Session Management

1. How can I get a session if only one already exist?
    a. HttpServletRequest's getSession() method when called passing false as request.getSession(false) will not create a new session. It returns a session if one already exist. Else it will return null.
2. How can I get the current session and if one not present creates a new one?
    a. HttpServletRequest's getSession() method when called passing true as request.getSession(true) or called without any arguments as request.getSession() will create a new session if one does not exist and return the session if it is already there.
3. What is the type of a session object?
    a. It is javax.servlet.HttpSession
4. What are the non http variants of HttpSession?
    a. There are none.
5. Give any situation where you would use getSession(false)?
    a. Consider that your application is designed so that initial requests should pass through a "Login" servlet, which establishes a session. As a security measure, all other servlets in the application make a getSession(false) call when they need the session object. Only if the user has legitimately passed through the "Login" servlet will the other servlets get the session object they need for the application to function.
6. What do you mean by newness of a session? How can you find whether a session is new?
    a. A newly created session is deemed as new. We can find whether a session is new by using the isNew() method on a HttpSession object.
7. Give two circumstances whereby sessions remain "new."?
    a. The first is when the client doesn't yet know about the session because this is a first request to a web application.
    b. The second is because a client declines to join the session, which it typically does by refusing to return the session tracking information. Under these circumstances, the web application treats each later request from the client as if it were the first, providing a new session each time.

8. What constitutes a “new browser session”?
9. It depends on the browser. Here’s an observation on Internet Explorer’s behavior. If you launch Internet Explorer afresh, then access a session-aware servlet—that’s a new session. If Internet Explorer itself launches a new Internet Explorer window (e.g., by running File | New Window or by running some appropriate script) and that new window accesses a session-aware servlet—it shares the session object with the Internet Explorer window from which it was launched.
10. Can session span across JVMs in distributed application? Explain briefly.
    a. Yes. If the architecture had migrated your session object from one JVM to another, session is replicated from the first running instance of Web App 1 to the second running instance of Web App 1. And with the session go all the objects attaching to the session. The only condition is that the attributes you place in a session should implement the Serializable interface. The exact mechanism by which this is achieved varies from one application server to another.
11. If you use the RequestDispatcher mechanism to get at a servlet in another web application and that servlet accesses the session object, will the session be same?
    a. No. A session is available to be shared between web resources in a single web application. A session cannot cross web application boundaries.
12. How does a session die?
    a. The HttpSession API provides an invalidate() method which we can use to end a session. Invalidation works by making (almost) all of the methods on HttpSession unworkable: If you try to use them, an IllegalStateException is thrown.
    b. If a session has not been used for a prescribed amount of time, the web container invalidates the session itself. Time will be calculated from the point where all requests using the session must have come to an end.
13. What are the different ways in which the session time-out value is controlled?
    a. The time-out value is controlled in one of three ways:
        i. **Application Server Global Default**
            1. Most application servers provide their own mechanism for imposing a global default on session length.
        ii. **Web Application Default**

1. You can set up a default value in the deployment descriptor:
    a. <web-app>
    b. <session-config>
    c. <session-timeout>60</session-timeout>
    d. </session-config>
    e. </web-app>

    The value in the <session-timeout> element is expressed as whole minutes. Any integer value is fine. A value of 0, or a negative value, denotes that the default for sessions created in the web application is to never expire.

iii. Individual Session Setting

1. On obtaining a session, your servlet code can setthe time-out value for that individual session only using the HttpSession.setMaxInactiveInterval(int seconds) API. Here the unit of time is in seconds. Again, a negative value supplied as an argument causes the session to never expire. A value of zero causes the session to expire immediately.

14. Consider that we have HttpSessionBindingListener and it is added as an attribute to the session. We also have a HttpSessionAttributeListener configured. Which all methods of the session event listeners will be called when the session time out?
    a. When a session times out below methods will be called in order if they are applicable for the situation:
        i. HttpSessionListener.**sessionDestroyed**
        ii. HttpSessionBindingListener.**valueUnbound**
        iii. HttpSessionAttributeListener.**attributeRemoved**
15. Consider that we have HttpSessionBindingListener and it is added as an attribute to the session. We also have a HttpSessionAttributeListener configured. Which all methods of the session event listeners will be called when the session is invalidated?
    a. When a session gets invalidated below methods will be called in order if they are applicable for the situation:
        i. HttpSessionListener.**sessionDestroyed**

ii. HttpSessionBindingListener.**valueUnbound**

iii. HttpSessionAttributeListener.**attributeRemoved**

16. What will getMaxInactiveInterval() do?
    a. HttpSession.getMaxInactiveInterval() method returns an int primitive representing the number of **seconds** permitted between client requests before the web container invalidates the related session.
17. List down three methods on HttpSession that don't throw IllegalStateException?
    a. get and setMaxInactive Interval(), and getServletContext()
18. We can get the time the session was created through which session method?
    a. getCreationTime()
19. We can get the last time the client sent a request associated with the session which session method?
    a. getLastAccessedTime()
20. List down two principal methods for session management "offi cially" recognized by the servlet API?
    a. cookie exchange
    b. rewriting URLs
21. What is **Session Management**?
    a. Session management helps you to associate a group of requests. Each of these requests needs to carry a unique ID, which identifies the session to which it belongs. The web application will allocate this unique ID on the first request from the client. The ID must be passed back to the client so that the client can pass it back again with its next request. In this way, the web application will know to which session the request belongs. Session management is basically used to store this information on the server-side between HTTP requests. This implies that the client must also need to store the unique ID somewhere for session management. Various session management mechanisms provide means for this.
22. How are cookies used for session management?
    a. The web application passes session IDs back to a client is via a cookie in the response which is a small text file that the client can store somewhere. **The web application will allocate this unique ID on the first request from the client.**

b. Cookies are used to support all kinds of session-like activity on the Web, regardless of whether the back-end technology is Java based. In the case of **J2EE web applications,** the cookie returned has a standard name—**JSESSIONID**.

23. How can you see the cookie information?
    a. Use HttpServletRequest.getCookies() to get an array of Cookie objects. We can then iterate through the array to see the name and value of cookie objects.
24. In the case of J2EE web applications, the value of which cookie matches the value returned by HttpSession.getId()?
    a. JSESSIONID
    b. HttpSession.getId() returns the session ID. The web application passes session IDs back to a client is via a cookie in the response. In the case of J2EE web applications, the cookie returned has a standard name—JSESSIONID.
25. Give a standard session tracking mechanism to use if a user has disabled cookies?
    a. URL Rewriting
26. What is URL Rewriting?
    a. URL rewriting is a standard session tracking mechanism mostly used when cookies are disabled in a client browser. A "pseudo-parameter" called jsessionid is placed in the URL between the servlet name (with path information, if present) and the query string. Example:
        i. http://localhost:8080/examp0401/SessionExample**;**jsessionid=58112645388D9380808A726A27F92997**?**name=value
27. Give two methods which will help you add the jsession id info for url rewriting to the URL?
    a. HttpServletResponse.encodeURL()
    b. HttpServletResponse.encodeRedirectURL()
28. What does HttpServletResponse.encodeURL() do?
    a. It accepts a String (representing the URL link on the web page minus session information) and returns a String (the same URL link, but now with the session information embedded).
29. Best practice dictates that every URL link you create in a servlet should be put through HttpServletResponse.encodeURL() method. Why?

a. HttpServletResponse.encodeURL() accepts a String representing the URL link and returns a String which has the same URL link, but now with the session information embedded.
b. This method is clever enough to know that if some other session mechanism is in force—at least one it recognizes, such as cookies—it returns the String representing the URL unchanged.
c. If every URL link you create in a servlet is put through this method, then, even if you are expecting your application to operate in a cookie friendly environment, it will still survive when it unexpectedly finds itself in cookie hostile territory.

30. What does an HttpServletResponse.encodeRedirectURL() do? How is the output string different from HttpServletResponse.encodeURL()?
    a. You give it a URL String; it gives back a URL String, with jsessionid embedded where necessary. This resulting URL String should then be used to plug into the HttpServletResponse.sendRedirect() method.
    b. The output String will look no different from a similar call to encodeURL(); the reason for providing encodeRedirectURL() is that the logic for determining whether or not to embed session information may be different when considering normal URLs versus redirect URLs.
31. Which is correct - encodeURL() or encodeUrl()? Explain.
    a. The method encodeURL() is the one we should use. The methods encodeUrl() and encodeRedirectUrl() are deprecated. These have "Url" in mixed case and were deprecated in a Java standardization exercise that mandated capitals for the abbreviation URL wherever it appeared in method or other names.
32. Give the methods that will identify which of the two standard session mechanisms are in use—cookies or URL rewriting?
    a. HttpServletRequest.isRequestedSessionIdFromCookie()
    b. HttpServletRequest.isRequestedSessionIdFromURL()
33. The methods that identify which of the two standard session mechanisms – cookies or url rewriting – belong to session object or request object? Give any advantage for doing so?
    a. Request object. There's a minor advantage in that you can execute these methods without having to first access the session object via the request.

34. Even when a session is present the methods isRequestedSessionIdFromCookie() and isRequestedSessionIdFromURL() called consecutively for the same request returned false. Give any possible reasons.
    a. This will happen:
        i. for SSL sessions.
        ii. for bespoke session mechanism logic (hidden form fields, for example).
        iii. when the session is new! Because at this point, the session ID isn't coming from URL or a cookie—but it has been generated by the web container.
35. Are management by cookie exchange and the management by rewriting URLs the only way to manage sessions? Can we use SSL?
    a. Management by cookie exchange and the management by rewriting URLs are are two principal methods for session management "officially" recognized by the servlet API. But we have more mechanisms.
    b. **Secure Sockets Layer (SSL),** which ensures secure communications between browsers and web applications, has its own session data built into it. If the web application uses HTTPS, then the web container may use the data on the HTTPS request stream to identify the client.
36. List down the attribute related methods of session.
    a. getAttribute(String name)
    b. getAttributeNames()
    c. setAttribute(String name, Object value)
    d. removeAttribute(String name)

Listeners

37. What is the basic idea behind listeners?
    a. Something of interest happens in your framework, and the framework lets the interested parties know. The interested parties are called "listeners" in Java (and design pattern) parlance. And whereas the Swing framework has listeners for mouse movements and keyboard strokes, the J2EE web application model has a set of server-side events that you can listen for.
38. List down the listener interface names for **listeners applied to request object**?
    a. ServletRequestListener

i. Responds to the life and death of each request.

b. ServletRequestAttributeListener

i. Responds to any change to the set of attributes attached to a request object.

39. List down the listener interface names for **listeners applied to context object**?

a. ServletContextListener

i. Responds to the life and death of the context for a web application.

b. ServletContextAttributeListener

i. Responds to any change to the set of attributes attached to the context object.

40. List down the two things you need to do to set up a listener in a web application?

a. Write a class that implements the appropriate listener interface.

b. Register the class name in the web application deployment descriptor, web.xml.

41. How can you configure your listener in the web.xml file? Give an example.

a. You need to place a <listener> element under the root element, and with this embed a <listener-class> element. The value held in the <listener-class> element is the fully qualified class name of a listener class.

b. No need to specify which type of listener you're talking about: The web container just works this out through Java's reflection capabilities.

c. Example:

i. <listener>

ii. <listener-class>com.abcpackage.RequestTrackingListener</listener-class>

iii. </listener>

42. Can a single listener class implement more than one kind of interface?

a. The same class could, potentially, implement more than one kind of interface.

43. Can you have more than one listener class implementing the same interface?

a. Yes.

44. If you have more than one listener class implementing the same interface, which order they will be triggered?

a. In the order in which they are declared in the deployment descriptor.

45. What will be the order of closedown events in session and context listeners, triggered during the closedown of a web application?

a. Closedown of a web application triggers a call to the matching closedown events in session and context listeners.
b. The order in which listeners are called is then in reverse order of deployment description declaration, with session listeners being processed before context listeners.

46. Is it necessary for listeners to have a no argument constructor?
    a. Listener classes must have a no-argument constructor. The web container is going to instantiate your listener only through the no-argument constructor. You can let the Java compiler supply one automatically if no other constructors are present.
47. List down the methods that need to be implemented for the ServletRequestListener?
    a. public void **requestInitialized**(ServletRequestEvent requestEvent)
        i. It is called the moment that any request in the web container becomes newly available—in other words, at the beginning of any request's scope. This is either at the start of the servlet's service() method or at the start of the doFilter() method for the first filter in a chain.
    b. public void **requestDestroyed**(ServletRequestEvent requestEvent)
        i. It is called for each request that comes to an end—either at the end of the servlet's service() method or at the end of the doFilter() method for the first filter in a chain.
48. List down two useful methods of **ServletRequestEvent**?
    a. getServletContext()
        i. returns the ServletContext for a web application.
    b. getServletRequest()
        i. returns the ServletRequest object itself (we can cast this to HttpServletRequest if you need to).
49. Write a listener class to preload an attribute into every request made to your web application?
    a. We can use a ServletRequestListener class and override the **requestInitialized()** method as:
        i. public void **requestInitialized**(ServletRequestEvent requestEvent) {
        ii. HttpServletRequest request = (HttpServletRequest)

iii. requestEvent.getServletRequest();
iv. request.setAttribute("foo", "bar");
v. }

50. List down the methods of **ServletRequestAttributeListener** interface?
    a. **attributeAdded(ServletRequestAttributeEvent srae)**
        i. is called whenever a new attribute is added to any request. In other words, any call to ServletRequest.setAttribute() will trigger a call to this method—provided that the name of the attribute being added to the request is a not a name already in use as an attribute of that request.
    b. **attributeRemoved(ServletRequestAttributeEvent srae)**
        i. is called whenever an attribute is removed from a request (as a result of any call to ServletRequest.removeAttribute()).
    c. **attributeReplaced(ServletRequestAttributeEvent srae)**
        i. is called whenever an attribute is replaced (as a result of any call to ServletRequest.setAttribute() for an attribute name already in use on that request).
51. Which all ServletRequestAttributeListener methods are called when we set an attributes value as null?
    a. **attributeRemoved**() method will be called. Setting a value as null effectively removes that attributes and is equivalent to **removeAttribute**().
52. Give the relation between ServletRequestAttributeEvent and ServletRequestEvent?
    a. **ServletRequestAttributeEvent** inherits from **ServletRequestEvent**.
53. List down the important methods of **ServletRequestAttributeEvent**?
    a. **getName()**
        i. It returns the String holding the name of the attribute being added, removed, or replaced.
    b. **getValue()**
        i. What is returned by getValue() varies slightly in meaning based on whether attribute is added, removed, or replaced:
            1. If **added** - returns the Object that is the value parameter on the setAttribute() call.

2. If **removed** - returns the Object that has been removed as a value from the request as a result of a removeAttribute() call.
3. If **replaced** - returns the old value of the attribute before a call to setAttribute() changed it.

54. Inside ServletRequestAttributeListener.attributeReplaced() method, how can you get hold of the new attribute value?
    a. **ServletRequestAttributeEvent** passed in as parameter to **attributeReplaced**() inherits from **ServletRequestEvent**, which has two handy methods we to get the context object and the request object. Having the request object, you can always get to the current value of an attribute using **getAttribute**() method on it.
55. Give the parent element of **ServletRequestEvent** and any important methods it inherits from that method?
    a. The "grandparent" of ServletRequestEvent is java.util.EventObject. This has one method—getSource()—which returns the object that is the source of the event. This—surprisingly perhaps—proves to be the ServletContext object: It represents the web application framework, which is, ultimately, the source of all events.
56. Give the code snippet that listens to removing attribute on a request object and print the values of old attribute value and new attribute value?
    a. public void **attributeReplaced**(ServletRequestAttributeEvent event) {
    b. String name = event.getName();
    c. Object oldValue = event.getValue();
    d. Object newValue = event.getServletRequest().getAttribute(name);
    e. System.out.println("Name of attribute: " + name);
    f. System.out.println("Old value of attribute: " + oldValue);
    g. System.out.println("New value of attribute: " + newValue);
    h. }
57. List down the methods that need to be implemented for the **ServletContextListener** ?
    a. **contextInitialized**(**ServletContextEvent** sce)
        i. It is called at the beginning of context scope. The context life cycle matches that of the web application: It's the first object made available on web application startup and the last to disappear at shutdown. So the

contextInitialized() method gets called before any servlet's init() method or any filter's doFilter() method.

b. **contextDestroyed**(**ServletContextEvent** sce)
    i. It is called at the end of context scope. And every filter and servlet destroy() method must have executed before the contextDestroyed() method is called.

58. List down the important methods of **ServletContextEvent**?
    a. Only one method **getServletContext**(), to get at the context object itself.
59. List down the methods of **ServletContextAttributeListener**?
    a. **attributeAdded**(**ServletContextAttributeEvent** scae)
    b. **attributeRemoved**(ServletContextAttributeEvent scae)
    c. **attributeReplaced**(ServletContextAttributeEvent scae)
60. List down the important methods of **ServletContextAttributeEvent**?
    a. getName()
        i. It returns the String holding the name of the attribute being added, removed, or replaced.

    b. getValue()
        i. What is returned by getValue() varies slightly in meaning based on whether attribute is added, removed, or replaced:
            1. If **added** - returns the Object that is the value parameter on the setAttribute() call.
            2. If **removed** - returns the Object that has been removed as a value from the request as a result of a removeAttribute() call.
            3. If **replaced** - returns the old value of the attribute before a call to setAttribute() changed it.
61. Give the parent hierarchy for **ServletContextAttributeEvent**?
    a. ServletContextAttributeEvent inherits from ServletContextEvent.
    b. ServletContextEvent inherits from EventObject.
62. List down the Session-Related Listeners?
    a. HttpSessionListener
    b. HttpSessionAttributeListener

c. HttpSessionBindingListener
    i. receives events when a value object is used as a session attribute.
d. HttpSesssionActivationListener
    i. receives events when a value object is transported across JVMs. This happens when the object is an attribute of a session in a distributed environment.

63. List down the methods of **HttpSessionListener**?

a. **sessionCreated**(HttpSessionEvent event)
    i. This method is called by the web container whenever a new session is provided. The sessionCreated() method receives an event, of type HttpSessionEvent. This has only the one method of its own, which is getSession()—to return the HttpSession object that has just been created.
b. **sessionDestroyed**(HttpSessionEvent event)
    i. This method is called by the web container at the moment a session is about to be invalidated—within the call to HttpSession.invalidate(), but before the session becomes invalid and unusable. An HttpSessionEvent object is passed as a parameter to the method, which gives access to the about-to-be-invalidated session through its getSession() method.

64. In your web application, you need to execute a block of code whenever the session object is first created. Can we create a Filter class, call the getSession(false) method, and if the result was null, then execute that block of code?
    a. No. The getSession() is defined on HttpServletRequest, not on ServletRequest. However a filter receives ServletRequet and not HttpServletRequest. The correct way would be to create an HttpSessionListener class and implement the sessionCreated method with that block of code.
65. List down the methods of **HttpSessionAttributeListener**?
    a. **attributeAdded**(HttpSessionBindingEvent hsbe)
        i. is called whenever a new attribute is added to any session.
    b. **attributeRemoved**(HttpSessionBindingEvent srae)
        i. is called whenever an attribute is removed from any session
    c. **attributeReplaced**(HttpSessionBindingEvent srae)
        i. is called whenever an attribute is replaced
66. List down the methods of HttpSessionBindingEvent?
    a. getName()
    b. getValue()
67. Give the parent hierarchy of **HttpSessionBindingEvent**?
    a. HttpSessionBindingEvent inherits from HttpSessionEvent.
    b. HttpSessionEvent inherits from EventObject.
68. List down the methods of HttpSessionAttributeEvent?
    a. HttpSessionAttributeEvent does not exist. **HttpSessionAttributeListener's** methods take an **HttpSessionBindingEvent** as a parameter.
69. List down two Session-Related Listeners Not Declared in the Deployment Descriptor?
    a. HttpSessionBindingListener
    b. HttpSessionActivationListener
70. List down the methods of **HttpSessionBindingListener**? When are they called?
    a. An HttpSessionBindingListener class has methods that are called when an object implementing HttpSessionBindingListener is added as an attribute value to a method or removed.
        i. valueBound(HttpSessionBindingEvent hsbe)

1. valueBound() is called whenever the object implementing the HttpSessionBindingListener interface is the value object passed to an HttpSession.setAttribute() call.

ii. valueUnbound(HttpSessionBindingEvent hsbe)

1. valueUnbound() is called whenever the object implementing the HttpSessionBindingListener interface is removed from the session as a result of an HttpSession.removeAttribute() call.

71. What happens if you have one or more objects implementing HttpSessionBindingListener, and have an HttpSessionAttributeListener defined in the deployment descriptor as well? Both have methods that are potentially called when session attributes are added, replaced, or removed—so which is called first?
    a. The web container must call all appropriate HttpSessionBindingListener valueBound() and valueUnbound() methods first, and only then call HttpSessionAttributeListener methods attributeAdded(), attributeReplaced(), or attributeRemoved().
72. What will happen if an object implementing HttpSessionBindingListener Obj1 is replaced by another object implementing HttpSessionBindingListener Obj2 in session?
    a. It will call valueBound() on Obj2 and valueUnbound() on Obj1.
73. What will happen to an object implementing HttpSessionBindingListener if the session to which it is bound times out?
    a. The **valueUnbound** method of the object implementing HttpSessionBindingListener instance is called when the session to which it is bound times out.
74. List down the methods of **HttpSessionActivationListener**? When are they called? What all conditions need to be satisfied?
    a. The methods of HttpSessionActivationListener are called in distributed environments on each HttpSessionActivationListener implementing object bound to the session, at the point where a session is moved from one JVM to another. The methods are:

i. **sessionWillPassivate(HttpSessionEvent hse)** is called on each HttpSessionActivationListener implementing object bound to the session **just prior to the serialization** of the session (and all its attributes).
ii. **sessionDidActivate(HttpSessionEvent hse)** is called on each HttpSessionActivationListener implementing object bound to the session **just after deserialization** of the session (and all its attributes).

b. In the source JVM, all objects bound to the session need to be serialized, and—of course—deserialized in the JVM that is the destination for the moved session. So objects should be serializable.
c. The object must be bound to the session as one of its current attributes.

75. List down the rules and restrictions on the handling of objects placed into instances of the HttpSession class using the setAttribute or putValue methods when a session is moved from one JVM to another?
    a. The container must be able to handle all objects placed into instances of the HttpSession class using the setAttribute or putValue methods appropriately. The following restrictions are imposed to meet these conditions:
        i. **The container must accept objects that implement the Serializable interface.**
        ii. The container **MAY choose to support storage of other designated objects in the HttpSession, such as references to Enterprise JavaBeans components and transactions.**
        iii. Migration of sessions will be handled by container-specific facilities.
        iv. The distributed servlet container must support the mechanism necessary for migrating objects that implement Serializable. If distributed containers persist or migrate sessions to provide quality of service features, **they are not restricted to using the native JVM Serialization mechanism for serializing HttpSessions and their attributes.** Developers are not guaranteed that containers will call readObject and writeObject methods on session attributes if they implement them, but are guaranteed that the Serializable closure of their attributes will be preserved.

v. Containers must notify any session attributes implementing the HttpSessionActivationListener during migration of a session. They must notify listeners of passivation prior to serialization of a session, and of activation after deserialization of a session.

vi. Application Developers writing distributed applications should be aware that since the container may run in more than one Java virtual machine, the developer cannot depend on static variables for storing an application state. They should store such states using an enterprise bean or a database.

76. Within an application marked as distributable, can we have requests that are part of a session to be handled by more than one JVM at a time?

    a. No. Within an application marked as distributable, all requests that are part of a session must be handled by one JVM at a time.