Common Response Headers

- **Content-Encoding**
  - The way the document is encoded.
- **Content-Length**
  - The number of bytes in the document.
- **Content-Type**
  - The MIME type of the document.
- **Expires**
  - The time at which document should be considered out-of-date and thus should no longer be cached.
Common Response Headers (cont.)

- **Last-Modified**
  - The time the document was last modified.

- **Location**
  - The URL to which browser should reconnect.

- **Refresh**
  - The number of seconds until browser should reload page.
  - Can also include URL to connect to.
Response Status Codes

- HTTP response status code:
  - 1xx : Informational
  - 2xx : Success
  - 3xx : Redirection
  - 4xx : Client error
  - 5xx : Server error

- Setting the *status code* allows you to:
  - Forward client to another URL
  - Indicate a missing resource
  - Instruct browser to use a cached copy
Common Status Codes

- **200 (OK)**
  - Everything is fine, document follows.
  - Default for servlets.

- **204 (No Content)**
  - Browser should keep displaying previous document.

- **301 (Moved Permanently)**
  - Requested document permanently moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
Common Status Codes (cont.)

- **302 (Found)**
  - Requested document temporarily moved elsewhere (indicated in Location header).
  - Browser goes to new location automatically.

- **304 (Not Modified)**
  - When the request header If-Modified-Since present, the requested document was available and not modified.

- **401 (Unauthorized)**
  - Browser tried to access password-protected page without proper Authorization header.

- **404 (Not Found)**
  - No such page.
HTTP Response API

- In class `HttpServletResponse`
  - `public void setContentLength(int len)`
    - Sets the `Content-Length` header.
  - `public void setContentType(String type)`
    - Sets the `Content-Type` header.
  - `public void setStatus(int statusCode)`
    - Sets the status code for this response. This method is used to set the return status code when there is no error, e.g., `SC_OK` or `SC_MOVED_TEMPORARILY`.
    - Should set status before sending document.
HTTP Response API (cont.)

- `public void sendError( int statusCode)`
- `public void sendError( int statusCode, String message)`
  - Sends an error response to the client using the specified status code and an (optional) descriptive message.
  - Wraps the message inside a small HTML document.
- `public void sendRedirect( String location)`
  - Sends a temporary redirect response to the client using the specified URL.
  - May specify relative URL.
HTTP Response API (cont.)

- public boolean containsHeader(String name)
  - Whether the named response header has already been set.
- public void setHeader(String name, String value)
  - Sets a response header with the given name and value.
  - If the header is already set, the new value overwrites the previous one.
- public void addHeader(String name, String value)
  - Adds a response header with the given name and value.
  - Response headers may have multiple values.
HTTP Response API (cont.)

- `public void setDateHeader(String name, long date)`
  - Sets a response header with the given name and date or integer value.
- `public void setIntHeader(String name, int value)`
- `public void addDateHeader(String name, long date)`
- `public void addIntHeader(String name, int value)`
  - Adds a response header with the given name and date or integer value.
Conditional Get

- When the request header `If-Modified-Since` is present in a request, it is called a conditional get.
  - Advantage: reduce network traffic and server load.
  - Web browser (client):
    - A browser may cache the pages it has visited.
    - When a user requests a page that has been cached, the browser sends a conditional get.
  - Web server:
    - When a conditional get is received, if the requested page is modified after the date, the page is sent to the client.
    - Otherwise, sends status code 304 (Not Modified)
Conditional Get for Servlets

- A servlet may take advantage of the conditional get.
  - Explicitly setting the Last-Modified header
    ```java
    long t = System.currentTimeMillis();
    response.setDateHeader("Last-Modified", t)
    ```
  - Overriding the `getLastModified()` method in class `HttpServlet`
    ```java
    long getLastModified( HttpServletRequest request) {
        return System.currentTimeMillis();
    }
    ```
Authentication

- Authentication
  - The mechanism by which communicating entities prove to one another that they are acting on behalf of specific identities.

- A web client can authenticate a user to a web server using one of the following mechanisms:
  - HTTP Basic Authentication
  - HTTP Digest Authentication
  - HTTPS Client Authentication
  - Form Based Authentication
HTTP Basic Authentication

- Based on a *username* and *password*.
  - A web server requests a web client to authenticate the user. *(Challenge)*
  - The web client obtains the username and the password *(Credentials)* from the user and transmits them to the web server.
  - The web server then authenticates the user in the specified realm.

- Basic Authentication is not a secure authentication protocol.
  - The user password is transmitted with a simple base64 encoding.
  - The target server is not authenticated.
A web client attempts to access a protected realm.

The web server may respond with a challenge:
- Status code: 401
- Response header: WWW-Authenticate: Basic realm="name"

The web client responds with a request that includes the user credentials in the following request header:
- Authorization: Basic <base64 encoded user-pass>
- Decoded user-pass is in the following form: userid: password
A basic Web Application
Reimbursement Architecture
ReimburseBase: The Base Class

- Enforce a consistent style of layout and look & feel.
- Simplify the responsibility and construction of individual pages.
- Make it easy for global changes or upgrades.
- Configurable and extensible.
Base Servlet Methods

- `makeHeader()`
- `makeFooter()`
- `makeLeftBar()`
- `makeRightBar()`

- The base servlet creates the standard template.
- Subclasses can override them to customize content
Base Servlet Methods (cont.)

- `abstract public boolean handleRequest(HttpServletRequest request, HttpServletServletResponse response, List content)`
  throws ServletException, IOException;

- To be overridden by each subclass to handle requests
- Content is returned as a list of HTML page elements.
- Returns true for normal conditions.
- Returns false to bypass the base servlet.
public void doPost( HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    makeHeader();
    if (handleRequest(request, response, content)) {
        makeLeft() and add it to the page;
        add the content to the page;
        makeRight() and add it to the page;
        makeFooter();
    }
}
Reimbursement

- **Class** ReimburseNew
  - Handles the Form for a new Reimbursement Request
  - **Contains Inner Class** ReimbursementForm
    - Fields for each form field
    - Form makeForm()
    - void fillForm(HttpServletRequest request)
    - String[] validateForm()
    - List processForm()
Reimbursement Form methods

- **makeForm()**
  - Builds an HTML form that consists of the input fields of the registration form.
  - If the a field in the RegistrationForm object contains a non-null value, the value will be filled in the corresponding input field in the HTML form.

- **fillForm()**
  - Retrieves the values posted from the HTML form and assigns the values to the corresponding fields of the RegistrationForm object.
Reimbursement Form methods (cont.)

- `validateForm()`
  - Checks whether all the required fields contain non-null values.
  - Returns null if all the required fields contain non-null values.
  - Otherwise, returns an array messages indicating what needs to be corrected on the form
    - Missing
    - Wrong format, size, length

- `processForm()`
  - Builds an HTML page that confirms the registration.
public boolean handleRequest(HttpServletRequest request,
                               HttpServletResponse response,
                               List content)
       throws ServletException, java.io.IOException {

    content.add(new H1("New Reimbursement"));
    String submit = request.getParameter("submit");
    ReimbursementForm form = new ReimbursementForm();
    if (submit == null) {
        // blank form
        content.add(form.makeForm());
    } else {
        // populate the form
        <next slide>
    }

    return true;
}
// populate the form
form.fillForm(request);

String[] missing = form.validateForm();
if(missing == null) {
    List confirmation = form.processForm();
    content.addAll(confirmation);
} else {
    content.add("The following field(s) need to be corrected or entered:");
    UL ulist = new UL();
    for (int i = 0; i < missing.length; i++) {
        ulist.addElement(new LI(missing[i]));
    }
    content.add(ulist);
    // give the form back, with data as submitted
    content.add(form.makeForm());
}
Patterns for Building the Web App

- **Page Builder Pattern**
  - Enforces a consistent style
  - Simplify page construction
  - Related GoF patterns: Template Methods, Builder

- **Form Handler Pattern**
  - A uniform way to validate and process posted form data.
Maintaining State

- HTTP is a stateless protocol
- Each request is completely independent of the previous
- Most applications need to understand the current state of the application
  - The pervasive "shopping cart"
  - Personalization
  - Workflow
How can you maintain state?

- State must either be completely part of the request, or maintained on the server
  - Cookies - simple data
  - Sessions - data for a current session (across pages)
  - Databases - persisting data across sessions (next lecture)
Cookies

- Defined in RFC 2109
- A cookie is basically a small piece of information stored by a web browser on the local client.
- Browsers are expected to only support 20 cookies per Web server, 300 total, and can be limited to 4kb each
- Cookies are returned to the server as HTTP request headers
Cookie uses

- Cookies are used on the web for a number of purposes
  - Tracking Sessions
  - Targeting advertising
  - Tracking usage
  - Personilization
- If used properly, not a security risk
Problems with Cookies

- A user can turn them off
- They can violate privacy (embedded images)
- If used improperly, a security risk
- As a developer, you cannot depend on them completely
Cookie API

- public `Cookie(java.lang.String name, java.lang.String value)`
  - Creates a Cookie

- public void `setMaxAge(int expiry)`
  - Set to age in seconds when Cookie expires
  - 0 deletes it
  - negative value deletes it when browser exits

- getters/setters for Name, Value, Path, Domain, etc.
Cookie API (cont.)

- Cookies are returned by the browser to the server if the Domain and Path match the server
- Getting Cookies from the request
  - public Cookie[] getCookies()
- Adding Cookies to the response
  - public void addCookie(Cookie cookie)
Session Tracking

- In web applications, the concept of a session is used to put consecutive requests and responses in context.
- Implicit in a *session* are the following concepts:
  - start and end
  - may be terminated by either party
  - state information is present
- The concept of *session tracking* allows stateful information to be stored on the server and associated with a specific client.
Session Tracking Approaches

- Since HTTP is stateless, the identification of a specific session is needed in each request.
- This can be accomplished using:
  - **Cookies**
    - Cookies have to be enabled
      ```java
      Cookie sessionCookie = new Cookie("jsessionid", "1234");
      ```
  - **URL-rewriting**
    - Need to encode the URL
      ```html
      http://host/path/file.html?jsessionid=1234
      ```
  - **Hidden form fields**
    - All pages need to be created dynamically to keep the session
      ```html
      <input type="hidden" name="jsessionid" value="1234"/>
      ```
Session Tracking API

- The Java Servlets API includes support for session tracking
- The interface HttpSession encapsulates the session information
- The session is stored on the server
- Tracking is performed using Cookies or URL-rewriting and is transparent to the servlet
  - To see this, block cookies in your browser and visit a page that uses servlets and sessions
Using the Session

- The scope of an HttpSession is the entire webapp
- The session persists until it expires on the server, or it is expired in code
- Programming issues
  - To accommodate URL rewriting, servlets should call `encodeURL()` for all URLs.
  - All session objects have to be Serializable (implement `java.io.Serializable`)
  - The HttpSession can be accessed from more than one thread at a time
Session API

- In the `HttpServletResponse` class:
  - `HttpSession getSession()`
  - `HttpSession getSession(boolean create)`
  - Returns the current `HttpSession` associated with this request, or if there is no current session and `create` is `true`, returns a new session.
  - Default: `create` is `true`

- `String getRequestedSessionId()`
  - Returns the session ID.
Session API (cont)

- **In the HttpSessionResponse class:**
  - `String encodeURL(String url)`
  - Encodes the specified URL by including the session ID in it, or
  - If encoding is not needed, returns the URL unchanged.
Session API (cont)

- **The HttpSession class:**

  `void setAttribute(String name, Object value)`
  - Binds an object to the specified name in this session.

  `Object getAttribute(String name)`
  - Returns the object bound to the specified name in this session, or
  - Returns `null` if no object is bound to the name.

  `void removeAttribute(String name)`
  - Removes the binding with the specified name and the object from this session.
  - Can do the same by calling `setAttribute(name, null);`
Session API (cont)

String getId()

- Returns the session ID.

Enumeration getAttributeNames()

- Returns an Enumeration of string objects containing the names of all the objects bound to this session.

long getCreationTime()

- Returns the time when this session was created in ms since the epoch

long getLastAccessedTime()

- Returns the last time the client sent a request associated with this session.
Session API (cont)

```java
int getMaxInactiveInterval()
```
- Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
- Tomcat default: 30 minutes

```java
void setMaxInactiveInterval( int interval)
```
- Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.
- A negative time indicates the session should never expire.
void invalidate()

- Invalidates this session and unbinds any objects in this session.

boolean isNew()

- Returns true
  - if the client does not yet know about the session, or
  - if the client chooses not to join the session (i.e. cookies are disabled).