Java EE

Cours 2

Les Servlets

Cours de 2e année ingénieur

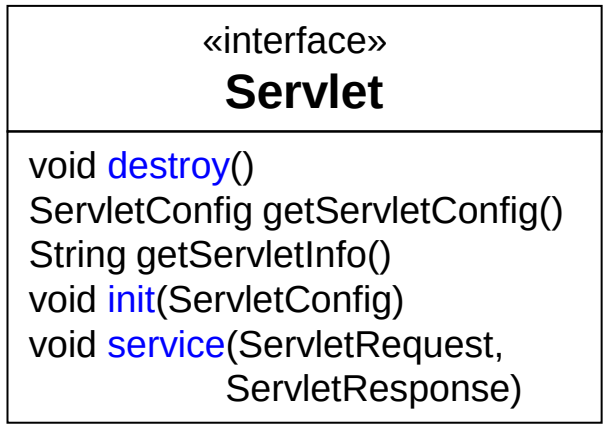
Servlets

• Une **servlet** est une application Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Ces données sont le plus généralement présentées au format HTML, mais elles peuvent également l'être au format XML ou tout autre format destiné aux navigateurs web.

• Une **servlet** s'exécute **dynamiquement** sur le serveur web et permet l'extension des fonctions de ce dernier. Typiquement : accès à des bases de données, transactions d'e-commerce, etc. Une **servlet** peut être chargée automatiquement lors du démarrage du serveur web ou lors de la première requête du client. Une fois chargées, les **servlets** restent actives dans l'attente d'autres requêtes du client.

Servlets

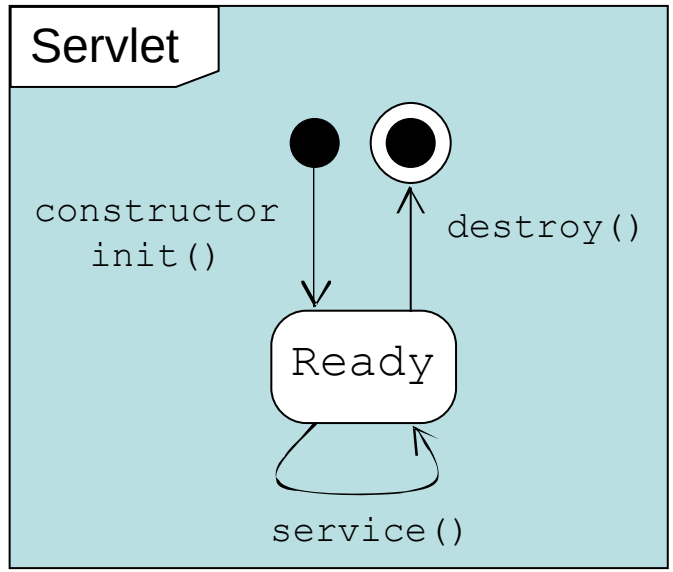
- Une servlet est un objet qui peut être manipulé par le conteneur via l'interface suivante:



- Lorsque le conteneur reçoit une requête, il la transmet au servlet qui correspondàl'URL pour que la requête soit traitée effectivement

Cycle de vie d'une servlet

1. Chargement de la classe
2. Instanciation du servlet
   - constructeur par défaut
3. Appel de init()
4. Appel(s) de service()
   - 1 thread par requête
5. Appel de destroy()



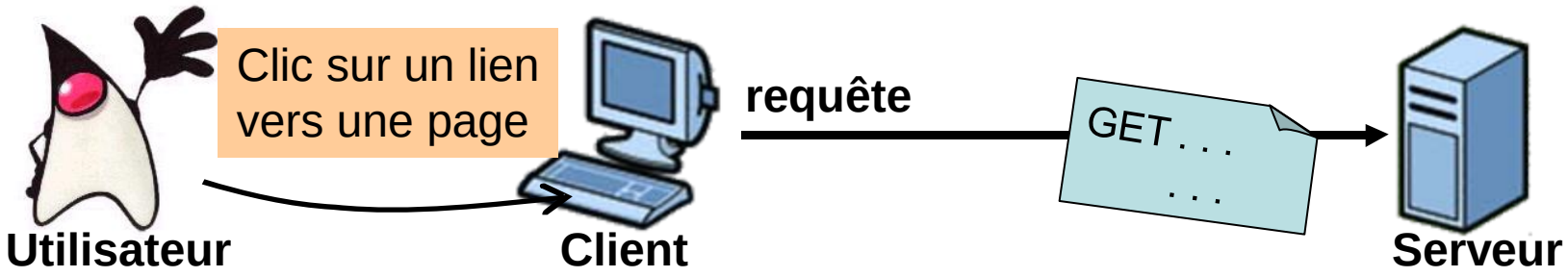
La méthode `service()`

- Lors de la réception d'une requête, le conteneur crée:
  - un objet `ServletRequest` (la requête), et
  - un objet `ServletResponse` (la réponse)
- Le conteneur appelle ensuite la méthode `service()` avec ces deux objets en paramètres pour permettre au servlet de répondre à la requête du client.

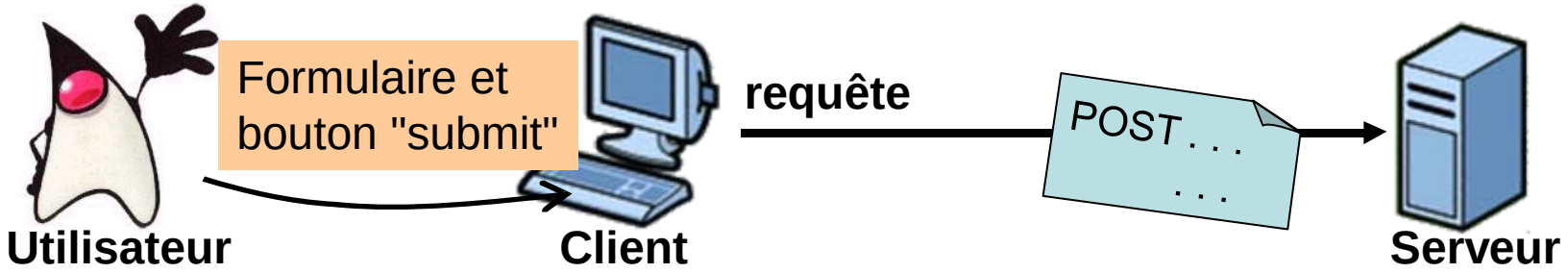
Requêtes HTTP (rappel)

- Deux méthodes principales: GET et POST

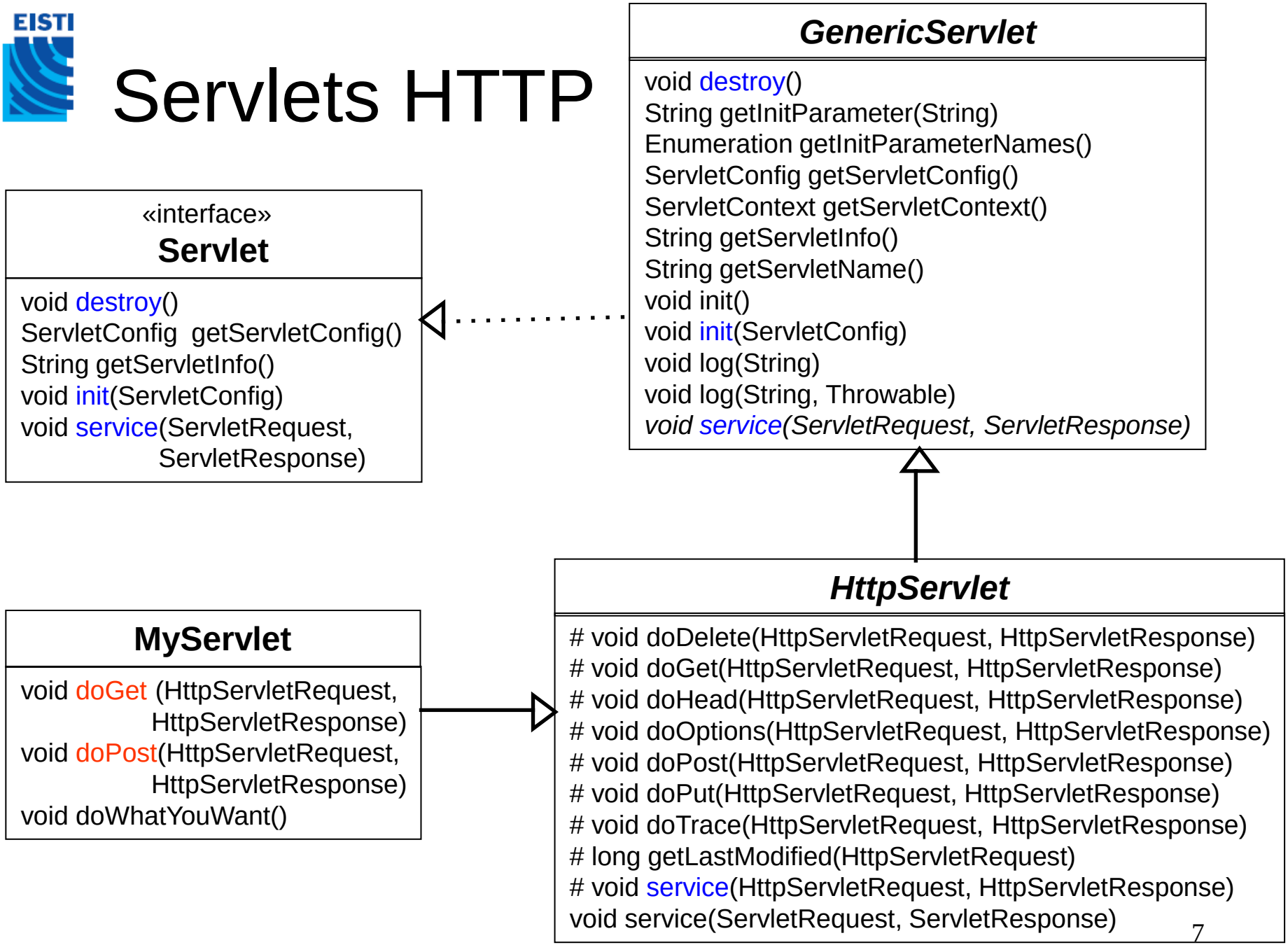
GET



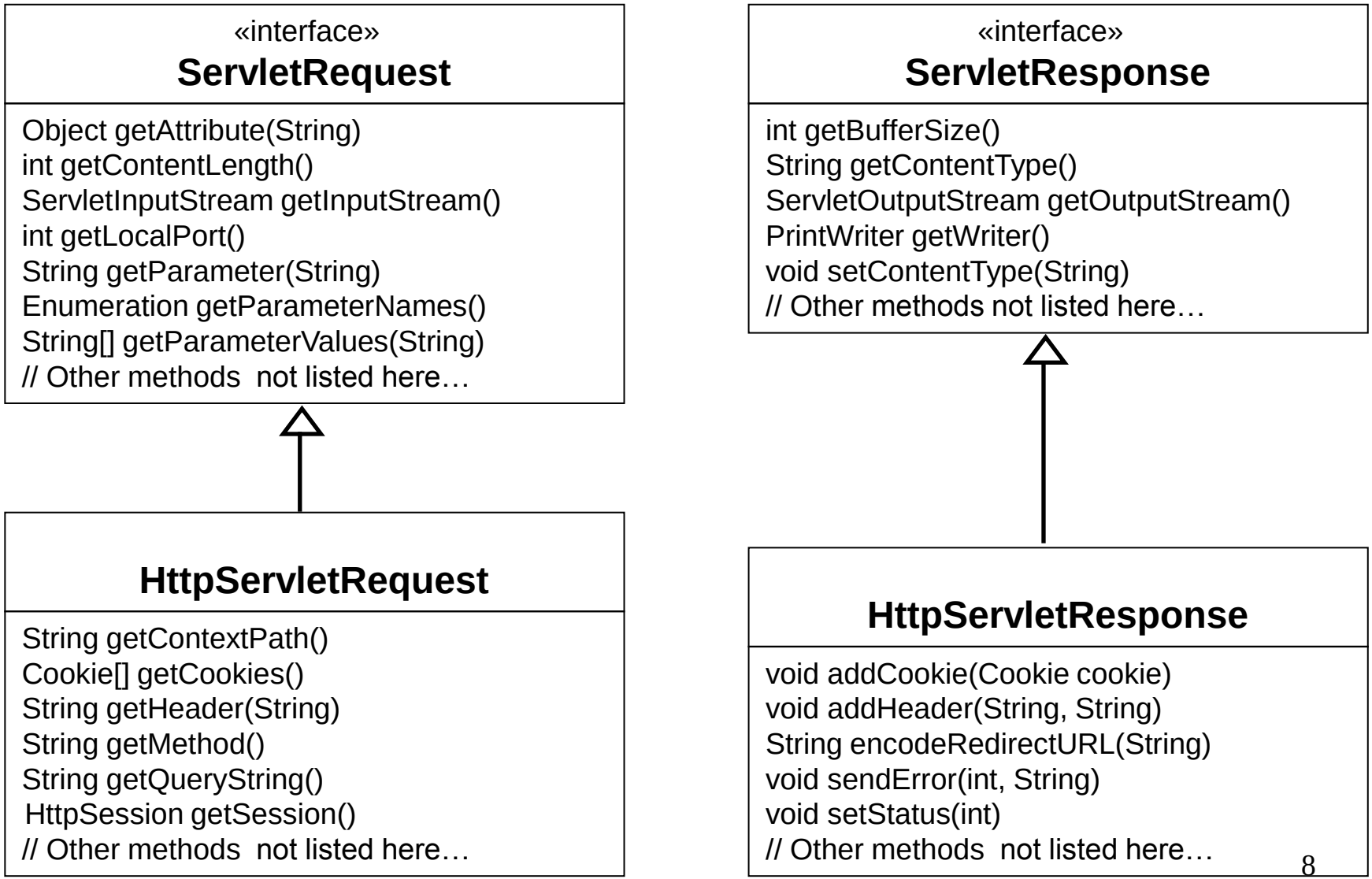
GET ou **POST**



Servlets HTTP



Requêtes et Réponses HTTP



Un servlet en 8 étapes (1/8)

Lors de la création d'un projet JEE, on a deux grandes phases :

- La phase de **développement** qui s'effectue la plupart du temps en local, dans un dossier permettant de compiler et tester les sources Java, les fichiers HTML, les CSS, …
- La phase de **déploiement** qui exporte le site local sur un serveur JEE afin de le tester.

- Ainsi, lors de la création d'une servlet, on commencera par créer un dossier de projet, avec un dossier src contenant les sources des servlets que l'on compilera dans un dossier build/classes (comme nous faisions dans les cours de Java ING1).
- C'est dans un second temps, que nous déploierons notre projet sur le serveur Tomcat.

- L'utilisation d'un outil de contruction de projets Java, tels que Ant ou Maven, est classique durant ces phases.

Un servlet en 8 étapes (2/8)

- Ecrire une servlet et la placer dans *src* (ici Clock.java)

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

public class Clock  extends  HttpServlet{
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
        throwsIOException{
        PrintWriter out=  response.getWriter();
         java.util.Date today=new   java.util.Date();

        String docType=  "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 "+
                         "Transitional//EN\">\n";

        out.println(docType);
        out.println("<html>");
        out.println("<head>\n<title>Clock</title>\n</head>");
        out.println("<body>\n"+
                    "<h1>Time on server</h1>\n"+
                    "<p>"+  today+  "</p>\n"+
                    "</body>");
        out.println("</html>");
    }
}
```

Servlet classique: redéfinit doGet() de HttpServlet

Code HTML incorporé dans Java

Un servlet en 8 étapes (3/8)

- Créer un dossier WEB-INF et y placer le fichier de configuration *web.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<servlet>
        <servlet-name>The  Clock</servlet-name>
        <servlet-class>Clock</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>The  Clock</servlet-name>
        <url-pattern>/Serv1</url-pattern>
</servlet-mapping>
</web-app>
```

classe Java

<servlet-name> permet de lier <servlet> et <servlet-mapping>

nom utilisé par le client dans sa requête

Un servlet en 8 étapes (2-3/8)

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

@WebServlet{"Serv1"}
public class Clock extends HttpServlet
{
        public  void doGet(HttpServletRequest request,
                HttpServletResponse response) throws  IOException
        {
                // instructions
        }
}
```

- Depuis la v.6, on peut configurer la Servlet par **annotations** (plutôt que dans le web.xml)

} Annotation définissant l'url-pattern de cette Servlet (remplace l'étape 3/8)

Un servlet en 8 étapes (4/8)

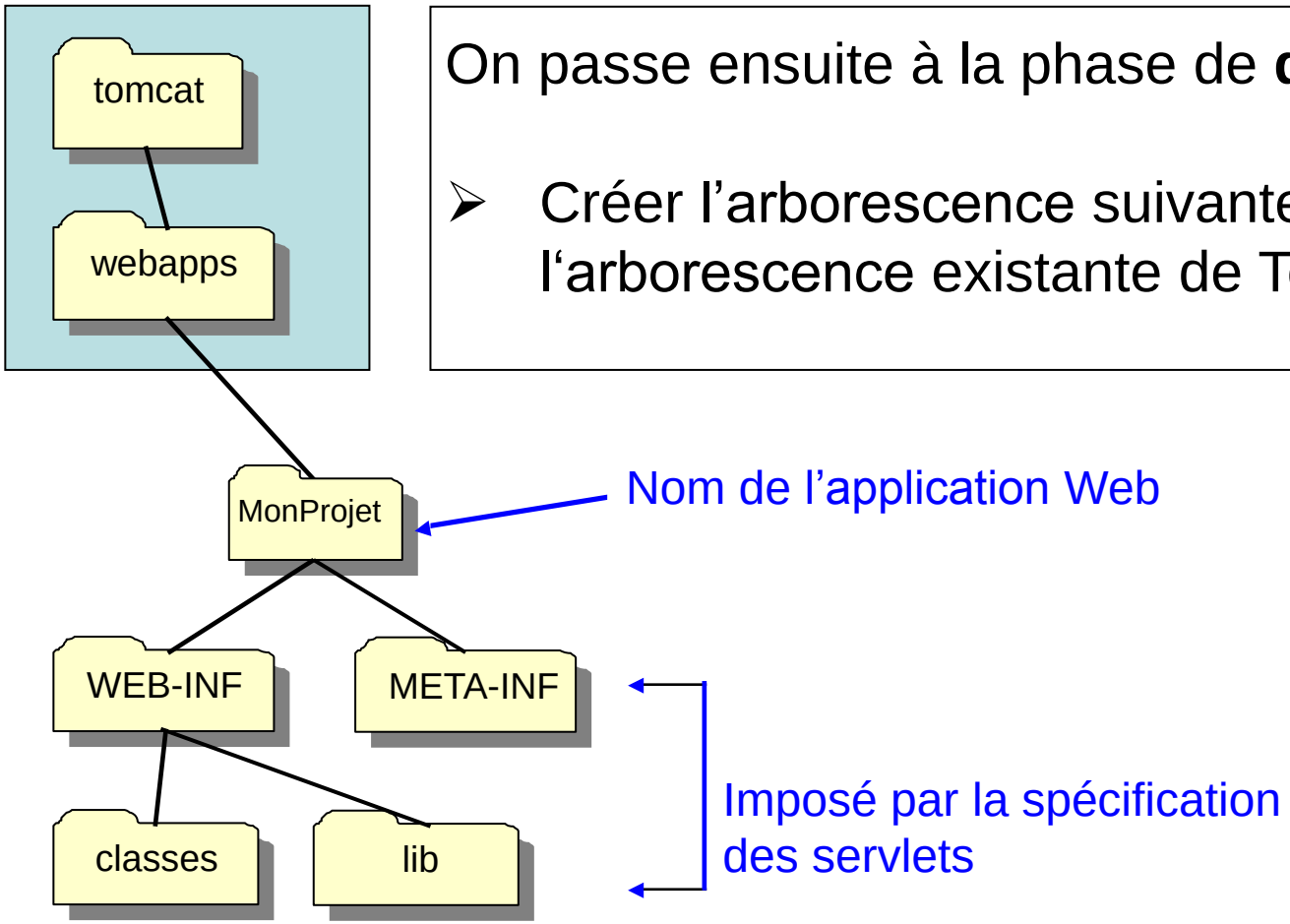
• A partir du répertoire de projet, compiler le servlet et placez-le dans le répertoire *build/classes*

```
% cd  AREL
% javac  –classpath  ~tomcat/lib/servlet-api.jar
–d  classes  src/Clock.java
```

Un servlet en 8 étapes (5/8)

On passe ensuite à la phase de **déploiement**

- Créer l'arborescence suivante **dans** l'arborescence existante de Tomcat



Un servlet en 8 étapes (6/8)

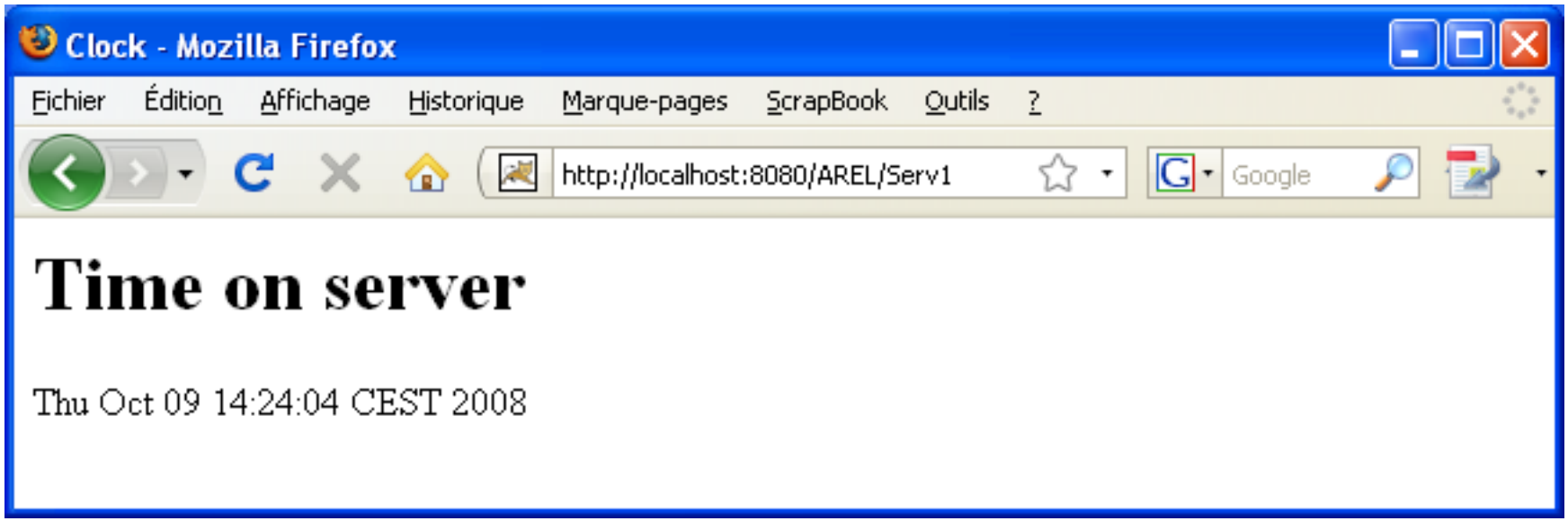
- Démarrer Tomcat

```
% cd  tomcat
% bin/startup.sh
```

ou bien lancer le Monitor si vous êtes sous Windows

Un servlet en 8 étapes (7/8)

- Lancer un navigateur et entrer l'URL suivante
  - `http://`**localhost:8080/MonProjet/Serv1**
- Le navigateur affiche:



Un servlet en 8 étapes (8/8)

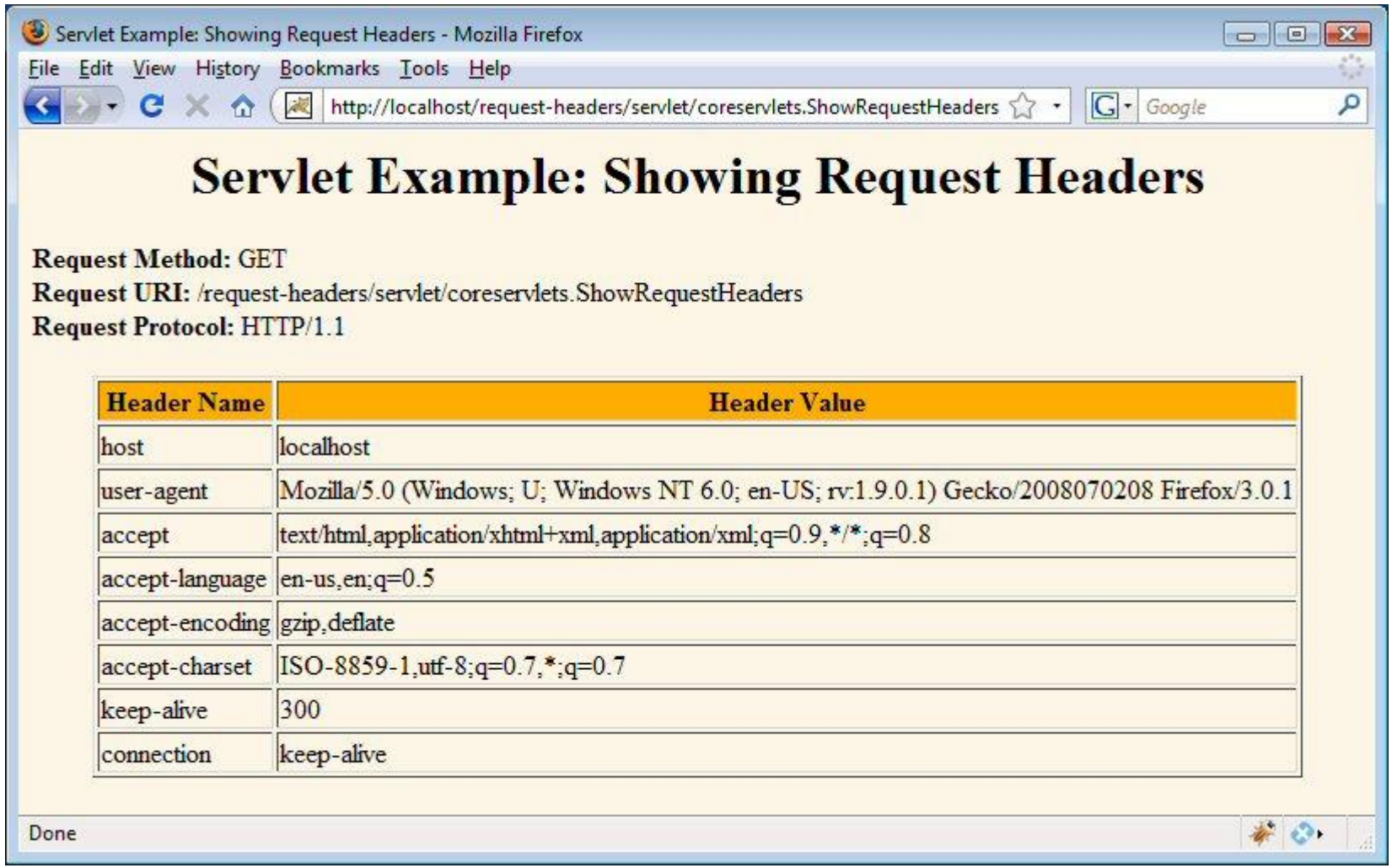
- Redémarrer Tomcat à chaque modification de la classe serlvet ou du descripteur de déploiement

```
% cd  tomcat
% bin/shutdown.sh
```

Entêtes d'une requête GET

```
public class ShowRequestHeaders extends HttpServlet{
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws IOException{
    //...
    out.println(doctype+
        "<html>\n<head><title>"+title+ "</title></head>\n"+
        "<body>\n<h1>"+title+  "</h1>\n"+
        "<b>Request Method: </b>"+request.getMethod()+ "<br />\n"+
        "<b>Request URI: </b>"+ request.getRequestURI()+ "<br />\n"+
        "<b>Request Protocol: </b>"+ request.getProtocol()+ "<br />\n"+
        "<table>\n"+
        "<tr><th>Header Name</th><th>Header Value</th></tr>");
    Enumeration<String> headerNames = request.getHeaderNames();
    while(headerNames.hasMoreElements()){
        String headerName = headerNames.nextElement();
        out.println("<tr><td>"+headerName+ "</td>");
        out.println("<td>"+
                    request.getHeader(headerName)+
                    "</td></tr>");
    }
    out.println("</table>\n</body></html>");
   }
}
```

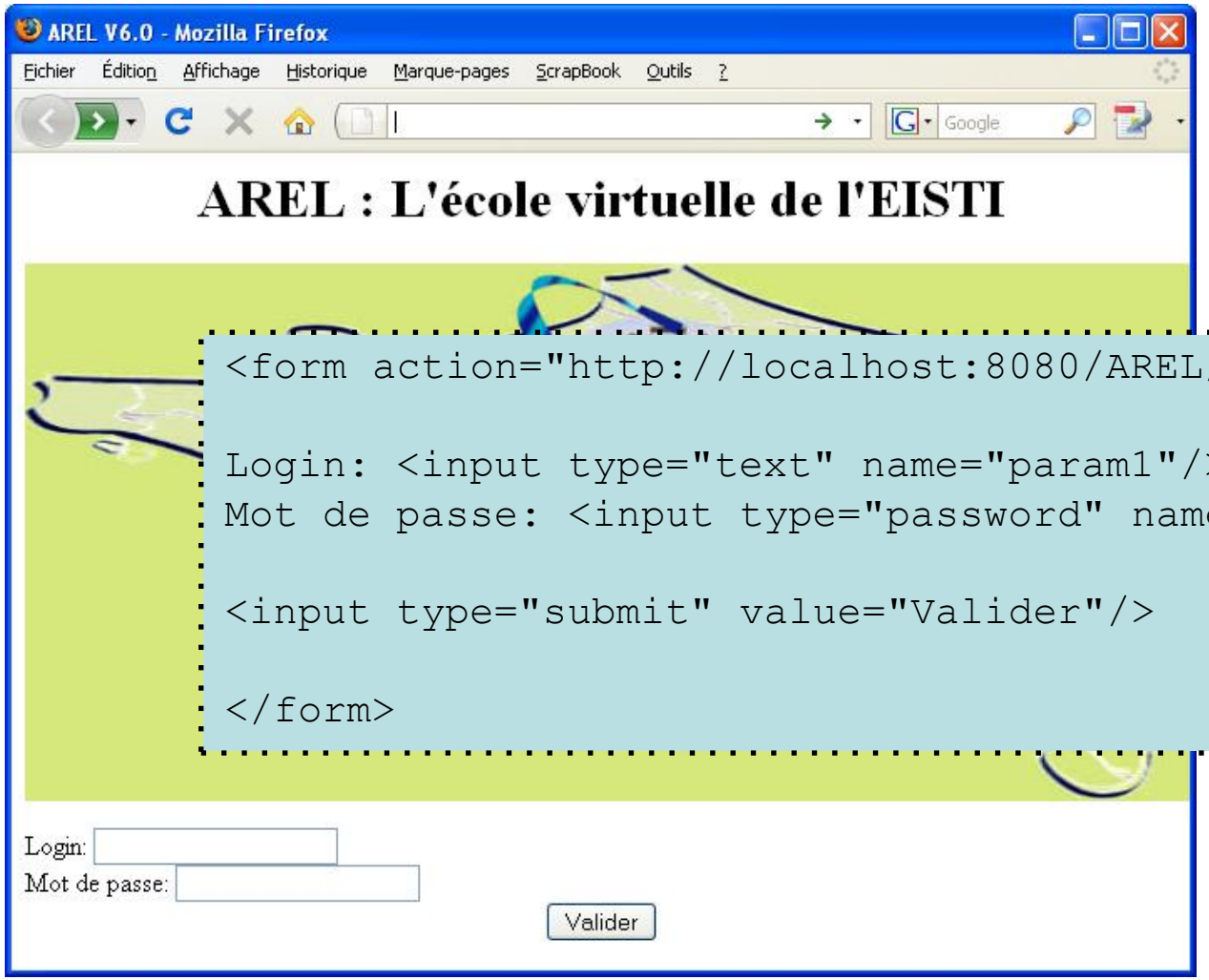
Entêtes d'une requête GET (2)



Entêtes d'une requête POST

```
public class ShowRequestHeaders extends  HttpServlet{
    public void doPost(HttpServletRequest  request,
                       HttpServletResponse response) throws IOException{

        doGet(request,response);

    }
}
```

Formulaire GET



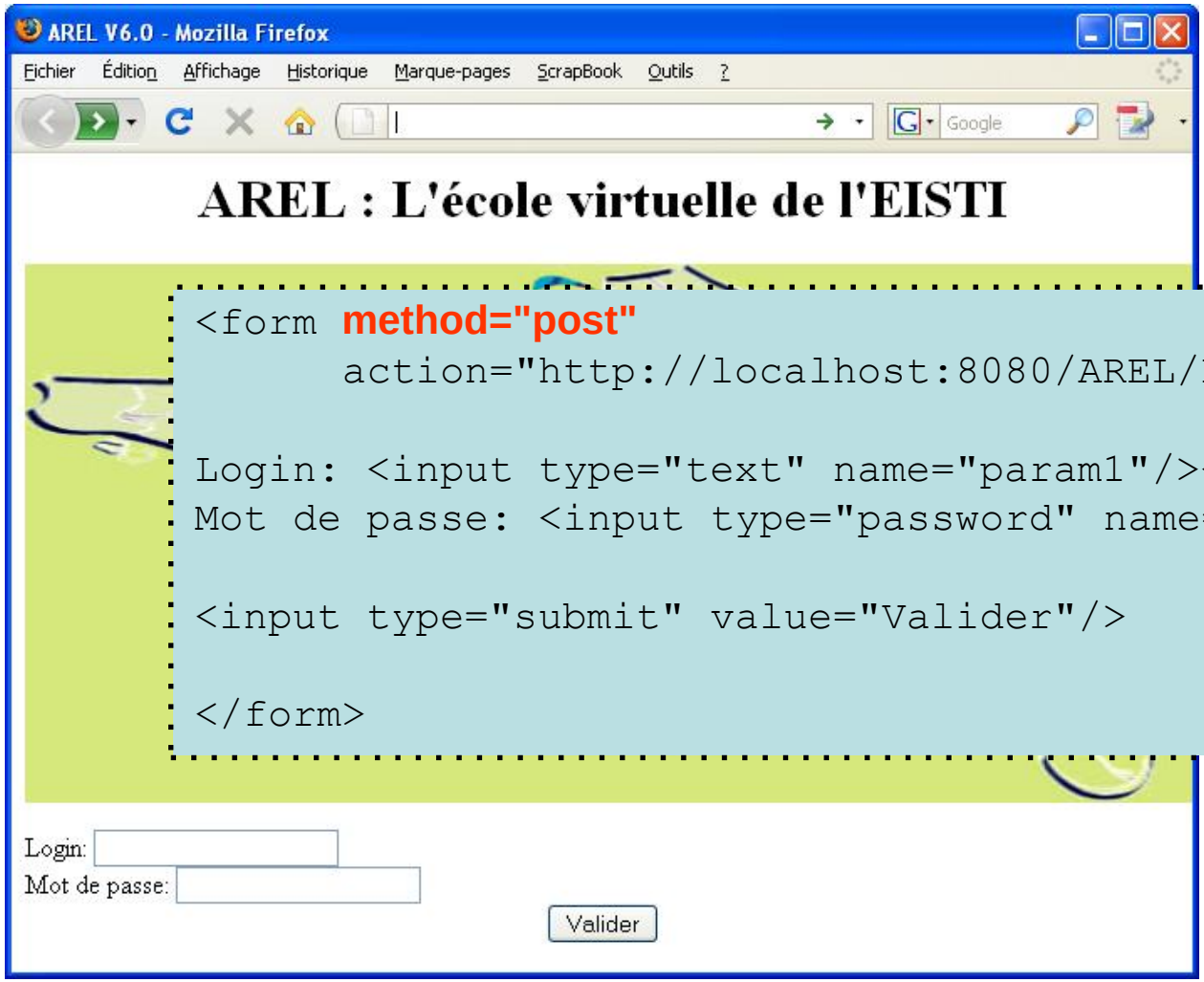

```
<form action="http://localhost:8080/AREL/LogServlet">

Login: <input type="text" name="param1"/><br/>
Mot de passe: <input type="password" name="param2"/><br/>

<input type="submit" value="Valider"/>

</form>
```

Traitement formulaire GET

GET /AREL/LogServlet?param1=monlogin&param2=monpass HTTP/1.1
host: localhost:8080
user-agent: Mozilla/5.0 (…) Gecko/2008092417 Firefox/3.0.3
accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
…

```
public class LogServlet extends HttpServlet{
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws IOException{
    String login=request.getParameter("param1");
    String password=request.getParameter("param2");
    if (checkUserAndPassword(login, password)){
        grantAccessTo(login);
    } else{
        sendAuthentificationFailure(login);
    }
  }
}
```

Formulaire POST




```
<form method="post"
      action="http://localhost:8080/AREL/LogServlet">

Login: <input type="text" name="param1"/><br/>
Mot de passe: <input type="password" name="param2"/><br/>

<input type="submit" value="Valider"/>

</form>
```

Traitement formulaire POST

POST /AREL/LogServlet HTTP/1.1
host: localhost:8080
user-agent: Mozilla/5.0 (…) Gecko/2008092417 Firefox/3.0.3
accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
…
content-type: application/x-www-form-urlencoded
content-length: 30

param1=monlogin&param2=monpass

```
public class LogServlet extends  HttpServlet{
    public void doPost(HttpServletRequest  request,
                       HttpServletResponse response)  throws IOException{

        doGet(request,response);

    }
}
```

Paramètres de formulaires

```
public class ShowParameters extends HttpServlet{
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)  throws IOException{
    //...
    out.println("<b>Query String: </b>"+
                request.getQueryString()+"<br />");

    out.println("<table>");
    Enumeration<?> parameterNames=request.getParameterNames();
    while(parameterNames.hasMoreElements()){
        String parameterName=(String)parameterNames.nextElement();
        out.println("<tr><td>"+parameterName+"</td>");
        String paramValue=request.getParameter(parameterName);
        out.println("<td>"+paramValue+"</td>");
    }
    out.println("</table>");
  }
}
```

Vérification de formulaires

- Données manquantes
  - Champ manquant dans le formulaire
    - **getParameter** retourne *null*
  - Champ renvoyé vide
    - **getParameter** retourne une chaine vide (ou une chaine avec des espacements)

```
String param=   request.getParameter("someName");
if((param == null) || (param.trim().equals("")){
   doSomethingForMissingValues(...);
} else{
   doSomethingWithParameter(param);
}
```

- Données malformées
  - Chaine non vide mais dans le mauvais format (ex: code HTML si le résultat doit être affiché)

Upload de fichiers

- Formulaire HTML
  - `<input` **`type="file"`** `name="nameFile" />`
  - `<form` **`method="post"`**
    `enctype="multipart/form-data"`
    `action="/servlet">`
  - Le choix du *enctype* impacte les autres champs du formulaire
    - **request.getParameter("name") ne fonctionne plus**
- Côté Servlet
  - Bas-niveau : parser l'inputStream
    `request.getInputStream()`
  - Haut-niveau : utiliser une librairie
    ex: Commons FileUpload du projet Jakarta
    (http://commons.apache.org/fileupload/)

Commons FileUpload

- Développement
  - import org.apache.commons.fileupload.*
  - import org.apache.commons.fileupload.servlet.*
- Déploiement
  - Dans le module Web de l'application…
    - *ie.* même chose pour chaque application!
  - …dans le répertoire WEB-INF/lib
    - commons-fileupload-1.2.1.jar
    - commons-io-1.4.jar

Upload simple (1)

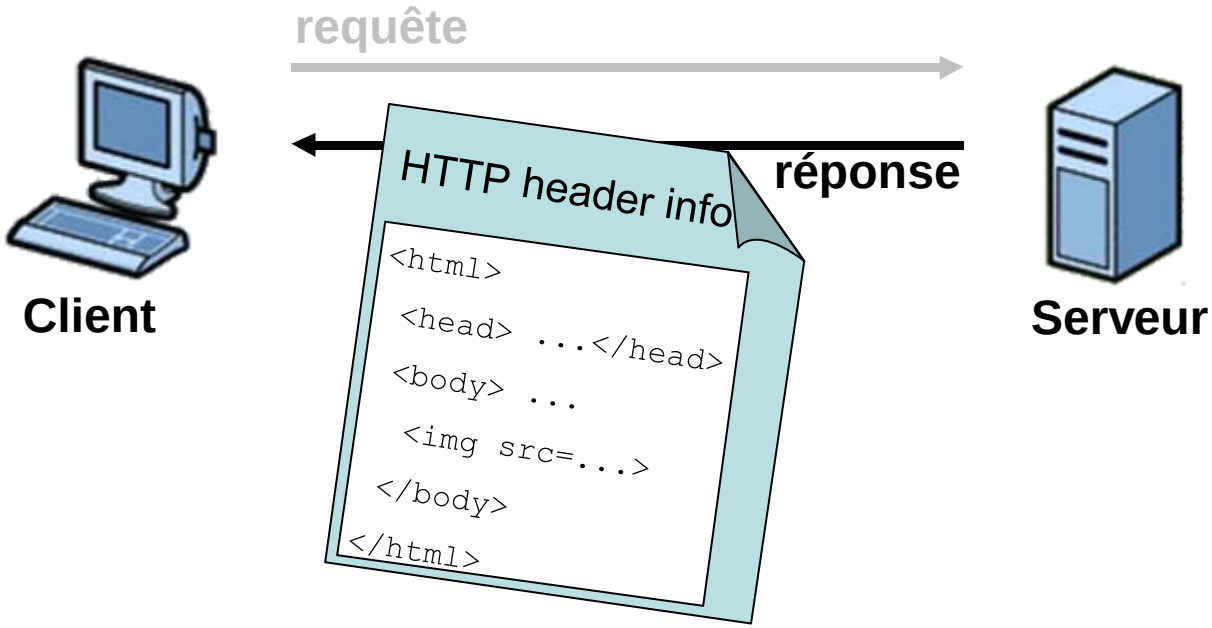
```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response) throws IOException{
 //...
 // check file upload request
 if(ServletFileUpload.isMultipartContent(request)){

      //   create a factory for disk-based (large) file items
      FileItemFactory fileItemFactory = new DiskFileItemFactory();
      fileItemFactory.setSizeThreshold(40960); /* the unit is bytes */

      //   create a new fileupload handler
      ServletFileUpload servletFileUpload =
           new ServletFileUpload(fileItemFactory);
      servletFileUpload.setSizeMax(81920); /* the unit is bytes */

      //   parse the request
      //   ...                    --->
 }
}
```

Upload simple (2)

```
// parse the request
try{
    List<?> fileItemsList=servletFileUpload.parseRequest(request);
    // Process file items
    Iterator<?> it = fileItemsList.iterator();
    while (it.hasNext()){
      DiskFileItem fileItem=(DiskFileItem)it.next();
      if(fileItem.isFormField()){ // classic form field (name = value)
        out.println("<b>Form field:</b><br />\n"+
                    fileItem.getString()+ "<br/>");
      } else{ // uploaded file
        out.println("<b>File:</b><br />\n<pre>"+
                    fileItem.getString()+ "</pre><br/>");
        // ex: save on disk
        File dest=new File(directoryPath,fileName);
        FileOutputStream fos = new FileOutputStream(dest);
        fos.write( fileItem.get() );
        fos.close();
      }
    }
} catch (FileUploadException e) {e.printStackTrace();}
```

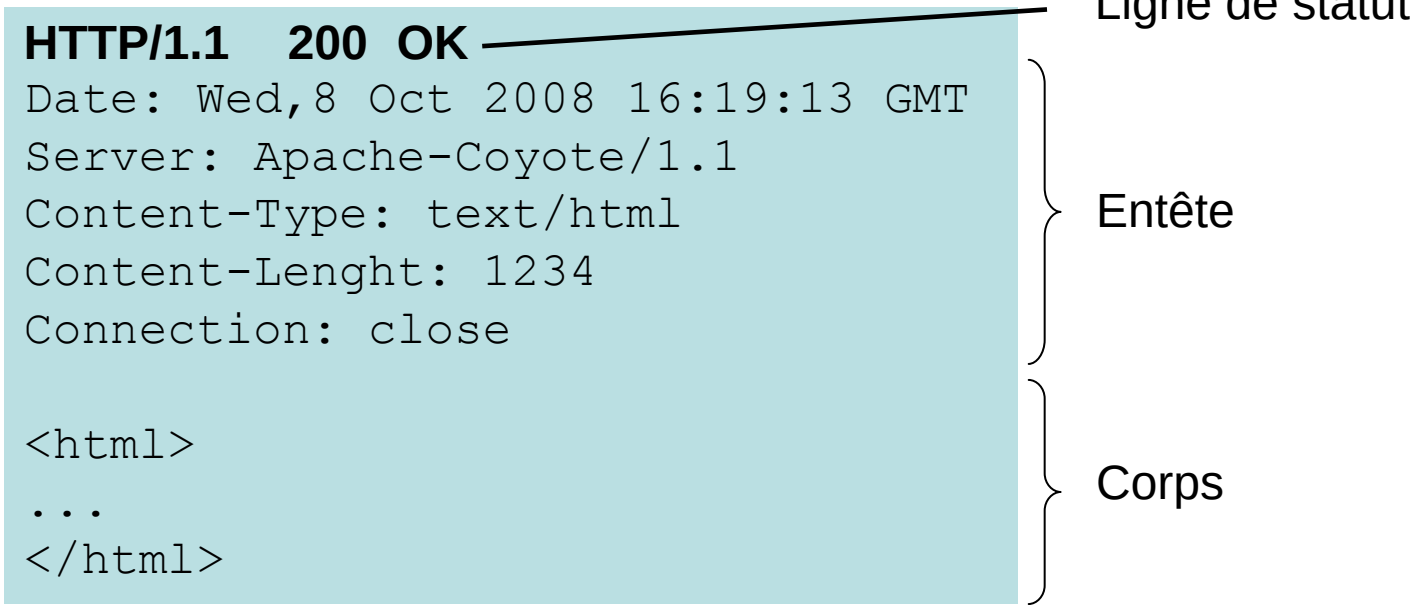
Réponse HTTP (rappel)

- Une réponse HTTP peut contenir du HTML
- HTTP rajoute des (meta)informations en entête du contenu de la réponse



Entête réponse HTTP

- Ex:



- Quelques codes réponses
  - 200 OK
  - 301 MOVED
  - 403 FORBIDDEN
  - 404 NOT FOUND
  - 503 SERVICE UNAVAILABLE

Status Codes

- response.setStatus(int statusCode)
  - Utiliser les constantes, pas d'entiers directement
  - Noms dérivés du message standard
    - Ex: SC_OK, SC_NOT_FOUND, etc…
- response.sendError(int code, String msg)
  - Englobe le message dans un petit document HTML
- response.sendRedirect(String url)
  - Le code de status est alors 302
  - L'attribut «Location» est également généré dans l'entête de la réponse

Exemple sendError

```
public class LogServlet extends HttpServlet{
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response) throws IOException{
    String login=request.getParameter("param1");
    String password=request.getParameter("param2");

    if ((param1 == null) || (param1.trim().equals(""))){
        response.sendError(HttpServletResponse.SC_NOT_FOUND,
                           "Empty login");
        return;
    }

    if (checkUserAndPassword(login, password)){
        grantAccessTo(login);
    } else{
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
                           "Access Denied to"+ login);
    }
  }
}
```

Exemple sendRedirect

```
public class WrongDestination extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException{
    String userAgent=  request.getHeader("User-Agent");
    if ((userAgent != null) &&
        (userAgent.contains("MSIE")){
      response.sendRedirect("http://home.netscape.com");
    } else{
      response.sendRedirect("http://www.microsoft.com");
    }
  }
}
```

Exemple sendRedirect (2)

- Même URL de départ pour les deux

setContentType

Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat (.pdf) file
application/postscript	PostScript file
application/vnd.ms-excel	Excel spreadsheet
application/vnd.ms-powerpoint	Powerpoint presentation
application/x-gzip	Gzip archive
application/x-java-archive	JAR file
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/x-aiff	AIFF sound file
audio/x-wav	Microsoft Windows sound file
audio/midi	MIDI sound file
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plain text
text/xml	XML document
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image
video/mpeg	MPEG video clip
video/quicktime	QuickTime video clip

Générer un fichier Excel

```
public class ApplesAndOranges extendsHttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException{
      response.setContentType
        ("application/vnd.ms-excel");
      PrintWriter out=  response.getWriter();
      out.println("\tQ1\tQ2\tQ3\tQ4\tTotal");
      out.println("Apples\t78\t87\t92\t29\t=SUM(B2:E2)");
      out.println("Oranges\t77\t86\t93\t30\t=SUM(B3:E3)");
  }
}
```