# Ch -7 Response Headers

## Setting Response Headers from Servlet

- **setHeader(String Name, String Value)**
  - This method sets the response header with the designated name to the given value.
  - There are two specialized methods to set headers that contain dates and integers

  - 1) setDateHeader
  - 2)setIntHeader

- **setDateHeader(String header, long milliseconds)**
  - This method saves you the trouble of translating a Java date in milliseconds since 1970 (as returned by System.currentTimeMillis, Date.getTime, or Calendar.getTimeInMillis) into a GMT time string.
- **setIntHeader(String header, int headerValue)**
  - This method spares you the minor inconvenience of converting an int to a String before inserting it into a header.

- HTTP allows multiple occurrences of the same header name, and you sometimes want to add a new header rather than replace any existing header with the same name.
  - The methods setHeader, setDateHeader, and setIntHeader replace any existing headers of the same name.
  - whereas addHeader, addDateHeader, and addIntHeader add a header regardless of whether a header of that name already exists.

- Finally, HttpServletResponse also supplies a number of convenience methods for specifying common headers.
- **setContentType(String mimeType)**
  - This method sets the Content-Type header and is used by the majority of servlets.
  - response.setContentType("type/subtype");
  - i.e
    - response.setContentType("image/jpeg");
    - response.setContentType(“text/html");
- **setContentLength(int length)**
  - This method sets the Content-Length header, which is useful if the browser supports persistent (keep-alive) HTTP connections.
- **addCookie(Cookie c)**
  - This method inserts a cookie into the Set-Cookie header. There is no corresponding setCookie method, since it is normal to have multiple Set-Cookie lines.

- **sendRedirect(String address)**
  - the sendRedirect method sets the Location header as well as setting the status code to 302.
  - response.sendRedirect(“http://localhost:8080/demo");
  - Response.sendRedirect(“http://www.microsoft.com”);

## HTTP 1.1 Response Headers and Their Meaning

- Allow
- Cache-control
- Connection
- Content-Encoding
- Content-Language
- Content-Length
- Content-Type
- Expires
- Last-Modified
- Location
- Pragma
- Refresh
- Retry-After
- Set-Cookie
- WWW-Authenticate

- **Allow :**
  - The Allow header specifies the request methods (GET, POST, etc.) that the server supports. It is required for 405 (Method Not Allowed) responses.
- **Connection** :
  - A value of close for this response header instructs the browser not to use persistent HTTP connections. Technically, persistent connections are the default when the client supports HTTP 1.1 and does *not specify* a “Connection: close” request header.
- **Content-Encoding**
  - This header indicates the way in which the page was encoded during transmission.
- **Content-Language**
  - The Content-Language header signifies the language in which the document is written

- **Content-Length**
  - This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection.
- **Content-Type**
  - The Content-Type header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
- **Last-Modified**
  - This very useful header indicates when the document was last changed.
  - The client can then cache the document and supply a date by an If-Modified-Since request header in later requests.
- **Location**
  - This header, which should be included with all responses that have a status code in the 300s, notifies the browser of the document address.
  - The browser automatically reconnects to this location and retrieves the new document.
  - This header is usually set indirectly, along with a 302 status code, by the sendRedirect method of HttpServletResponse.

- **Pragma**
  - Supplying this header with a value of no-cache instructs HTTP 1.0 clients not to cache the document.
  - However, support for this header was inconsistent with HTTP 1.0 browsers.
  - In HTTP 1.1, “Cache-Control: no-cache” is a more reliable replacement.
- **Retry-After**
  - This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.
- **Set-Cookie**
  - The Set-Cookie header specifies a cookie associated with the page.
  - Each cookie requires a separate Set-Cookie header.
  - Servlets should not use response.setHeader("Set-Cookie", ...), but instead should use the special-purpose addCookie method of HttpServletResponse.

- **WWW-Authenticate**
  - This header is always included with a 401 (Unauthorized) status code.
  - It tells the browser what authorization type and realm the client should supply in its Authorization header.
  - Frequently, servlets let password-protected Web pages be handled by the Web server's specialized mechanisms (e.g., .htaccess) rather than handling them directly.
- **Cache-Control**
  - This useful header tells the browser or other client the circumstances in which the response document can safely be cached. It has the following possible values.
  - **public** : Document is cacheable, even if normal rules (e.g., for password-protected pages) indicate that it shouldn't be.
    - response.setHeader("Cache-Control", " public ");

- **private.** Document is for a single user and can only be stored in private (nonshared) caches.
  - response.setHeader("Cache-Control", " private");

- **no-store.** Document should never be cached and should not even be stored in a temporary location on disk. This header is intended to prevent inadvertent copies of sensitive information.
  - response.setHeader("Cache-Control", " no-store");

- **max-age=xxx** : Document should be considered stale after xxx seconds. This is a convenient alternative to the Expires header but only works with HTTP 1.1 clients. If both max-age and Expires are present in the response, the max-age value takes precedence.

- **s-max-age=xxx :** Shared caches should consider the document stale after xxx seconds.
- The Cache-Control header is new in HTTP 1.1.

- **Refresh**
  - This header indicates how soon (in seconds) the browser should ask for an updated page.
  - Note that Refresh does not stipulate continual updates; it just specifies when the ***next*** update should be. So, you have to continue to supply Refresh in all subsequent responses. This header is extremely useful because it lets servlets return partial results quickly while still letting the client see the complete results at a later time.
  - response.setIntHeader("Refresh", 30);
  - Instead of having the browser just reload the current page, you can specify the page to load. You do this by supplying a semicolon and a URL after the refresh time. For example, to tell the browser to go to http://host/path after 5 seconds
  - response.setHeader("Refresh", "5; URL=http://host/path/");
  - Example: Refresh.java

- **Expires**

  - This header stipulates the time at which the content should be considered out-of-date and thus no longer be cached.
  - A servlet might use this header for a document that changes relatively frequently, to prevent the browser from displaying a stale cached value.
  - an Expires header with a date in the past is often used to prevent browser caching.

- long currentTime = System.currentTimeMillis();
- long tenMinutes = 10*60*1000; // In milliseconds

  response.setDateHeader("Expires", currentTime + tenMinutes);

## 7.5 Using Servlets to Generate JPEG Images

- Although servlets often generate HTML output, they certainly don't always do so.
- First, let us summarize the two main steps servlets have to perform to build multimedia content.
  1. **Inform the browser of the content type they are sending.** To accomplish this task, servlets set the Content-Type response header by using the setContentType method of HttpServletResponse.
  2. **Send the output in the appropriate format.** This format varies among document types, of course, but in most cases you send binary data, not strings as you do with HTML documents. Consequently, servlets will usually get the raw output stream by using the getOutputStream method, rather than getting a PrintWriter by using getWriter.

## 1. Create a BufferedImage.

- You create a java.awt.image.BufferedImage object.
  - BufferedImage(int width,int height,int type)
  - i.e.

```
BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
```

## 2. Draw into the BufferedImage.

- You accomplish this task by calling the image's getGraphics method, casting the resultant Graphics object to Graphics2D, then making use of Java 2D's rich set of drawing operations, coordinate transformations, font settings, and fill patterns to perform the drawing.
  - Graphics2D g2d = (Graphics2D)image.getGraphics();
  - i.e.
    - G2d.drawLine();
    - G2d.background();

## 3. Set the Content-Type response header.

- As already discussed, you use the setContentType method of HttpServletResponse for this task.
- The MIME type for JPEG images is image/jpeg. Thus, the code is as follows.
- response.setContentType("image/jpeg");

## 4. Get an output stream.

- As discussed previously, if you are sending binary data, you should call the getOutputStream method of HttpServletResponse rather than the getWriter method.
- For instance:
- OutputStream out = response.getOutputStream();

## 5. Send the BufferedImage in JPEG format to the output stream.

- When you use the ImageIO class, you just pass a BufferedImage, an image format type ("jpg", "png", etc.) and either an OutputStream or a File to the write method of ImageIO.
- Except for catching the required IOException.

```
try
{
    ImageIO.write(image, "jpg", out);
}
catch(IOException e)
{ System.out.println(e.getMessage());
}
```

Example : ImageDevelop.java

## Summary

- void setContentType(String mimeType)
- void setContentLength(int length)
- void addCookie(Cookie c)
- void sendRedirect(String address) throws IOException
- void setHeader(String Name, String Value)
- PrintWriter getWriter()
- OutputStream getOutputStream()

**OTHER METHODS of HttpServletResponse**

- void setStatus(int sc)
- String encodeURL(String url)