Chapter 01 – Servlet Model

Http Methods

1. What does an **HTTP request** contain?
    a. A request consists of a request line, some request headers, and an (optional) message body.
    b. The request line contains three things:
        i. The HTTP method
        ii. A pointer to the resource requested, in the form of a "URI"
        iii. The version of the HTTP protocol employed in the request
        iv. **Ex:** GET http://www.abcd.com/index.html HTTP/1.1
    c. A carriage return / line feed concludes the request line. After this come request headers in the form—name: value.
        i. **Ex-1:** Accept-Encoding: gzip, deflate
        ii. **Ex-2:** User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
    d. A blank line must follow the last request header and, after that, the request body—if there is one (there doesn't have to be).
2. What does an **HTTP response** contain?
    a. A response consists of a response line, some request headers, and an (optional) message body.
        i. The response line has three parts.
            1. First is the HTTP version actually used.
            2. Next is a response code (200 denotes success)
            3. After that is a brief description of the response code.
        ii. A carriage return / line feed denotes the end of the response line. Response header lines follow headers in the form—name: value.
        iii. Finally, separated from the headers by a blank line, is the response body.
3. Which are the seven HTTP methods?
    a. The seven methods are:
        i. **GET -** It's meant to be a "read-only" request for information from a server. With a GET requested, parameters are appended to the URL. The three standard triggers for a GET request are

1. Typing into the address line of the browser and pressing go
2. Clicking on a link in a web page
3. Pressing the submit button in an HTML <form> whose method is set to GET.

ii. **POST -** Triggered from forms in browsers, when the form method is set to POST. Whereas GET is intended as read-only, POST is for add/update/delete operations. When there are parameters with a POST request, they are put into the request body, not appended on to the query string of the URL, as for GET.

iii. **HEAD -** identical to the GET method, but doesn't return a message body. However, all the response headers should be present just as if a GET had been executed.

iv. **OPTIONS -** Tell the requester what HTTP methods can be executed against the URL requested.

v. **TRACE -** The purpose of TRACE is to return the request back to the requester in the state it was in at the point where it reached the last computer in the chain. The primary reason for doing so is to debug some problem that you attribute to request header change.

vi. **PUT -** Take a client resource and put it in a location on a server as specified in the URL of the request. If there's anything already on the server in that location a PUT will overwrite the current contents of the URL with the file it is uploading. You can determine from the response codes you get back whether the resource was replaced (typically, response code 200) or created for the first time (response code 201).

vii. **DELETE -** It causes the server to delete the contents of the target URL. A server has the right to delay its response to a DELETE method, as long as it responds later. A response code of 200 (OK) indicates that the delete has been done; 202 (accepted) means that the request has been accepted and will be acted on later.

4. Give any two advantages of sending parameters within the request body for a POST?
   a. A URL is limited in length. However the request body can be any big we want.

   b. Putting parameters into the URL is very visible and public and is usually recorded by the "history"-keeping nature of most browsers.

5. What is an idempotent request?
   a. An idempotent request is meant to have the same result no matter how many times it is executed. GET is supposed to be idempotent, whereas POST is not.
6. List down the methods that are considered idempotent?
   a. GET, TRACE, OPTIONS, HEAD, PUT and DELETE.
7. How can you divide the seven http methods as safe and not safe?
   a. GET, TRACE, OPTIONS, and HEAD should leave nothing changed, and so are safe. Even if a GET request can have irreversible side effects, a user should not be held responsible for them. Therefore, from a user perspective, the methods are safe.
   b. PUT, DELETE, and POST are inherently not safe: They do cause changes, and a user can be held accountable for executing these methods against the server.
8. Is a GET request guaranteed to be idempotent and POST guaranteed to be NOT idempotent?
   a. GET is supposed to be idempotent, whereas POST is not. That's how things are meant to be, but there is no absolute guarantee that any given GET request won't have irreversible side effects. The outcome depends on what the server program that receives the GET request actually does with it. Equally, a POST request may not result in an update. However GET and POST methods in general obey idempotency rules and your web applications should observe them.
9. Differnetiate between **original CGI model and FastCGI?**
   a. The **original CGI model** would die once the request was completed. **FastCGI**, which followed **original CGI model** along with other technologies, has the benefit of keeping processes alive—usually, one for each different server-side program that your server supports.
10. Give the name and parameters of **servlet methods** corresponding to the HTTP methods?
    a. The seven HTTP **methods map on to servlet methods** of the same name with a "do" in front (e.g., POST maps on to doPost()). The doXXX() methods receive

two objects as parameters—the first representing the HTTP request and the second the HTTP response.

11. Which HTTP method you use to check that a resource is valid and accessible, without its content?
    a. Using the **HEAD** method is an economical way of checking that a resource is valid and accessible, or that it hasn't been recently updated.

12. More notes on Http Methods
    a. **HTTP** is a simple request /response protocol underpinning most web applications on the Internet, regardless of whether they are written in Java.
    b. Beyond the seven main HTTP methods, there is **CONNECT** which is officially listed in the RFC, but it is essentially reserved for future use.
    c. In an HTTP request URL, question mark introduces **the query string** (ie the parameter list). Ex: http://dictionary.reference.com/search?q=idempotent
    d. **PUT and DELETE** are **disallowed** on most web servers as they are "unsafe" methods.
    e. Of the seven methods, **POST** is the only one that is not considered "**idempotent**."

Form Parameters

13. Give any two uses of a hidden input field?
    a. A servlet can temporarily store values to hidden fields. These values may be useful as parameters to the next servlet requested. It's one approach to "session control".
    b. Script running within the web page can also set the hidden values.

14. Describe an **HTML <form>** on a web page?
    a. An **HTML <form>** passes parameters along with the request. With a GET requested, these parameters are appended to the URL. When form control data are passed to the server, the name attribute supplies the parameter name. The value attribute supplies the parameter value.
    b. A **form** is defined on a web page with the tag **<form>**. A typical form opening tag might look like this:
        i. <form action="someServlet" method="POST">
    c. Major attributes are:

    i. **action** - contains some target resource in the web application.
    ii. **method** - denotes the HTTP method to execute. The default is HTTP GET.

15. How can you make a checkbox as already selected? Give an example.
    a. We can use the checked attribute as:
        i. **<input type**="checkbox" **name**="musicians" **value**="BTHVN" **checked="checked"** />Beethoven

16. What will happen if we pre-check more than one radio buttons with same name as:
<INPUT type="radio" name="volume" value="MED" checked="checked" />Medium
<INPUT type="radio" name="volume" value="LOW" checked="checked" />Low
    a. Only one choice is possible. The browser resolves this by letting the last one marked as checked take precedence.

17. What does a select tag do? Give examples.
    a. The <select> tag allows you to set up a list of values to choose from in a web page—the style is either a pop-up menu or a scrollable list. There are two "modes" for this control: The user is either restricted to one choice from the list or has multiple choices from the list.
        i. The single-choice, pop-up menu:
            1. <**select name**="Countries">
            2. <**option value**="FR">France</option>
            3. <option value="GB" selected>Great Britain</option>
            4. </select>
        ii. The multiple-choice:
            1. <**select name**="Countries" **size**="3" **multiple**>
            2. <**option value**="FR" **selected**>France</option>
            3. <option value="GB">Great Britain</option>
            4. <option value="DK" **selected**>Denmark</option>
            5. </select>

18. What are the different types of the **input** elements?
    a. **<input type="text" />** - allows a user to input a single line of text. The size attribute specifies the width of the text field in characters. The maxlength attribute

controls the maximum number of characters that a user can type into the text field.

b. **<input type="password" /> -** works just like text. The browser should however mask the characters typed in by the user.
c. **<input type="hidden" /> -** not available for user input. Such a form control is hidden. You see only the contents of a hidden field if you view the HTML source of the web page.
d. **<input type="checkbox" /> -** Use checkbox to put one or more small boxes on screen. By clicking and selecting a checkbox, its name and value will be sent as parameters. If there are more than one checkbox with same name selected, all selected values will be sent for that parameter name.
e. **<input type="radio" /> -** the choice made is mutually exclusive for all radio buttons with same name. Only one name can be selected and sent.
f. **<input type="submit" /> -** will create a button to send the form data to the URL designated by the action attribute of the opening <form> tag.
g. **<input type="reset" /> -** a button to reset all the values within the form to the way they were when the page was first loaded.
h. **<input type="button" /> -** a "custom button," which is connected to some sort of script within your page. You typically harness this by defining an onclick attribute within the input tag and making the value correspond to some JavaScript function.

19. To enter more than one line of text input from the user in a form you can use the ………. tag.
    a. <textarea>
20. List down the major attributes of the **textarea** element?
    b. **name**—as elsewhere, the parameter name
    c. **rows**—the number of visible lines (a scroll bar activates when access to further rows is needed)
    d. **cols**—the number of characters to displayed across the width of the area (based on an average-width character)

e. Another attribute—**wrap**—offers some flexibility in the treatment of carriage return and line feed characters, introduced either by the user pressing the enter key or the browser wrapping the text. There is no HTML standard for this attribute, and implementations vary slightly from browser to browser.

21. What was the disadvantage of **CGI**?
    a. CGI has been very useful, but had performance issues. The original CGI model demanded a heavyweight process on the web-serving host machine for each request made. Furthermore, this process would die once the request was completed.
22. Can we have more than one form in a page?
    a. There can be more than one form on the same web page. However, only one can be submitted at once.
23. Is it necessary to have name and value attributes for <input type="submit">?
    a. Not necessarily. However, it is quite usual to supply the value attribute, which allows you to define your own text on the button. If you supply a name attribute as well, then the name/value data for the submit button are passed just like any other parameter as part of the submitted form data.
24. Can we have separate type of form controls share the same name within a form?
    a. There is nothing illegal about entirely separate form controls sharing the same name. You can have, say, a checkbox, a set of radio buttons, and a text field that all share the same name—and so give rise to multiple values for the same parameter name when retrieved in the target servlet code.
25. List down the API methods to retrieve the request parameters?
    a. ServletRequest.**getParameterValues**(String parmName) - returns a String array with all values present, or null if no value exists for the parameter name.
    b. ServletRequest.**getParameter**(String parmName) - returns the first value for the given parameter.
    c. ServletRequest.**getParameterNames**() - returns an Enumeration of String objects representing the names of all the parameters in the request. If there are no parameters Enumeration will be empty.

   d. ServletRequest.**getParameterMap**() - returns a java.util.Map object, where the keys in the map are of type String (and represent each unique parameter name) and the values in the map of type String array (representing the values for the parameter).

26. What does this input tag type do : <input type ="file" name="fileForUpload" />?
    a. The tag—<input type ="file" name="fileForUpload" />—creates a button and a text field in your browser. The button can be used to launch a “choose file” type dialog allowing you to browse your local file system—the result of the choice is stored in the text field. When you submit the form, the chosen file is uploaded in the post body.
27. Give an occasion when you might need to work directly on the POST data using getReader() or getInputStream?
    a. When a POST request is being used to upload a file, usually in conjunction with an HTML <input> type.
28. Can you use getInputStream() after you have used getReader() or use getReader() after using getInputStream()?
    a. No. We should not call getReader() after getInputStream() or the other way.
    b. If you attempt to use getInputStream() after you have used getReader(), you are likely to get a message such as the one below:
       i. java.lang.IllegalStateException: getReader() has already been called for this request.
29. Can we have parameters with same name in the query string and within the POST request? How will getParameterValues() method handle this?
    a. When there are more values for a request parameter, getParameterValues(String parmName) should make sure that parameters in the query string should be placed before parameters in a POST request body.
30. List any cases where the methods ServletRequest.getReader() or ServletRequest.getInputStream() - can blow up?
    a. with an **IOException** if something goes wrong with the input /output process.

   b. —for getReader() only— with an **UnsupportedEncodingException** if the character set encoding used is not supported on the platform, and the text therefore remains un-encodeable.

31. While using ServletRequest.**getParameterMap**() can we add data to it?
    a. While using ServletRequest.**getParameterMap**(), you cannot put() key/value pairs in the Map retrieved from this method.
32. *The data from a POSTed form are transferred to a "parameter set" and, once there, are available to convenience methods such as getParameter().* List down the conditions for the data to be present in this set?
    a. The request follows the HTTP or HTTPS protocol, using the POST method.
    b. The content type for the request must have a value of "application/x-www-formurlencoded."
    c. Finally, a call must be made to one of the getParameter* convenience methods: The first such call causes the transfer.
33. Once data have been transferred to the parameter set with a call made to one of the getParameter* convenience methods, can we use the request's getInputStream() or getReader() to get hold of the parameters? What about using getParameter* convenience methods after a call to getInputStream() or getReader()?
    a. Once data have been transferred to the parameter set, you must not use the request's getInputStream() or getReader() to get hold of the parameters. Although you may not get any exception, the servlet specification promises unpredictable results.
    b. A call to a getParameter* method is unlikely to do you much good if the request body has been treated as a raw byte or character stream.
34. **More notes on Form Parameters:**
    a. By varying the value for the type attribute of <input> tag, a wide range of field types are available. All <input> tags have the name attribute in common.
    b. The APIs to retrieve parameters are found on javax.servlet.ServletRequest, the parent interface for javax.servlet.http.HttpServletRequest interface. Your servlet engine provides a class implementing the HttpServletRequest interface, and it

passes an instance of this as a parameter to whichever of a servlet’s doXXX() methods is targeted by an HTTP request.

c. The parameter value returned by getParameter() must be the first value in the array returned by getParameterValues().

d. You can read the request body directly by using ServletRequest.getReader() or ServletRequest.getInputStream(). It’s then up to you to examine the resulting character or input stream for name/value pairs separated by ampersands.

   i. **BufferedReader reader** = new BufferedReader(new InputStreamReader(
   ii. request.**getInputStream**()));
   iii. String line;
   iv. while ((line = reader.readLine()) != null) {
   v. StringTokenizer st = new StringTokenizer(line, "&");
   vi. while (st.hasMoreTokens()) {
   vii. out.write("<br /" + st.nextToken());
   viii. } } }

e. To get at parameters that are sent in a GET request, we can also use HttpServletRequest.getQueryString() method.

Requests

35. How can you classify HTTP headers? Give the common ones in each category?

a. **Headers** can be categorized into four types:

   i. **Request Headers:** pertaining strictly to the request.

      Common request headers are Accept, Accept-Charset, Accept-Encoding, Accept-Language, Authorization, From, Host, If-Modified-Since, Max-Forwards, Referer, User-Agent.

   ii. **Response Headers:** pertaining strictly to the response.

      Common response headers are Age, Location, Retry-After, Server, Set-Cookie, WWW-Authenticate.

   iii. **General Headers:** can occur on either the request or the response.

      Common general headers are Cache-Contro, Connection, Date, and Transfer-Encoding.

iv. **Entity Headers:** again, applicable to both request and response. These headers have information about the request or response body.

Common Entity Headers are Allow, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-Type, Expires, Last-Modified.

36. List down the methods on HttpServletRequest to get request header information?
    i. String **getHeader**(String name) - Returns the (first or only) value of the specified request header as a String.
    ii. Enumeration **getHeaders**(String name) - Returns all values of the specified request header as Strings within an Enumeration.
    iii. Enumeration **getHeaderNames**() - Returns the names of all request headers as Strings within an Enumeration.
    iv. long **getDateHeader**(String name) - Returns the value of the specified request header as a long primitive representing a date.
    v. int **getIntHeader**(String name) - Returns the value of the specified request header as an int primitive.
37. If we supply a header name that doesn't convert to a long date or an int to getDateHeader() or getIntHeader(), what will happen?
    a. If we supply a header name that doesn't convert to a long date or an int to getDateHeader() or getIntHeader(), getDateHeader() will throw an IllegalArgumentException, while getIntHeader() will throw a NumberFormatException.
38. What will happen if the requested header is not present for getDateHeader() or getIntHeader()?
    a. Both methods - getDateHeader() & getIntHeader() - return a value of -1 if the requested header is missing.
39. List down the Cookie attributes?
    a. Cookie attributes are:
        i. Name
        ii. Value
        iii. Domain

    iv. Path
    v. Comment
    vi. MaxAge
    vii. Secure
    viii. Version

40. How can you get cookies passed with the request from code?
    a. For getting cookies passed with the request, we can use the HttpServletRequest.getCookies() method. This returns an array of javax.servlet.http.Cookieobjects, or null if none are sent with the request.
41. What are cookies? How can you set the cookie attributes programatically?
    a. Cookies are small text files passed between client browsers and web servers. They are used in part as an extended parameter mechanism. A cookie can store a great deal of essential information— often enough to identify the client and to store information about choices made.
    b. A Cookie has a two-argument constructor passing in the mandatory name and value—thereafter, you can set any of its attributes except the name.
42. I want to get the cookie names available in the request header and compare with a required cookie name. Give the steps?
    a. We **get** the **Cookie array** from a **request** object using **getCookies()** method. Once we get the array we can go through the array and use the getName() method of the cookie to get the name of the cookie and compare with the required cookie name.
43. How can you get a Writer from a request object?
    a. We cannot get a writer from a request object, but only from a response object.
44. How can you make a copy of a cookie?
    a. You can clone() a cookie to make a copy of it.
45. Within an HTTP message, headers' names are separated from their values by a …(a)….., multiple values are separated by…(b)…., and each header is separated from the next by a …(c)…….. A ……(d)….finishes the header section, separating this from any attached request body.
    a. Colon, Comma, Carriage return, Blank line.

46. More notes on Request

a. There's an assumption there that getIntHeader() could never legitimately return a negative value, while any dates sought with getDateHeader() are after midnight on January 1, 1970, Greenwich Mean Time.

b. There is NO add-methods in **request** object. Only two setter methods are

  i. setAttribute(String arg0, Object arg1)

  ii. setCharacterEncoding(String arg0)

c. request.**getInputStream()** returns a **ServletInputStream**.

d. request.**getReader**() returns a **BufferedReader**

Response

47. How can you achieve the same effect as a sendRedirect() using response headers?

a. The call to sendRedirect() causes the servlet container to fill out a "Location" header with the alternative URL and set a status code telling the client to go to that url. If you wanted to achieve the same effect as a sendRedirect() in code, you could execute code such as the following:

  i. response.setStatus(HttpServletResponse.SC_TEMPORARY_REDIRECT)

  ii. response.setHeader("Location", "http://www.alternateurl.com/index.jsp")

48. How is redirection using sendRedirect() different from forwarding?

a. Redirection achieves roughly the same effect as forwarding. The difference is that sendRedirect() actually sends a message back to the client, so the client makes a re-request for the actual resource required. With forwarding, all the action stays on the server.

49. List down the methods that can be used to set header information on a response object?

a. **addHeader**(String name, String value) – keep adding values, even where the name is the same.

b. **setHeader**(String name, String value) - overwrites existing values. setHeader() just replaces one of the existing values with its own setting if there are multiple values for a header. However the API documentation is silent on this point.

c. **containsHeader**(String headerName) can be used to determine if a value has been set for a given header name already.

d. **addIntHeader**(String name, int value) and **addDateHeader**(String name, long value) - can be used for adding an int value and a date respectively.
e. **setIntHeader**(String name, int value) and **setDateHeader**(String name, long value) - They replace one of the existing values with its own setting if there are multiple values for a header. Else add an int value and a date respectively.
f. **setContentLength**(int arg0) method on the response sets the **Content-Length** header to the int value passed. We can also manually set the header using **setHeader** or **setIntHeader methods.** However setHeader will require us to pass the header value as a String, whereas **setIntHeader** will accept the integer value. So we can send the same integer value as is we send for setContentLength**.**

50. What is the best way to add a Cookie to a response? What are the alternatives?
    a. To attach a cookie to the response we can use the **addCookie**(Cookie cookie) method as many times.
    b. The **addCookie**() is a convenience method to add a header with the name "Set-Cookie" and a correctly formatted cookie content. So we can use addHeader() or setHeader() with the header name as Set-Cookie and value as a correctly formatted cookie content. But then you would have to do the formatting of the value yourself as a String.
51. Describe the HttpServletResponse.sendRedirect(String pathname) method?
    a. When a client makes a call to a servlet, it can send an alternative URL back and tell client to look there instead. This redirection can be achieved through the **HttpServletResponse.sendRedirect(String pathname)** method. The pathname can be either relative or absolute or a completely different domain url.
52. Differentiate the usage of sendError() and setStatus() methods of HttpServletResponse?
    a. sendError() sends an error response to the client using the specified status code and clearing the buffer. If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.
    b. setStatus() sets the status code for this response. This method is used to set the return status code when there is no error (for example, for the status codes SC_OK or SC_MOVED_TEMPORARILY). If there is an error, and the caller

wishes to invoke an error-page defined in the web application, the sendError method should be used instead. The container clears the buffer and sets the Location header, preserving cookies and other headers.

53. Give the method to remove a cookie?
    a. There is no "remove cookie" method.
54. Describe the ServletResponse.setContentType(String mimeType) method?
    a. The ServletResponse.setContentType(String mimeType) method can be used to set the content type of the response. You should provide a registered MIME type.
55. Is Content type same as the encoding you are using for your files? How can you set encoding?
    a. Content type is not exactly same as the exact encoding you are using for your files. The encoding could range from simple ASCII through to full Unicode, with plenty of variants in between. You can set the encoding along with the content type as part of the String passed to setContentType(). You can set the encoding separately through the setCharacterEncoding().
56. What will happen to the methods that get request header information if there is no header or parameter or attribute of the given name?
    a. In general, if there is no header or parameter or attribute of the given name, methods returning a String will return null. Methods returning Enumerations will return an instance of an Enumeration object, but it will be empty.
57. Will getHeaders(String Name) and getHeaderNames() return null in any case?
    a. For getHeaders(String Name) and getHeaderNames(), it's possible for the servlet container to deny access to the request headers. In this case null may be returned instead of an Enumeration object.
58. When and how you will use a PrintWriter and an OutputStream associated with the response?
    a. The output content can be character-based or byte-based and we can use a PrintWriter and an OutputStream associated with the response respectively for them, but you can't use both at once.
        i. To get hold of the PrintWriter from response object you can use:
            1. PrintWriter out = response.getWriter();

        ii. If it’s a binary file format you want to send in your response, then you can use
            1. OutputStream out = response.getOutputStream()

59. Can we create a cokkie as setHeader("Set-Cookie", newValue) ?
    a. In theory, you could use setHeader("Set-Cookie", newValue) to change the value of an existing cookie, but then you would have to do the formatting of the value yourself as a String.

Servlet Life Cycle

60. List down the stages of the servlet life cycle?
    a. **Servlet class loading**—the point where static data (if any) in your servlet are initialized
    b. **Servlet instantiation**—the point where instance data (if any) in your servlet are initialized, and the no-argument constructor (if present) is called. The servlet must be loaded and instantiated before the first request for it is processed.
    c. **init()**—the initialization method called when a servlet instance is created.
    d. **service()**—the method called to perform the work. The **service**() method convert HTTP requests into appropriate doXXX() method calls.
    e. **destroy()**—When a servlet instance is taken out of service, the **destroy**() method is called—only once for that instance.
61. What is the significance of UnavailableException? What are the two ways in which you can create UnavailableException? What is the significance of each ways?
    a. UnavailableException is a subclass of ServletException. The init() method might throw a ServletException or UnavailableException.
    b. UnavailableException can be constructed in two ways:
        i. **With a message**—this denotes permanent unavailability (like ServletException). The servlet container should log the fault and not necessarily try to create another instance.
        ii. **With a message and a time limit (in seconds)**— this denotes temporary unavailability. There’s no absolute definition of what the servlet container should do under these circumstances. A sensible resolution would be to

block requests to the servlet until the time limit has expired, then allow them through if the init() method has successfully completed.

62. Can we have servlets with parameterized constructor?
    a. We can have, provided we also have a no argument one also. The containers will instantiate servlets using the no argument constructor. If we provide a constructor with arguments, container won't provide the default no arg one and hence won't be able to instantiate a servlet and we will get a java.lang.InstantiationException.
63. Describe the init() method of the servlet?
    a. **init()** method called when a servlet instance is created.
    b. The servlet container is guaranteed to call **init**() method once—and only once—on instantiation of the servlet.
    c. The **init**() method must complete successfully before the servlet container will allow any requests to be processed by the servlet.
64. What should be the continer behaviour if the init() method throw a ServletException or UnavailableException?
    a. The **init**() method might throw a ServletException, or it might run out of time. A straight ServletException is a failure, and the servlet container can abandon this instance and try to construct another. It can also throw UnavailableException which is a sub class of ServletException in which case container can make additional attempts.
65. Can the initialization code for a servlet go in the constructor?
    a. Initialization code for a servlet could go in the constructor. There's nothing wrong in having a zero-argument constructor for a servlet, but it's more usual to override the public void **init**() method and place initialization code there.
66. Differentiate between **init**(ServletConfig config) and init()?
    a. The servlet container actually calls Servlet.**init**(ServletConfig config) which does some essential initialization of its own, including loading any initialization parameters associated with the servlet.
    b. The **init**() method is a convenience method in the GenericServlet class. The **init**(ServletConfig config) method calls **init()** after completing its own work. If

you have overridden the no-parameter init(), your version of the method will be found polymorphically.

67. Describe the service method of the servlet?
    a. The method is called by the servlet container in response to a client request.
    b. The method is defined in the Servlet interface. The method is implemented in the GenericServlet and HttpServlet classes.
    c. The method accepts a ServletRequest and ServletResponse as parameters.There is an overloaded version of the method in HttpServlet that dispatches to the appropriate doXXX() methods. The overloaded version accepts an HttpServletRequest and HttpServletResponse as parameters.
    d. Multiple threads may simultaneously access the service() method for a single servlet instance.
    e. There may be no client requests at all, and that means the service() method may never get executed.
68. Give the signature for the servlet life cycle methods init(), service() and destroy?
    a. The signature of servlet life cycle methods init(), service() and destroy are:
        i. public **void init**(**ServletConfig** config) throws ServletException
        ii. public **void service**(**ServletRequest** req, ServletResponse res) throws ServletException, IOException
        iii. public **void destroy**()
69. Give the conditions for the destroy() method to be called?
    a. When a servlet instance is taken out of service, the **destroy**() method is called—only once for that instance.
    b. All threads executing a service() method on this instance must have ended—or if not ended, gone beyond a server-defined time limit.
    c. destroy() is never called if init() failed. It’s inappropriate to call a method destroy what was never set up in the first place.