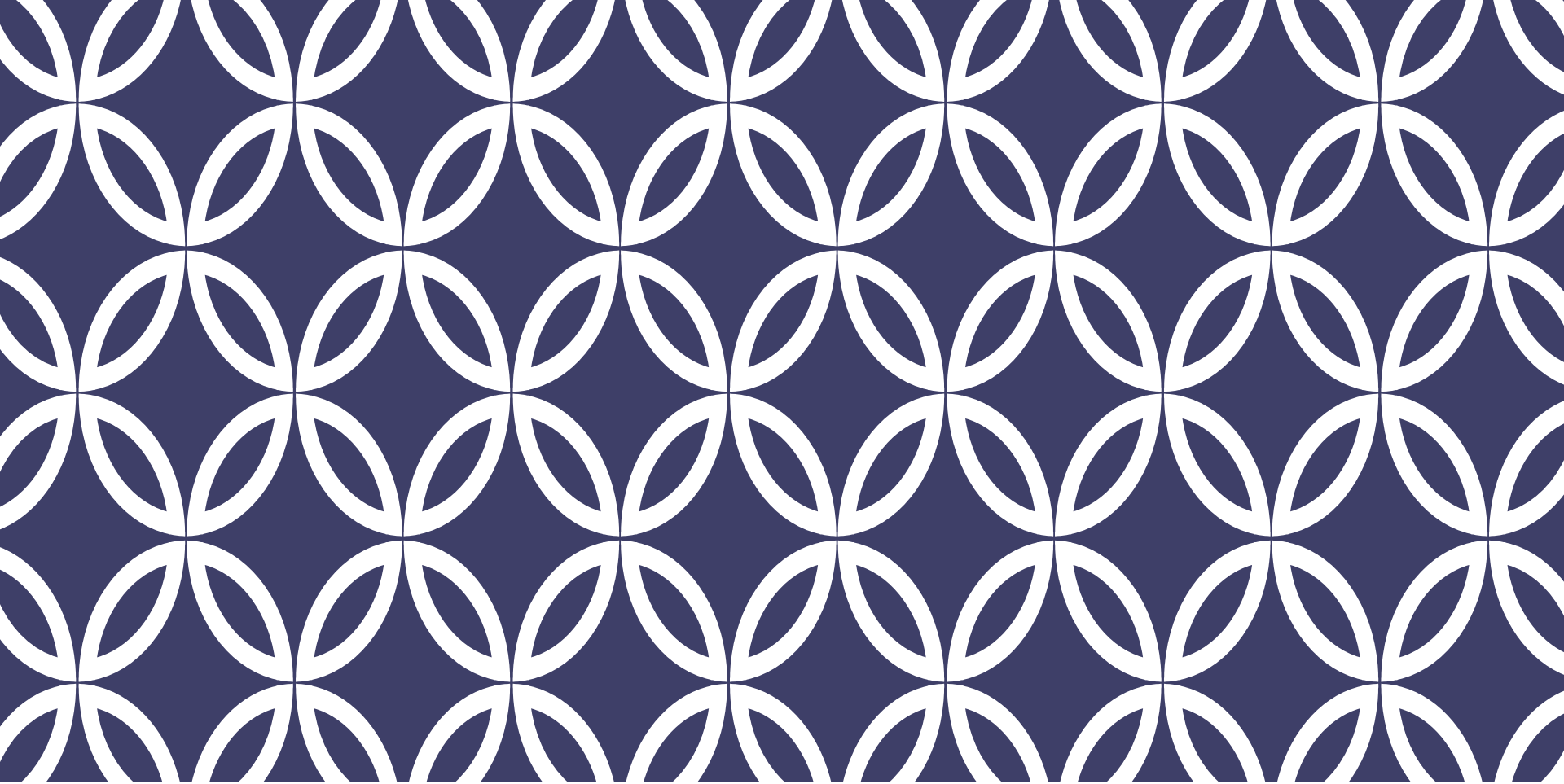
ACADGILD

Presents

Introduction to Advance Java J2EE



Session 20 – Servlets

Agenda – Servlets

1. HTTP Headers
2. Request Dispatching
3. Servlet Filter
4. Events & Listeners

HTTP Headers

- Servlet can set HTTP headers into it just to provide extra information about its response

- For setting HTTP headers in Servlets, **setHeader** method is used.

- Few of HTTP headers that are most often set by Servlets are as follows:
  - **Cache-Control**: tells caching system how to treat a document
  - **Pragma**: same as cache-control header
  - **Connection**: indicates willingness to maintain connection with client
  - **Retry-After**: specifies when server can again handle request
  - **Expires**: shows when information becomes invalid
  - **Location**: specifies new location of document
  - **WWW-Authenticate**: indicates authorization scheme
  - **Content-Encoding**: specifies scheme used to encode response body

- Example, response.setHeader(“cache-control”, “no-cache”);

Request Dispatching

- As name suggests, it dispatches (redirects) requests coming from client to different locations or pages in web applications
- Interface named **RequestDispatcher** is one which is used for this purpose
- This interface has got some methods in it who actually sends requests from client to another resource
- Another resource can be HTML page, JSP or Servlet
- We can also include content of another resource to response
- **RequestDispatcher** enables Servlet communication within web applications
- There are two methods defined in this interface Forward Include
  - forward method
  - include method

Request Dispatching

- **Forward() method**:
  - **Method signature**
  - void forward(ServletRequest request, ServletResponse response)
  - forwards request to another resource on the server
- **Include() method:**
  - **Method signature**
  - void include(ServletRequest request, ServletResponse response)
  - includes the content of resource (servlet, JSP page, HTML file) in the response
- To use these methods for redirecting requests, we must get RequestDispatcher
- We can get RequestDispatcher by calling getRequestDispatcher() method

Request Dispatching

- **Forward() method**:
  - **Method signature**
  - void forward(ServletRequest request, ServletResponse response)
  - forwards request to another resource on the server
- **Include() method:**
  - **Method signature**
  - void include(ServletRequest request, ServletResponse response)
  - includes the content of resource (servlet, JSP page, HTML file) in the response
- To use these methods for redirecting requests, we must get RequestDispatcher
- We can get RequestDispatcher by calling getRequestDispatcher() method

Request Dispatching Example

Code snippet to redirect control to another resource

```
RequestDispatcher dispatcher =
request.getRequestDispatcher("ServletTwo");
// ServletTwo is the url-pattern of the second Servlet

dispatcher.forward(request, response);
// Forwards from current Servlet to second Servlet
```

Servlet Filters

- A **filter** is an object that is invoked at pre processing & post processing of request
- It is mainly used to perform filtering tasks such as
  - authentications
  - Conversions or formatting body
  - logging
  - compression
  - encryption & decryption
  - input validation
- The Servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the Servlet.
- We can have multiple filters for a single resource and we can create a chain of filters for a single resource
- We can create our own Servlet Filter by implementing **javax.servlet.Filter** interface

Servlet Filter Life Cycle

- **void init(FilterConfig paramFilterConfig)** – When container initializes Filter, this method is invoked. This method is called only once in the lifecycle of filter & we should initialize any resources in this method. FilterConfig is used by container to provide init parameters & Servlet context object to Filter. We can throw ServletException in this method

- **doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain)** – This method is invoked every time by container when it has to apply filter to a resource. Container provides request & response object references to filter as argument. FilterChain is used to invoke the next filter in the chain. This is a great example of Chain of Responsibility Pattern.

- **void destroy()** – When container offloads the Filter instance, it invokes the destroy() method. This is the method where we can close any resources opened by filter. This method is called only once in the lifetime of filter.

Events & Listeners

- In simple terms, **event is occurrence of something.**
- In world of web application, an event can be **initialization** of application, destroying an application, **request from client**, creating/destroying a session, attribute **modification** in session etc.
- For example, consider an application which has single entry point i.e. **user login**.

  We can do login process in first Servlet request but if we have multiple entry points then doing user login process everywhere will result in code redundancy.
- In above scenario we can create a listener for the application start up **event**.

  So here, starting up an application is an event & entity who will catch it is **listener**
- Servlet API provides different kind of listeners for different types of Events.

Events provided by Servlet API

- **javax.servlet.AsyncEvent** – Event that gets fired when asynchronous operation initiated on a ServletRequest has completed, timed out, or produced an error

- **javax.servlet.http.HttpSessionBindingEvent** – Events of this type are sent to an object when it is bound or unbound from a session

- **javax.servlet.http.HttpSessionEvent** – This is the class representing event notifications for changes to sessions within a web application.

- **javax.servlet.ServletContextAttributeEvent** – Event class for notifications about changes to the attributes of the ServletContext of a web application.

- **javax.servlet.ServletContextEvent** – This is the event class for notifications about changes to the servlet context of a web application.

Listeners provided by Servlet API

- **javax.servlet.AsyncListener** – Listener that will be notified in the event that an asynchronous operation initiated on a ServletRequest to which the listener had been added has completed, timed out, or resulted in an error.

- **javax.servlet.ServletContextListener** – Interface for receiving notification events about ServletContext lifecycle changes.

- **javax.servlet.ServletContextAttributeListener** – Interface for receiving notification events about ServletContext attribute changes.

- **javax.servlet.ServletRequestListener** – Interface for receiving notification events about requests coming into and going out of scope of a web application.

- **javax.servlet.ServletRequestAttributeListener** – Interface for receiving notification events about ServletRequest attribute changes.

Lets Discuss Assignments