attached using ‘POST’ method

File path, name and data will be stored in the request

Use *request.getInputStream()* method in order to read data and than parse it.

There are many upload-file-Beans available that know how to parse the request for you

Ex: *labs\html\fileUpload.html [FileUploadServlet.java]*

Cookies

Servlet manages every connection as a new one

Cookie is the only way to identify a revisit client

Cookies

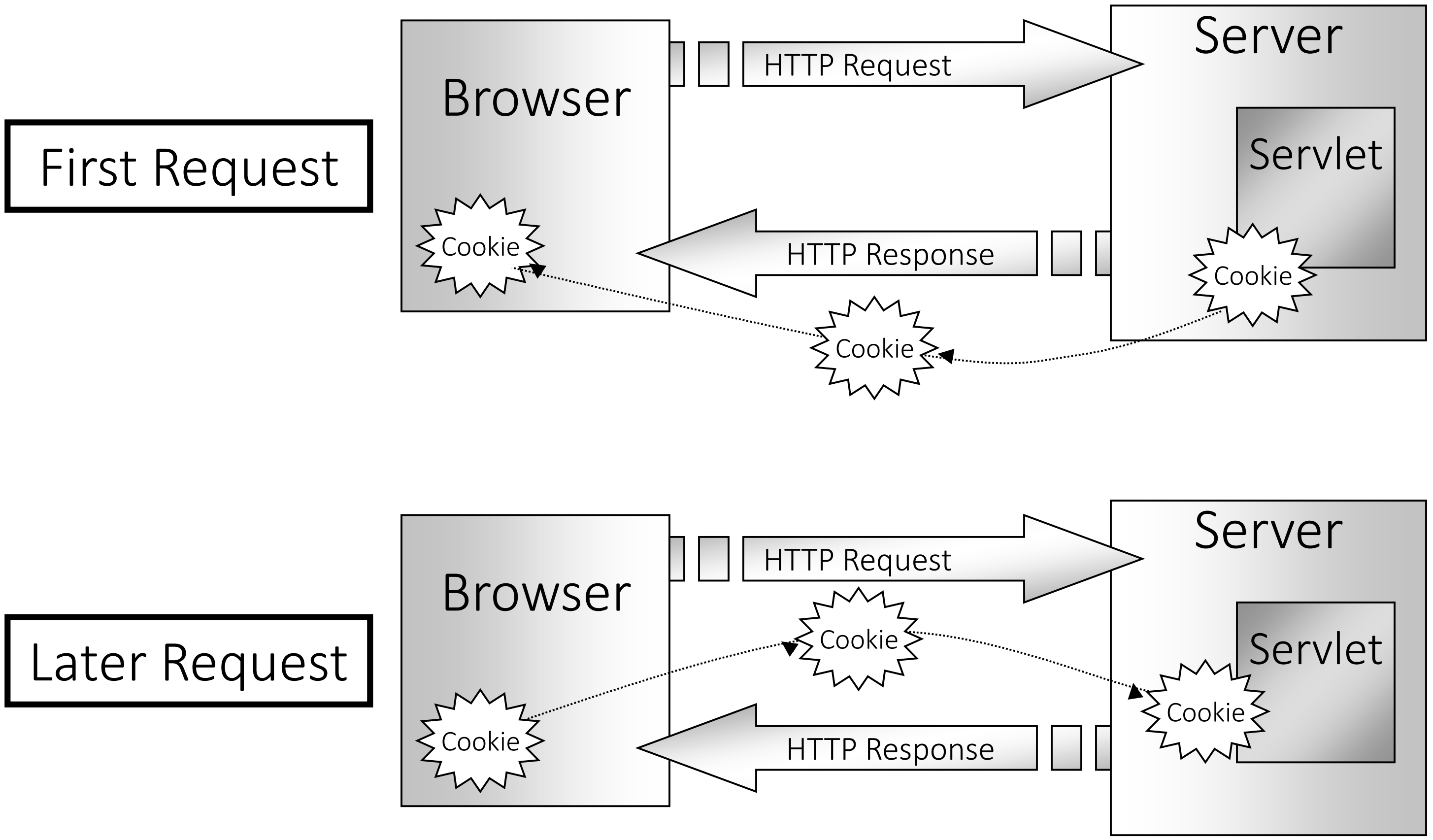
Cookies are sent by browsers request (if existed)

Server can implant many cookies within the response

Use to save information on client side

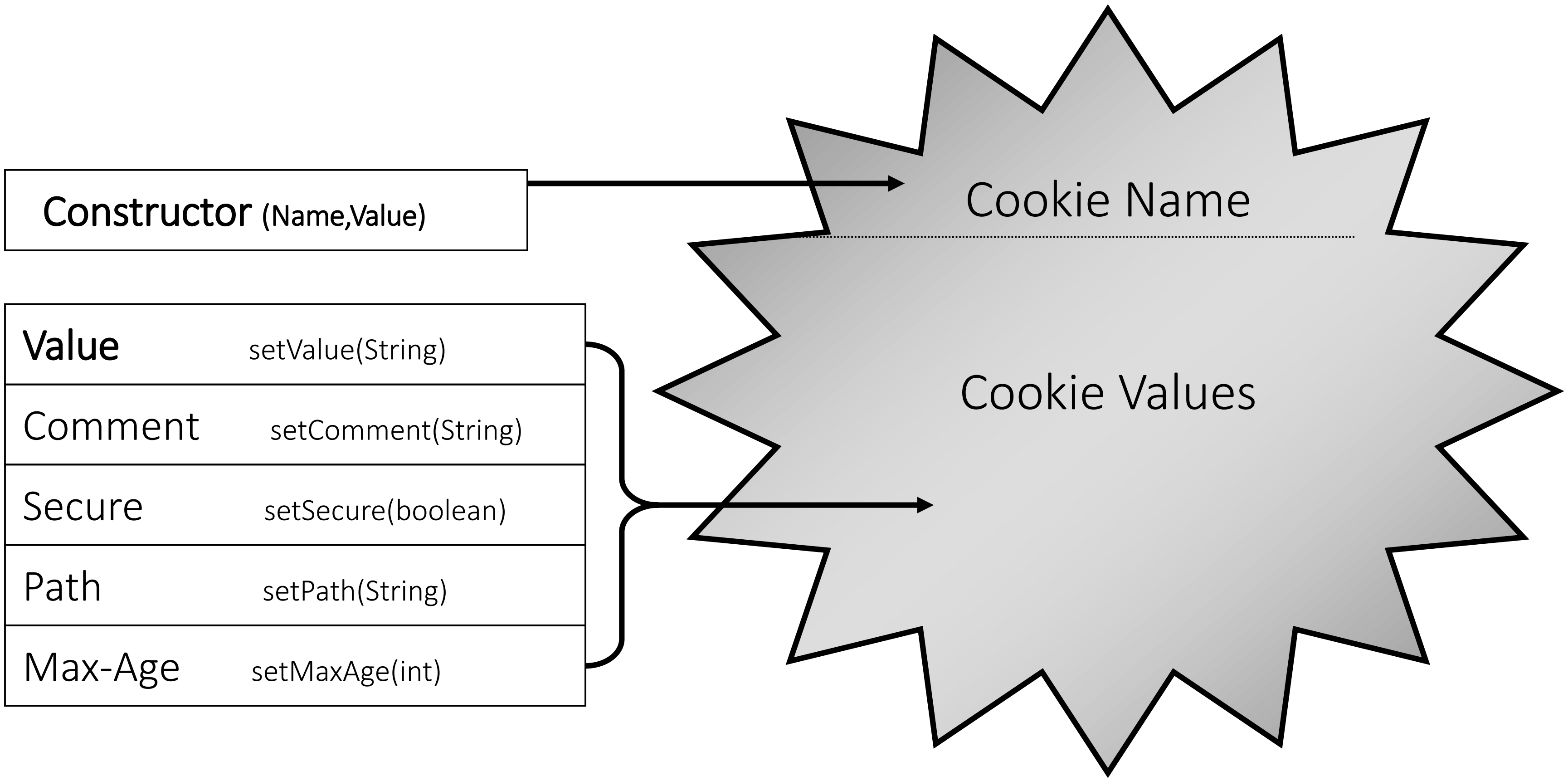
Can be blocked by the client

Cookies


First Request
Browser
HTTP Request
Server
Servlet
Cookie
HTTP Response
Cookie
Cookie
Later Request
Browser
HTTP Request
Server
Cookie
Servlet
Cookie
Cookie
HTTP Response

Cookies

Cookie Structure:



Value	setValue(String)
Comment	setComment(String)
Secure	setSecure(boolean)
Path	setPath(String)
Max-Age	setMaxAge(int)

Cookies

Creating & sending cookies:

Creating Cookie
• Cookie c = new Cookie (“Visits”,“1”);

Setting Cookie
• c.setDomain(“johnbryce.co.il”);
• c.setComment (“Counts user visits”);
• c.setSecure(false);
• c.setPath(“http://site.com/MySite/index.html”);
• c.setMaxAge(60*60*24*365);

Sending Cookie
• res.addCookie(c);

This cookie will be sent by the browser only to: http://site.com/MySite directory or its sub directories

The age of this cookie is set to a year
- 0 value: browser will delete the cookie
- negative value:cookie will last during session
- positive value: lifetime in seconds
! not used – cookie valid in **session scope**

Cookies

Getting & parsing cookies:

Getting Cookies

•Cookie [] cookies= req.getCookies()

Getting Cookie Fields Values

- String comment = cookie.getComment ();
- boolean secure = cookie.getSecure();
- String path = cookie.getPath();
- int maxAge = cookie.getMaxAge();
- String name = cookie.getName();
- String value = cookie.getValue();

Ex: *labs\html\cookie.html [CookieServlet.java]*

Ex: *labs\html\nameCookie.html [NameCookieServlet.java]*

Adding a cookie example

```
// Check to see if cookietest parameter is set
if (req.getParameter("cookietest") == null) {
         resp.addCookie(new Cookie("CookieTest", "ok"));
         String url = req.getRequestURI() + "?cookietest=ok";
         resp.sendRedirect(url);
         return;
}
// Callback from sendRedirect() above - check for cookie
ServletContext ctx = getServletContext();
if (req.getCookies() != null) {
         // Cookies were accepted, so handle appropriately
         ctx.getRequestDispatcher("/HandleCookie").
         forward(req, resp);
} else {
         // Cookies were declined, so handle appropriately
         ctx.getRequestDispatcher("/HandleNoCookie").
         forward(req, resp);
}
```

Add cookie

Retrieving a cookie example

```
public void doGet(HttpServletRequest req,
                                              HttpServletResponse res) {
  String CookieTest = "";
  Cookie[] cookies = req.getCookies();
  if (cookies != null) {
    for (int i=0; i<cookies.length; i++) {
      if(cookies[i].getName().equals("CookieTest"))
                   CookieTest = cookies[i].getValue();
    }
  }
  if (CookieTest.equals("error"))
      // do error HTML
  else    // do ok HTML
}
```

Lab 5

The stateless problem

HTTP protocol is stateless

Servlets and JSPs should be stateless

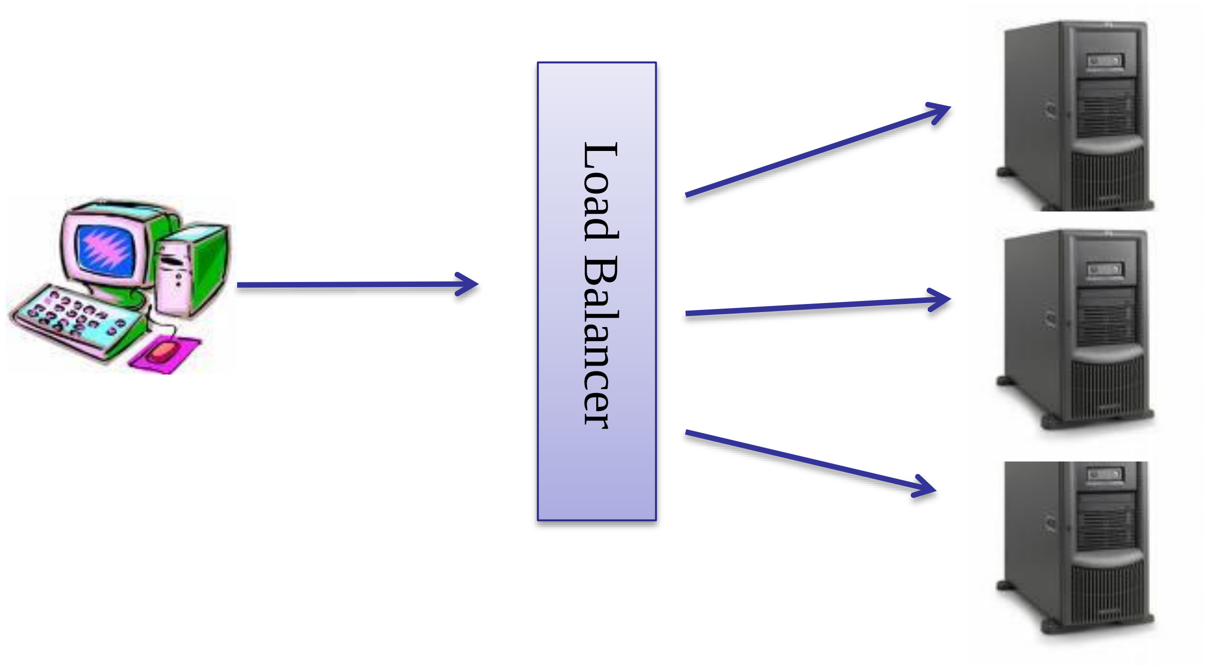
- No instance variables
- Multi thread support

Storing of user state is done outside the servlet or JSP and is called **session state**

Distributed environment

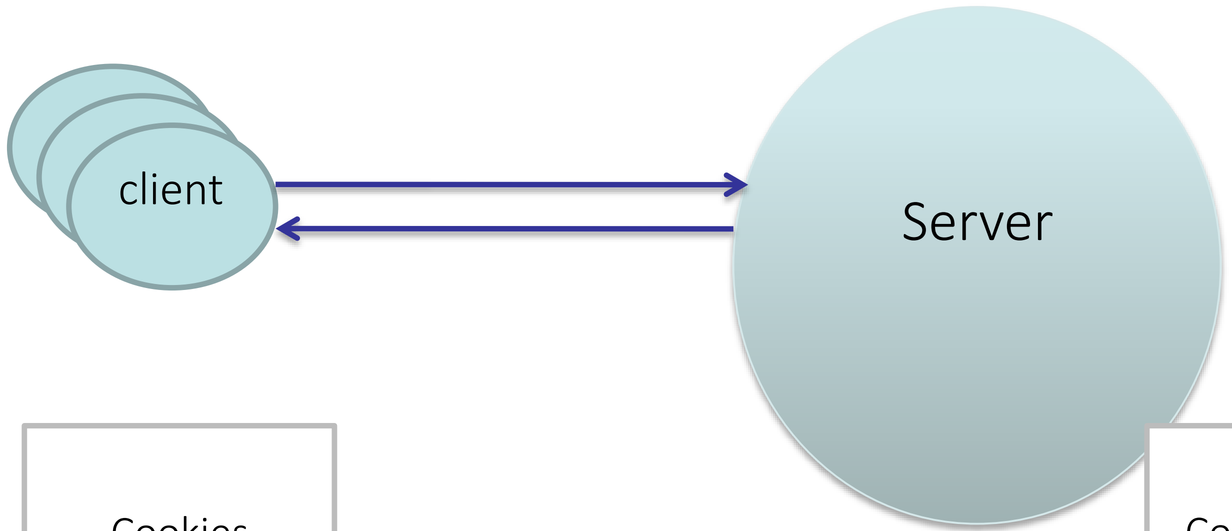
HTTP requests are spread across multiple servers

Application should be server independent – flexible for server changes



Solutions for session state

Can be done on server side or client side



Cookies
Hidden fields
URL rewriting

Content based routing
Database storage
Distributed sessions
Session Affinity

Sessions

The period of time that a server & a client are connected

In order to keep a connection identity - the client must be marked

'Marking' a client can be done by:

- Cookies - that can be blocked
- Marking URL - like:

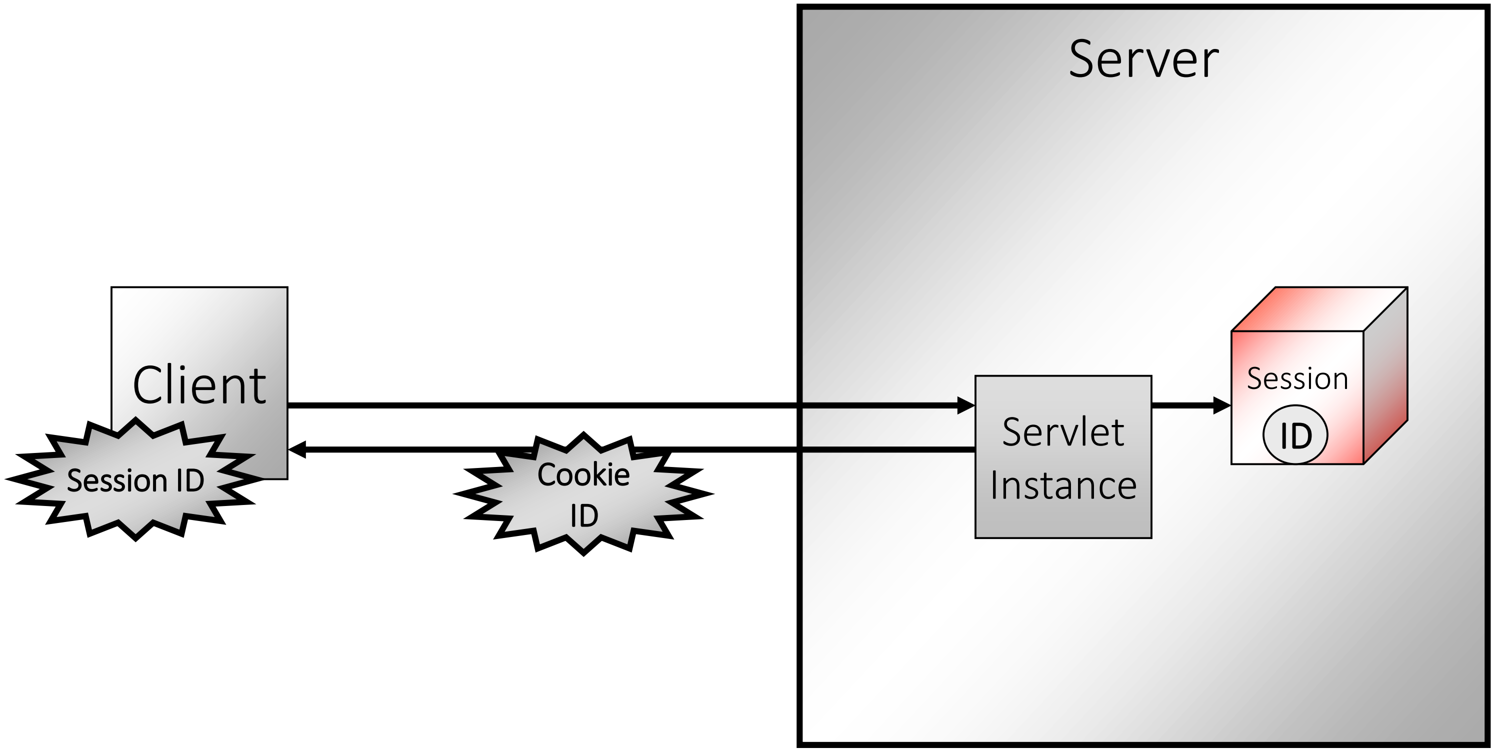
  that will hold as long as client uses Back & Next buttons

```
http://localhost:8080/MySite/res.html;sessionid=1234
```

The servlet engine choose the right way automatically

Sessions

How does it work ?



Sessions

JOHN BRYCE
Leading in IT Education
a matrix company

Getting old or creating new Session:

```
HttpSession session = req.getSession (true);
```

HttpSession
- Class that represents session instances created by the servlet engine per client
- Contain the session of current client

getSession (boolean b)
- Return a Session instance belong to the current client - if exists
- true = if doesn't exists - create new one

! Is this HttpSession new or old ?
- Use this method to get the answer: `public boolean isNew()`
- true = session was just created, false = session was existed already

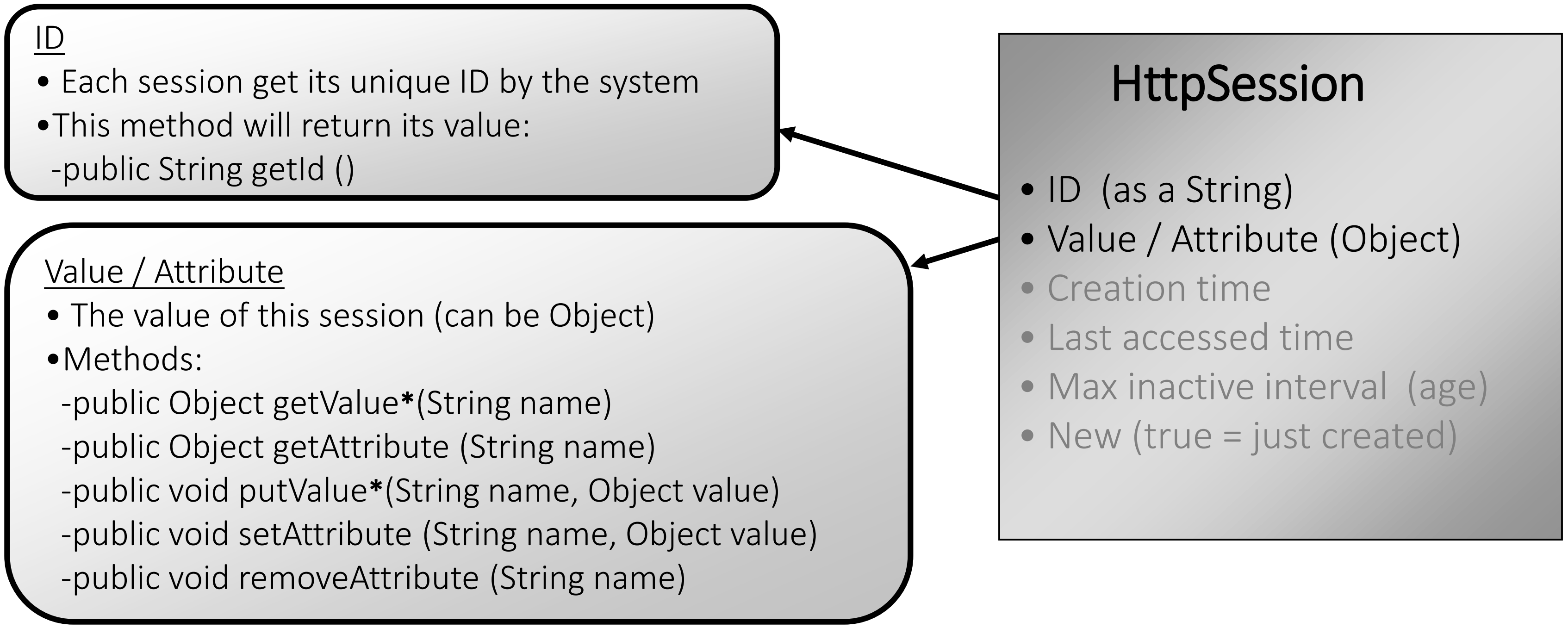
Terminating a session programmatically:

```
session.invalidate();
```

Sessions

JOHN BRYCE
Leading in IT Education
a matrix company

Session structure:



ID
- Each session get its unique ID by the system
- This method will return its value:
  -public String getId ()

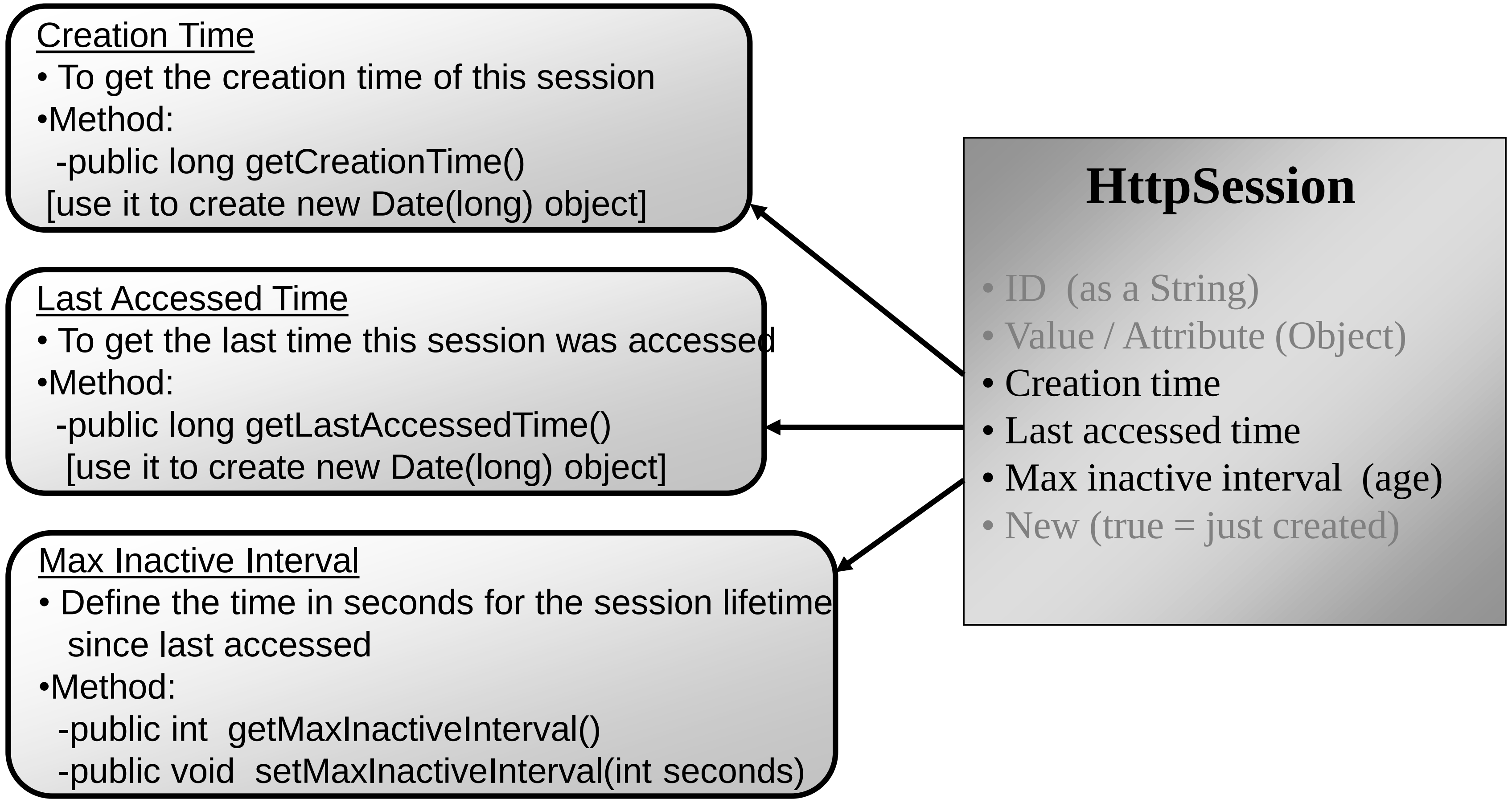
Value / Attribute
- The value of this session (can be Object)
- Methods:
  -public Object getValue*(String name)
  -public Object getAttribute (String name)
  -public void putValue*(String name, Object value)
  -public void setAttribute (String name, Object value)
  -public void removeAttribute (String name)

HttpSession
- ID (as a String)
- Value / Attribute (Object)
- Creation time
- Last accessed time
- Max inactive interval (age)
- New (true = just created)

* - deprecated methods

Sessions

- Session structure:



HttpSession

- ID (as a String)
- Value / Attribute (Object)
- Creation time
- Last accessed time
- Max inactive interval (age)
- New (true = just created)

<u>Creation Time</u>

- To get the creation time of this session
- Method:
  - -public long getCreationTime()
  - [use it to create new Date(long) object]

<u>Last Accessed Time</u>

- To get the last time this session was accessed
- Method:
  - -public long getLastAccessedTime()
  - [use it to create new Date(long) object]

<u>Max Inactive Interval</u>

- Define the time in seconds for the session lifetime since last accessed
- Method:
  - -public int getMaxInactiveInterval()
  - -public void setMaxInactiveInterval(int seconds)

Ex: *labs\html\session.html [SessionServlet.java]*

Session Invalidation

Programmatically invalidating – *session.invalidate()*

Session timeout – *session.setMaxInactiveInterval(int)*

  Can also be set for the application as a whole through web.xml

Session and multi threading

HttpSession is a shared resource and therefore access should be synchronized

```
User u = (User)session.getAttribute(“user”);
synchronized(u){
        //do something
}
```

Session serialization

In a clustered environment, sessions might need to be passed around servers

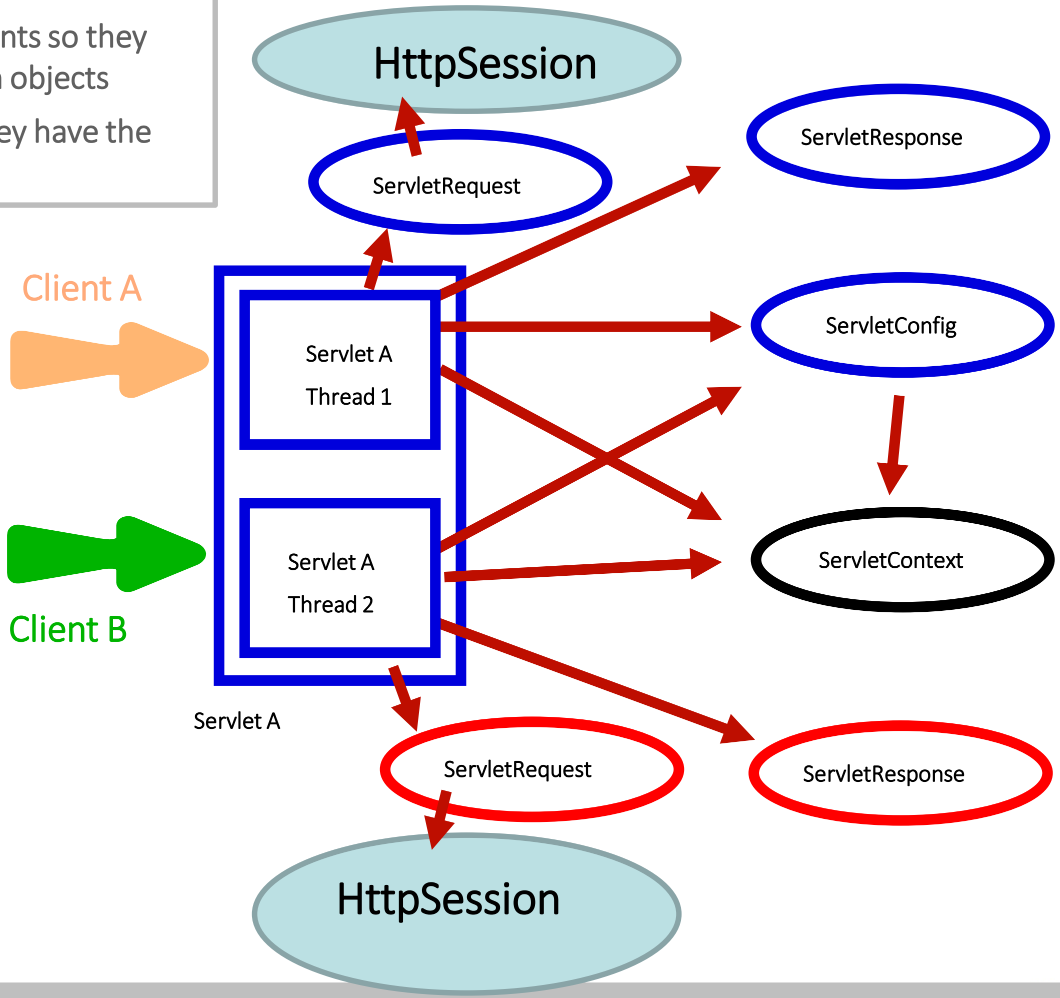
This requires objects stored in sessions to be Serializable

Sharing objects

The two clients are running the same servlet – servlet A.

Still, they are different clients so they have different HttpSession objects

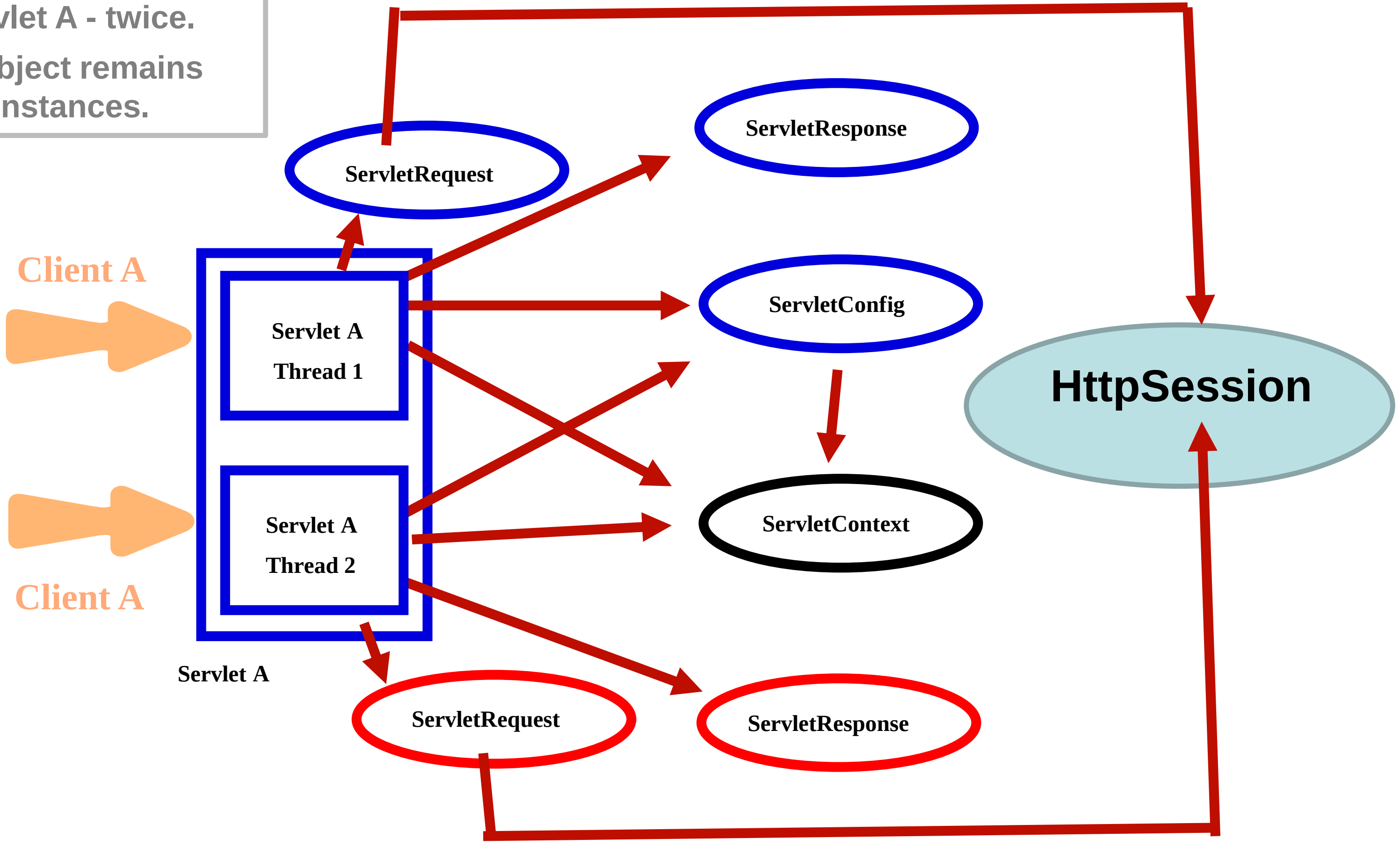
It's the same Servlet so they have the same ServletConfig object



Sharing objects

The same client is running the same servlet – servlet A - twice.

The HttpSession object remains the same for both instances.

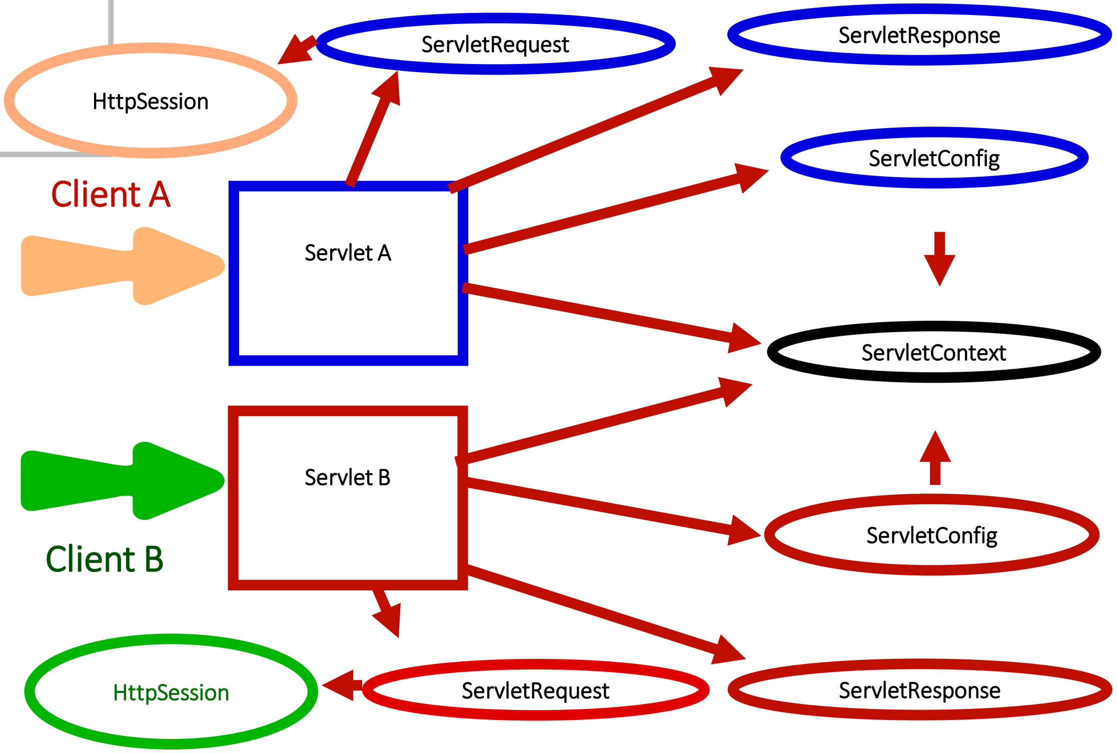


Sharing objects

The two clients are running the different servlets.

They are different clients so they have different HttpSession objects.

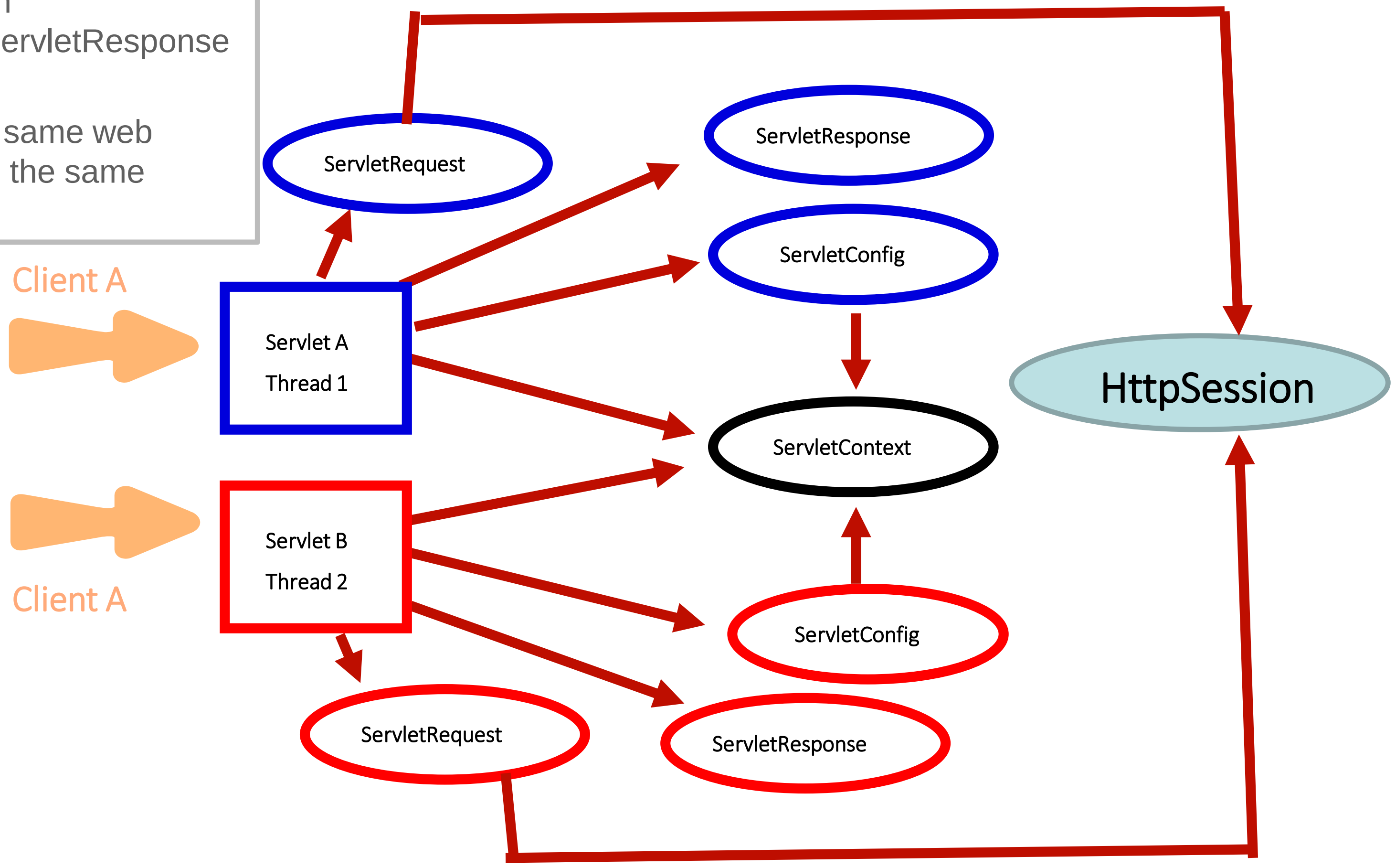
They have different ServletConfig objects



Sharing objects

The two clients are running different servlets each has his own ServletConfig and own ServletRequest and ServletResponse objects.

•Since both are in the same web application they share the same ServletContext



Session Events

The server can notify session events

Listener class will be registered in the *web.xml* file

Listener can listen to:

Session Events

Session Attribute Events

Session Events

Listener may implement the follow interfaces:

Session Events - HttpSesstionListener

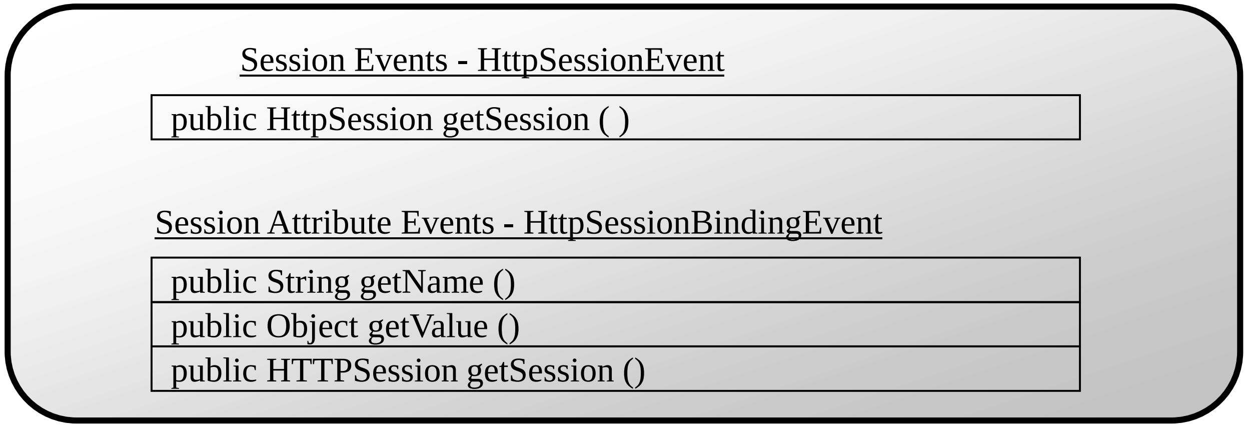
public void sessionCreated (HttpSessionEvent e)
public void sessionDestroyed (HttpSessionEvent e)

Session Attribute Events - HttpSessionAttributeListener

public void attributeAdded(HttpSessionBindingEvent e)
public void attributeReplaced(HttpSessionBindingEvent e)
public void attributeRemoved(HttpSessionBindingEvent e)

Session Events

Event classes provides information about the session / attribute


Session Events - HttpSessionEvent

public HttpSession getSession ( )

Session Attribute Events - HttpSessionBindingEvent

public String getName ()
public Object getValue ()
public HTTPSession getSession ()

Session Events

Registering Listener(s) in *web.xml* :

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>

   <listener>
     <listener-class>
          myPackage.MyListener
     </listener-class>
   </listener>

   <servlet>…
       ….
<web-app>
```

The name of the listener class located in *WEB-INF\classes*

Ex: *labs\html\sessionEvent.html [SessionEvensServlet.java & SessionListener]*

URL Rewriting

Useful when session cookies are

- Disabled
- Not supported (like in most cellular phones)

Server takes a given URL and insert in it the session ID

The new URL (re-written URL) will be used as the next client form action

Session ID will be part of the next request submitted by the client

Session will be bounded to the request automatically by the container

Still – not always safe since client might write the URL manually – without the session ID

URL rewriting

URL rewriting for session ID passing

- Requires special encoding API
- Site page flow should include the encoded information
- The flow has to include dynamically generated pages
- Stores the session identifier in the page returned to the user

URL Rewriting

Rewriting URL is done using the response method:

encodeURL

```
..
String url = response.encodeURL(“http://localhost:8080/mysite/nextSevlet”);
out.print(“<form action=“+url”+” method=‘GET’>”) …
```

Done to encode the session ID

Pages using redirect need to encode the session ID as well:

```
response.sendRedirect(response.encodeRedirectURL(
                              “http://localhost:8080/mysite/nextSevlet”));
```

Session management alternatives

	Advantages	Disadvantages
Cookies	Easy to use, automatically passed in the header, no coding, simpler app	Can be turned off, not secure, limit of 4096 bytes size of cookie
Hidden Fields	Unique for each client, larger data, no turning off, relatively small	Not secure under HTTP, increase network traffic, not supported by framework, text only, requires dynamic form
HttpSession	Stored on server, secure, supported by Servlet API	Problematic across servers, maintenance of session ID, large
Distributed sessions environment	Session advantages plus enables clustered environment	

Lab 6

Servlet Context

Defines a set of methods that a servlet uses to communicate with its servlet container

One instance is shared among all application components

Useful for:

- Get MIME type of files
- Load files as InputStream
- Dispatch requests
- Use log file services
- Manage attributes in the Context scope
- Get versions of servlet and its container

Obtaining Servlet Context

ServletContext is available from:

- Any servlet
- Servlet Request

Is handed to the servlet in the original init(config) method

- Use the method: getServletContext() to obtain it
- BUT ! Make sure you call super.init() from your init()

Servlet Context

ServletContext methods:

ServletContext public methods:

Object getAttribute (String attName)	get an attribute located in the Context
void setAttribute (String attName, Object value)	Set an attribute in the Context scope
ServletContext getContext (String url)	Gets other Context in the server (url begin with "/")
String getInitParameter (String paramName)	Returns an init param of the servlet – can be shared
int getMajorVersion () **2**.3	Servlets API version supported by the container
int getMinorVersion () 2.**3**	Servlets API version supported by the container
String getMimeType (String fileName)	Returns the MIME type of the given file
RequestDispatcher getRequestDispatcher(String path)	Returns the dispatcher for JSP/Servlet (url begin with "/")
RequestDispatcher getNamedDispatcher(String name)	Same as previous –but done according to in mapped name
String getRealPath (String relativePath)	Returns the full URL of the resource (http://.../index.html)
String getResource (String mappingName)	Returns the URL to the mapped resource
InputStream getResourceAsStream (String mappingName)	Returns an InputStream of the resource
String getServerInfo ()	Returns the Version of the server
void log (String msg)	Sends message to a log file managed by the server

Servlet Context

ServletContext can also be initiated in values specified in web.xml

web.xml

```
<!-- do not use these tags within the servlet element definition -->
<context-param>
      <param-name>supportMail</param-name>
      <param-value>support@comPany.com</param-value>
</context-param>
```

Use getInitParameter(..) method to get values

Servlet Context

Dispatching request to JSP example:

```
import javax.servlet.jsp.*;
public void doGet (HttpServletRequest req,  HttpServletResponse res) {
        try {
                ServletContext context =getServletContext();
                RequestDispatcher rd=context.getRequestDispatcher("/jsp/reqParam.jsp");
                rd.forward(req, res);
        } catch (Exception ex) {
            ex.printStackTrace ();
        }
}
```

RequestDispatcher

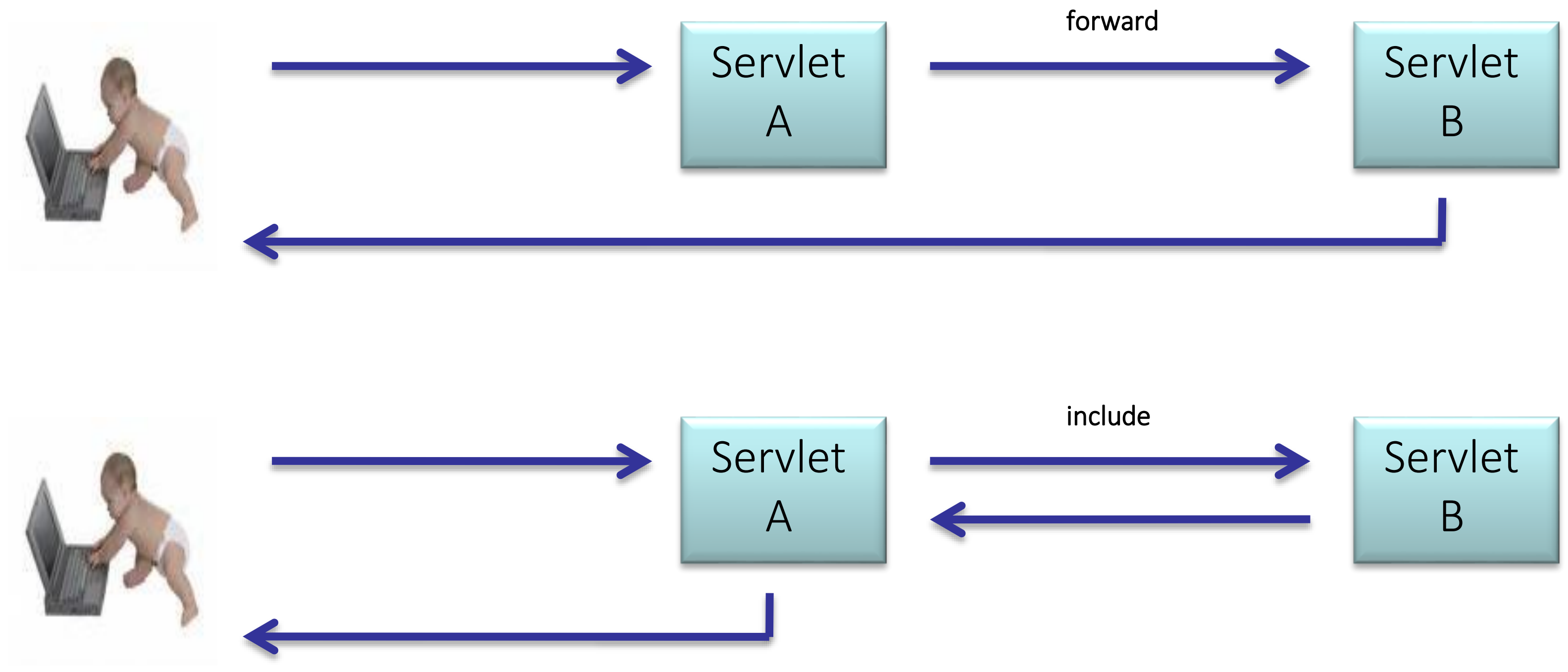
- call specified source and send HttpServletRequest & Response
- Source can be:
  - another servlet
  - JSP file

Ex: *labs\html\jspServlet.html   [JSPServlet.class]   [request.jsp]*

Request Dispatching

- Allows one servlet to delegate requests to another servlet
- Done on server side
  - Clients are not aware of this operation
  - Unlike – redirect
- Two methods are supported in the RequestDispatcher interface:
  - Forward
  - Include

Request Dispatcher flow



Forwarding

Servlet A should not write to the out stream – considered a flush

The request object can be changed – headers and status code can be set

Parameters from the original request stay throughout

Execution continues after the forwarding

Example:

```
        getServletContext().getRequestDispatcher("/pages/showBalance.jsp").forward(request, response);
```

Forwarding

To have another resource build the response use RequestDispatcher object forward

```
getRequestDispatcher(resourceName).forward(request,response)
```

IllegalStateException would be thrown if the source servlet tries to access the OuputStream or Writer object.

Forward vs. sendRedirect –

with forward the request object of the sender is passed to the reciever.

sendRedirect is a temporal redirect – therefore it is a new request object!

Including

Http headers should not be written by Servlet B

The request and response objects should not be changed by servlet B

Servlet A may generate response content before or after the include call to servlet B

```
getServletContext().getRequestDispatcher("/pages/navigation_bar.html").
include(request, response);
```

Sharing objects

ServletContext approach

```
getServletContext().setAttribute("objectName",anObject);
getServletContext().getAttribute("objectName");
```

When groups of servlets need to work with the same object

setAttribute replaces if value exists already and updates listeners accordingly. Passing null is like removeAttribute

Request object approach

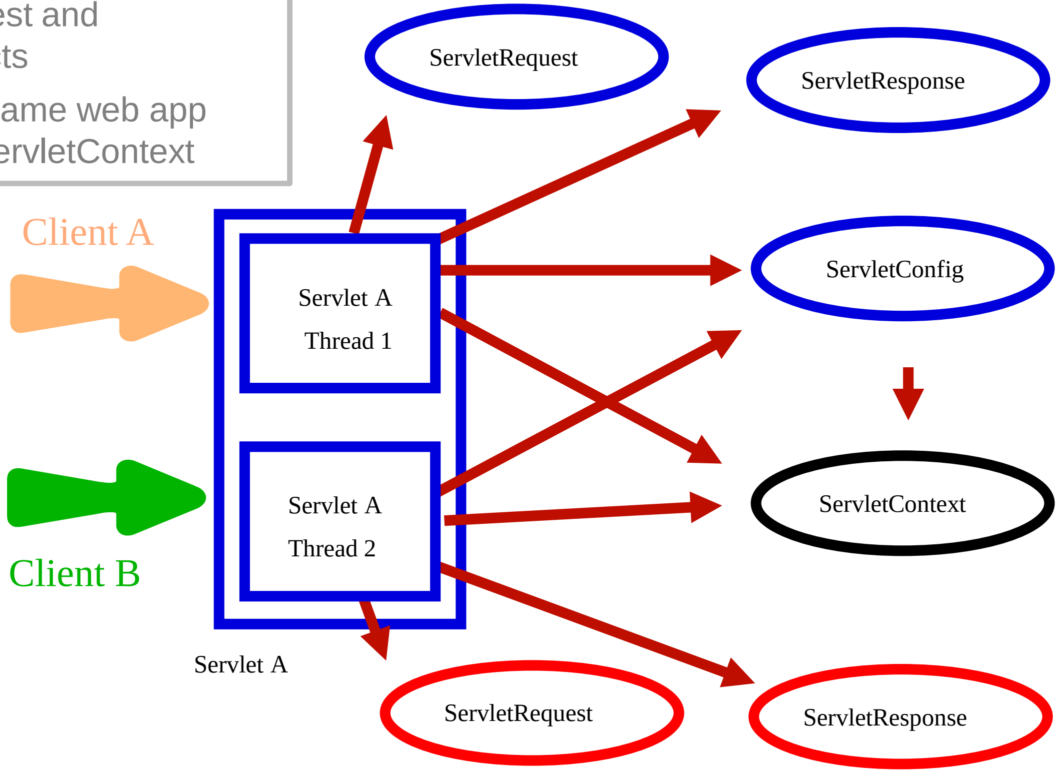
```
                request.setAttribute("objectName",anObject);
request.getAttribute("objectName");
```

For sharing objects between servlets when doing a forward or include

Can be done by the container or programmatically

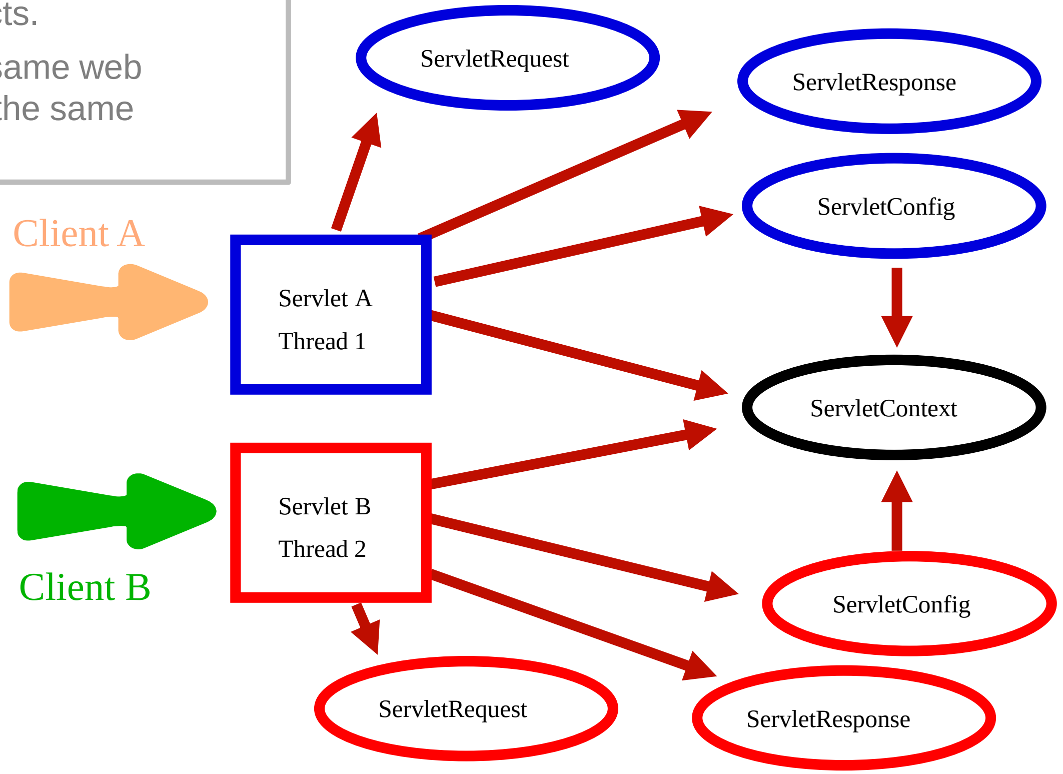
Sharing objects

- The two clients are running the same servlet – servlet A. Therefore, they share the same ServletConfig but each have their own ServletRequest and ServletResponse objects
- Since they are in the same web app they share the same ServletContext



Sharing objects

- The two clients are running different servlets each has his own ServletConfig and own ServletRequest and ServletResponse objects.
- Since both are in the same web application they share the same ServletContext



Servlet Listeners

A class that listens to certain events in a web application

Listening to state changes in web modules

Provide more control over application, session and request objects.

Monitoring is centralized – increases reuse

Servlet Listeners interfaces

Object	Actions	Interface	Methods

ServletContext	Create
	Destroy
		javax.servlet.
		ServletContextListener
			contextInitialized(ServletContextEvent e)
			contextDestroyed(ServletContextEvent e)

ServletContext	Add attribute
	Remove attribute
	Replace attribute
		javax.servlet.
		ServletContextAttributesListener
			attributeAdded(ServletContextAttributeEvent e)
			attributeRemoved(ServletContextAttributeEvent e)
			attributeReplaced(ServletContextAttributeEvent e)

HttpSession	Create
	Destroy
		javax.servlet.http.
		HttpSessionListener
			sessionCreated(HttpSessionEvent e)
			sessionDestroyed(HttpSessionEvent e)

HttpSession	Add attribute
	Remove attribute
	Replace attribute
		javax.servlet.http.
		HttpSessionAttributesListener
			attributeAdded(HttpSessionBindingEvent e)
			attributeRemoved(HttpSessionBindingEvent e)
			attributeReplaced(HttpSessionBindingEvent e)

ServletRequest	Create
	Destroy
		javax.servlet.
		ServletRequestListener
			requestInitialized(ServletRequestEvent e)
			requestDestroyed(ServletRequestEvent e)

ServletRequest	Add attribute
	Remove attribute
	Replace attribute
		javax.servlet.
		ServletRequestAttributesListener
			attributeAdded(ServletRequestAttributeEvent e)
			attributeRemved(ServletRequestAttributeEvent e)
			atttributeReplaced(ServletRequestAttributeEvent e)

Defining Listeners

Listeners are defined in the web.xml file

A <listener> tag defines one listener\

Example:

```
<listener>
        <listener-class>
                mypackage.SessionCounter
        </listener-class>
</listener>
```

Lab 7

Servlet Filters

Servlet Filters are used to decorate the requests & responses

Reusable components which transform the content of HTTP requests, responses and headers

Are indirectly invoked by the client

Can be used dynamically for each servlet

Are loaded on server startup

Part of Servlet API 2.3

Servlet Filters

Pre-processing operations that filters can do:

- User authentication
- Session validation
- Denial of service (blocking heavy requests)
- Request decryption
- Request decompression
- Request auditing

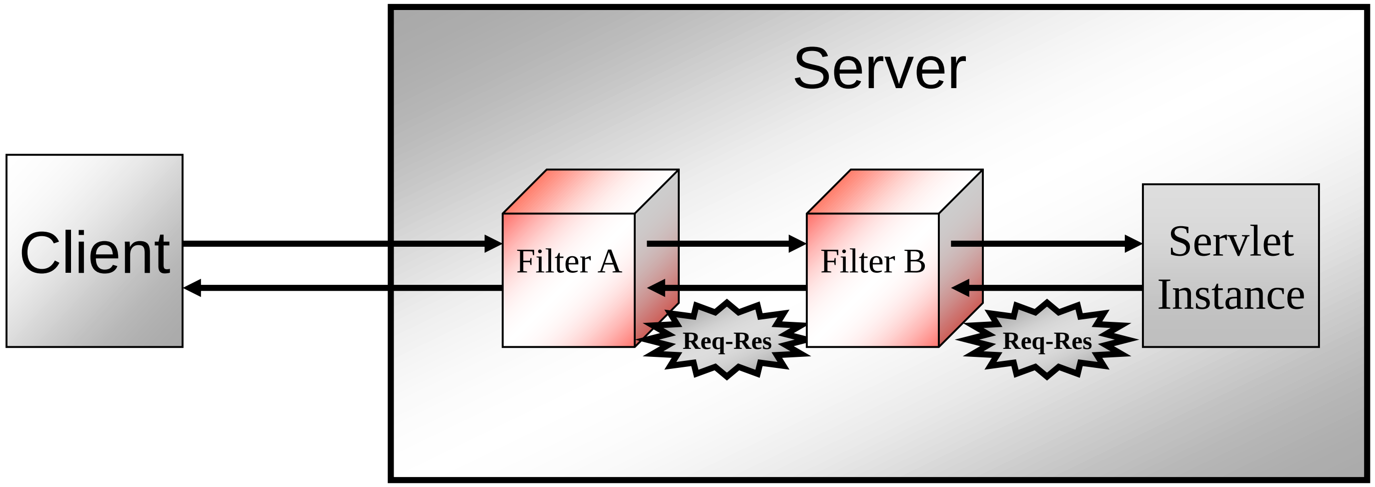
Servlet Filters

Post-processing operations that filters can do:

- Response customization (according to the type of the client)
- Response encryption
- Response compression
- Language translation
- Debugging

Servlet Filters

How does filters work ?



Servlet Filters

Writing Filters:

Class is denoted with @WebFilter annotation

Filter class must implement *Filter* interface

Implement the methods:

- *init (filterConfig) throws SerlvetException –*
  - Called by container when placing filter in service
  - Parameter – FilterConfig
- *doFilter (request, response, FilterChain)*
  - Called each time request/response is passed through the chain
  - Parameters – ServletRequest, ServletResponse, FilterChain
- *destroy ()*
  - Performs filter cleanup and is called when placing filter out of service
  - No parameters

Servlet Filters

Writing Filters:

FilterChain holds the chain of filter mapped to a particular servlet.

```
import javax.servlet.*;
import javax.servlet.http.*;
@WebFilter("/MyFilter")
public class  MyFilter implements Filter{
   …..
   public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
              pre-processing tasks
              chain.doFilter(req,res);
             post processing tasks
   }
   …..
}
```

FilterConfig interface

Includes the following methods:

getFilterName() – returns name of filter as String

getInitParameter() – returns initialization parameter value as String

getInitParameterNames() – Returns an enumeration of Strings with the names of the init parameter

getServletContext() – Returns a reference to the ServletContext when the filter is operating

Servlet Filters

Defining Filters in the DD

- Declaring Filter name & class (optional)
- Mapping the Filter to a servlet(s)

Chain is determined according to the mapping order for each servlet

Servlet Filters

Example:

web.xml

```
<filter>
     <filter-name>MyFilter</filter-name>
     <filter-class>myPackage.MyFilter</filter-class>
  </filter>
  <filter-mapping>
               <filter-name>MyFilter</filter-name>
               <servlet-name>/MyServlet</servlet-name>
  </filter-mapping>
```

Defining filter's name & class (optional – can be done via @WebFilter)

Mapping filter to a particular servlet
(use the servlet URL pattern to map)

Filter chaining configuration

According to the order of filter mapping elements in the DD

Web resource is invoked by last filter in the list

Filters that match a certain url pattern are executed before filters that match servlet name element of the requested web resource

Order it determined by order of appearance in web.xml

Filter chaining configuration

```
<filter-mapping>
          <filter-name>LoginChecker</filter-name>
  <servlet-name>Login</servlet-name>
</filter-mapping>
<filter-mapping>
          <filter-name>Logger</filter-name>
          <url-mapping>/*</url-mapping>
</filter-mapping>
<filter-mapping>
  <filter-name>LoginTrailer</filter-name>
          <servlet-name>Login</servlet-name>
</filter-mapping>
```

Mapping Order for Prime Servlet:

(1) Logger

(2) LoginChecker

(3) LoginTrailer

Lab 8

JSP

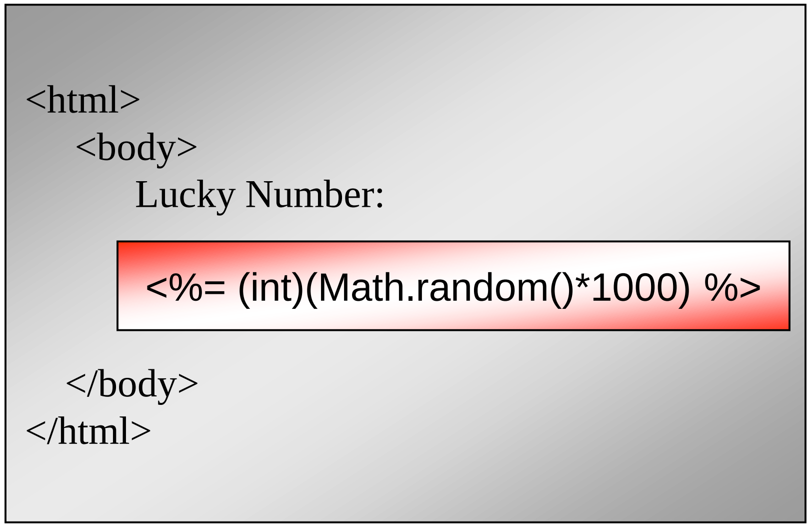
Introduction

JSP

JSP file contain:

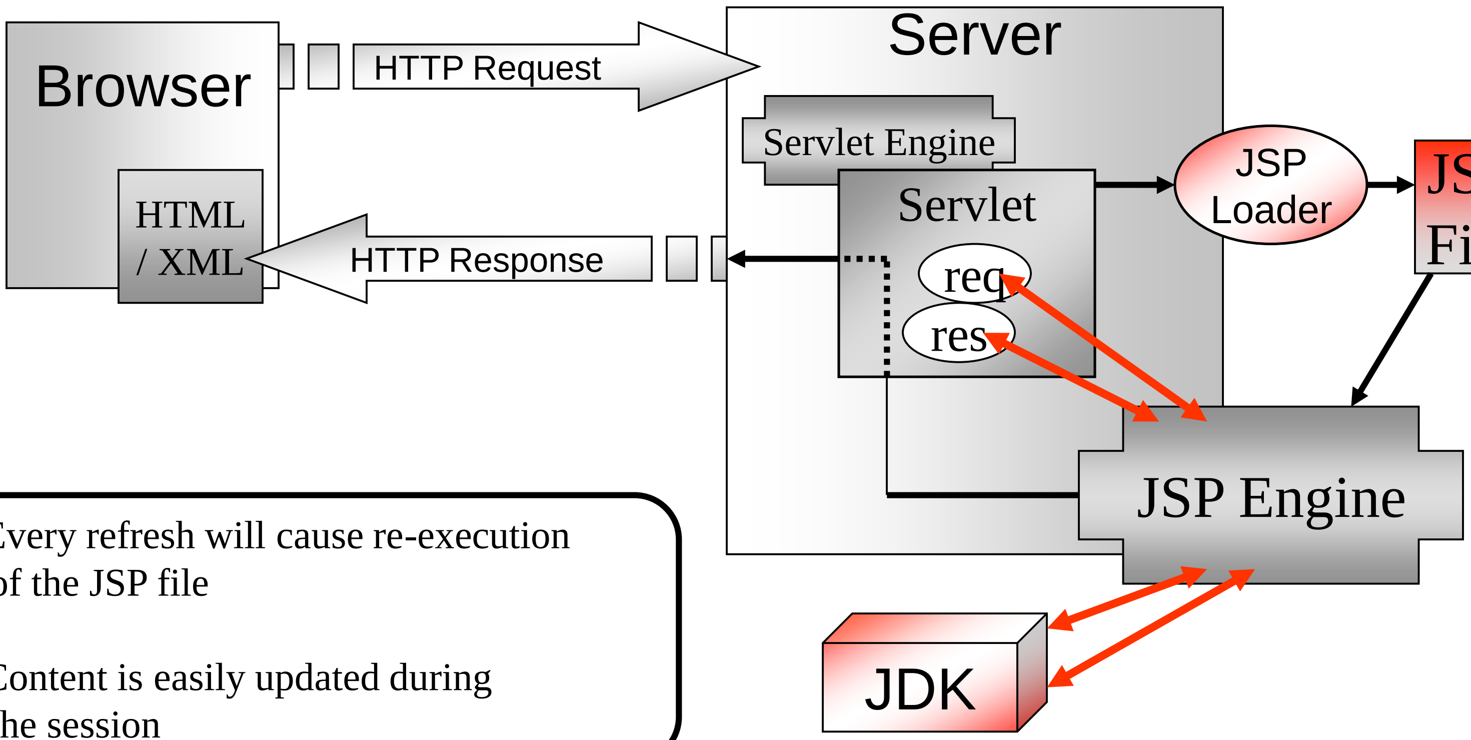
▨ static blocks (HTML / XML)

🟥 dynamic blocks (JSP)



```
<html>
    <body>
        Lucky Number:
          <%= (int)(Math.random()*1000) %>
    </body>
</html>
```

How Does It Work ?



- Every refresh will cause re-execution of the JSP file
- Content is easily updated during the session

Calling JSP Files

JSP files can be saved in the same directory of html files.

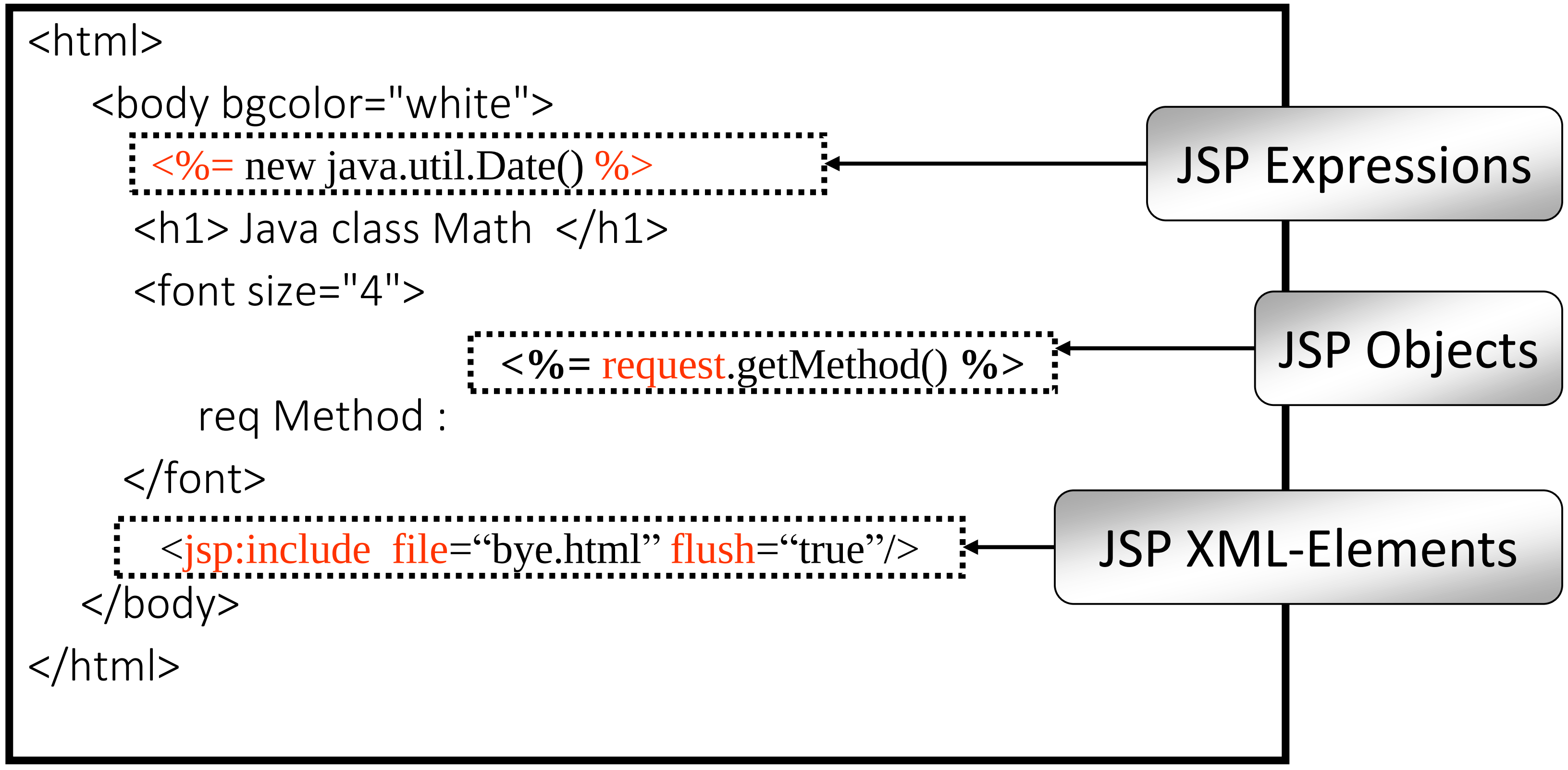
To this location:

*c:\TomCat\jakarta-tomcat-4.0-b7\webapps\MySite\**jsp\myFile.jsp***

the URL is:

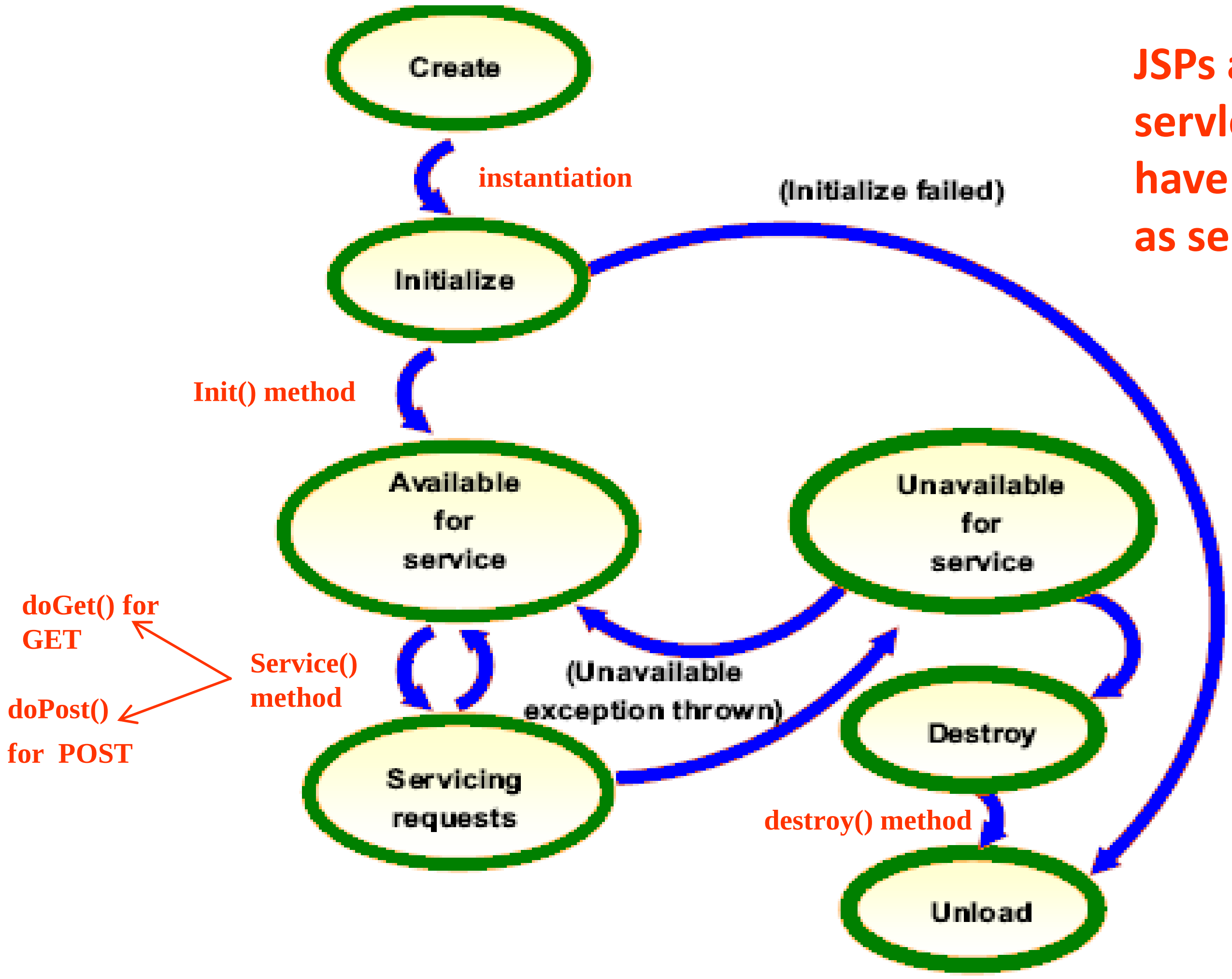
*http://localhost:8080/MySite/**jsp/myFile.jsp***

JSP Document



```
<html>
    <body bgcolor="white">
      <%= new java.util.Date() %>
      <h1> Java class Math  </h1>
      <font size="4">
                          <%= request.getMethod() %>
          req Method :
      </font>
      <jsp:include  file="bye.html" flush="true"/>
    </body>
</html>
```

JSP Lifecycle



JSPs are comiled into servlets – they actually have the same life cycle as servlets!

JSP Syntax

<% *code* **%>** - run Java process (code executed for each socket)

<%= *code* **%>** - run Java process and print result

<%! *code* **%>** - global servlet statements (not per socket)

<%@ directive %> - using JSP Directives

<%-- *comment* **--%>** - will not be added to the result

JSP Objects

request - represent the HttpServletRequest object

response - represent the HttpServletResponse object

out - represent the HttpServletResponse PrintWriter object

session - represent the Session object of the current connection

config - represent the ServletConfig of the Servlet

pageContext - a JSP object PageContext for dealing with JSP documents

exception - a thrown Exception instance if there is any

application - represent the ServletContext object

```
<html>
    <body bgcolor="white">
            Req Method : <%= request.getMethod() %>
    </body>
</html>
```

JSP Objects - Session

Session created by using cookies or URL rewriting

Session are created by default – unless disabled [later]

HttpSession instance named 'session'

JSP XML-Elements

```
<!ELEMENT jsp:setProperty>
    <!ATTLIST  jsp:setProperty
                    name     CDATA   #REQUIRED
                    property CDATA #REQUIRED
                    value    CDATA   #REQUIRED>
```

and some more …

<jsp:scriptlet>
<jsp:include>
<jsp:plugin>
<jsp:param> & <jsp:params>
<jsp:fallback>
<jsp:forward>
<jsp:setProperty> <jsp:getProperty>
<jsp:useBean>

```
<jsp:setProperty name="customer1"
                 property="id"
                 value="1234"/>
```

Using Java Code With JSP

JOHN BRYCE
Leading in IT Education
a matrix company

<% code %> and *<%= code %>*

Java Expressions

- **<%=** new java.util.Date() **%>**
- **<%** String word = "Hello"; **%>**
  <%= word %>
- **<%** if (Math.random()>0.5){ **%>**
  <b> you can do this </b>
  <% } else { **%>**
  <b>& you can do that </b>
  <% } **%>**

Using JSP Objects

- **<%=** request.getMethod() **%>**
- **<%** if (Math.random()>0.5){
  out.print("you can do this");
  } else {
  out.print("& you can do that");
  } **%>**

Naming security roles to access restricted resource

<auth-constraint> - tag that tells the container "only the mentioned role has access to this area.“

<role-name> - enclosed by *<auth-constraint>.* Name of role to be later used in restriction

Mapping Security Roles in web.xml

Mapping roles to an existing servlets of JSP defined in *web.xml*

Mapping is done according to one or more URL patterns.

Predefined roles is linked to the resources here

Example:

```
…
<servlet-mapping>
      <servlet-name>ConnectServlet</servlet-name>
      <url-pattern>/con</url-pattern>
</servlet-mapping>
…
<security-constraint>
   <web-resource-collection>
      <web-resource-name>Connect Task</web-resource-name>
           <url-pattern>/con/*</url-pattern>
           <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
           <role-name>manager</role-name>
      </auth-constraint>
  </security-constraint>

<security-role>
     <role-name> manager </role-name> …
```

Defining Security Roles in web.xml

Example:

Defining roles according to the authorization policy desired for the web application

The security roles are actually role references that will be mapped to real role define as part of the web-server security policy using vendor specific tools and configuration files.

```
…
<servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
       <servlet-name>HelloServlet</servlet-name>
       <url-pattern>/Hello</url-pattern>
</servlet-mapping>
…
<security-role>
     <role-name> manager </role-name>
     <role-name> employee </role-name>
     <role-name> client </role-name>
</security-role>
```

Naming users/groups in the role

A "role" is just this abstract thing, but we need to tie it to the real security system.

This is where we move away from the standards, and the particular server takes over.

```
              <servertpye-web-app>
    <security-role-assignment>
           <role-name>manager</role-name>
                 <principal-name>system</principal-name>
     </security-role-assignment>
</servertype-web-app>
```

We tie to the role name manager, and give the usernames, or groups that are part of that role. In this case, granting access to /secure url to the user "system.“

When a browser accesses /con, only the users in the manager role will get through!

Authentication Options

We have defined the users that are allowed access to a resource

Now, we need to tell the container how we want to authenticate the users.

There are four authentication methods to choose from:

Authentication Method	Description
BASIC	Use HTTP basic authentication. Pop up window will show trying to access /secure.
FORM	For building your authentication into your Web pages using DD
CLIENT-CERT	We can use client digital certificates to authenticate against.

Form based authentication

In the DD, tell the container to use FORM-based authentication

Specify "FORM" as the auth-method (instead of BASIC, DIGEST, or CLIENT-CERT),

Specify Web page LoginForm.html has the <FORM> which will authenticate a user.

Accessing a page under /con we require filling out the form in LoginForm.html and authenticating

If authentication fails redirect to LoginError.html will occur

```
              <login-config>
              <auth-method>FORM</auth-method>
                  <form-login-config>
                  <form-login-page>/LoginForm.html</form-login-page>
                  <form-error-page>/LoginError.html</form-error-page>
                  </form-login-config>
 </login-config>
```

Building the form

HTML form in LoginForm.html:

```
<form method="POST" action="j_security_check">
                    Username: <input type="text" name="j_username"><br />
                    Password: <input type="password" name="j_password"><br /> <br />

                    <input type="submit" value="Login">
                    <input type="reset" value="Reset">
</form>
```

If the authenticated user is part of the admin role (e.g. system user), the original resource will be sent back to the user, otherwise the LoginError.html will.

Authentication form convections:

1. Our <form>'s action
   field must be j_security_check
2. We must have form
   fields j_username, and j_password that hold the username and password to authenticate with

Enforcing SSL

For controlling the level of security in the transport mechanism using the following tag in web.xml:

```
        <user-data-constraint>
          <description>SSL not required</description>
          <transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
```

There are three possible values for the <transport-guarantee>

NONE - No encryption is required (http is fine)

CONFIDENTIAL - The data must be encrypted, so that other parties can not observe the contents (e.g. enforce SSL)

INTEGRAL - The data must be transported so that the data cannot be changed in transit.

Web Application Archive

Web apps. Can be packed and deployed as one unit

Application is zipped to a *war* file

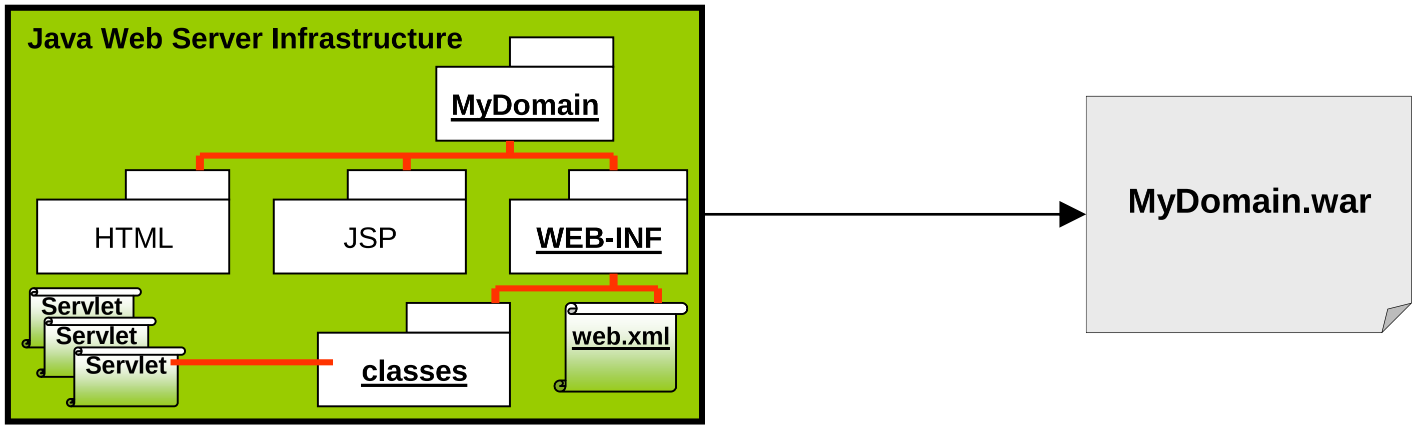
Web containers can extract and work with *war* files

war stands for Web Application Archive

In order to pack – use Java jar utility

The name of the war file defines the application’s name

Web Application Archiving



```
C:\MyDomain> jar –cf  MyDomain.war   *.*
C:\MyDomain>
```