Ch-03 Web Applications

1. What is ServletContext?
    a. ServletContext is an interface that defines a set of methods that helps us to communicate with the servlet container. There is one context per "web application" per JVM and all the components of the web application can share it.
    b. We can use the deployment descriptor to set up initialization parameters at the ServletContext level which can then be shared by all components of the web application.
2. Who will create the ServletContext object for a web application?
    a. The ServletContext is created by the container when the web application is deployed.
3. When and how the ServletContext will be made available to a servlet by the container?
    a. The ServletContext is created by the container when the web application is deployed and after that only the context is available to each servlet in the web application.
    b. The ServletContext object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.
4. The ……. is the closest thing our application has to an object that represents the web application itself.
    a. ServletContext
5. How can we declare ServletContext parameters in the deployment descriptor?
    a. Each initialization parameter is declared in the <context-param> sub element of the root element <web-app>. The <context-param> element contains mandatory <param-name> and < param-value> sub elements which hold the name and value of the parameters respectively.
    b. Example:
        i. <web-app>
        ii. <context-param>
        iii. <param-name>machineName</param-name>
        iv. <param-value>ABCDEF</param-value>
        v. </context-param>

vi. <context-param>
vii. <param-name>secretParameterFile</param-name>
viii. <param-value>/WEB-INF/xml/secretParms.xml</param-value>
ix. </context-param>
x. </web-app>

6. How can you retrieve ServletContext initialization parameters from your code?
    a. We can use the below two methods of ServletContext:
        i. getInitParameter(String paramName) – will return a String or null.
        ii. getInitParameterNames() – will always return an enumeration of names.
7. Write a code snippet inside a servlet that gets a ServletContext object and use it to print all ServletContext initialization parameters and values?
    a. PrintWriter pw = response.getWriter();
    b. ServletContext sc = getServletContext();
    c. Enumeration<String> e = sc.**getInitParameterNames**();
    d. while(e.hasMoreElements()){
    e. String paramName = (String)e.nextElement();
    f. pw.write("Param="+paramName+"Val="+sc.**getInitParameter**(paramName));
    g. }
8. Consider the scenario - Parameters are put in a file and use a single ServletContext parameter to hold its location rather than define all parameters as ServletContext initialization parameters. Give any advantages of this scenario?
    a. A change to a servlet context parameter necessitates redeploying the DD and restarting the web application.
    b. However by defining the parameters in a file, we can read the file again from the same location to reload the values without restarting the application.
9. What is the major difference between parameters and attributes?
    a. Parameters come into our application from the client request or DD elements and are read only.
    b. Attributes can be read, created, updated and deleted. Web container as well as our application code can set attributes.

10. Which are the different scopes for an attribute?
    a. Request
    b. Session
    c. Application (or Context)
    d. Page
11. List down the attribute manipulation methods for request, session and application scopes?
    a. The attribute manipulation methods for request, session and application scopes are identical and differ only on the interfaces they are defined. They are:
        i. public void **setAttribute**(String name, Object value)
        ii. public Object **getAttribut**(String name)
        iii. public Enumeration **getAttributeNames**()
        iv. public void **removeAttribute**(String name)
    b. The interfaces on which they are defined are:
        i. **Request** – javax.servlet.**ServletRequest**
        ii. **Session** – javax.servlet.http.**HttpSession**
        iii. **Application** – javax.servlet.**ServletContext**
12. What will happen if we pass null for the value in **setAttribute**?
    a. If the object passed as a value is null, it has the same effect as removeAttribute().
13. What will happen if any of the attribute manipulation functions are invoked in session scope when the session is invalid?
    a. IllegalStateException will be thrown.
14. What will happen if the attribute doesn't exist while calling getAttribute() function?
    a. Null is returned if no attribute of the given name exist.
15. What will happen if we call getAttributeNames() when no attributes are there in various scopes?
    a. In request and session scopes, an empty enumeration will be retuned if no attribute exist.
    b. Since some attributes will be always supplied by the web container to the context, the enumeration returned by getAttributeNames() will never be empty in the application scope.

16. What will be the order of the attributes within the enumeration returned by the getAttributeNames() method.
    a. There is no guarantee of the order.
17. What do you mean by scope?
    a. Scope describes the lifetime and availability of web application artifacts, especially attributes. Different scopes are request, session, application and page.
18. What is the lifetime of request scope in an http environment?
    a. Request scope last from the moment as HTTP request hits a servlet in our web container to the moment the servlet is done with delivering the HTTP response. With respect to code, request scope begins on entry to a servlet's service() method and ends on the exit from that method.
19. In an HTTP environment request scope is represented by which object?
    a. HttpServletRequest
20. A request scope is represented by which interface?
    a. ServletRequest
21. Is the request scope thread safe?
    a. Yes. Request scope is bound to a single request thread. So it is thread-safe.
22. What is the lifetime of a session scope object?
    a. A session scope starts when a browser window establishes connection with our web application till the point where the browser window is closed. It can span multiple requests from the same client.
23. Session scope is represented by which object?
    a. HttpSession object.
24. How can you get access to the session object from your servlet code?
    a. Using getSession() method of the HttpServletRequest object.
        i. HttpSession session = request.getSession();
25. What are the non-http equivalents of session scope?
    a. There are no non-http equivalents for the session scope.
26. What is the lifetime of the context scope?
    a. Context scope starts from the point where a web application is put into service till it is removed from service or the web application is reloaded.

27. What will happen to the context scope if the web application is reloaded?
    a. The context scope is destroyed and recreated.
28. Are context attributes thread-safe?
    a. No. Every thread in our application can access them.
29. Context scope is represented by which object?
    a. An implementation of the ServletContext interface.
30. In a distributed application, will the ServletContext object in different clones, be same or different?
    a. Different. The web container provides one ServletContext object per web application per jvm.
31. If information is not needed beyond one production of an Http response, which is the most, suited scope for this information?
    a. Request
32. You have an xml file that stores essential information about how your application works, and only changes when the web application is redeployed? Which is the most, suited scope for this?
    a. Context
33. We have information spanning across multiple requests from the same client. Which is the most, suited scope for this?
    a. Session
34. Are session attributes thread-safe?
    a. Session attributes are not fully thread safe as they can be accessed by more than one thread from the same client. However in most cases, they are thread-safe.
35. List any two approaches to solve the multithreading problems with context attributes?
    a. Like ServletContext parameters, treat them as read only. Set up the attributes in the init() method of the servlet and use only the getAttribute() method after that.
    b. Surround the ServletContext attribute updates with synchronization blocks. To make sure no other thread reads the value of these attributes mid-update, we will need to synchronize the getAttribute() calls as well.

36. What is a SingleThreadModel?
    a. If a servlet implement this interface, the web container would guarantee that only one thread at a time would be able to access any given instance of the servlet.
    b. However this interface is deprecated in the version 2.4 of the servlet specification.
37. What is request dispatching?
    a. Dispatching is a means of delegating control from one web resource to another. We use a RequestDispatcher to represent the web resource to which we want to delegate.
38. How can you obtain a RequestDispatcher object?
    a. A RequestDispatcher can be obtained from either ServletRequest or ServletContext:
        i. From ServletRequest
            1. **getRequestDispatcher**(String path) – Will accept a path relative to the context root (starting with '/') or a path relative to the current resource.
                a. **Example:** RequestDispatcher **rd** = request.getRequestDispatcher("/servlet/ServletA");
        ii. From ServletContext
            1. **getRequestDispatcher**(String path) – Only full paths (starting with '/') are allowed.
            2. **getNamedDispatcher**(String name) – Name must match one of the <servlet-name> values we have set up in the DD so that it can refer to a named servlet or JSP.
39. Can we specify a path outside the context to which RequestDispatcher belongs while getting RequestDispatcher from ServletRequest or ServletContext?
    a. No
40. Can we specify relative paths in **getRequestDispatcher**(String path) method of ServletContext?
    a. No. It will throw IllegalArgumentException.

41. We have a servlet set up in DD without <servlet-mapping>. Give any way of accessing the servlet and any advantages with that approach?
    a. We can use **getNamedDispatcher()** method of ServletContext to get that servlet or jsp. We can thus avoid public url access to some resources and provide controlled access based on situations.
42. What will happen if an incorrect url is fed to the RequestDispatcher methods?
    a. The method returns null. No exception will be thrown.
43. Can we obtain the RequestDispatcher for other web applications? If yes, how?
    a. We can obtain RequestDispatcher for other web applications by obtaining another web application's context and getting a RequestDispatcher from that.
    b. However our web containers default security settings may prevent us from this by causing request dispatchers returned from other contexts to be null.
44. After obtaining a RequestDispatcher for a servlet what should we do?
    a. Having obtained a RequestDispatcher for a servlet, we can either call the forward() or include() method on it passing the ServletRequest and ServletResponse objects.
45. What is the major difference between a forward() and include() method of a RequestDispatcher?
    a. The **forward**() method passes the responsibility to another web resource. The response **will come entirely from the target of the forward()**. Any work that the forwarding servlet has done on the response will be discarded.
    b. The **include**() method includes the output of the including web resource inside the including servlet. On the return from the include() call, the including servlet can still add more to the response.
    c. If we call the getXXX() methods of the request object from the **forwarded-to** servlet, it will return the values of forwarded-to servlet. This is because a forwarded-to servlet has complete control over the request.
    d. If we call the getXXX() methods of the request object from the **included** servlet, it will return the values of including servlet and NOT the included servlet.
46. What will happen if we call forward() or include() after committing the response in a servlet?

a. If the response is already committed we won't be able to forward it. Doing so will give IllegalStateException.
b. However a call to include will still work even if the response is already committed.

47. Give one way to commit a response manually?
    a. We can use the flush() method of PrintWriter object.
48. Will the code written after the forward() call in a page executed?
    a. The code will be executed, but nothing will be printed to the response. So that code unrelated to the response such as outputting to console, setting attributes etc can be done without any issues.
49. If we call getXXX() methods of the request object from within a forwarded servlet, what values will it print – Those of the forwarded-to servlet itself or the forwarding servlet?
    a. Forwarded-to servlet
    b. This is because a forwarded-to servlet has complete control over the request—it's as if the forwarding servlet had never been called.
50. If we call getXXX() methods of the request object from within an included servlet, what values will it print – Those of the included servlet itself or the including servlet?
    a. Including servlet
    b. An including servlet takes the contents of the included web resource and adds this to its own response. Here including servlet doesn't completely give the control to the included servlet.
51. Since the getXXX() method calls on request object within a forwarded-to servlet return the forwarded-to request details, how can you get the request details of the forwarding servlet?
    a. The web container set up five special attributes that reflect "original" values about the request path, instead of the request path, which has been modified to fit the forwarded to servlet. We can use the request.getAttribute() method to get these values.
        i. javax.servlet.forward.request_uri
        ii. javax.servlet.forward.context_path
        iii. javax.servlet.forward.servlet_path

iv. java.servlet.forward.path_info
v. java.servlet.forward.query_string

52. Since the getXXX() method calls on request object within a included servlet return the including request details, how can you get the request details of the included servlet?
    a. The web container set up five special attributes that reflect values about the included servlet request path, instead of the including request path.
        i. javax.servlet.include.request_uri
        ii. javax.servlet. include.context_path
        iii. javax.servlet. include.servlet_path
        iv. java.servlet. include.path_info
        v. java.servlet. include.query_string
53. What will be the value of javax.servlet.include.request_uri attribute from within the forwarding and including servlet?
    a. null
54. Will the special attributes like javax.servlet.include.request_uri always be set while forwarding or including using RequestDispatcher?
    a. Not always. When we use getNamedDispatcher() on ServletContext, special attributes are not set. This is because we are not forwarding or including via an external url request. Because all these special attributes pertain to features of external requests (mostly URL information), they are not deemed relevant to an internal server call to a named resource.
55. What are filters?
    a. Filters are used for preprocessing requests or post processing responses before they reach a target resource in a web application. They receive request and response, which they can manipulate. They also have an inclusion mechanism whereby a filter can pass control to another filter or servlet.
56. Do filters have access to ServletContext?
    a. Yes

57. Give few common uses of filters?
    a. Authentication, logging, data compression, caching.
58. To write a filter we need to implement which interface?
    a. javax.servlet.Filter
59. List down the methods we need to implement while implementing javax.servlet.Filter interface?
    a. public void **init**(FilterConfig config) throws ServletException
    b. public void **doFilter**(ServletREquest request, ServletResponse response, FilterChain chain) throws IOException, ServletException
    c. public void **destroy**()
60. When is a Filter's init() method called?
    a. A Filter's init() method is called only once, at some point between server startup and before the first request is intercepted by the filters.
61. Which is the only place in a Filter life cycle where we can capture the FilterConfig object and keep it available for later use?
    a. The init() method of a filter.
62. List down the importance methods of FilterConfig which are useful for a web developer?
    a. getFilterNames()
    b. getInitParameter(String name)
    c. getInitParameterNames()
    d. getServletContext()
63. When is a filter's doFilter() method called?
    a. A filter's doFilter() method is called by the web container whenever it intercepts an appropriate request for the filter.
64. What is a FilterChain?
    a. The sequence of filters that needs to be executed before a request reaching a resource, together with the requested resource forms a chain known as FilterChain.
65. What is a FilterChain object?
    a. The FilterChain object represents the next thing in the FilterChain. The next thing can be another filter or the web resource ultimately requested.

b. The FilterChain object allows a filter to pass control or deny access to other filters and web resources. The FilterChain object which is passed inside the doFilter() method of a filter had a method doFilter() which accepts only request and response. If we need the request to pass through the chain, we need to call the doFilter() method.

66. How can you deny request to a resource from the filter?
    a. The FilterChain object allows a filter to pass control or deny access to other filters and web resources. If we don't call the doFilter() method on the FilterChain object passed inside the doFilter() method of a filter, the chain is broken and access will be denied.
67. List down the important things we can do in a Filter's doFilter() method?
    a. We can do many things, which are not limited to:
        i. Look into the request
        ii. Wrapper the request and response objects if required
        iii. Add or change things about the request through the wrapper
        iv. Call the next filter or servlet in the chain using doFilter() on the FilterChain object passed in as a parameter.
        v. Block the request by not calling the FilterChain's doFilter() method.
        vi. On the return from the FilterChain's doFilter() method (or ven if it wasn't called), amend the response headers or content through the wrapper.
68. When will a Filter's destroy() method be called?
    a. A Filter's destroy() method is called only once, at some point after the last filter request is processed in the doFilter() method, and before the web application closes.
69. Describe the <filter> element in DD?
    a. Filters are declared in the DD using the <filter> element. The first three elements <description>, <display-name> and <icon> are optional. Next two elements which are also mandatory are <filter-name> and <filter-class>. The <filter-name> gives a logical name to the filter and <filter-class> specifies the fully qualified name of our filter class. After that there can be as many <init-param> elements as we want to set up initialization parameters in filters.

70. Describe the <filter-mapping> element in DD?
    a. The first is **<filter-name>**, which must tie back to a <filter-name> specified in a <filter> element.
    b. Then it can have any number of occurrences of **<url-pattern>** or **<servlet-name>** elements with a condition that at least one of them should be available.
    c. The third optional sub-element, **<dispatcher>** specifies the routes into a web resource on your web application to kick in a filter dependent on one of these routes. Here are the four valid values for the dispatcher element, with a description of the route:
        i. **REQUEST** (the default)—a direct client request for a web resource.
        ii. **FORWARD**—an internal web server request for a web resource via the forward() method on a RequestDispatcher.
        iii. **INCLUDE**—an internal web server request for a web resource via the include() method on a RequestDispatcher.
        iv. **ERROR**—an internal web application request for a resource that has been set up as an <error-page>.
71. What will happen if one request that matches more than one filter mapping?
    a. Given a request that matches more than one filter mapping,
        i. First, all matching filters will run for <filter-mappings> with <url-pattern> matches, in order of <filter-mapping> declaration.
        ii. Second, all matching filters will run for <filter-mappings> with <servletname> matches, in order of <filter-mapping> declaration.
72. For each <filter> element in the web application deployment descriptor, will multiple instances of a filter be created by the web container?
    a. No (**Need to try out and/or verify**).
73. Consider the below filter mappings and say which filter will be called first if a request for First come as a direct URL request?
        i. <filter-mapping>
        ii. <filter-name>FirstFilter</filter-name>
        iii. <servlet-name>First</servlet-name>
        iv. </filter-mapping>

v. <filter-mapping>
vi. <filter-name>SecondFilter</filter-name>
vii. <url-pattern>/*</url-pattern>
viii. </filter-mapping>

b. SecondFilter will be executed first and then FirstFilter. URL pattern matching will happen first for all matching filters before servlet name matching happens.

74. Give the url pattern within a <filter-mapping> that will catch all requests to a web application?
    a. <url-mapping>/*<url-mapping> or <url-mapping>*<url-mapping>
75. Which of the below will map a request to a servlet, jsp or html will be mapped?
    i. <servlet-name>*</servlet-name>
    ii. <url-mapping>/*<url-mapping>
    iii. <url-mapping>*<url-mapping>
    a. All (Tried out in tomcat).
76. Can we work directly and modify the request object passed in as a parameter to the doFilter() method of a Filter?
    a. No.
77. How can we modify the request or response objects passed in as parameters to the doFilter() method of a Filter?
    a. We cannot work directly and modify the request or response objects passed in. However we can wrapper the request in suitable wrapper objects. We pass real request or response to the constructer of the wrapper class. The wrapper class may override methods in the request or response and add specialized methods of its own.
78. What are the types of the wrapper objects used to wrap request and response in a Filter?
    a. Suitable wrapper objects extend appropriate wrapper classes in javax.servlet:
        i. ServletRequestWrapper
        ii. ServletResponseWrapper
    b. Or corresponding wrapper classes in javax.servlet.http:
        i. HttpServletRequestWrapper
        ii. HttpServletResponseWrapper

79. Write the minimal code to create a wrapper object for request and send it in the doFilter() method?
    a. Class extending HttpServletRequestWrapper
        i. package wrappers;
        ii. import javax.servlet.http.HttpServletRequest;
        iii. import javax.servlet.http.HttpServletRequestWrapper;
        iv. public class MyRequestWrapper extends HttpServletRequestWrapper{
        v. public MyRequestWrapper(HttpServletRequest request) {
        vi. super(request);
        vii. }
        viii. }
    b. doFilter() method
        i. public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        ii. HttpServletRequest httpRequest = (HttpServletRequest)request;
        iii. MyRequestWrapper myRequestWrapper = new MyRequestWrapper(httpRequest);
        iv. chain.doFilter(myRequestWrapper, response);
        v. }
80. A servlet can throw an UnavailableException, which is a subclass of …….
    a. ServletException
81. What is the significance of isPermanent() method of UnavailableException ?
    a. If this returns true, the web container gives up calling this filter; if it returns false the web container will try again after a specified time interval.
82. What determines whether an UnavailableException returns true or false?
    a. It depends on how the Exception is created.
    b. If we use the version of constructor that accepts only a string message, the exception is constructed as permanent.
    c. If we use the two parameter constructor that accepts a string message and an integer seconds, then the web container should deem the exception as temporary and try to call the servlet again after specified number of seconds.

83. What are Servlet chains?
    a. Before filters we could create servlet chains to create a sequence of servlets. However the methods to construct Servlet chains are now deplrecated.
84. Give any advantages of filter chains over servlet chains?
    a. A filter can be shuffled easily by moving entries up or down in a DD.
    b. We can set additional filters later without any programming.
85. Are the context params duplicated across JVMs in a distributed application?
    a. No. There is no mechanism to duplicate context params from one jvm to another in distributed applications. However we can have identical DDs with same parameters set up, which will effectively make the params available in all JVMs.
86. How can you remove request parameters?
    a. We cannot remove request parameters. There are no methods for this.
87. Can session scope span JVMs?
    a. Session scope can span JVMs in a distributed application.
88. Can session scope span web apps?
    a. No
89. Can request scope span web apps?
    a. Yes. Requests can span web apps when a request dispatcher is used from another context.
90. Can request scope span JVMs?
    a. No. There is no mechanism to carry requests across JVMs even in distributed applications
91. Can context scope span JVMs?
    a. No. There is one context per web application per JVM.
92. Using wrappers for request and response is an example of the Decorator pattern. Explain?
    a. While using wrappers for request and response, you take a class, then wrap around it another class, which might mimic, extend, or change the functionality. Hence this is an example of the Decorator pattern.
93. Can HttpServletRequestWrapper modify the header of a request within an object implementing the javax.servlet.Filter interface?

a. Yes. Wrappers can might mimic, extend, or change the functionality of the class that it wraps.

94. Can we use HttpServletRequestWrapper for the request passed to the RequestDispatcher.include method?
    a. An HttpServletRequestWrapper **CAN** be used on the request passed to the RequestDispatcher.include() method. It can be used in any place where an HttpServletRequest or ServletRequest is expected as HttpServletRequestWrapper implements HttpServletRequest which in turn extends ServletRequest.
95. How can we add parameters to the request while using request dispatcher?
    a. We can add parameters to the JSP's URL when generating the request dispatcher and thus pass information in the servlet to the JSP.