

# Fundamentals of Java Enterprise Components

## **Session: 5**

Java Servlets

For Aptech Centre Use Only

# Objectives



- ▶ Describe servlet and its lifecycle
- ▶ Describe how various events in the lifecycle of the servlet are handled
- ▶ Explain configuration of the Servlet
- ▶ Explain creating and handling HTTP requests and responses
- ▶ Describe writing service methods for Servlets
- ▶ Describe how Web resources can be accessed by the Servlet
- ▶ Explain finalization of a servlet
- ▶ Explain asynchronous processing in a servlet
- ▶ Describe implementing non-blocking I/O in servlets

# Servlet



A server-side component of Web application.

Receive a request from client and process it.

Client can be JSF, JSP, HTML, or any other Web client.

Servlets receive requests over Internet through HTTP protocol.

Java servlets are implemented through Java Servlet API.

Servlets construct response pages and return it to the client.

# Lifecycle of a Servlet 1-2



## Initialization

- Initialized when a request is received.
- `init()` method of `javax.servlet` interface is used to initialize a servlet.
- Initialized after servlet is loaded by the container of the application.
- Initialized only once in the lifecycle of the servlet.

## Service

- Service provided by the servlet is defined in the `service()` method.
- The service method has two parameters of type `ServletRequest` and `ServletResponse`.
- `Service()` method can be invoked any number of times after initialization.

# Lifecycle of a Servlet 2-2



## Destruction

- Servlet is removed from the memory using `destroy()` method.
- `destroy()` method deallocates memory allocated for the servlet.
- Invoked only once during the lifecycle of the memory.

# Servlet Listener Classes 1-3



Following are the classes which create listener objects to keep track of the lifecycle events of a servlet:

## **javax.servlet.AsyncListener**

- This listener object is notified if there is a change of state of an asynchronous operation which is initiated by a `ServletRequest` object. The asynchronous event which is triggered due to a `ServletRequest` is of type `javax.servlet.AsyncEvent`.

## **javax.servlet.ServletContextListener**

- This listener object is notified when a lifecycle event such as initialization of a servlet occurs. The corresponding event object is, `javax.servlet.ServletContextEvent`.

# Servlet Listener Classes 2-3



## `javax.servlet.ContextAttributeListener`

- This listener object is notified when there is a change in the attributes of the lifecycle events of the servlet. The corresponding event is, `javax.servlet.ContextAttributeEvent`.

## `javax.servlet.ServletRequestListener`

- This listener object is notified if the servlet receives a request or sends out a request from the Web application. The corresponding event is, `javax.servlet.ServletRequestEvent`.

## `javax.servlet.ServletRequestAttributeListener`

- This listener object is notified if there is a change in the attributes of the servlet requests. The corresponding event is, `javax.servlet.RequestAttributeEvent`.

## `javax.servlet.http.HttpServletRequestListener`

- This listener object is notified when there are HTTP requests being received from other components or when HTTP requests are sent out to other components of the application.

# Servlet Listener Classes 3-3



## `javax.servlet.http.HttpSessionBindingListener`

- This listener object is notified when a certain object is bound or unbound to an HTTP session. The corresponding event is, `javax.servlet.http.HttpSessionBindingEvent`.

## `javax.servlet.http.HttpSessionAttributeListener`

- This listener object is notified when there is a change in the attributes of an HTTP session.

## `javax.servlet.http.HttpSessionActivationListener`

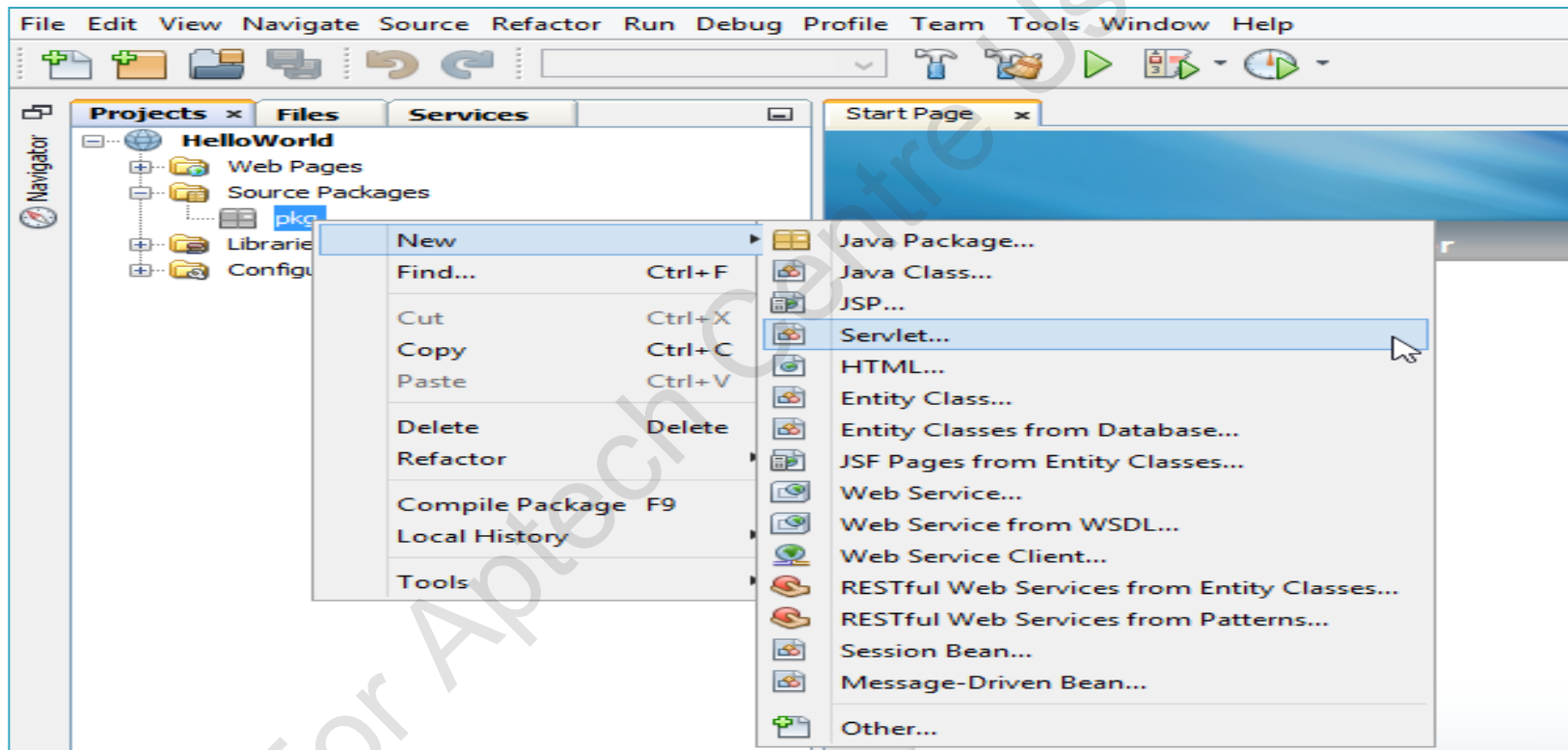
- HTTP sessions can be passivated instead of removing them from the memory and again activated where there is a request for the servlet. When HTTP session activation occurs, this listener object is notified.



# Creating a Servlet in NetBeans IDE 1-5



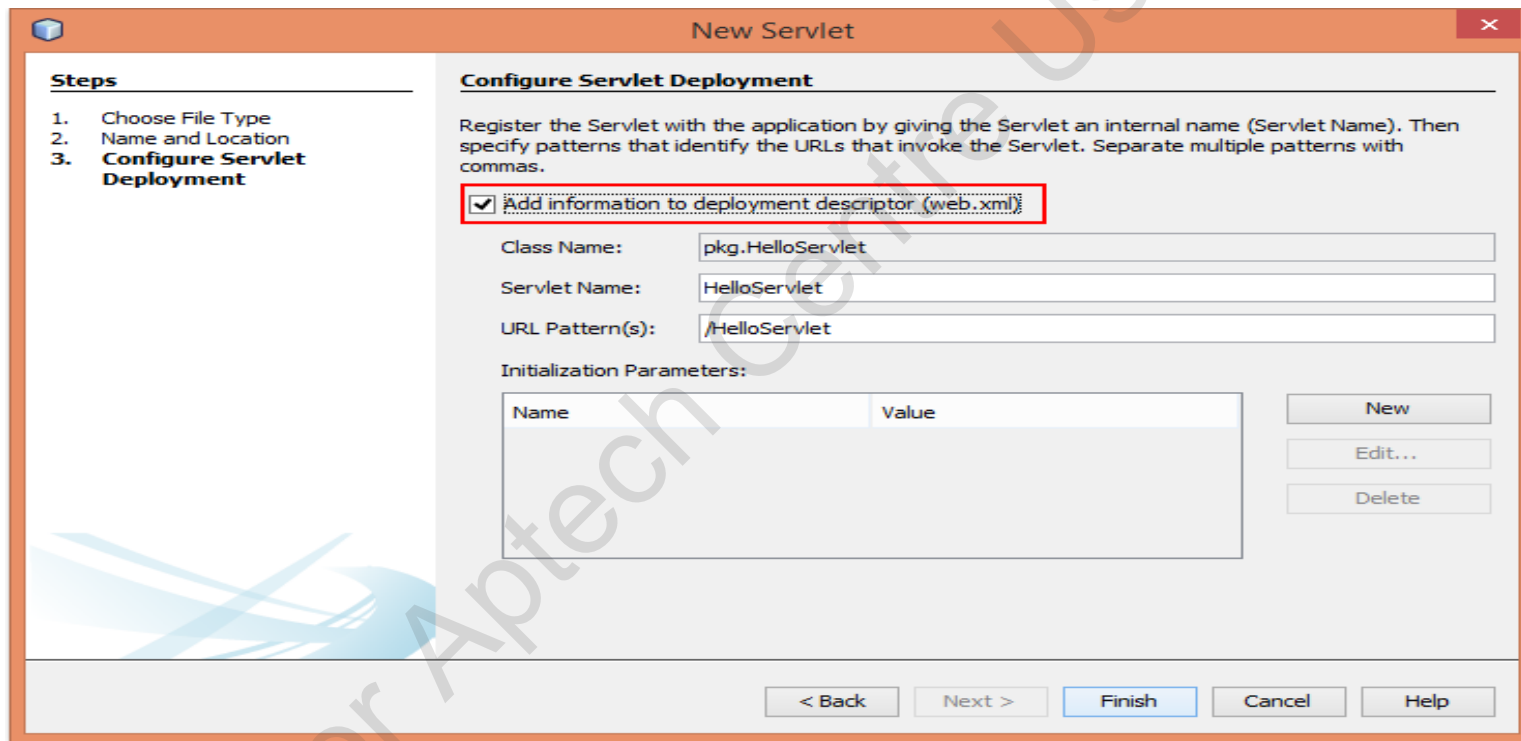
To create a new servlet, create a new package in the Web application by right-clicking the Source Packages folder and selecting New → Java Package. Next, right-click the package and select New → Servlet as shown in the following figure:



# Creating a Servlet in NetBeans IDE 2-5



The web.xml file is created automatically if the Add information to deployment descriptor (web.xml) checkbox is selected while creating a servlet, as shown in the following figure:



The image shows the 'New Servlet' dialog box in NetBeans IDE. The 'Steps' pane on the left shows three steps: 1. Choose File Type, 2. Name and Location, and 3. Configure Servlet Deployment. The 'Configure Servlet Deployment' pane is active. It contains a checkbox labeled 'Add information to deployment descriptor (web.xml)' which is checked and highlighted with a red rectangle. Below this, there are text fields for 'Class Name' (pkg.HelloServlet), 'Servlet Name' (HelloServlet), and 'URL Pattern(s)' (/HelloServlet). There is also a section for 'Initialization Parameters' with a table with two columns: 'Name' and 'Value'. To the right of the table are buttons for 'New', 'Edit...', and 'Delete'. At the bottom of the dialog are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**Steps**

1. Choose File Type
2. Name and Location
3. **Configure Servlet Deployment**

**Configure Servlet Deployment**

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ **Add information to deployment descriptor (web.xml)**

Class Name: pkg.HelloServlet

Servlet Name: HelloServlet

URL Pattern(s): /HelloServlet

Initialization Parameters:

Name	Value
------	-------

New Edit... Delete

< Back Next > Finish Cancel Help

# Creating a Servlet in NetBeans IDE 3-5



Following code is generated by the container when a servlet is created:

```
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(urlPatterns = {"/Servlet_Example"})
public class HelloServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet First_Servlet</title>");
        }
    }
}
```

# Creating a Servlet in NetBeans IDE 4-5



```
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet First_Servlet at " +
request.getContextPath() + "</h1>");
        out.println("</body>"); out.println("</html>");
    }
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    processRequest(request, response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    processRequest(request, response);
}

@Override
public String getServletInfo() {
    return "Short description";
}

} // </editor-fold>
```

# Creating a Servlet in NetBeans IDE 5-5



Following are the methods present in the code generated by the NetBeans IDE:

- **processRequest()** - is used to write the processing code to process the incoming request.
- **doGet()** – used when the incoming request is a HTTP GET operation, in turn invokes processRequest() method.
- **doPost()** – used when the incoming request is a HTTP POST operation, it also invokes processRequest() method.
- **getServletInfo()** – Returns information about the servlet.

# Servlet Context and Configuration 1-7



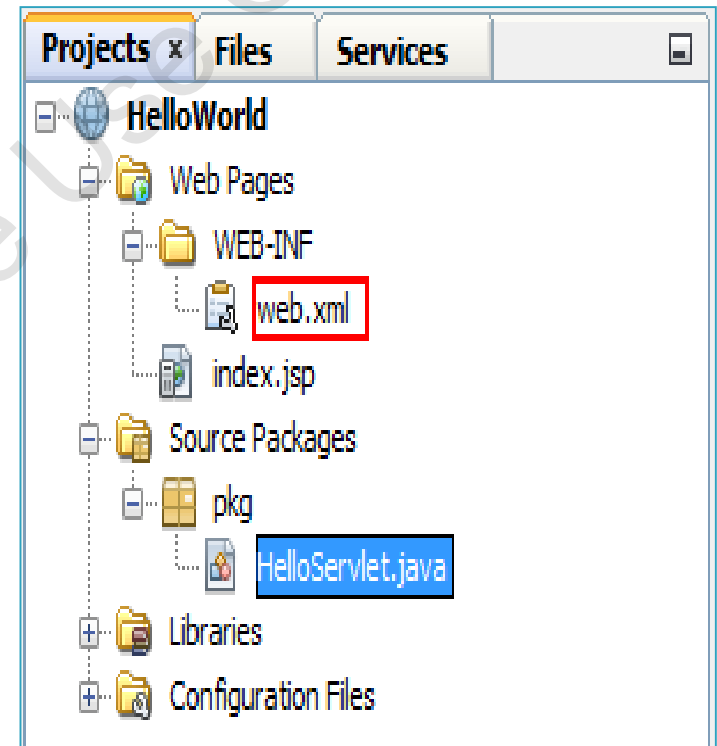
A servlet context defines various attributes of the servlet and the various resources accessed by a servlet.

It is defined through class `javax.servlet.ServletContext` in Java Servlet API.

The context is initialized when the servlet is initialized.

When a servlet is created through NetBeans IDE, the context of the servlet is set during the deployment of the application through the deployment descriptor `web.xml`.

Location of `web.xml` is shown in the figure.



# Servlet Context and Configuration 2-7



The web.xml file contains the deployment configuration information of the Web application. Following code is used for the deployment of the Web application:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
```

# Servlet Context and Configuration 3-7



```
<url-pattern>/HelloServlet</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
</web-app>
```



# Servlet Context and Configuration 4-7



Context parameters can be added to the deployment descriptor, which are values referring to the context of the servlet.

Following code can be added to the deployment descriptor to define the context parameters of the servlet:

```
<context-param>
    <param-name>n1</param-name>
    <param-value>100</param-value>
</context-param>
<context-param>
    <param-name>n2</param-name>
    <param-value>200</param-value>
</context-param>
```

# Servlet Context and Configuration 5-7



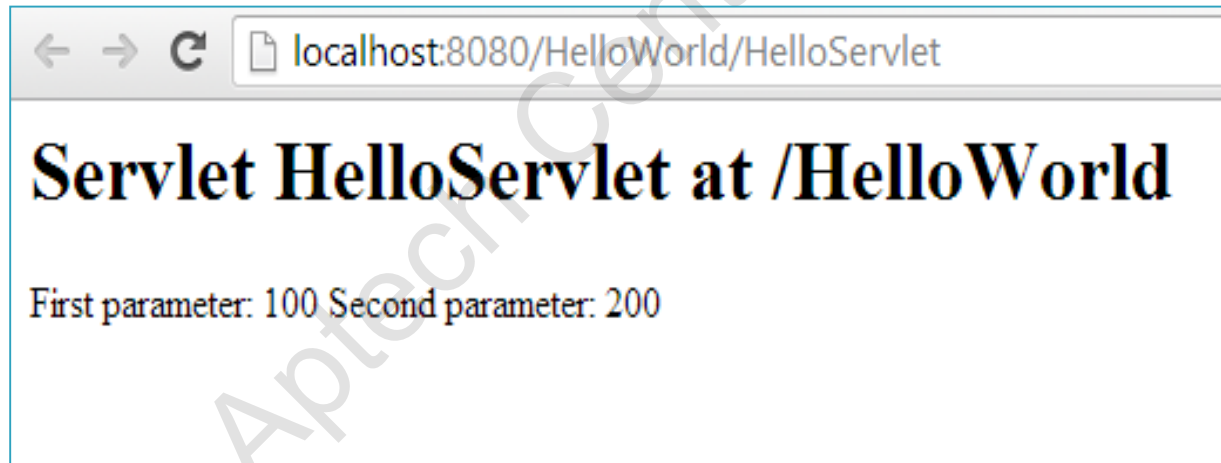
Following code enables accessing the context parameters from the `processRequest()` method of the servlet:

```
.....  
ServletContext sc = getServletContext();  
    String para1 = sc.getInitParameter("n1");  
    String para2 = sc.getInitParameter("n2");  
    out.println("First parameter: "+para1);  
    out.println("Second parameter: "+para2);  
.....
```

# Servlet Context and Configuration 6-7



To execute the servlet code, right-click anywhere on the servlet and select Run File. Following figure shows the output of the application:



# Servlet Context and Configuration 7-7



Every `ServletContext` object has an associated `ServletConfig` object, which has all the configuration details such as initialization parameters and other values generated by the container. Following are the methods defined in the `ServletConfig` interface:

- **`getInitParameter()`** – This method accepts the initialization parameter name and returns the initialization parameter value.
- **`getInitParameterNames()`** – This method returns the names of all the initialization parameters.
- **`getServletContext()`** – This method returns the context in which the servlet is executing.
- **`getServletName()`** – This method returns the name of the servlet.

# Request and Response Methods 1-4



The request can be a `ServletRequest` or `HttpServletRequest` object and as a response, the servlet may generate `ServletResponse` or `HttpServletResponse` object.

The format of an HTTP request is as follows:

- `http://[host]:[port][request-path]?[query-string]`

Non-HTTP requests implement the `ServletRequest` interface. The methods implemented in the interface access information such as parameters of the `ServletRequest`, object valued attributes which are used for communication between container and servlet, protocol information.

# Request and Response Methods 2-4



A response is a communication from the servlet to the client. Following are different steps involved in generating response to the client requests:

The data to the client can be sent through an output stream such as `PrintWriterStream` and multimedia content through a `ServletOutputStream` object.

Methods in the `ServletResponse` interface are used to indicate the content type of the response in case of multimedia content.

There are methods to communicate the requirement of buffer at the receiver end.

# Request and Response Methods 3-4



- ▶ The ServletRequest interface provides the `getParameter` method to fetch the request parameters on the servlet. Request parameters are extra information sent with the request. The syntax of the method is as follows:
  - `String getParameter(String name)`
- ▶ The method returns the value of a request parameter as a `String` or `null` if the parameter does not exist.

# Request and Response Methods 4-4



HTTP status codes are also included in case of HTTP responses. Various status codes are shown in the given table.

HTTP sessions may also require cookies to be set on the client-side. These cookies have the session information to be sent to the client from the server.

HTTP Status Code	Message
400 Bad Request	Syntax error
404 Not Found	The requested page could not be found on the server
408 Request Timeout	The client has waited for more than the time out period and the server did not respond during this time
412 Precondition Failed	The required pre-condition for the request is not fulfilled



# RequestDispatcher



- ▶ A RequestDispatcher object is used to forward a request to the resource or to include a resource in a response. The resource can be dynamic or static.
- ▶ A relative pathname can be specified, however, it cannot extend outside the current servlet context.
- ▶ If the path begins with a "/" it is interpreted as relative to the current context root.
- ▶ It consists of two methods as follows:
  - **forward**
    - `void forward(ServletRequest request, ServletResponse response)`
  - **include**
    - `void include(ServletRequest request, ServletResponse response)`

# Filtering Requests and Responses 1-2



Following are the functions performed by filters:

- ▶ Performing authentication on the headers and blocking the requests, if required
- ▶ Auditing the users of the applications and also, profiling the users who are accessing the application
- ▶ Compressing data
- ▶ Applying localization methods such as transforming the character set
- ▶ Interacting with external resources

# Filtering Requests and Responses 2-2



Filters are defined on the requests and responses with the help of filter API. Following are the steps implemented by the filter:

- ▶ Querying the request and response headers
- ▶ Blocking the requests from propagating further based on the data in the header
- ▶ Modifying the request and response headers and their data

# Programming Filters



The filter API is implemented as a part of javax.servlet package and consists of the Filter, FilterChain, and FilterConfig interfaces. In order to define a filter, the Filter interface is implemented.

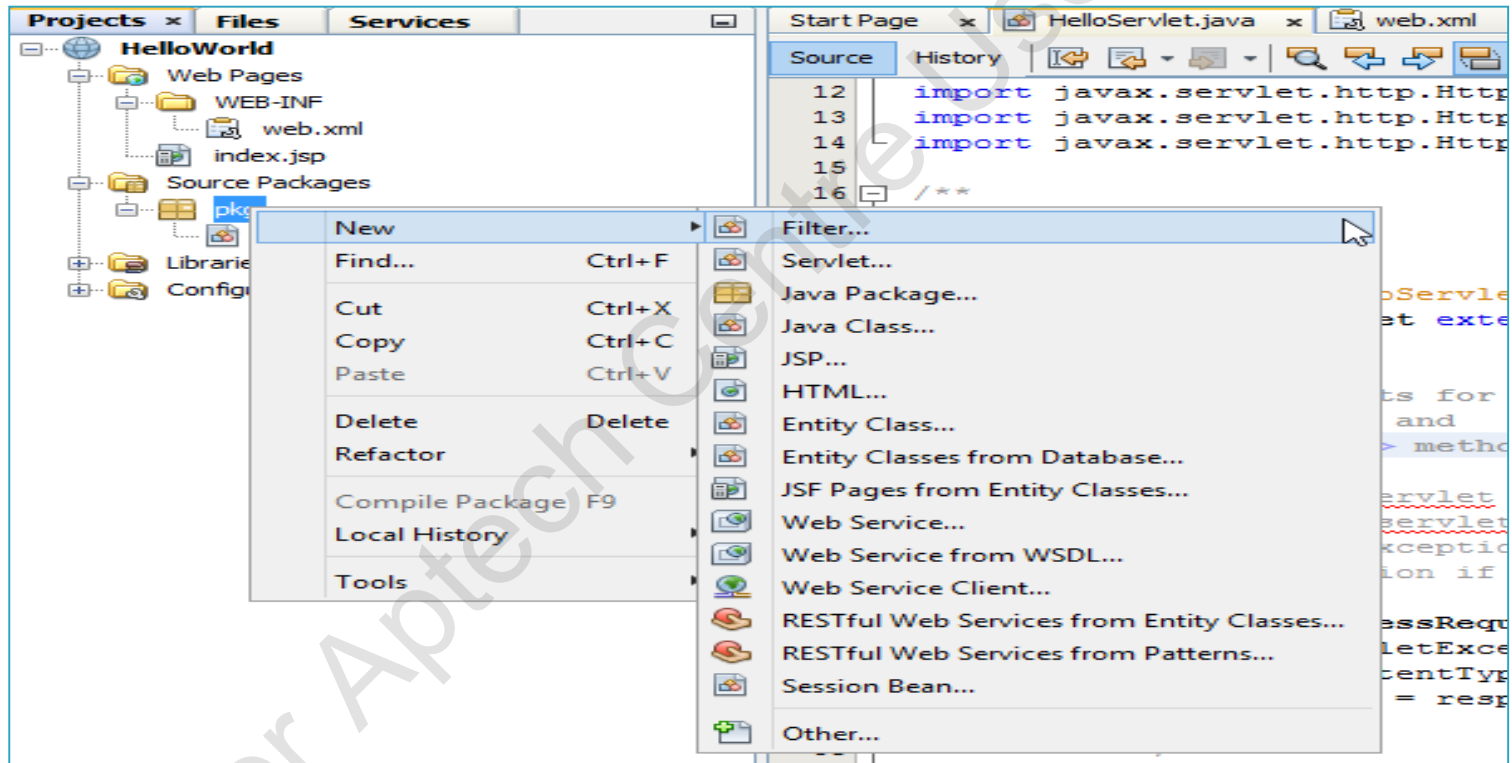
When the Filter interface is implemented the doFilter() method has to be defined. Following are the actions performed by the doFilter() method:

- Examines the request headers
- Customizes or blocks the request
- Customizes or blocks the response of the servlet
- In case of multiple filters applied, the next filter in the sequence is invoked after the current filter completes its processing
- The filter objects also have an init() and destroy() methods which are invoked when the service of the filter is required and when the service is terminated

# Adding Filter in NetBeans IDE



While using NetBeans, IDE Filter object can be created by adding a new file to the Web application context as shown in the following figure:



# Programming Customized Requests and Responses



The filter passes an additional stream to the servlet, to customize the requests and responses.

This additional stream is passed to the servlet by defining a wrapper to the servlet generated stream.

`ServletRequestWrapper` and `HttpServletRequestWrapper` are used to implement the filter actions on the request.

`ServletResponseWrapper` and `HttpServletResponseWrapper` are used for customizing the responses.

# Specifying Filter Mappings



Mapping information specifying which filter to apply to which resource is essential.

Filter mappings are done in the deployment descriptor of the application, that is web.xml.

Following is an example of filter mapping in the deployment descriptor.

```
.....
    <filter>
        <filter-
name>HelloServletFilter</filter-
name>
        <filter-
class>pkg.HelloServletFilter</filte
r-class>
    </filter>
    <filter-mapping>
        <filter-
name>HelloServletFilter</filter-
name>
        <servlet-
name>HelloServlet</servlet-name>
    </filter-mapping>
.....
```

# Maintaining Client State 1-2



Every HTTP session with the servlet is represented by an `HttpSession` object. The `getSession()` method is used to return the session associated with the current servlet request.

## Associating objects with a session

- A session tracking mechanism is required which will map the objects of the session to their respective values.
- Listener objects can be defined to keep track of object mappings when they change. Following are the listener objects:
  - `javax.servlet.http.HttpSessionBindingListener` interface
  - `javax.servlet.http.HttpSessionActivationListener` interface



# Maintaining Client State 2-2



## Session management and tracking

- Every HTTP session is associated with a time out period, after which the resources allocated to the session can be reclaimed.
- To avoid session expiry due to time out, the session should be periodically accessed.
- The time out interval can be accessed and set through `getMaxInactiveInterval()` and `setMaxInactiveInterval()` methods.
- After the purpose of the session is served, the session has to be invalidated through `invalidate()` method.
- Session time out can be set in the `web.xml` file by modifying the `<Session-timeout>` value.

# Finalizing a Servlet



The Web container removes the servlet from the memory by invoking the `destroy()` method.

The Web container has to check on the following parameters of the servlet before removing it:

- All the service methods of the servlet should have returned calls
- Any pending service methods should be given certain grace period to complete the service requests
- Any service request that is pending must be identified even after the grace period
- The Web container must notify the long running service requests and wait until these threads complete their operation

The servlet class must maintain a counter to track the number of service requests being served.

The Web container must notify all the active methods to shutdown through `setShuttingDown()` method.

# Developing and Deploying a Servlet Application



Creation of a new project in NetBeans creates various sub-directories which are used for purposes such as testing the application, holding the class files pertaining to the application, and so on.

Application is developed by creating all class files in the Source packages directory.

All the Web components such as Servlets, HTML pages etc are created in the Web pages directory.

Deployment information of the application is stored in WEB-INF directory in web.xml file.

To build the application, right-click the application and select 'Clean and Build'.

Once the application is successfully built choose 'Deploy' to deploy the application on the Web server.

# Asynchronous Processing 1-2



While executing servlets and filters in an application, if the execution reaches a blocking state, then the execution thread is handed over to an asynchronous context and returns without generating the response.

The Asynchronous context completes the blocking operation and returns the response, else it hands over the thread to another servlet.

The `javax.servlet.AsyncContext` interface provides the required functionality for asynchronous processing.

# Asynchronous Processing 2-2



Following are some of the methods provided by the AsyncContext class:

**void start(Runnable r)** – Through this method, the container provides a new thread instance to perform the operation which is blocking the application servlet from being removed from the container.

**getRequest()** – returns a ServletRequest object to return the request which has initiated the process of setting up an asynchronous context.

**getResponse()** – returns a ServletResponse object. It can be used to generate the response to the servlet request from the asynchronous context.

**complete()** – completes the asynchronous operation and returns the response to the client which has requested the operation.

**dispatch()** – accepts the dispatch path as an argument and dispatches the requests and responses of asynchronous context.

# Non-blocking I/O 1-2



Non-blocking I/O is used to process request and responses to improve performance of Java applications. Following steps summarize the implementation of non-blocking I/O:

- Initiate an asynchronous mode for the request/response
- For the request or response objects in the service methods, obtain corresponding input/output streams
- Assign listeners to the input and output streams, a read listener to the input stream and a write listener to the output stream
- Process the requests and responses in the listener callback methods

# Non-blocking I/O 2-2



Following are the methods provided by the `javax.servlet.ServletInputStream` for non-blocking I/O:

- **`void setReadListener(ReadListener r1)`** – assigns a listener object to the input stream, which has callback methods that asynchronously read data from the input stream
- **`isReady()`** – returns a boolean value indicating that data can be read without blocking
- **`isComplete()`** – returns a boolean value to indicate completion in reading all the data in the stream

Following are the methods provided by `javax.servlet.ServerOutputStream` for non-blocking I/O:

- **`void setWriteListener(WriteListener w)`** – assigns a listener object to the output stream which has callback methods that asynchronously write data to the output stream
- **`isReady()`** – returns whether the write operation can be completed without blocking

# Summary



- ▶ Servlets can process both HTTP and non-HTTP requests.
- ▶ A servlet is loaded by the container and initialized with init parameters. The servlet services the client requests and is removed by the container when the servlet is no longer requested by clients.
- ▶ Filters can be defined on the servlet requests and responses to perform tasks such as authentication, logging, and so on.
- ▶ Servlet API supports sessions and allows tracking of the sessions through different listener objects.
- ▶ A servlet, when removed from the container, must complete all the requests it is currently processing before releasing the memory space.
- ▶ Asynchronous processing of requests is implemented to avoid blocking of resources by the servlet.