Web Component Development Using Java

Session: 2

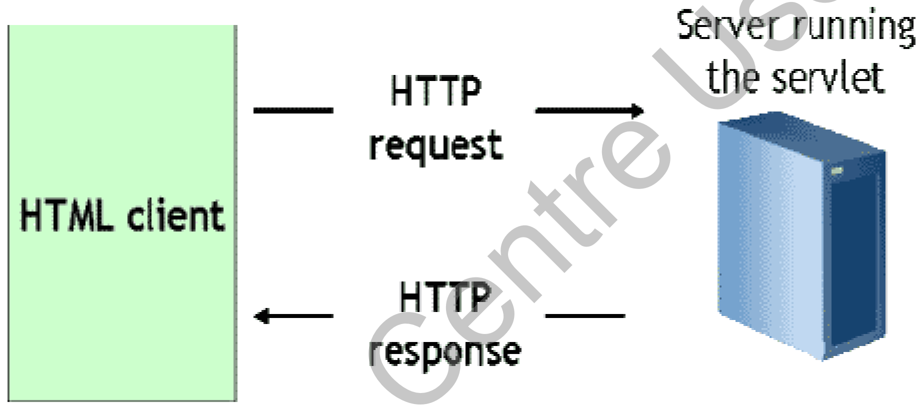
Java Servlet

Objectives

- Explain the Servlet API
- Explain the servlet architecture and life cycle of Servlet
- Explain the methods of ServletRequest and HttpServletRequest interfaces
- Explain the methods of ServletResponse and HttpServletResponse interfaces
- Describe the use of response headers
- Explain how to read text and binary data from a request
- Explain the ServletConfig and ServletContext interface
- Explain redirection of client requests
- Explain RequestDispatcher interface
- Explain error handling in Servlet

Introduction

- **Servlet:**
  - Server-side program written in Java programming language.
  - Processes the client requests and generates dynamic content in response.



- **Clients of Servlet:**
  - A Web client which sends request through HTTP protocol.
  - A Web service endpoint sending request data through SOAP protocol.

Servlet API 1-5

❖ All the classes related to developing and managing servlets are provided in two packages, they are as follows:

`javax.servlet`

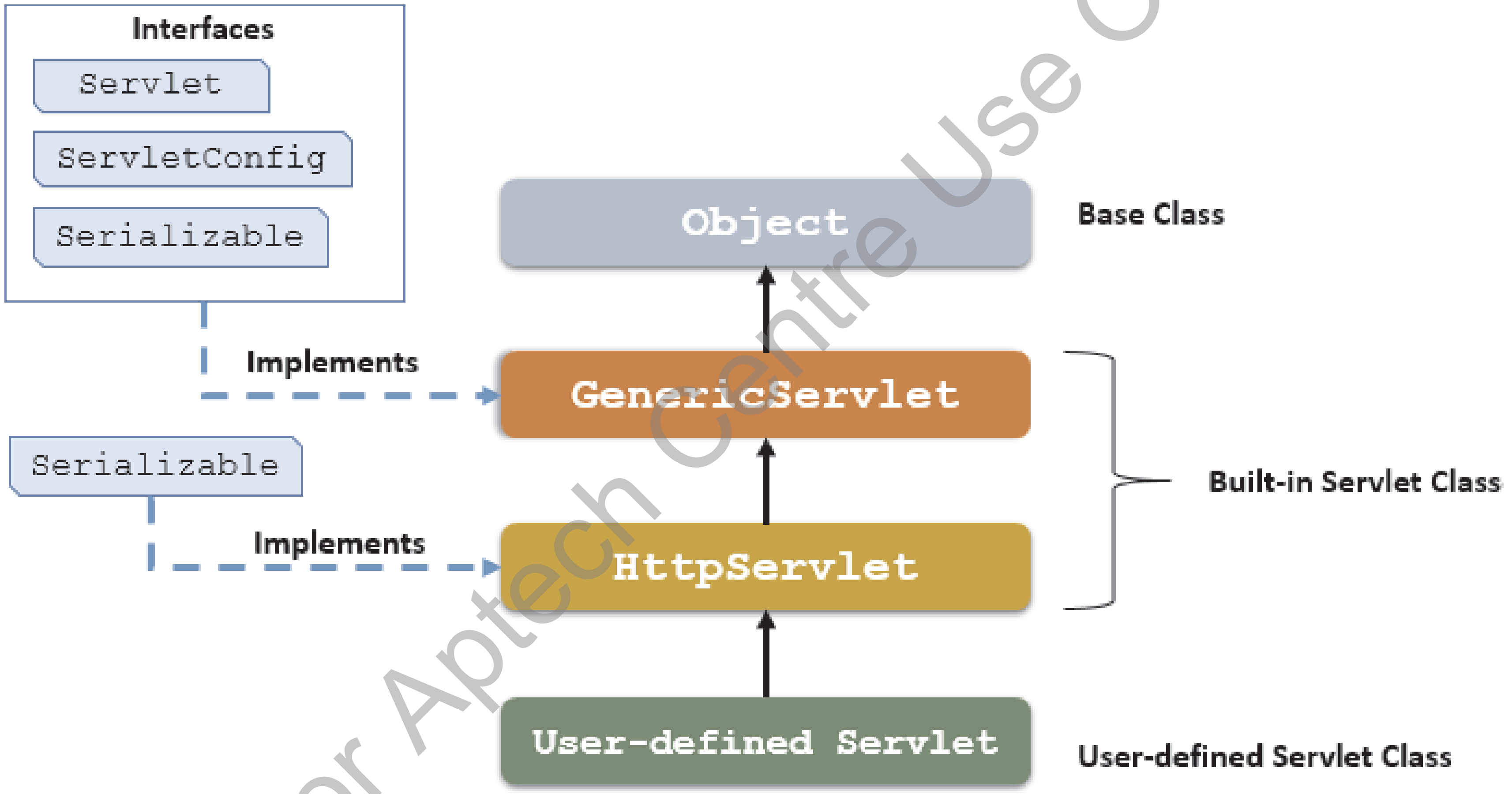
- It contains generic interfaces and classes that are implemented and extended by all servlets.
- It provides a framework for the Servlet operations.
- It includes an interface named Servlet which defines the life cycle methods for a servlet.
- Every servlet must implement the `javax.servlet.Servlet` interface directly or indirectly.

`javax.servlet.http`

- It contains classes and interfaces used for developing a servlet that works with HTTP protocol.

Servlet API 2-5

- Figure displays the Servlet API hierarchy.



Servlet API 3-5

- The `javax.servlet` package contains an abstract class named `GenericServlet.`
- The `GenericServlet` inherits from `Object` class and helps to design a protocol independent servlet.
- The `GenericServlet` implements three interfaces, they are as follows:

`Servlet`

- It defines the life cycle methods for a Servlet.

`ServletConfig`

- It defines methods that are used by Servlet container to pass information to a Servlet instance during its initialization.

`Serializable`

- It is defined in `java.io` package to serialize the state of an object.

Servlet API 4-5

- The code snippet demonstrates how to create a protocol-independent login servlet.

```
public class LoginServlet extends GenericServlet{
. . .
}
```

- In the given code, the class named `LoginServlet` is a generic servlet.
- To develop a servlet which is accessible only through a Web browser, user can create an HTTP Servlet by extending `HttpServlet` class.
- The `HTTPServlet` class is a subclass of `GenericServlet` and enables to create an HTTP-based Servlet as part of a Web application.

Servlet API 5-5

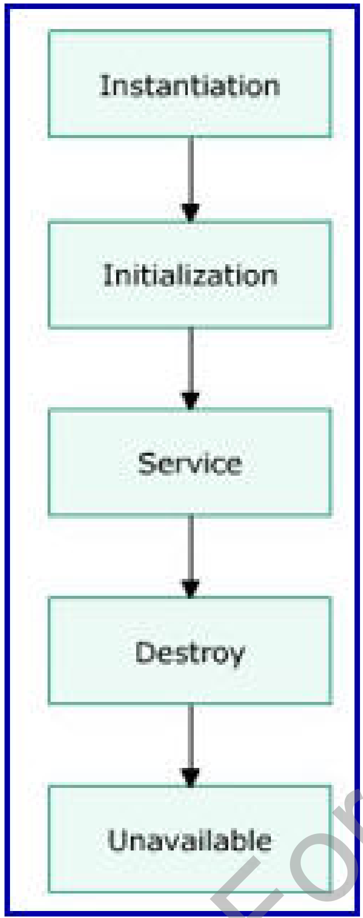
- The code snippet shows the creation of login servlet to be accessed on the Web.

```
public class LoginServlet extends HttpServlet{
. . .
}
```

- In the given code, the `LoginServlet` is defined as HTTP servlet, so it will be accessed through HTTP protocol.

Servlet Life Cycle

- The life cycle defines how the servlet is loaded, initialized, handles requests from clients and is taken out of service.



The servlet container creates an object of the servlet.

The servlet container calls the `init()` method which begins the life cycle of the servlet.

The servlet processes the request given by the client and generates the dynamic response to be sent back to the client.

The servlet container destroys the servlet instance.

The servlet container frees up the memory occupied by the servlet.

Servlet Life Cycle Methods

- The servlet life cycle is defined by the following methods:

`init()` **Initializes the servlet**

- The life cycle of the servlet begins with the `init()` method.
- This method is called only once by the server and not again for each user request.

`service()` **Processes the request made by the client**

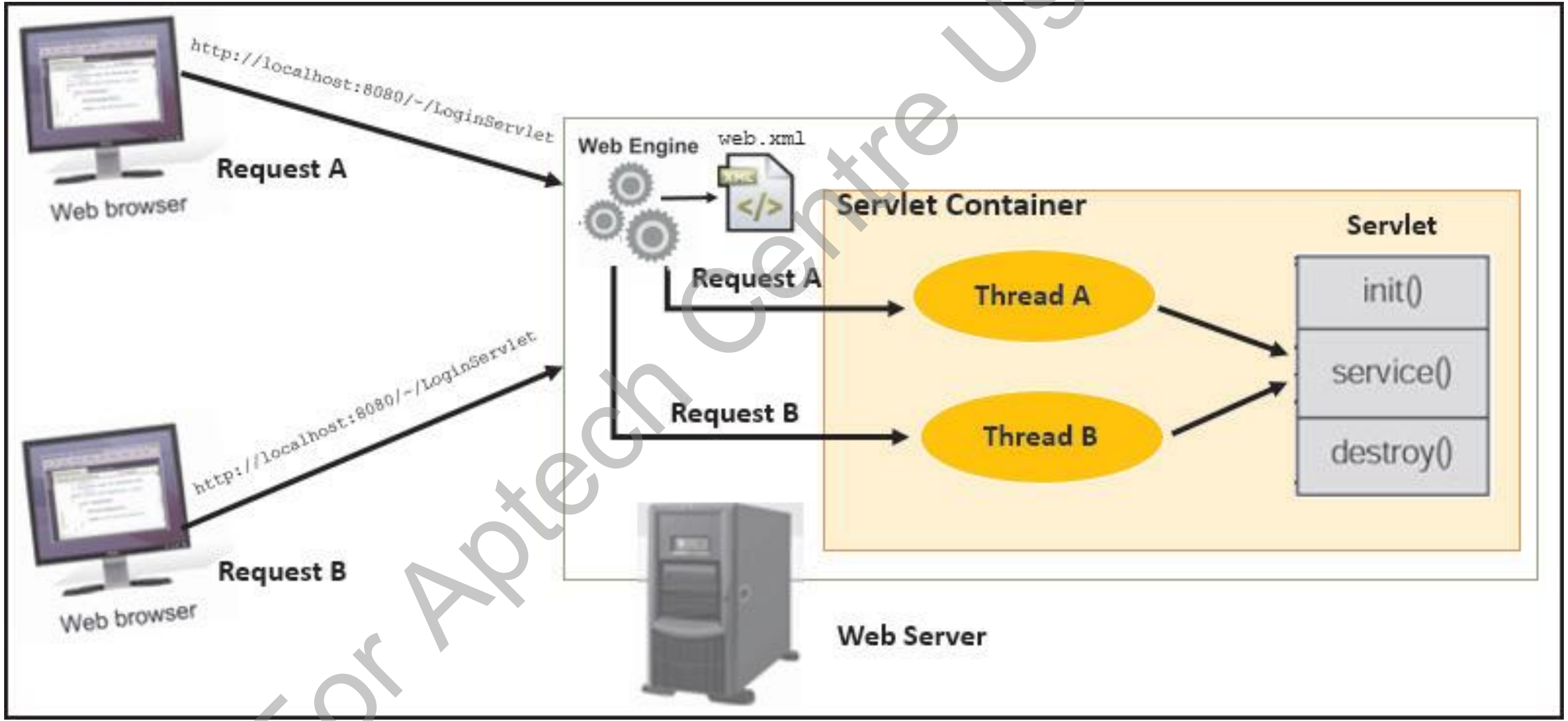
- It passes the `ServletRequest` and `ServletResponse` objects which collect and pass information on which the request was received.

`destroy()` **Destroys the servlet interface, if there are no more requests to be processed**

- A server calls this method after all requests have been processed or a server-specific number of seconds have passed, whichever comes first.

Servlet Architecture 1-2

- The servlet container is responsible for handling servlet execution which is based on multithreaded model.
- Figure shows the handling of multiple clients request by a servlet.



Servlet Architecture 2-2

- The flow of information in a servlet is as follows:

The Web browser sends an HTTP request to the Web server through a Uniform Resource Locator (URL)

↓

On receiving the request, the Web engine looks into the web.xml deployment descriptor file to match the URL with the registered Servlets

↓

Once the requested Servlet is found, the Web engine forwards it to the Servlet container which is responsible for instantiating the Servlet instance

↓

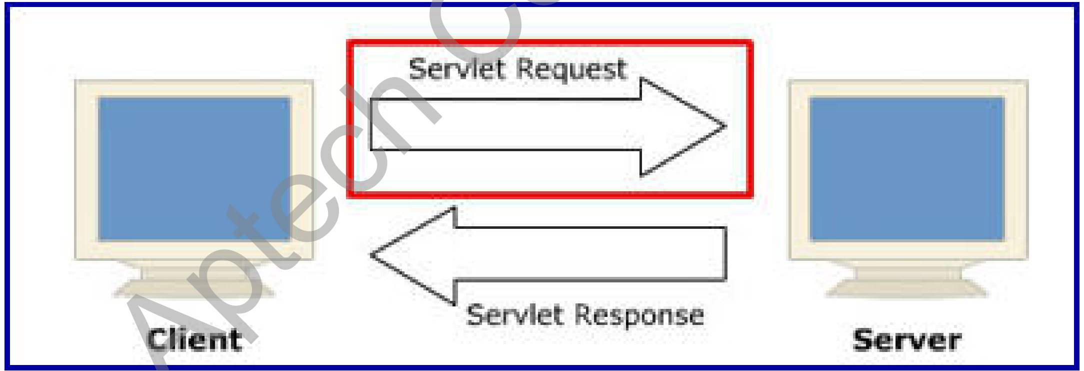
The container first locates the servlet class, loads the servlet, and creates an instance of the servlet in the memory

↓

After the Servlet is loaded by the Servlet container, it is initialized completely and then the client request is processed

Handling Servlet Request

- The `ServletRequest` interface:
  - Retrieves client-specific request parameters such as data entered in online forms.
  - Provides access to the specific information required to process the request appropriately.
  - Handles a request, the server passes a request object, which implements the `ServletRequest` interface.
  - Has two access parts of the raw request such as headers and input stream.
- Figure depicts a Servlet request.



ServletRequest Interface 1-3

- The `ServletRequest` object is passed as an argument to the `service()` method of the servlet.
- The methods defined in the `ServletRequest` interface are as follows:

```
public String getParameter(Strings)
```

The method returns the value of a specified parameter that is sent along with the request information.

```
public String getParameter(Strings)
```

The method retrieves the value of an attribute specified by name, that was set using the `setAttribute()` method and returns null when no attribute with the specified name exists.

ServletRequest Interface 2-3

`public int getContentLength()`

The method returns the length of content in bytes and returns -1, if the length is not known.

- The code snippet shows the use of `getContentLength()` method.

```
/* The length of the content is passed to len1 and
compared with the size of file limit */
if((len1 = request.getContentLength()) > fileLimit){
System.out.println ("Name1 "+login +" Id1 " +id +
"Expected Upload size is more than specified file
Limit");
}
```

ServletRequest Interface 3-3

```
public ServletInputStream getInputStream() throws
IOException
```

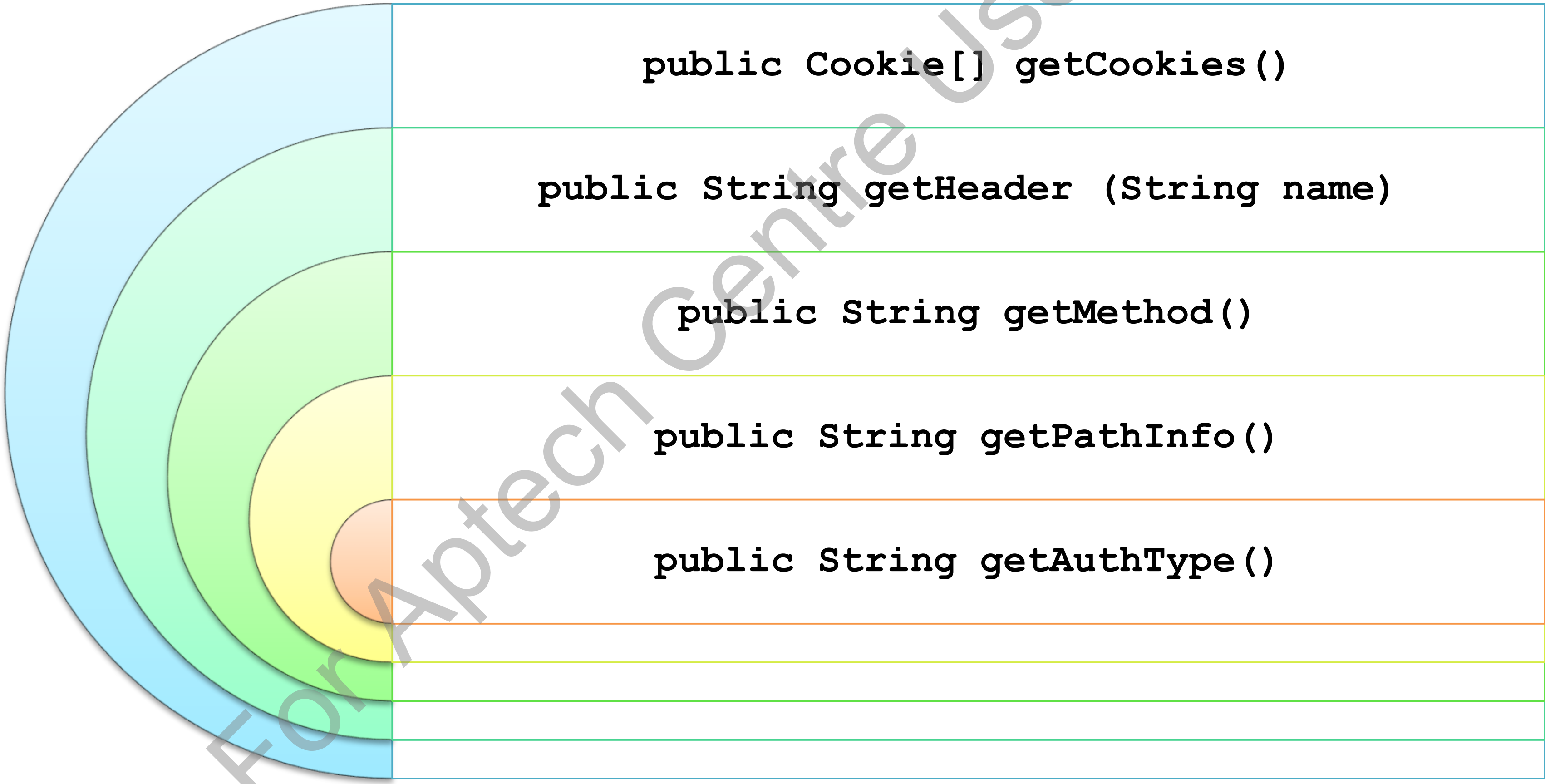
The method returns the binary data of the body of request, requested by the client and stores it in a `ServletInputStream` object.

```
ServletInputStream inStream1 = request.getInputStream();
public String getServerName()
```

The method returns the host name of the server to which the client request was sent.

HttpServletRequest Interface

- It defines an `HttpServletRequest` object, which is passed as an argument to the `service()` method.
- The methods defined by the `HttpServletRequest` interface are as follows:



```
public Cookie[] getCookies()
public String getHeader (String name)
public String getMethod()
public String getPathInfo()
public String getAuthType()
```

Reading Parameters from Request

- Request parameters:
  - Are strings sent by the client to a servlet container as a part of its request.
  - Are stored as a set of name-value pairs.
- Following methods of the `ServletRequest` interface are available to access parameters:

```
public java.lang.String
getParameter(java.lang.String name)
```

- Returns the value of a request parameter as a String, or null if the parameter does not exist.

```
public java.util.Enumeration
getParameterNames()
```

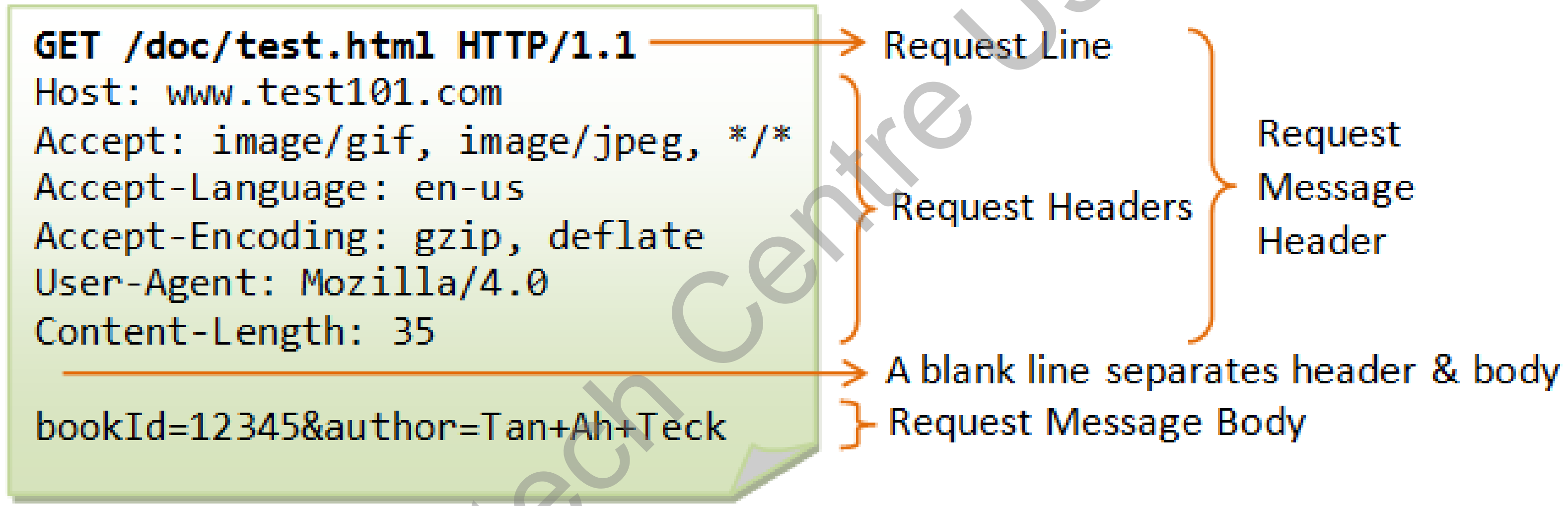
- Returns an enumeration of string objects containing the names of the parameters of this request.

```
public java.lang.String[]
getParameterValues(java.lang.Stringname)
```

- Returns an array of string objects containing all of the parameter values or null if parameters do not exist.

HttpRequest Headers

- It allows the client to pass additional information about the request, and about the client itself, to the server.
- Few of the request headers available are as follows:



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Reading Headers form Request

❖ A servlet can access the headers of an HTTP request through the following methods of the `HttpServletRequest` interface.

```
public String getHeader (String name)
```

- Returns the value of the specified request header as a String.

```
public java.util.Enumeration getHeaders(java.lang.String name)
```

- Returns all the values of the specified request header as an enumeration of String objects.

```
public java.util.Enumeration getHeaderNames()
```

- Returns an enumeration of all the header names this request contains.

HTML Forms 1-3

- HTML forms:
  - Are used to collect user input data.
  - Are stored as form fields and passed through browser URL for further processing in the Servlet.
- Several input types are provided by HTML to be used with `<form>` element. Some of them are as follows:

Text input

- Allows the user to enter a single line of text.

Drop-down list input

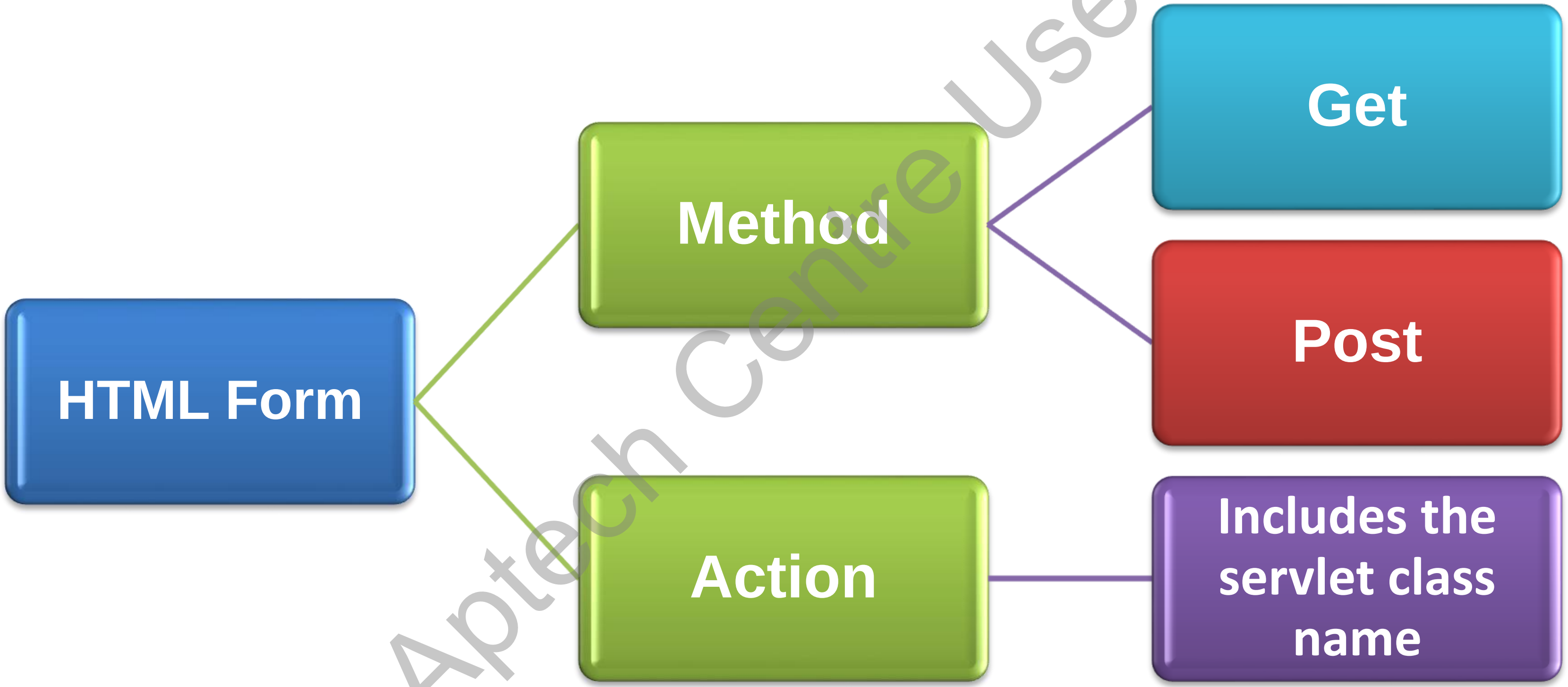
- Provides a list of options to the user to be selected.

Submit button

- Allows the user to send the data to the server for processing.

HTML Forms 2-3

- The HTML form has the following two major attributes:



HTML Forms 3-3

- The code snippet shows the HTML page for accepting the user choice.

```
<form method="GET" action="Login">
   <Username: <input type="text" name="username"/>
   <br/>

<select name="color" size="1">
   <option> light
   <option> amber
   <option> brown
   <option> dark
</select>

<center>
   <input type="SUBMIT">
</center>
</form>
```

Handling Form Data Using Request Object

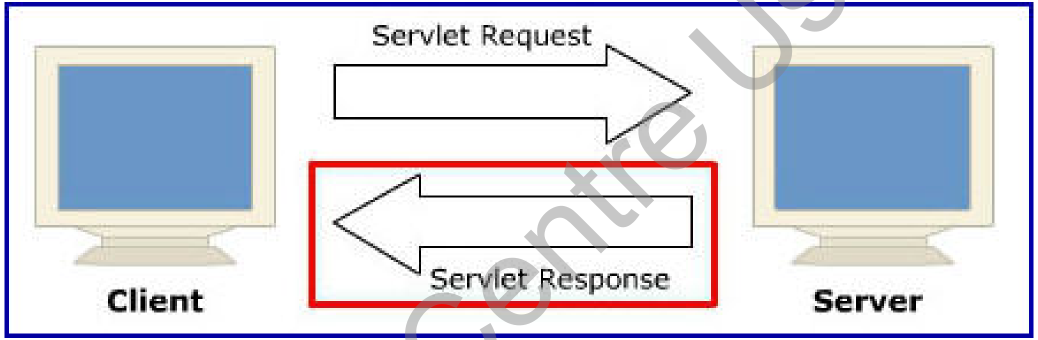
- The `request` object:
  - Allows the servlet to obtain the user data from the form.
  - Calls the data reading methods of the request object.
- The code snippet shows the use of `getParameter()` method in handling data.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException,
ServletException

{
   String name = request.getParameter("username");
   String selectedColorValue =
request.getParameter("color");

// data processing code here
}
```

- The code invokes the `getParameter()` method to get value of the fields `name` and `color` and stores them as `String`.

ServletResponse Interface 1-2

- The `ServletResponse` and `HttpServletResponse` interfaces include all the methods needed to create and manipulate a servlet's output.
- Figure depicts servlet response.



- The `ServletResponse` interface:
  - Defines methods that allow to retrieve an output stream to send data to the client, decide on the content type, and so on.
  - Is passed as an argument to `service()` method of a servlet.

ServletResponse Interface 2-2

- The methods defined by the `ServletResponse` interface are as follows:

```
public java.lang.String getContentType()
```

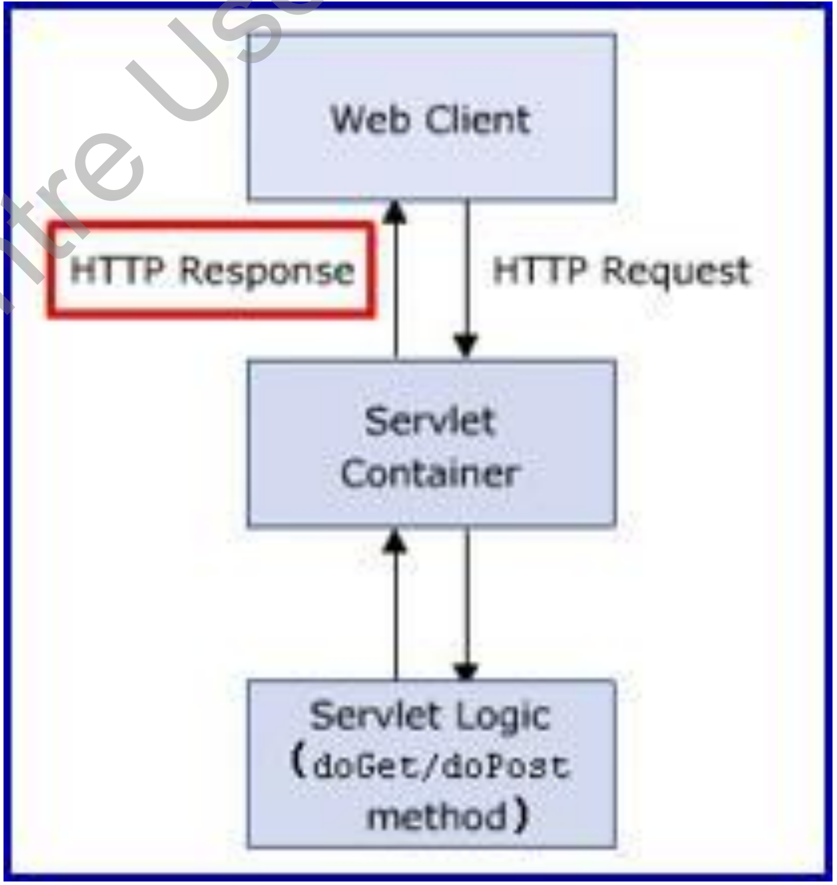
```
public PrintWriter getWriter() throws IOException
```

```
public ServletOutputStream getOutputStream() throws
IOException
```

```
public void setContentType(java.lang.String str)
```

HttpServletResponse Interface

- The `HttpServletResponse` interface extends `ServletResponse` interface and provides information to an `HttpServlet`.
- It defines an `HttpServlet` object, which is passed as an argument to the `service()` method of a servlet.
- Figure depicts HTTP servlet response.



Methods in HttpServletResponse Interface 1-2

- The methods defined by the `HttpServletResponse` interface are as follows:

```
public void addCookie(Cookie cookie)
```

- Adds the specified cookie to the response, sent to the client.
- The code snippet shows the use of `addCookie()` method.

```
/* Adds a cookie named cookie2 with color red to
the existing cookies */
Cookie cookie2 = new Cookie("color", "red");
response.addCookie(cookie2);
```

Methods in HttpServletResponse Interface 2-2

```
public void addHeader(java.lang.String name,
java.lang.String value)
```

- Used to add name and value to the response header.

```
public boolean containsHeader(String name)
```

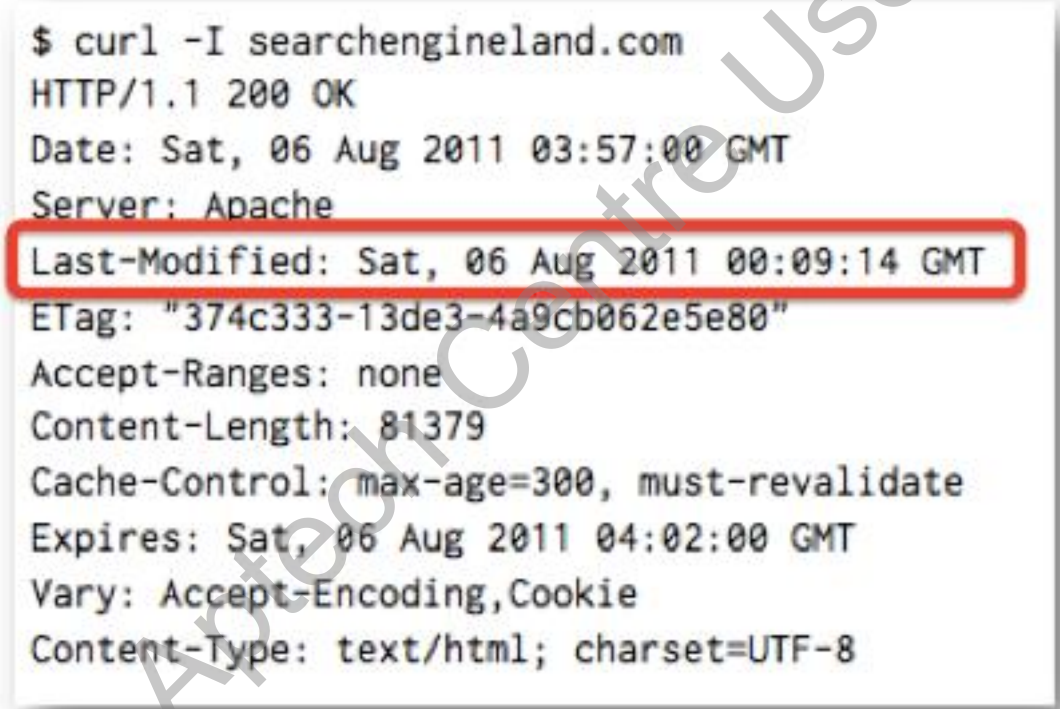
- Used to verify if the response header contains any values.
- It returns true if the response header has any values.
- It returns false otherwise.

```
public boolean containsHeader(String name)
```

- Sends an error response to the client using the specified status code and clearing the buffer.
- Status codes are used to indicate the reason, a request is not satisfied or that a request has been redirected.

Response Headers

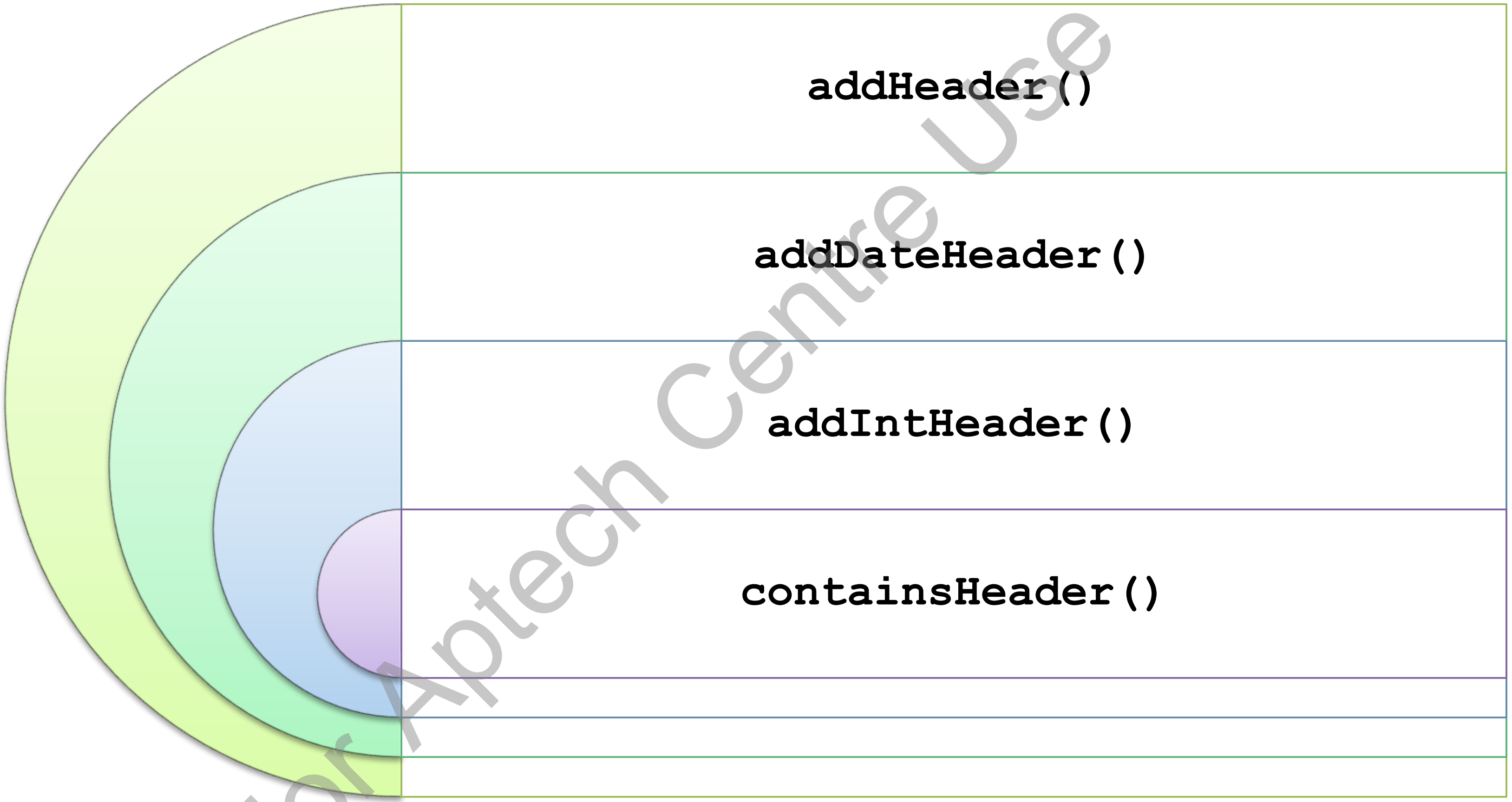
- Figure shows the parameters of response header sent in response to the client.



```
$ curl -I searchengineland.com
HTTP/1.1 200 OK
Date: Sat, 06 Aug 2011 03:57:00 GMT
Server: Apache
Last-Modified: Sat, 06 Aug 2011 00:09:14 GMT
ETag: "374c333-13de3-4a9cb062e5e80"
Accept-Ranges: none
Content-Length: 81379
Cache-Control: max-age=300, must-revalidate
Expires: Sat, 06 Aug 2011 04:02:00 GMT
Vary: Accept-Encoding,Cookie
Content-Type: text/html; charset=UTF-8
```

Sending Headers

- The methods of `HttpServletResponse` interface that are used to send header information to the client are as follows:



Sending Data from Servlet Using Response Object

- The `response` object should perform the following steps:
  - Inform the browser, the type of file/data that is getting transferred.
  - Convert the object data to stream.
- The code snippet shows how to send data using `response` object.

```
public class CodeReturnJARfile extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
   response.setContentType("application/jar");
   ServletContext ctx = getServletContext();
   InputStream is = ctx.getResourceAsStream("/JAVAcompactV3.jar");
   int read = 0;
   byte[] bytes = new byte[1024];
   OutputStream os = response.getOutputStream();

   while ((read = is.read(bytes)) != -1) {
       os.write(bytes, 0, read);
   }
os.flush();
os.close(); }
```

Reading Binary Data

- Multipart/form-data is an Internet media type that returns a set of values as the result of a user filling out a form.
- These set of values are transmitted as binary data.
- Each request handled by a servlet has an input stream associated with it.
- Use `getInputStream()` to retrieve the input stream as a `ServletInputStream` object.
- **Syntax:**

```
public ServletInputStream
ServletRequest.getInputStream() throws
IOException
```

Producing Text and Binary Data

- The methods belonging to `ServletResponse` interface for producing output streams are as follows:

```
public ServletOutputStream getOutputStream() throws
java.io.IOException
```

- The `getOutputStream()` returns a `ServletOutputStream`, which can be used for binary data.

```
public java.io.PrintWriter getWriter()throws
java.io.IOException
```

- The `getWriter()` returns a `java.io.PrintWriter` object, which is used only for textual output.
- The `getWriter()` method examines the content-type to determine what character set to use.

Sending Text and Binary Data

- `ServletOutputStream` provides an output stream for sending binary data to the client.
- The methods supported by the `ServletOutputStream` class are as follows:

```
public void print(boolean b)throws java.io.IOException
```

- Writes a boolean value to the client with no Carriage Return-line Feed (CRLF) character at the end.

```
public void println(char c)throws java.io.IOException
```

- Writes a character value to the client, followed by a Carriage Return-line Feed (CRLF).

Redirecting Requests

❖ Requests are redirected by following methods:

```
public void sendRedirect(java.lang.String location) throws
java.io.IOException
```

- Sends a redirect response to the client using the specified redirect location URL.

```
public java.lang.String encodeRedirectURL(java.lang.String url)
```

- Encodes the specified URL for use in the `sendRedirect()` method, or if encoding is not needed, returns the URL unchanged.
- All URLs sent with the `sendRedirect()` method should be run encoded using this method.

Request Dispatcher 1-4

- A servlet runs in an environment called the servlet context, which describes various parameters associated with the servlet.
- A servlet belongs to only one servlet context.
- A servlet cannot access resources such as static HTML pages, which are stored in local files using the `RequestDispatcher` objects.
- The local resource can be accessed using the method `getResource(String path)` of `ServletContext` object that returns a URL object for a resource specified by a local Uniform Resource Identifier (URI), for example, `'/html'`.
- The method `ServletRequest.getRequestDispatcher(java.lang.String)` allows using relative path, whereas the method `ServletContext.getRequestDispatcher(java.lang.String)` allows only absolute path.

Request Dispatcher 2-4

- The servlets belonging to same application can share data using the following methods of `RequestDispatcher` interface are as follows:

`forward()` method

- The method is used to forward request from one servlet to another resource on the same server.
- **Syntax:**

```
public void forward(ServletRequest request1,
ServletResponse response1) throws
 ServletException, IOException
```

where,

- `Request1` – is the request made by the client to the servlet.
- `response1` – is the response made by the servlet to the client.

Request Dispatcher 3-4

- The code snippet retrieves the servlet context from its `RequestDispatcher` instance, and then forwards control to the JSP page.

```
public class MyServlet extends HttpServlet
{
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
   request.setAttribute("title", "Home Page");
   ServletContext servletContext = getServletContext();
    RequestDispatcher dispatcher = servletContext.
getRequestDispatcher("/cart.jsp");
dispatcher.forward(request,response);
 }
}
```

Request Dispatcher 4-4

include() method

- The method is used to include the contents of another servlet, JSP page, or an HTML file.
- **Syntax:**

```
public void include(ServletRequest request1, ServletResponse
response1) throws ServletException,IOException void
println(char c)throws java.io.IOException
```

- The code snippet demonstrates how to include the content from the specified path.

```
// Include static page or servlet/JSP page from the specified path,
. . .
RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/contactus.jsp");
if (dispatcher == null)
  out.println(path + " not found");
else
  dispatcher.include(request, response);
. . .
```

init Parameters 1-2

- The `ServletConfig` interface provides various methods to configure a servlet, before processing the data requested by the client.
- The servlet has the direct access to Servlet initialization parameters using the `getInitParameter()` method.
- **Syntax of `getInitParameter()` method:**

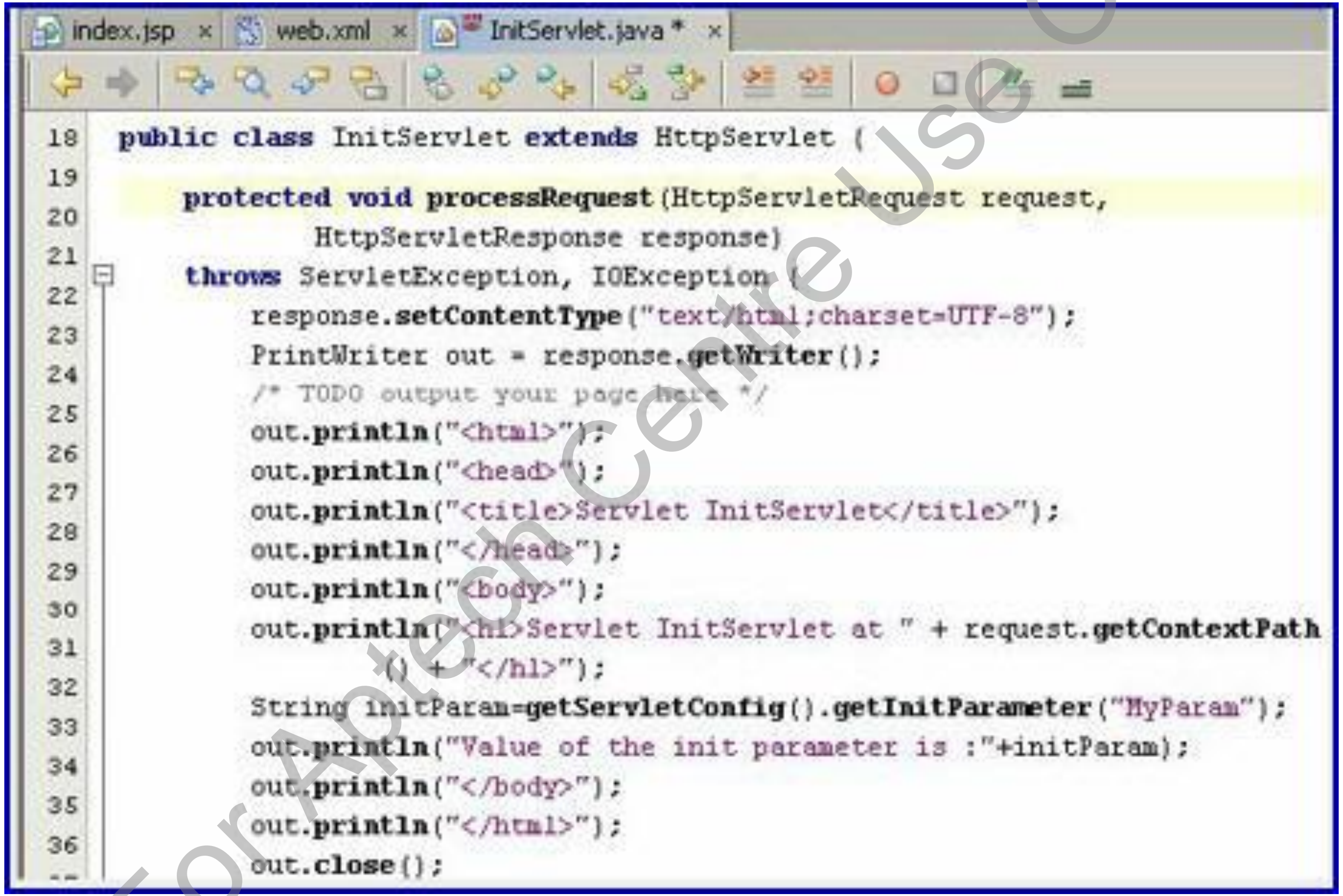
```
public String getInitParameter(String
name)
```

- The code snippet demonstrates how to retrieve the parameters of the servlet.

```
//Retrieves the parameters user and password
super.init(config);
String user1 = getInitParameter("user");
String password1 = getInitParameter("password");
. . .
```

init Parameters 2-2

- Figure depicts servlet initialization.



```
public class InitServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        /* TODO output your page here */
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet InitServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet InitServlet at " + request.getContextPath
            () + "</h1>");
        String initParam=getServletConfig().getInitParameter("MyParam");
        out.println("Value of the init parameter is :"+initParam);
        out.println("</body>");
        out.println("</html>");
        out.close();
```

Reading Parameters in init() Method

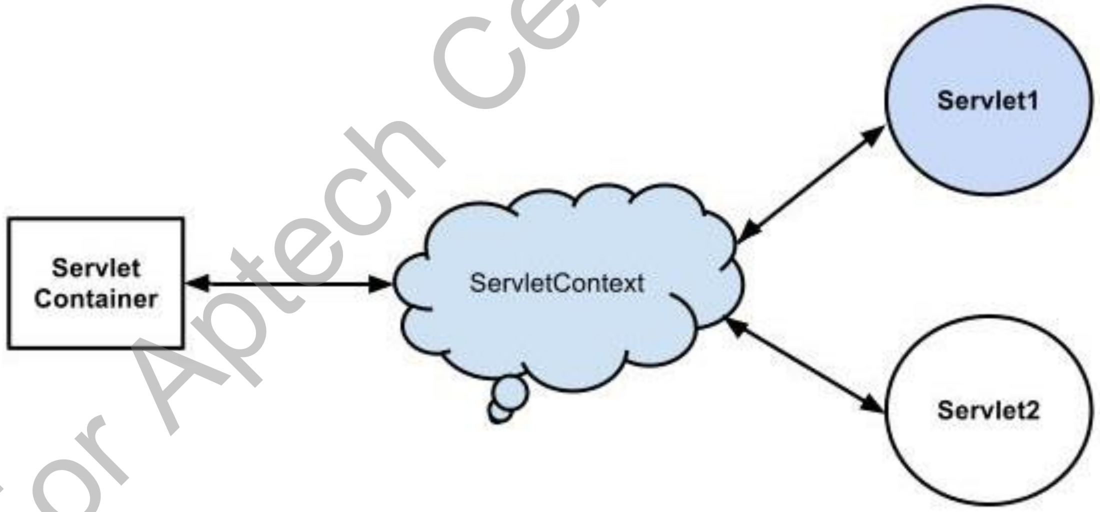
```
public void init() throws ServletException
```

- The code snippet illustrates how a servlet reads initialization parameters.

```
int count;
public void init(ServletConfig config) throws
ServletException {
super.init(config);
String initial = config.getInitParameter("initial");
try {
count = Integer.parseInt(initial);
} catch (NumberFormatException e) {
count = 0;
}
}
```

ServletContext

- Provides access to various resources and facilities, which are common to all servlets in the application.
- Servlets running in the same server sometimes share resources, such as JSP pages, files, and other servlets.
- The servlet API also knows about resources to the servlets in the context.
- The attributes of `ServletContext` class can be used to share information among a group of servlets.



ServletContext Methods 1-3

- The `ServletContext` methods are used to get and set additional information about the servlet.
- Some commonly used methods of `ServletContext` interface are as follows:

```
public String getInitParameter(String name)
```

```
public void setAttribute(String name,Object object)
```

```
public Object getAttribute(String name)
```

```
public Enumeration getInitParameterNames()
```

```
public void removeAttribute(String name)
```

ServletContext Methods 2-3

- The code snippet shows the `web.xml` file for specifying the context parameters and servlet initialization parameters.

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
. . .
<context-param>
    <param-name>Global</param-name>
    <param-value>ValueToAll</param-value>
</context-param>
<!-- other stuff including servlet declarations -->
<servlet>
    <servlet-name>ParamServletTests</servlet-name>
    <servlet-class>TestInitParams</servlet-class>
<init-param>
    <param-name>ServletSpecific</param-name>
    <param-value>ServletValue</param-value>
</init-param>
</servlet>
</web-app>
```

ServletContext Methods 3-3

- The code snippet demonstrates how to read the context parameters from the `web.xml` file.

```
. . .
public class ContextParameters extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse
res)
throws ServletException,IOException
{
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter();
// Creating ServletContext object
    ServletContext context=getServletContext();
//Getting the value of the initialization parameter and
printing it
    String str = context.getInitParameter("Global");
    pw.println("driver name is="+ str);
    pw.close();
}
}
```

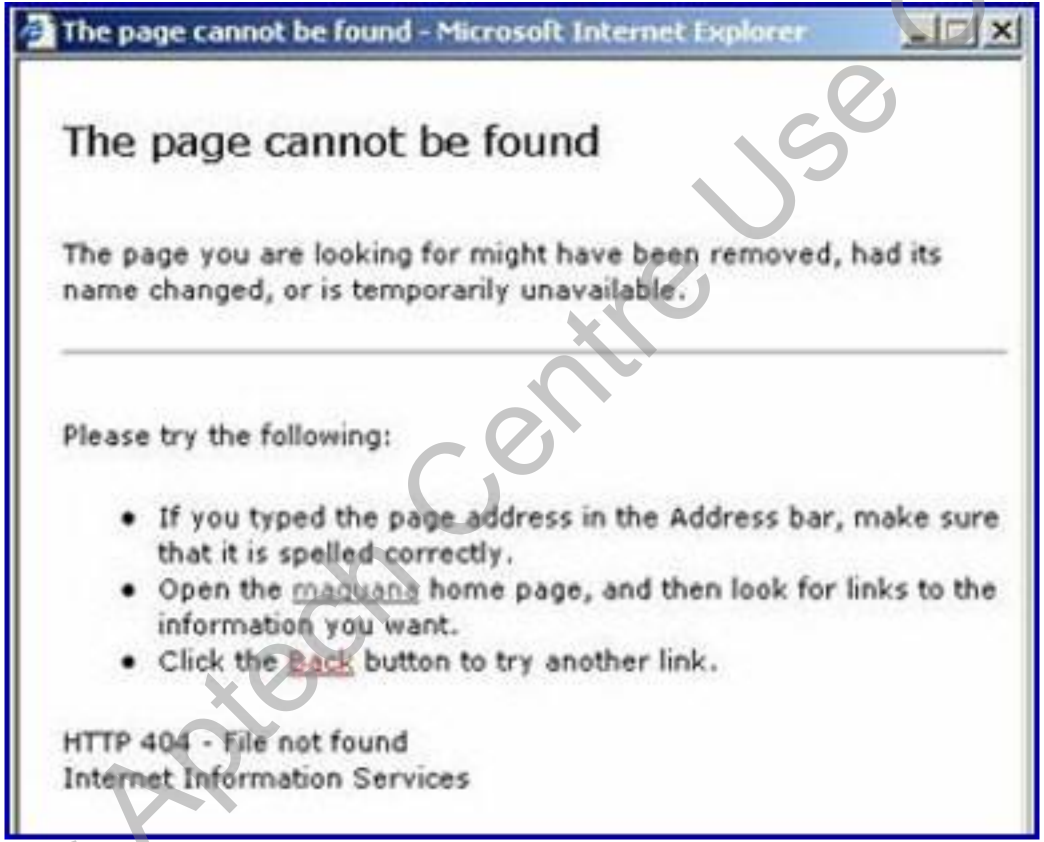
Status Codes 1-2

- Table lists some of the error codes along with their associated message and meaning.

Status Code	Associated Message	Meaning
301	Moved Permanently	Document is moved to a separate location as mentioned in the URL. The page is redirected to the mentioned URL.
302	Found	Temporary replacement of file from one location to the other as specified.
400	Bad Request	The request placed is syntactically incorrect.
401	Unauthorized	Authorization is not given to access a password protected page.
404	Not Found	Resource not found in the specified address.
408	Request Timeout	Time taken by client is very long to send the request (only available in HTTP 1.1).
500	Internal Server Error	Server is unable to locate the requested file. The servlet has been deleted or crashed or has been moved to a new location without informing.

Status Codes 2-2

- Figure depicts an example of status code.



The page cannot be found

The page you are looking for might have been removed, had its name changed, or is temporarily unavailable.

Please try the following:

- If you typed the page address in the Address bar, make sure that it is spelled correctly.
- Open the maquans home page, and then look for links to the information you want.
- Click the Back button to try another link.

HTTP 404 - File not found
Internet Information Services

Error Methods in 'HttpServlet' Class 1-2

- Errors during the execution of a Web application are reported using the following methods:

`sendError()`

- This method checks for the status code and sends to the user the specified response message.
- After sending the error message, the buffer is cleared.
- **Syntax:**

```
public void sendError(int sc1) throws
java.io.IOException
```

- The code snippet shows the use of `sendError()` method.

```
// Return the file file1
try {
    ServletUtils.returnFile(file1, out);
}
catch (FileNotFoundException err) {
    res.sendError(res.SC_NOT_FOUND);
}
```

Error Methods in 'HttpServlet' Class 2-2

`setStatus()`

- This code is specified earlier so that on receiving the `setStatus()` method, the error message is thrown or redirected to another default Web page.
- **Syntax:**

```
public void HttpServletResponse.setStatus(int sc1)
```

- The code snippet shows the use of `setStatus()` method.

```
/* If the client sent an If-Modified-Since header equal or
after the servlet's last modified time, send a short "Not
Modified" status code Round down to the nearest second since
client headers are in seconds */

if (servletLastMod == -1) {
    super.service(req, res);
}
else if ((servletLastMod / 1000 * 1000) <=
    req.getDateHeader("If-Modified-Since")) {
    res.setStatus(res.SC_NOT_MODIFIED);
}
```

Logging Errors 1-3

- Servlets can store the actions and errors through the `log()` method of the `GenericServlet` class.
- The `log()` method also assists in debugging by describing all the details about an error.
- The `log()` method can be used by passing either a single argument or two arguments.
- **Syntax:**

```
public void log(String msg1)
```

- The error can be viewed by using `log()` to record in a server.

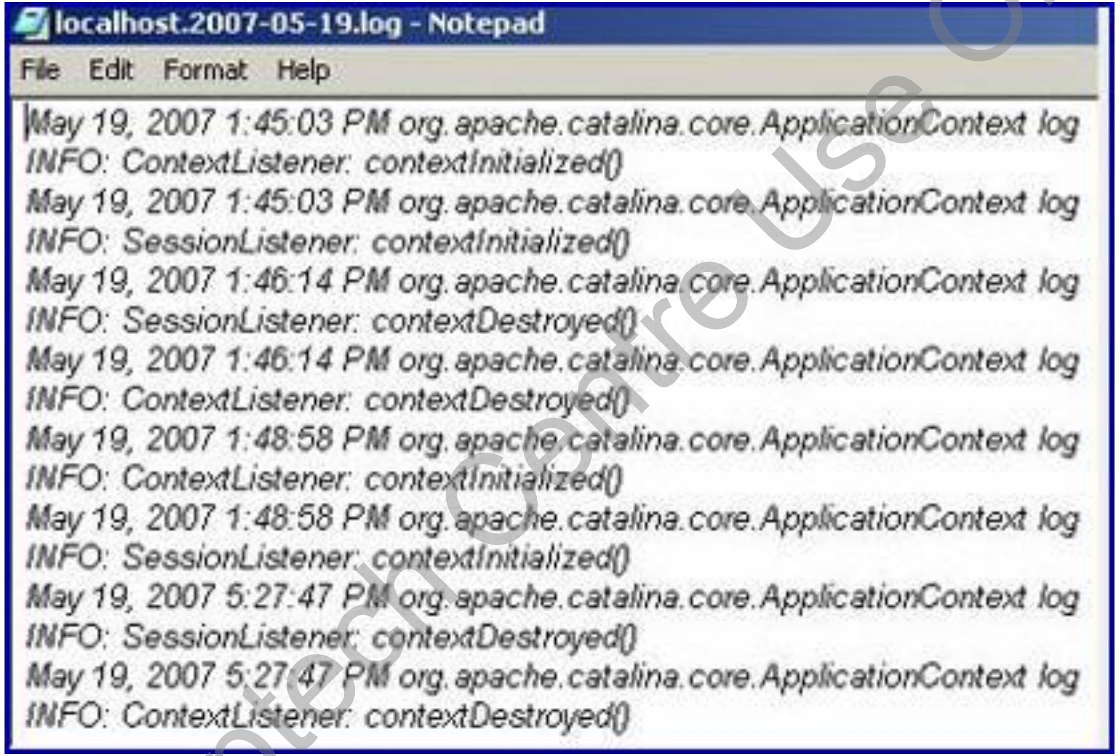
Logging Errors 2-3

- The code snippet shows the use of `log()` methods.

```
// Return the file
try {

    ServletUtils.returnFile(file, out);

}catch (FileNotFoundException e){

    log("Could not find file: " + e.getMessage());
    res.sendError(res.SC_NOT_FOUND);
}
catch (IOException e) {
    log("Problem sending file", e);
    res.sendError(res.SC_INTERNAL_SERVER_ERROR);
}
```

Logging Errors 3-3

- Figure depicts a log file.



```
May 19, 2007 1:45:03 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextInitialized()
May 19, 2007 1:45:03 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextInitialized()
May 19, 2007 1:46:14 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextDestroyed()
May 19, 2007 1:46:14 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextDestroyed()
May 19, 2007 1:48:58 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextInitialized()
May 19, 2007 1:48:58 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextInitialized()
May 19, 2007 5:27:47 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextDestroyed()
May 19, 2007 5:27:47 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextDestroyed()
```

Forwarding to Error Page 1-2

- Following methods create object for the `RequestDispatcher` interface:

`'ServletContext.getRequestDispatcher(String path)'`

- This method takes a string argument as a path, which is within the scope of the `ServletContext`.
- The path, relative to root is passed as an argument.
- This path, in turn, is used to locate the servlet to which it is redirected.

`'ServletContext.getNamedDispatcher(String name)'`

- The method takes a string argument indicating the name of servlet to the `ServletContext`.
- It returns a `RequestDispatcher` object for the named servlet.

`'ServletRequest.getRequestDispatcher(String path)'`

- This method uses relative path which is the location of the file relative to current path.

Forwarding to Error Page 2-2

- The code snippet shows the dispatching of error object.

```
public class Dispatcher extends HttpServlet {
    public void doGet(HttpServletRequest req1,
    HttpServletResponse res1) {
        RequestDispatcher dispatcher1 =
            request.getRequestDispatcher("/error_page. jsp");
        if (dispatcher1 != null)
    dispatcher1.forward(req1, res1);
        }
    }
}
```

Summary

- The init(), service(), and destroy() methods are the servlet's lifecycle methods. The init() method intimates the servlet that it is being placed into service.
- The service() method is called by the servlet container to allow the servlet to respond to a request. The servlet container calls the destroy() method after all threads within the servlet's service method have exited or after a timeout period has passed.
- A GenericServlet class defines a servlet that is not protocol dependent. To have better control over the required servlets HttpServlet is extended to GenericServlet.
- A servlet request contains the data to be passed between a client and the servlet. All requests implement the ServletRequest interface, which defines the methods for accessing the relevant information.
- A servlet response contains data to be passed between a server and the client. All responses implement the ServletResponse interface. This interface defines various methods to process response.
- You use getInputStream() to retrieve the input stream as a ServletInputStream object. ServletOutputStream provides an output stream for sending binary data to the client.
- Resources are included to a servlet by forwarding request from one servlet to another by using the forward() and include() methods along with RequestDispatcher interface. Inter-servlet communication can be used by the servlets to gain access to other currently loaded servlets and perform some tasks on other servlets.
- Errors may occur due to one reason or the other but the status code for each type of error, makes it easy for identifying the error. Errors are reported using sendError() and setStatus() methods. Errors are logged using the log() method of ServletContext and forwarded to an error page using the RequestDispatcher interface.