H2K Infosys, LLC
Dream, Strive & Achieve Victory

H2K Infosys, LLC
Dream, Strive & Achieve Victory

H2K Infosys, LLC
Dream, Strive & Achieve Victory

www.H2KINFOSYS.com
USA: 770-777-1269
Training@H2KInfosys.com
UK : (020) 3371 7615

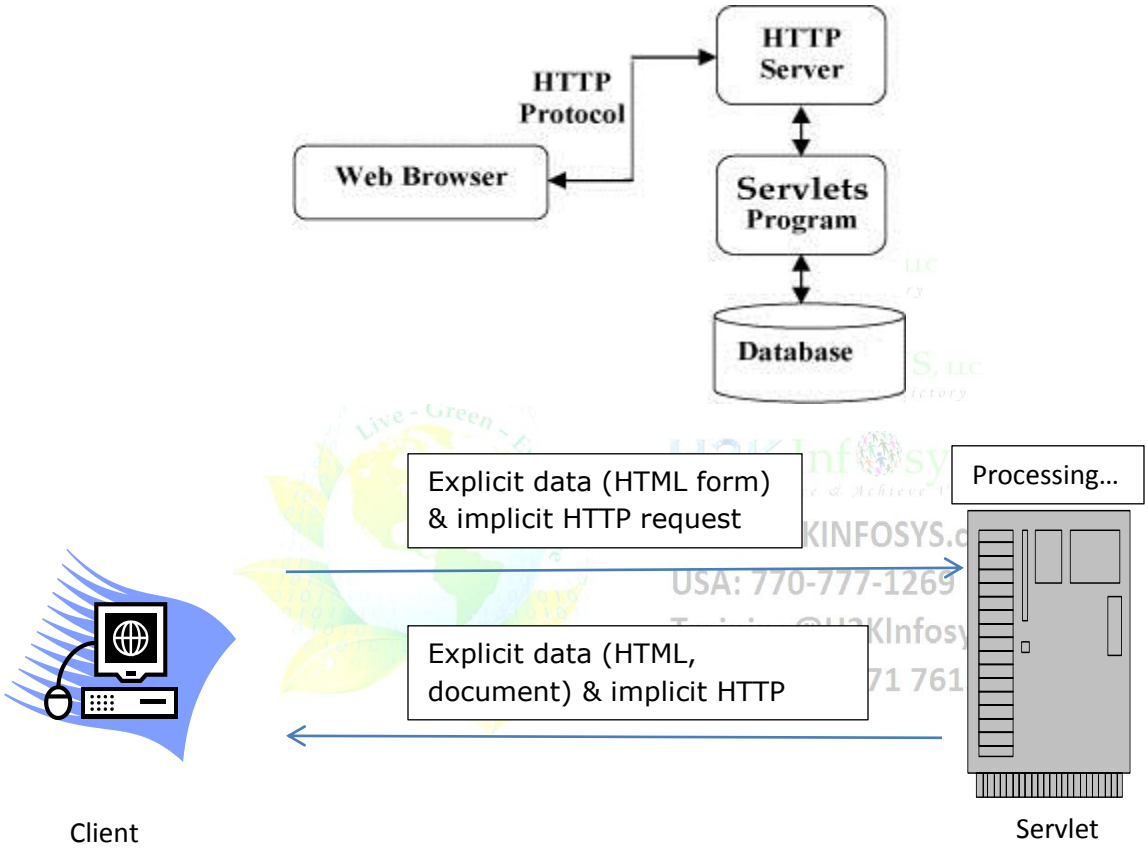
.H2KINFOSYS.com
770-777-1269
Training@H2KInfosys.com
UK : (020) 3371 7615

Servlets

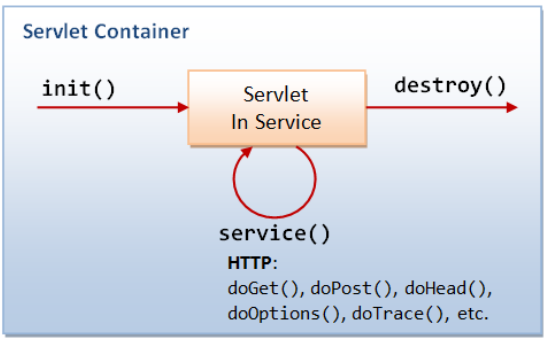
Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Servlets Architecture:



Servlets - Life Cycle



A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.

- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in details.

The init() method :

- The init method is designed to be called only once.
- Called when the servlet is first created

The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The servlet is normally created when:

- A user first invokes a URL corresponding to the servlet.
- If the servlet is set to be loaded when the server is first started

```
public void init() throws ServletException {
  // Initialization code...
}
```

The service() method :

- The service() method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate

```
public void service(ServletRequest request,
                    ServletResponse response)
      throws ServletException, IOException{
}
```

The doGet() and doPost() are most frequently used methods with in each service request.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to perform cleanup activities. After the destroy() method is called, the servlet object is marked for garbage collection.

```
public void destroy() {
    // Finalization code...
  }
```

Sample Code for Hello World:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException{
      // Do required initialization
      message = "Hello World";
  }

  public void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
  }

  public void destroy()  {
      // do cleanup activities if required.
  }
}
```

Web.XML settings:

```
<servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

What you get from HTTP Request?

Anything you want to get from is request! Request object has everything you need to serve your client. Important methods from HTTPRequest which are very helpful while programming are stated below:

Method	Description
Cookie[] getCookies()	Returns an array containing all of the Cookie objects the client sent with this request.
Enumeration getAttributeNames()	Returns an Enumeration containing the names of the attributes available to this request.
Enumeration getHeaderNames()	Returns an enumeration of all the header names this request contains.
Enumeration getParameterNames()	Returns Enumeration of String objects containing the names of the parameters contained in this request.
HttpSession getSession()	Returns the current session associated with this request, or if the request does not have a session, creates one.
HttpSession getSession(boolean create)	Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.
String getContentType()	Returns the MIME type of the body of the request, or null if the type is not known.
String getContextPath()	Returns the portion of the request URI that indicates the context of the request.
String getHeader(String name)	Returns the value of the specified request header as a String.

String getParameter(String name)	Returns the value of a request parameter as a String, or null if the parameter does not exist.
String[] getParameterValues(String name)	Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
int getIntHeader(String name)	Returns the value of the specified request header as an int.

What you do with HTTP Response?

Serlvet responds with HTTPResponse Object. Response object carries all the details required to make client browser understand the response from web.

Methods to Set HTTP Response Header:

Method	Description
String encodeRedirectURL(String url)	Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
String encodeURL(String url)	Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
void addCookie(Cookie cookie)	Adds the specified cookie to the response.
void addHeader(String name, String value)	Adds a response header with the given name and value.
void flushBuffer()	Forces any content in the buffer to be written to the client.
void sendError(int sc, String msg)	Sends an error response to the client using the specified status.
void sendRedirect(String location)	Sends a temporary redirect response to the client using the specified redirect location URL.
void setContentType(String type)	Sets the content type of the response being sent to the client, if the response has not been committed yet.
void setContentLength(int len)	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.

Examples 1: Code shows Request Headers

```java
public class ShowHeader extends HttpServlet{

      // Method to handle GET method request.
        public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                  throws ServletException, IOException
        {
            // Set response content type
            response.setContentType("text/html");

            PrintWriter out = response.getWriter();
              String title = "HTTP Header Request Example";
            String docType = "";
            out.println(docType +
              "<html>\n" +
              "<head><title>" + title + "</title></head>\n"+
              "<body>\n" +
              "<h1 align=\"center\">" + title + "</h1>\n" +
              "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
              "<tr>\n" +
              "<th>Header Name</th><th>Header Value(s)</th>\n"+
              "</tr>\n");

            Enumeration<String> headerNames = request.getHeaderNames();

            while(headerNames.hasMoreElements()) {
               String paramName = (String)headerNames.nextElement();
               out.print("<tr><td>" + paramName + "</td>\n");
               String paramValue = request.getHeader(paramName);
               out.println("<td> " + paramValue + "</td></tr>\n");
            }
            out.println("</table>\n</body></html>");
        }
        // Method to handle POST method request.
        public void doPost(HttpServletRequest request,
                           HttpServletResponse response)
            throws ServletException, IOException {
           doGet(request, response);
        }
      }
```

Examples 2: Servlet Creates a Browser Clock

```
public class MyBrowserClock extends HttpServlet{

	public void doGet(HttpServletRequest request,
		 HttpServletResponse response) throws ServletException, IOException {
		/*
		 * This header specifies how soon the browser should
		 * ask for an updated page.You can specify time in number
		 * of seconds after which a page would be refreshed.
		 */
		response.setIntHeader("Refresh", 1);

		// Set response content type
		response.setContentType("text/html");

		// Get current time
		Calendar calendar = new GregorianCalendar();
		String am_pm;
		int hour = calendar.get(Calendar.HOUR);
		int minute = calendar.get(Calendar.MINUTE);
		int second = calendar.get(Calendar.SECOND);
		if(calendar.get(Calendar.AM_PM) == 0)
		  am_pm = "AM";
		else
		  am_pm = "PM";

		String CT = hour+":"+ minute +":"+ second +" "+ am_pm;

		PrintWriter out = response.getWriter();
		String title = "Auto Refresh Header Setting";
		String docType ="";
		out.println(docType +
		  "<html>\n" +
		  "<head><title>" + title + "</title></head>\n"+
		  "<body>\n" +
		  "<h1 align=\"center\">" + title + "</h1>\n" +
		  "<p>Current Time is: " + CT + "</p>\n");
	}


	public void doPost(HttpServletRequest request,
			HttpServletResponse response)
	throws ServletException, IOException {
		doGet(request, response);
	}
}
```

Understanding sessions

The web server does not maintain information about the client, so it cannot determine when another request comes from the same client. This inability to track clients and their navigation through a website makes it difficult to do any complex transactions on a website. A session object can be used to track a user throughout the entire interaction with the web server.

A session object gives you a single location to store and retrieve information throughout a user's connection to your website.

You can access session information in servlets and JSPs:

- In a Java Servlet, use the javax.servlet.http.HttpSession object.
- In JSPs, use the implicit JSP session object.

Handling session:

```
HttpSession thisSession = req.getSession(true); // Or
HttpSession thisSession = req.getSession();

// Setting an Attribute in Session
thisSession.setAttribute("name", thisName);

// getting an Attribute from a session (from anywhere where session is available
String userName = (String)thisSession.getAttribute("name");
```

ServletContext

Interface **ServletContext** defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

There is one context per "web application" per Java Virtual Machine.

The **ServletContext** object is contained within the **ServletConfig** object, which the Web server provides the servlet when the servlet is initialized.

```
// getting context in Servlet
ServletContext context = getServletContext();
// Setting Attribute in context
context.setAttribute("DBName", "MySQL");
// Setting Attribute in context from Any Servlet later
context.getAttribute("DBName");
```

ServletConfig

ServletConfig is a servlet configuration object used by a servlet container to pass information to a servlet during initialization.

ServletConfig as four important methods which we will need to get initialization parameters.

Method	Description
String getInitParameter(String name)	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
Enumeration getInitParameterNames()	Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
ServletContext getServletContext()	Returns a reference to the ServletContext in which the caller is executing.
String getServletName()	Returns the name of this servlet instance.

```
// getting Servlet Config
ServletConfig config = getServletConfig();
// getting init parameter
config.getInitParameter("DBName");
```

Setting Init Parameter

```
<servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>com.dashboard.services.MyServlet</servlet-class>
        <init-param>
          <param-name>DBName</param-name>
          <param-value>MySQL</param-value>
        </init-param>
</servlet>
```


Servlets Filters

Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:

- To intercept requests from a client before they access a resource at back end.
- To manipulate responses from server before they are sent back to the client.

Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor.

```
<filter>
  <filter-name>ShowHeaderFilter</filter-name>
  <filter-class>ShowHeaderFilter</filter-class>
</filter>
<filter-mapping>
```

```
  <filter-name>ShowHeaderFilter</filter-name>
  <url-pattern>/ShowHeader</url-pattern>
</filter-mapping>
```

A filter is simply a Java class that implements the javax.servlet.Filter interface. The javax.servlet.Filter interface defines three methods:

Method	Description
public void doFilter (ServletRequest, ServletResponse, FilterChain)	The doFilter method can examine and modify the data and headers of the request and response objects. This method uses the FilterChain object to invoke the next filter in the chain.
public void init(FilterConfig filterConfig)	Init() is invoked by container only once on first request. FilterConfig object is used to access init-parameters and session details.
public void destroy()	This method is called by the web container to indicate to a filter that it is being taken out of service.

Understand FilterConfig:

Container initializes the filter using the FilterConfig object. The FilterConfig object provides the following methods that let you access the initialization parameters and the ServletContext object:

- getFilterName() – get the name of current Filter
- getInitParameter() – read init parameter from Deployment Descriptor
- getInitParameterNames() – read init parameter names
- getServletContext() – returns servlet context object

Understanding the FilterChain

Container passes the **FilterChain** object into the filter's **doFilter** method. Each filter passes control to the next filter or to the target resource, but eventually regains control when the downstream filters finish processing and pass control back up the chain. You can think of using multiple filters in a chain as winding and unwinding a stack.

As each filter finishes processing the request, it returns control to the chain. From there, processing can occur in the next filter (if there is one), the previous filter, or at the target resource.

Filters can include logic to determine when or whether to return control to the chain. If the last filter calls the chain and there are no more filters in the chain, then the request is passed to the target resource.

If a filter's logic prevents it from returning control to the chain, the chain is broken and the response is sent back through the filter chain and to the client. The remaining downstream filters (if any) in the chain are ignored.

Examples 3: Log IP Address Filter

```
public class LogIPFilter implements Filter {

	@Override
	public void init(FilterConfig config) throws ServletException {
		// Get init parameter
		String testParam = config.getInitParameter("test-param");

		//Print the init parameter
		System.out.println("Test Param: " + testParam);

	 }


	@Override
	public void doFilter(ServletRequest request, ServletResponse response,
			FilterChain chain) throws IOException, ServletException {
		  // Get the IP address of client machine.
		String ipAddress = request.getRemoteAddr();

		// Log the IP address and current timestamp.
		System.out.println("IP "+ ipAddress + ", Time "
					+ new Date().toString());

		// Pass request back down the filter chain
		chain.doFilter(request,response);

	}

	@Override
	public void destroy() {
		 /* Called before the Filter instance is removed
		from service by the web container*/
		 System.out.println("Calling Destroy");

	}


}
```

Servlets - Session Tracking

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server:

A. Cookies:

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Setting and getting cookies is a very easy process but this method has browser dependency. Many browser settings do not allow cookies at all.

Writing a Cookie:

```
// Create cookies for first and last names.
Cookie firstName = new Cookie("first_name", “Rogger”);
Cookie lastName = new Cookie("last_name", “Narayan”);

// Set expiry date after 24 Hrs for both the cookies.
firstName.setMaxAge(60*60*24);
lastName.setMaxAge(60*60*24);

// Add both the cookies in the response header.
response.addCookie( firstName );
response.addCookie( lastName );
```


Reading a Cookie:

```
Cookie[] cookies = request.getCookies();
```

B. SessionId field in HTML

A web server can send a hidden HTML form field along with a unique session ID as follows:

```
<input type="hidden" name="sessionid" value="12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends

request back, then session_id value can be used to keep the track of different web browsers.

This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

C. URL Rewriting:

You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. For example, with http://www.h2kinfosys.com/syllabus.htm;sessionid=12345, the session identifier is attached as sessionid=12345 which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies but here drawback is that you would have generate every URL dynamically to assign a session ID.

HttpSession:

HttpSession Interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

You can get the Session object from request.

```
HttpSession session = request.getSession();
```

Here is a summary of the important methods available through HttpSession object:

Method	Description
public Object getAttribute(String name)	This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
public Enumeration getAttributeNames()	This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
public long getCreationTime()	This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public int	This method returns the maximum time interval, in seconds, that the servlet container will keep this

getMaxInactiveInterval()	session open between client accesses.
public void invalidate()	This method invalidates this session and unbinds any objects bound to it.
public boolean isNew()	This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
public void removeAttribute(String name)	This method removes the object bound with the specified name from this session.
public void setAttribute(String name, Object value)	This method binds an object to this session, using the name specified.
public void setMaxInactiveInterval(int interval)	This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Setting Session Timeout:

Two ways to achieve timeout for session:

1. With session.setMaxInactiveInterval(600) // in seconds
2. With Web.xml Entry: (in Minutes)

```
<session-config>
   <session-timeout>10</session-timeout>
 </session-config>
```


Event Listeners

The servlet specification includes the capability to track key events in your Web applications through event listeners.

There are two levels of servlet events:

Level	Description
Servlet context-level (application-level) event	This event involves resources or state held at the level of the application servlet context object.
Session-level event	This event involves resources or state associated with the series of requests from a single user session; that is, associated with the HTTP session object.
Servlet Request event	Life Cycle and Events of Servlet Request

Each of these two levels has two event categories:

- Lifecycle changes
- Attribute changes

You can create one or more event listener classes for each of the four event categories. A single listener class can monitor multiple event categories.

Event Listener Categories and Interfaces:

Category	Descriptions	Interface
Servlet context lifecycle changes	Servlet context creation, at which point the first request can be serviced Imminent shutdown of the servlet context	ServletContextListener
Servlet context attribute changes	Addition of servlet context attributes Removal of servlet context attributes Change of servlet context attributes	ServletContextAttributeListener
Session lifecycle changes	Session creation Session invalidation Session timeout	HttpSessionListener
Session attribute changes	Addition of session attributes Removal of session attributes Replacement of session attributes	HttpSessionAttributeListener
Request lifecycle changes	Request initialized Request Destroyed	ServletRequestListener
Request attribute changes	Addition of Request attributes Removal of Request attributes Replacement of Request attributes	ServletRequestAttributeListener

Be aware of the following rules and guidelines for event listener classes:

- In a multithreaded application, attribute changes may occur simultaneously. There is no requirement for the servlet container to synchronize the resulting notifications; the listener classes themselves are responsible for maintaining data integrity in such a situation.
- Each listener class must have a public zero-argument constructor.

Defining Filters in web.xml:

```
<web-app>
.
.
<listener>
    <listener-class>com.h2k.test.ServletContextListenerTest</listener-class>
  </listener>
 .
</web-app>
```

Writing Listeners:

********** **ServletContextListener** **********

```
/**
 * Application Lifecycle Listener implementation class ServletContextListenerTest
 */
public class ServletContextListenerTest implements ServletContextListener {
    /**
     * Default constructor - Its Mandatory to have one.
     */
    public ServletContextListenerTest() {
        // Even if you don't have to write anything here, just have an empty
Constructor.
    }
      /**
       * Called with Servlet Context gets Initialized.
     * @patam ServletContextEvent - Event class gives you access to Servlet Context
     * It has getServletContext() method to access it.
     */
    public void contextInitialized(ServletContextEvent event) {
      // Do something with context gets Initialized
    }
      /**
       * Called with Servlet Context gets Destroyed.
       */
    public void contextDestroyed(ServletContextEvent arg0) {
      // Do something with context gets Destroyed
    }

}
```

********** **ServletContextAttributeListener** **********

```
/**
*  ServletContextAttributeListener implementation
 *
 */
public class ServletContextAttributeListenerTest implements
ServletContextAttributeListener {
      /**
     * Default constructor - Its Mandatory to have one.
     */
    public ServletContextAttributeListenerTest() {
      // Even if you don't have to write anything here, just have an empty
Constructor.
    }
/**
 * attributeAdded -  When New attribute gets added
 * ServletContextAttributeEvent has two methods to get details about Attribute
 * String getName() - Use method this to get the name of the attribute that was
added, removed, or replaced.
* Object getValue() - Use this method to get the value of the attribute that was
added, removed, or replaced.
*/
    public void attributeAdded(ServletContextAttributeEvent event) {
        // Do something
    }
/**
  * When existing attribute is replaced / changed
  */
    public void attributeReplaced(ServletContextAttributeEvent event) {
        // Do Something
    }
/**
  * When existing attribute is removed
  */
    public void attributeRemoved(ServletContextAttributeEvent event) {
      // Do Something
    }

}
```

********** **HttpSessionListener** **********

```
/**
 * Application Lifecycle Listener implementation class HttpSessionListenerTest
 *
 */
public class HttpSessionListenerTest implements HttpSessionListener {

      /**
     * Default constructor - Its Mandatory to have one.
     */
    public HttpSessionListenerTest() {
      // Even if you don't have to write anything here, just have an empty
Constructor.
    }

      /**
     * When session is created.
     * HttpSessionEvent - gives access to Session with getSession() method
     */
    public void sessionCreated(HttpSessionEvent event) {
       // Do something
    }

   /**
     * When Sesison gets Destroyed
     */
    public void sessionDestroyed(HttpSessionEvent event) {
        // Do something
    }

}
```

********** **HttpSessionAttributeListener** **********

```
/**
 * Application Lifecycle Listener implementation class
HttpSessionAttributeListenerTest
 *
 */
public class HttpSessionAttributeListenerTest implements HttpSessionAttributeListener
{
       /**
     * Default constructor - Its Mandatory to have one.
     */
    public HttpSessionAttributeListenerTest() {
      // Even if you don't have to write anything here, just have an empty
Constructor.
    }

       /**
     * When Attribute is remove.
     * HttpSessionBindingEvent provides getName(), getValue() and getSession()
methods.
     */
    public void attributeRemoved(HttpSessionBindingEvent event) {
        // Do Something
    }

       /**
     * When new attribute gets added
     */
    public void attributeAdded(HttpSessionBindingEvent event) {
        // Do Something
    }

       /**
     * when attribute gets changed or replaced
     */
    public void attributeReplaced(HttpSessionBindingEvent event) {
      // Do Something
    }


}
```

********** **HttpSessionActivationListener** **********

```
/**
 * Objects that are bound to a session may listen to container events notifying them
* that sessions will be passivated and that session will be activated. A container
* that migrates session between VMs or persists sessions is required to notify all
* attributes bound to sessions implementing HttpSessionActivationListener.
 *
 */
public class HttpSessionActivationListenerTest implements
HttpSessionActivationListener {

    /**
     * Default constructor.
     */
    public HttpSessionActivationListenerTest() {
        // TODO Auto-generated constructor stub
    }

      /**
     * Notification that the session has just been activated.
     */
    public void sessionDidActivate(HttpSessionEvent event) {
        // TODO Auto-generated method stub
    }

      /**
     *Notification that the session is about to be passivated.
     */
    public void sessionWillPassivate(HttpSessionEvent event) {
        // TODO Auto-generated method stub
    }

}
```

********** **HttpSessionBindingListener** **********

```
/**
 * Causes an object to be notified when it is bound to or unbound from a session.
 * The object is notified by an HttpSessionBindingEvent object. This may be as a
 * result of a servlet programmer explicitly unbinding an attribute from a session,
 * due to a session being invalidated, or due to a session timing out.
 */
public class HttpSessionBindingListenerTest implements HttpSessionBindingListener {

    /**
     * Default constructor.
     */
    public HttpSessionBindingListenerTest() {
        // TODO Auto-generated constructor stub
    }

      /**
     * Notifies the object that it is being bound to a session and identifies the
session.
     */
    public void valueUnbound(HttpSessionBindingEvent event) {
        // TODO Auto-generated method stub
    }

      /**
     *Notifies the object that it is being unbound from a session and identifies the
session
     */
    public void valueBound(HttpSessionBindingEvent event) {
        // TODO Auto-generated method stub
    }

}
```

********** **ServletRequestListener** **********

```
/**
 * Way for receiving notification events about requests coming
 * into and going out of scope of a web application.
 *
 */
public class ServletRequestListenerTest implements ServletRequestListener {

    /**
     * Default constructor.
     */
    public ServletRequestListenerTest() {
        // TODO Auto-generated constructor stub
    }

/**
*  Receives notification that a ServletRequest is about to go out of scope of the web
application.
*  A ServletRequest is defined as going out of scope as it exits the last servlet or
*  the filter in the chain.
*/
public void requestDestroyed(ServletRequestEvent event) {
    // TODO Auto-generated method stub
}

/**
  * Receives notification that a ServletRequest is about to come into scope of the
web application.
  * A ServletRequest is defined as coming into scope of a web application when it is
about to enter
  * the first servlet or filter of the web application
  */
 public void requestInitialized(ServletRequestEvent event) {
        // TODO Auto-generated method stub
 }

}
```

********** **ServletRequestAttributeListener** **********

```
/**
 *  ServletRequestAttributeListener Interface for receiving notification events
 *  about ServletRequest attribute changes.
 */
public class ServletRequestAttributeListenerTest implements
ServletRequestAttributeListener {

    /**
     * Default constructor.
     */
    public ServletRequestAttributeListenerTest() {
        // TODO Auto-generated constructor stub
    }

      /**
     *  Receives notification that an attribute has been added to the ServletRequest.
     *  ServletRequestAttributeEvent - provides getName() and getValue() for
attribute
     *  for which event is generated
     */
    public void attributeAdded(ServletRequestAttributeEvent event) {
        // TODO Auto-generated method stub
    }

      /**
     * Receives notification that an attribute has been removed from the
ServletRequest.
     */
    public void attributeRemoved(ServletRequestAttributeEvent event) {
        // TODO Auto-generated method stub
    }

      /**
     *Receives notification that an attribute has been replaced on the
ServletRequest.
     */
    public void attributeReplaced(ServletRequestAttributeEvent event) {
        // TODO Auto-generated method stub
    }


}
```