# CS 378 Modern Web Apps - Collaborative Notes & Exam Study

---

| | |
|---|---|
| Instructor: | Dev (Devdatta Kulkarni) |
| Email: | devdatta@cs.utexas.edu |
| Office: | GDC 4.812 (but office hours are not held here) |
| Office Hours: | WED 8-9:30pm @ GDC 2.104 |

---

| | |
|---|---|
| TA: | Venket (Venketaram Ramachandran) |
| TA Email: | v.ram28@gmail.com |
| TA Office Hours: | THU 12:30-2pm @ GDC 1.302 #1 and FRI 2pm-3:30pm @ GDC 1.302 #3 |

---

| | |
|---|---|
| Github: | https://github.com/devdattakulkarni/ModernWebApps |
| Course books: | Professional Java For Web Applications<br>RESTful Java with JAX-RS 2.0<br>Both available from this GitHub repo |

---

ALL slides converted to PDF and combined into one file

---

| | |
|---|---|
| Final Exam: | A link to **YOUR** Final Exam Schedule<br>Monday, May 16, 7:00-10:00 pm |

---

To add to the table of contents
highlight something to add → apply a heading style (menu next to font select)→ click the table of contents→ click the refresh button inside the table

## Table of contents:

FINAL EXAM STUDY

# HTTP

## HTTP Resources

A **resource** is an entity maintained by the server.

- may have a "real" manifestation
  - an actual file that is stored on the server
  - defaults to serve index.html
- may be virtual
  - result of a query is rendered during the request
    - e.g. highlighting of a search query on the page, dynamic/virtual in nature
- they are identified / located by URL (Uniform Resource Locator)
  - http_url: "http:" "//"host[":"port ][ abs_path ]
  - host: domain name or IP address
  - port: 0 or more digits, **port 80 is default for HTTP, port 443 for HTTPS (default, but optional)**
  - abs_path: path of resource on the server
- Etag in the header is used for caching / client identification, often stored in a cookie

## HTTP Methods

- a **method** indicates what to do with a resource.
- Method types **(CRUD, create, read/retrieve, update, delete)**:
  - **C**reate a new resource
  - **R**etrieve info
  - **U**pdate/modify info
  - **D**elete a resource
- HTTP has several methods that have predefined meanings (**Bold** are the most common)
  - **GET** (read)
  - HEAD
    - get methods that you can do with that resource
  - **POST**
    - create a resource
  - PUT
    - modify a resource
  - **DELETE**
    - delete a resource
  - OPTIONS
  - TRACE

## Socket Programming

- remote process running on a remote server listening on an IP address. client connects to that IP and sends some data and reads some data
- to connect you need an **address** and a **port**
- UA (User Agent)
- O (Origin Server)
- Intermediary points along the chain may cache responses
  - can be proxies, gateways, or tunnels
- Proxy servers
  - something which is closer to a client that creates requests in place of the client
- Gateway server (reverse proxy)
  - vise versa of a proxy, a reverse proxy.
  - receives requests and sends responses in front of a server
    - used to balance load
    - routing requests
- HTTP needs reliable transport mechanism (meaning that all packets will arrive at the client or resend missed packets).
  - protocols that are defined at different layers of the application stack.
    - TCP (transmission control protocol) - **ok**
    - SMTP (simple mail transfer protocol) - **ok**

- UDP (user datagram protocol) **- not ok**
  - not possible to implement HTTP on top of UDP (not reliable)
  - A connectionless transmission model with no guarantee of delivering, ordering, or duplicate protection

# HTTP requests

- Request-line
  - Method <space> URI <space> HTTP-Version CRLF
  - CLRF = newline
- Request Headers
  - allow client to pass additional info about the request or the client itself to the server
  - http 1.0 request headers
    - User-agent: info about agent originating the request. Ie. send different file to android vs. a laptop
    - authorization
      A user agent can authenticate itself with a server by using this
      - Basic Authentication
        - username/password base64 encoded -> send to server
          - not encrypted! -> not secure! (not really used because of this)
      - Digest Authentication
        - "digest" refers to a "message digest" method, such as md5
        - you "digest" your username/password with the given method before you send it, and the server knows these values (or can generate them by using the same method on your username/password), then compares them to authenticate you
    - if-modified-since
- CRLF (Newline)
  - stands for carriage-return-line-feed if you were curious - MUST be an explicit newline, not a character (\n)
- Request Body
- Example of HTTP Request
  - **HTTP 1.0:** GET /irclogs/ HTTP/1.0
  - **HTTP 1.1:** GET /irclogs/ HTTP/1.1
    Host: <host-name> **Host header required for 1.1

# HTTP Responses

- Status-Line
  - [HTTP-version] <space> Status-code <space> Reason-phrase CRLF
  - Example: HTTP/1.1 200 OK
  - 3 Digit integers, first digit defines class of response
    - 1xx - informational
    - 2xx - success (202 - No content, accepted but nothing to send back)
    - 3xx - redirection (further action must be taken. 301 - Moved Permanently)
    - 4xx - client error (404 - Not found, others - bad syntax or cannot be fulfilled, 401 - Authorization[login] required)
    - 5xx - server error
    - Detailed HTTP code list at: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- HTTP 1.0 Response Headers
  - Date
    - Indicates the date when the message originated
  - Last-modified
    - Date/time when the resource was last modified
      - For files, it may be the just the file system last-modified time
      - How to deal with a group of resources?
  - Content-length
    - Indicates the size of the entity=body
    - The value is decimal
      - Example: java TestClientSocket eacesdrop.openstack.org /
        - Show Content-Length
        - Show serveroutput.txt
  - Content-type
    - Indicates the media type in the Entity-Body
  - Example HTTP 1.0 Response:

```
HTTP/1.0 200 OK
Date: Fri, 08 Aug 2003 08:12:31 GMT
Server: Apache/1.3.27 (Unix)
MIME-version: 1.0
Last-Modified: Fri, 01 Aug 2003 12:45:26 GMT
Content-Type: text/html
Content-Length: 2345
** a blank line *
<HTML> ...
```

- CRLF (newline)
  Same as in requests, stands for carriage-return-line-feed if you were curious - MUST be an explicit newline, not a character (\n)
- Response Body
  The content of the response

- HTTP 1.1: Chunked Transfer encoding
  Transfer encoding header
- breaks the data into chunks, server doesn't know how many chunks, sends the chunks to the client, first thing it sends is the chunk size (in hex), followed by the actual chunk, repeats chunk size and chunk until chunk size zero appears
- Transport layer will handle errors
  - (correct me if I screwed any of this up) → TCP works like this: If you didn't get a chunk (packet, actually), TCP asks for it again, when you have all of them, it moves on, TCP sits on top of IP, which does not guarantee order (or that data reaches your destination at all, actually), so TCP checks that for you.
- Transfer coding
  - encoding transformation that has been applied to an entity-body
  - chunked transfer encoding
    - transfer-coding: chunked
    - Chunked-body: chunk, last-chunk (has chunk-size 0), trailer, CLRF
      - Chunk = chunk-size (string of Hex digits), CRLF, chunk-data, CLRF

# HTTP Caching

- Goal
  - client - eliminate the need to send extra requests and to reduce # of round-trips (request → responses) required
  - server - eliminate the need to send responses/resources more than once - client holds onto resources
  - *caching is managed via request and response headers
- Overall Operation
  - Step 1: User Agent (UA) requests a resource on behalf of the user
  - Step 2: Server sends the resource along with some caching related headers
  - Step 3: Intermediate cache/proxy server save the resource and headers
  - Step 4: Cache/proxy server uses a cache algorithm to satisfy subsequent requests
- Caching headers HTTP 1.0
  - Expires (Response header)
    - tells the client when the resource should be erased from the cache and reloaded from the server
    - if the expires header is **not present** the resource should **not be cached**
    - Interesting factoid about the "Sun 19 Nov, 1978" Expires header from the cs.utexas.edu website
      - https://en.wikipedia.org/wiki/Dries_Buytaert
      - ^^this guy's birthday^^ is the default value if the expires header is not set
  - If-modified-since (Request header)
    - GET the resource only if it's been modified since X date
    - the if-modified-since field lets us know If the client has a cached copy that is the same as the server's copy, The server responds with 304 if not modified and the client uses its cached copy.
    - Use the Last-Modified header value for this
  - Pragma: no-cache
    - request header
    - Meaning: an intermediary should forward the request toward the origin server even if it has a cached copy of what is being requested
    - Basically meaning a client can request to skip the cache from the server and get a fresh copy of the resource
- Problems with HTTP 1.0 Caching
  - Won't work if the the clocks on the server and client are not synchronized

- Not fine-grained enough
  - does not support mechanisms to control contents of a cache
- Caching headers HTTP 1.1
  - Entity-tag (ETag) cache validator headers
    - An entity tag is a "signature" of the resource
      - I'm thinking like a hash of the resource, a change to the resource changes the hash
    - used to validate a resource as "good enough to use"
    - a client can never ask a server to calculate a new Etag, it is all up to the server to calculate and re-calculate
    - Two types - it is up to the server when to calculate the entity tag:
      - Strong: changes for every change
        - the server will calculate a new Etag for any little change to the resource like:
          - modifying the file
          - how many times the resource was requested (access counter)
      - Weak: changes for only semantically significant changes
        - has a 'W' in front of the Etag, example→ Etag: W/"12342344523-1"
        - when 'ticket name' is changed from lower case to mix case, server may not calculate new validator value
        - when more than one update is likely to happen to a resource within a second and the timestamp is being used a validator, then it will be a weak validator
          - implies timestamps never have a resolution finer than 1 second
  - Cache-control headers
    - allows a client or server to transmit a variety of directives in either requests or responses
      - these directives typically override the default caching algorithms
      - must be obeyed by all caching mechanisms along the request/response chain
    - max-age header
      - tells the client to cache the response for only X amount of seconds, if the value is 0, then the client should not cache this resource
      - Uses 'expires' header or 'max-age' directive
        - If both are present, 'max-age' takes precedence
      - seconds are in decimal
      - max-age=0
        - client's way to force a recheck with the origin server by an intermediate cache
        - same as the 'Pragma: no-cache' header of HTTP 1.0
        - can be sent in as part of a request OR response header
    - s-maxage header
      - for shared cache, the specified maximum age overrides max-age and expires header
      - takes precedence over other cache-control header conditions (max-age & expires)
  - if-none-match (Request header)
    - only send the request if the Etag sent in this header doesn't exist on the server ("does not match server's resource Etag value") - resource has been modified
    - semantically the same as if-modified-since, but if-modified-since used as a tiebreaker if both headers are sent or if there is a server running HTTP 1.0 and doesn't understand the if-none-match header
      - –So modification date takes priority over the entity tag
  - if-match (Request header)
    - return the resource only if the value of the Etag of the resource is the same as what was sent in this header
    - used with POST and PUT requests, only do the update if the resource has not been modified, server sends back a 412 (precondition failed) if they don't match
    - might be used if you want to update a resource, such as a wiki page - you wouldn't want to spend 3 hours writing a wiki page edit, only to submit it and overwrite a person who submitted a 5 minute change while you were working

# Servlets

- A Servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP
  - the servlet specification is defined for Java, not other languages
  - but this doesn't mean you can't write a program that does the same thing in another framework
    - django (python), ruby on rails, etc.

(notes on creating servlets in eclipse below, under "useful stuff")

- Web component (understands HTTP)
- generates **dynamic content** (content that depends on input data)

- managed by a **container** throughout it's lifecycle
    - instantiates a servlet, runs it, then destroys it
        - corresponding methods inside the servlet: **init**, **service**, **destroy**
    - provides network services for requests and responses
    - must implement both HTTP 1.0 & 1.1
    - may support HTTPS or other response/requests-based protocols, but not required
    - may perform caching
        - may respond to requests without delivering them to the servlet (delivers resources from servlet container cache instead)
        - may modify requests or responses before sending them to the servlet/client (ie. adding 'Date' header)
    - Containers DO NOT SYNCHRONIZE access to your servlet
        - Solution for Java: simply add "synchronized" to the method definition
- Specification: http://download.oracle.com/otndocs/jcp/servlet-3_1-fr-eval-spec/index.html
- Context Root: identifies a web apps in a Java EE server. Think of this as a pointer/reference that allows the web container to identify a web application
- Web.xml: 'Deployment descriptor' which tells the Servlet container where to route incoming requests. Use web.xml to define: Servlet's name, Servlet mapping, Servlet's parameters, when to start a Servlet (ie <load-on-startup>)
    - <load-on-startup> (see definition in Servlets question section below)
- Servlet Interface:

```
public interface Servlet {
        void init(ServletConfig config)
        void service(ServletRequest req,   ServletResponse res)
        void destroy()
}
```

- HTTPServlet:

```
public abstract class HttpServlet {
        protected void doGet(HttpServletRequest req, HttpServletResponse res)
        protected void doPost(HttpServletRequest req, HttpServletResponse res)
}
```

# Passing Parameters to Servlets

- Query Parameters
    - passed with resource URL as a query string
        - query strings start with "?"
        - parameters are separated by "&"
        - example: www.google.com/search?q=modern+web+apps&fat=your+mom
            - note: "fat" is not a real parameter that works on google, "q" is
        - Awesome: http://www.google.com/search?q=%73%75%70%65%72%6C%6F%67%6F%75%74&btnI
        - If we allow URL parameters we could be vulnerable to URL injection, if we don't defend against it
- Request Headers
    - we can set arbitrary headers
    - convention is to start the header with "X-"
- Request Body
    - Params can be sent in the request body. (ie. use 'query-parameters' doPost)

# Sessions

- What is a session?
    - Application defined set of logical actions that have some app-specific semantic meaning
- Why are sessions needed
    - To distinguish between different users in a user based web app
    - To save cart items in a shopping website
    - To customize and personalize displayed information (ie. Netflix recommendations)
- Persistent connections
    - One approach for server to track requests BUT a persistent connection socket is not a necessary condition
    - Different approach than Sessions:
        - Persistency is a property of the transport mechanism (a long-lived Socket)
        - A Session is a property of a group of HTTP requests (either request in part of Session or its not) * requests belonging to a Session use the same persistent connection
- How to implement (to track requests)
    - generates a unique session_id and sends it to the client
    - client sends that session_id to the server with every request, so the server associates those requests with that client

- Session Cookies:
  - Why are cookies called cookies
  - Approach 1: embed in a response header (cookies)
    - What is a cookie?
      - small piece of info sent by the server in a response header for tracking / identification purposes
    - Want to see your cookies at any site? Type this in the url bar and hit enter:
      - javascript:alert(document.cookie)
  - Max age for cookies have "seconds" for units. Negative value means cookie won't be persistently stored
  - Server uses response header to send Client uses request header to send in subsequent requests
  - What he means by sub-resources and parent resources when setting cookies
    - Say you were visiting cs.utexas.edu → if you set a cookie for, say, *.utexas.edu, then visiting yourmom.utexas.edu would also have the cookie sent as part of the request, but if you set a cookie for cs.utexas.edu then visiting yourmom.utexas.edu would NOT have the cookie sent
  - **Security issues:**
    - Session Fixation: attacker sends links to victim with sessionID embedded.. When user clicks it the attacker gets control
    - Session Hijacking: attacker injects attack JavaScript into site, and JS copies sessionID cookie. Prevent with "HttpOnly" cookie attribute
    - Insecure Cookies: "Man in the middle attack" where attacker observes network traffic between Client and Server. Prevent this with the "Secure" attribute (only transfer cookie over HTTPS)
  - **Cookie attributes:** (see questions section at bottom)
- URL Rewriting: Embed within the resource URL itself
  - Encoding step appends to the url
  - Useful when browser does not support cookies
  - How to expire a session: remove session id from all URLs
  - Issues: Unsafe
    - Session id is visible within the urls
    - Must consider ALL possible urls within a workflow to encode

# Unit Testing

- How to ensure a "unit of code" is working as expected? → unit testing
  - In other words, unit testing is how to make sure your program works even in situations that you have .
- Example:
  - doGet() from HelloServlet
    - what would unit testing this require?
      - verifying that:
        - when the method is called, "Hello world." is returned (println() is called)
        - HttpServletResponse's getWriter() method is called
      - if these two requirements are met, then the unit is behaving as expected
- A single unit test is only supposed to test a single case (i.e. if(true) separate unit test from if(false))
- Some unit testing libraries: Junit (j-unit), Mockito
- unit testing examples on course github
  - Servlets/servlet-unit-test
  - Servlets/mockito-example

# Stages of Unit Testing

1. Decompose your app into small units
2. Set up the expectations for the code unit under test
3. Invoke the code unit under test
4. Verify the result

(Step 2) Expectation setting & verification

- How do we set expectations on dependencies?
  -

# Mock

- an object that can be used for testing purposes in place of the real dependency
  - provides a way to set values that would be returned when a method is called on the object
  - provides a way to verify that a method was called on an object (even if the method was called multiple times)
- See Piazza @102 for general motivation and usage

# Spring Framework

- is an application container
  - needs to be put in pom.xml
- `ApplicationContext`
  - Central interface to provide configuration for an application
  - `WebApplicationContext`
    - interface to provide configuration for a web app
    - contains getServletContext()
  - How to get access to ApplicationContext within controller bean?.....
    - implement the ApplicationContextAware interface
      - public class EmailController implements ApplicationContextAware{...}

# ServletConfig

- contains initialization parameters
- A servlet container creates a ServletConfig when initializing a servlet
- the container passes the ServletConfig to the servlet when the init() method is called
  - i.e. `void init(ServletConfig config) throws ServletException{...}`
- in web.xml

```
<servlet>
	<servlet-name />
	<servlet-class />
	<init-params>
		<param-name>password</param-name>
		<param-value>abcdefg</param-value>
	</init-params>
	<load-on-startup>1</load-on-startup>
</servlet>
```

*Load-on-startup: A zero value on **load**-on-**startup** means that point 1 is executed when a request comes to that **servlet**. Other values means that point 1 is executed at container **startup**.

# ServletContext

- The context that the servlet is running in
- can be obtained from a ServletConfig (config.getServletContext());
- contains the metadata of a web application, basically the parameters that apply to ALL servlets defined in the web.xml
- in web.xml

```
<context-param>
	<param-name>password</param-name>
	<param-value>abcdefg</param-value>
</context-param>
```

- if you try to `getServletContext()` directly from doGet() it won't work unless you have saved it to a variable in your init()

# Servlet Container

- where all servlets live, like the actual web server (e.g. tomcat).

# Logging

- Record interesting events in our applications
  - config messages, debug messages, warnings, exceptions, diagnostic messages
- The old way:
  - System.out.println()
    - issues: not fine-grained or flexible enough - cannot selectively control which messages are printed or where they are printed to
- How it should be:
  - should support different categories of messages
  - should be able to specify where the messages go (file, socket, console, etc)
- Java Logging Architecture
  - **Loggers:** generate messages
    - decides whether a message should be handled or not and passes them to the handler

        - a logger is **not** responsible for sending a message to where it needs to go
    - **Handlers**: filter messages
        - decides whether or not a message should be written, then passes it to the appender
        - Some types: {Stream, Console, File, Socket, Memory}Handler
    - Why filter at both the Logger level and the Handler level for the (severity)? Why not use just one filter?
        - Because ~~each **Handler** has it's own **Appender** (corresponding to an individual output)~~
            - each handler corresponds to an individual output (see types above)
        - Explained by example:
            - Pretend we want to log ALL severities of log messages to **file**, but we only want to see WARNING or SEVERE come to the **console** (or pick any other two outputs and severities instead). We would have to allow all severities of log messages to pass the **Logger** level filter, but then we'd have a **console**Log**Handler** that would filter out all logging less severe than WARNING and a **file**Log**Handler** that would filter out … nothing.
        - This way it gives us one big, powerful filter at the Logger level for which we can turn off all log messages of a certain severity level or lower, and smaller filters that let us control each output. This is happens in Eclipse, sometimes it gives you warnings or error messages that didn't show up in your console.
    - **Appenders**: write messages to outputs (correction: appenders are not part of the java logging framework)
- **Log level severity order:** (Source: https://docs.oracle.com/javase/6/docs/api/java/util/logging/Level.html):
    - SEVERE: a message level indicating a serious failure. In general SEVERE messages should describe events that are of considerable importance and which will prevent normal program execution. They should be reasonably intelligible to end users and to system administrators. This level is initialized to 1000.
    - WARNING: a message level indicating a potential problem. In general WARNING messages should describe events that will be of interest to end users or system managers, or which indicate potential problems. This level is initialized to 900.
    - **INFO:** a message level for informational messages. Typically INFO messages will be written to the console or its equivalent. So the INFO level should only be used for reasonably significant messages that will make sense to end users and system admins. This level is initialized to 800.
    - **CONFIG:** a message level for static configuration messages. CONFIG messages are intended to provide a variety of static configuration information, to assist in debugging problems that may be associated with particular configurations. For example, CONFIG message might include the CPU type, the graphics depth, the GUI look-and-feel, etc. This level is initialized to 700.
    - **FINE**: a message level providing tracing information. In general the FINE level should be used for information that will be broadly interesting to developers who do not have a specialized interest in the specific subsystem.FINE messages might include things like minor (recoverable) failures. Issues indicating potential performance problems are also worth logging as FINE. This level is initialized to 500.
    - **FINER:** indicates a fairly detailed tracing message. By default logging calls for entering, returning, or throwing an exception are traced at this level. This level is initialized to 400.
    - **FINEST**: indicates a highly detailed tracing message. This level is initialized to 300.
    - **ALL:** indicates that all messages should be logged. This level is initialized to Integer.MIN_VALUE.

# REST

- he recommended the second book because he didn't feel that the one we have been reading from covered it well enough
    - we will be using the second book for this topic
- REST: REpresentational State Transfer
    - A methodology (HTTP methods) for the manipulation and management of web resources through different "state representations"
    - State representation: different values for attributes of a resource (example → account=AE vs account=YOURMOM)
- SOAP: Simple Object Access Protocol
    - Obsolete
- REST principles
    - Addressability
        - Every object / resource is uniquely identified (i.e. xyz.com/departments/cs/yourmom/herphonenumber.txt)
    - Uniformed Constrained Interface
        - Essentially, we are going to stick to GET, POST, PUT, DELETE, and PATCH methods
            - each resource is accessible with only a few familiar actions
        - Interoperability - HTTP is universal
        - POST ( = create = C ) - create a new resource
        - GET ( = read = R ) - get info about resource
        - PUT ( = update = U ) - update existing resource
        - DELETE ( D ) - delete existing resource
        - **Idempotent actions** - more than one request has the same effect on the system as only one request
            - GET, HEAD, PUT, DELETE are idempotent

        - GETting a resource multiple times doesn't change the system
        - PUT (using the same data) multiple times doesn't change the system more than once
        - DELETE - obvious, can only delete something once
        - POST?? → changes last modified time?
    - Representation Oriented
        - REST service is addressable through a specific URI and representations are exchanged between the client and service
    - Communicate Statelessly- should not keep client state on the server
        - Track sessions and data using: URL rewriting, Cookies, Request body params
    - Hypermedia As The Engine of Application State (HATEOS)
        - One resource may point to another resource via a <link>
- Recipe for designing REST APIs
    - Identify potential resources on the system
        - Hint: look for nouns.
    - Map actions to the identified resources
    - Identify actions that cannot be mapped to resources
    - Refine resource definitions by adding new resources in the resource model if required
    - Introduce resource representations
    - Re-map actions by establishing resource hierarchies
    - Repeat if needed

# XML / HTML parsing

- One option → Tree parsing
    - DOMParser
- Another → search for a string ("parsing using callbacks")
    - SAXParser
    - Register callback handler with parser
    - Read file line by line
    - Invoke callback handler
- We can use these parsing techniques for both XML, and for HTML that is "well formed"
- XPath parser
    - Queries are specified using "path expressions"
        - example: /cs378/assignments/assignment
    - Also cannot parse HTML that is not well-formed
    - this is the "current way of doing things" and is recommended
- Parsing HTML
    - expression based parser (regex)
        - RegexParser → match patterns
    - using a library such as jsoup (http://jsoup.org)
        - JSoupParser
    - parsing disadvantages
        - "brittle" - can break easily, if the webpage changes our program might not work anymore
        - no formal contract defined, cannot validate the html document

# Functional Testing

- Testing from the end user's perspective whether or not the program is behaving as expected
- Unit testing is for individual layers - Functional testing is for the entire application
- No mocks in functional testing
- If functional tests are written correctly, changing the code should not affect if they pass or fail
    - meaning the steps in between / underneath can change as long as the overall outcome is the same

## Databases

- Persistent Storage
    - Supports storage whether application program is running or not (i.e. files on disk, databases)

- Non-persistent storage
    - RAM
    - Example: Objects in Java applications only exist while the program is running
- A database is persistent storage that is contained in one or more tables and is managed by a database server
- SQL is a declarative language to retrieve data from a database table
    - Declarative language is something that allows you to tell the computer what you want or need, and it performs the computation on its own. The opposite would be writing instructions on how to retrieve the data
    - CRUD == SQL == REST analogy
        - Create == Insert == POST
        - Read == Select == GET
        - Update == Update == PUT
        - Delete == Delete == DELETE
- Helpful MySQL Commands
    - Create Database
    - Create Table
    - Grant Privileges
        - GRANT ALL ON <dbname> TO "user"@"localhost"
    - Use <db>
        - Switches context over to that db
    - Show tables
        - Gives some ASCII art of the available tables in the current db
- Helpful SQL Commands
    - Select *
    - Select column name
    - Auto_increment
    - Allow nulls
    - Set primary key
    - Insert
    - Update
    - Delete
    - Join
- Joining
    - Left outer
    - Right outer

  Keys - a way of indexing and pointing to a unique entry in a table
    - P.S. this key stuff gets really silly and complicated, take a database class or read about it
    - Natural - already exists in the table
    - A **primary key** is a **key** in a relational database that is unique for each record. It is a unique identifier, such as a driver license number, telephone number (including area code), or vehicle identification number (VIN). A relational database must always have one and only one **primary key**. A natural key can be a primary key
    - A candidate key is a key that uniquely identifies a row, and a primary key is one of the candidate keys in a table (your choice). A table can have many candidate keys, but only one primary.
    - Surrogate - we generate them, can be random or ordered, just extra data
        - (is this a foreign key?) A foreign key is a column from another table. When you make and declare a foreign key if that other column gets updated, it is generally supposed to cascade updates to the other tables that are using that foreign key.
- Join - A way of combining more than one table, think of it sorta like multiplying matrices - we just combine the data so we can cross reference it
- DBs are faster than file systems because of indexing?
    - Indexing, the fact that they skip OS calls and handle storage themselves, and having highly optimized algorithms
- Param-value should be empty when we turn it in
    - Should anything else be empty?
- Get the ID after insert use Statement.RETURN_GENERATED_KEYS
- Transactions
    - Execute multiple statements simultaneously
    - They all work or they all don't
    - Atomic (done as a unit)
        - Transaction is the abstraction for allowing atomicity

# JDBC

- DataSource
    - Setup data source
- Connection Class

# Hibernate ORM

- ORM → object / relational mapping
    - Maps (objects ←→ database tables)
- JPA - java persistence architecture - an API specification for java ORM
    - Hibernate, iBatis, MyBatis, EclipseLink
- Configuration:
    - hibernate.cfg.xml
        - src/main/resources/hibernate.cfg.xml
        - Important fields:
            - Connection settings
            - Cache provider
            - ...more (examples on github)
- Entities = Tables and relationships
    - @Entity - annotation - hibernate treats all instances of these as rows that need to be saved in the database
- Sessions
    - session.save(newAssignment); = a commit - ready to be sent to database
    - transaction.commit() = a push - send to database, create rows in table
    - transaction.rollback() = discard previous stuff (up to beginTransaction()) - for use when errors occur
    - session.close() - hibernate won't close the session for you when you go outside of the scope of the method
-

# Useful stuff seen in class

## Tools

Chrome extension he used to see HTTP status code: REST Console
Recommends Eclipse/IntelliJ/Netbeans for HW 1 and will be using Eclipse/IntelliJ/Netbeans in the futureEclipse Luna
Tomcat (Servlet container)

## Programs

TestClientSocket

```
$> java TestClientSocket  [base domain/url] [path to file] [format output, i.e. “txt”]
         or in his words  [host]            [resource]     [outputType]
       Example: java TestClientSocket www.cs.utexas.edu /~devdatta/index.html txt
```

TestUserAgentHeader

```
$> java TestUserAgentHeader [url] [type]
```

TestURLConnection

```
$> java TestURLConnection [url]
```

## Console

curl
-i <url>
output <url> to terminal in text form
-H <header>
specify headers conditions that must be satisfied for the request to generate a response

## Eclipse

See Servlet-1 slides starting at 12 for setting up Eclipse + Tomcat
Create a new “Dynamic Web Project”

- Name it
- Target runtime set to: Apache Tomcat v8.0

This creates
Web content directory
Java Resources → src directory

- Right click inside to create Servlet

in the doGet method
response.getWriter().println(“Hello world.”);
**OR**
Go to the class github and download the pre-created projects
Go to import → existing Maven projects → browse to the project

- (Apache) Maven - allows you to specify dependencies for your project
- right click on the project → Maven → update project

**pom.xml** - defines libraries and dependencies that our program needs so that they can be imported

- stands for “project object model”
- include GSON library in your pom.xml if you want to use it

**web.xml** - defines the url(s) that the servlet should handle requests to

# Extra notes (anything you want, homework notes, whatever)

2/1/2016 Extra Assignment

Read **Persistent Connections -** will not be covered in lecture.

Begins on Slide #58 in HTTP lecture.

2/3/2016

HW2 on Canvas

2/24/2016

Read about request mapping from the textbook

example: @RequestMapping(value="/email")

@RequestMapping("/email") → these two are the same thing

simple explanation: put before a method or class to make that method or class be called when the value / resource is met

## Assignment 2:

@ResponseBody

spring generates response body in text/html

this can be changed to json (somehow, didn't specify in class - maybe in chapter 13 of the book)

## Assignment 3:

Create a REST API (there is a step by step slide on this)

eavesdrop.openstack.org/meetings

also we will implement functional tests

## Assignment 4:

## Assignment 5:

## Assignment 6:

## Assignment 7:

# Midterm Review

- Book Chapters
  - Java for Web Applications: 1, 2, 3, 5, 9, 11, 12, 13, 14
  - RESTful Java with JAX-RS 2.0: 1, 2, 3, 4, 5
- Framework allows you to split your code into resources and services, or controllers and services
- Setter injection - you have your dependency introduced through a setter method
- Why is developing against an interface good? http://stackoverflow.com/questions/4456424/what-do-programmers-mean-when-they-say-code-against-an-interface-not-an-obj
- REST reuses http methods to define an architecture on the state of the resources on the server
- Know the different logging levels
- Know the difference between unit testing and functional testing
  - Unit testing tests a specific part of your code, functional testing tests your code as a whole if it works or not
- Marshalling - converting Java objects into JSON/XML
  - Advantage is that you are able to search very easily by querying against the java object
- Unmarshalling - going the other way around (JSON/XML to POJO)
- If the logger default level is null, it uses its parent's level of logging

## Topic Notes from the slides on canvas

(there are questions and concepts that aren't covered in the other questions)

### HTTP Protocol

Request/response protocol
Versions 1.0, 1.1
Headers
    Request/response headers
    Cache control headers
Transfer-encoding: chunked
Session tracking mechanisms

### Servlets

Deployment descriptor
Servlet container
ServletContext
    Represents the context for an entire web application (all Servlets)
    Initialization information to all the servlets can be obtained from ServletContext
ServletConfig
    Use to initialize a specific servlet
Servlets
    doGet, doPost
    HttpServletRequest, HttpServletResponseobjects
    Concurrency
Servlet filters

### Layered Architecture

Splitting your code into layers
    Controllers and Services (Spring)
    Resources and Services (JAX-RS)
Developing against an Interface
Concept of dependencies

- Dependency Injection
  - Setter injection
  - Constructor injection

## Spring

- Framework that eases writing of web applications
- DispatcherServlet
- Annotation-based
  - @Controller, @Service, @PathVariable
- Beans
- Java classes that satisfy certain criteria
  - Have getter and setter methods for the properties
  - Setter method names follow a specific format

## REST and RESTEasy

- Framework that eases writing of REST services
- REST architecture principles
- Annotation-based
  - @Request, @RequestBody

## Logging

- Purpose
- Java logging architecture
  - Loggers, Handlers
- Logging Levels
  - Message levels
  - Logger and handler levels

## Unit Testing

- What is it?
  - Testing logical units of code
    - Comes down to testing specific methods
- How to do?
  - Identity "code under test"
  - Identify the dependencies
  - Create mock dependencies
  - Set expectations
  - Invoke the "code under test"
  - Assert output is as expected
  - Verify that certain methods were called/not called

## Marshalling/Unmarshalling

- Marshalling
  - Converting Java objects to JSON/XML
- Unmarshalling
  - Converting JSON/XML strings to Java objects

## JSON/XML/HTML Parsing

- Parsing techniques
  - JSON
    - Gson

XML
- DOM Parsing
- SAX Parsing
- XPath
- Java regular expression parsing

HTML
- Jsoup

## Questions from the slides on canvas

# HTTP

**1. When is a particular HTTP header used?**

Authorization:
A user agent can authenticate itself with the server by including "authorization" header; usually but not always used after receiving a 401 header

If-Modified-Since:
The If-Modified-Since request-header field is used with a method to make it conditional: if the requested variant has not been modified since the time specified in this field, an entity will not be returned from the server; instead, a 304 (not modified) response will be returned without any message-body. If the content has changed the server responds to the request with a 200 status code and the entire requested document / resource.

Set-Cookie:
used to track the history of the user. Included in the cookie is a description of the **range of URLs** for which the state is valid. Any future HTTP requests made by the client that fall in that range will include a transmittal of the current value of the state object from the client back to the server.

User Agent:
Used to let the server know what software is being used. Essentially, a user agent is a way for a browser to say "Hi, I'm Mozilla Firefox on Windows" or "Hi, I'm Safari on an iPhone" to a web server.

**2. What is the meaning of a particular HTTP header?**

User-Agent :
Identifies what software is acting on behalf of the user.
Most web browsers use a user string like:

Example:

```
Mozilla/5.0 (iPad; U; CPU OS 3_2_1 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML, like Gecko) Mobile/7B405
```

***Is the above a single example?***

Cookie:
A small piece of data sent from a website and stored in the user's web browser while the user is browsing.
Every time the user loads the website, the browser sends the cookie back to the server to notify the user's previous activity

Date:
The Date and Time the messages was sent.

If-Range :
If entity is unchanged, just send me the parts that i'm missing; otherwise send me the entire new entity

If-Unmodified-Since:
only send response if entity has not been modified since a specific time.

Accept-Ranges :
What partial content range types this server supports via byte serving

Range-Request :
Request only part of an entity. Bytes are numbered from 0.

Age :
The age the object has been in a proxy cache in seconds

max-age :
let caches know when an asset is expired by using the aptly-named "expires" header.

**3. What will be response if the request contains a particular HTTP header?**

Last Modified:
shows the age of requested object. Is returned if request has a modified type header

---

**4. As a web application developer what are some of the ways to ensure that the clients never have to work with stale data?**

Pragma : no cache or Cache-Control : no cache will stop the browser from keeping cache but will have to load every time, including when there is no new data.
The better way is set the Last-Modified header appropriately and let the client decide if it's stale.

---

**5. What mechanisms are available for tracking user session across requests?**

To track a user's session, the server asks the client to store a cookie using Set-Cookie response header,
Now that the user has been assigned a unique cookie, all subsequent requests will be tracked to that
User by checking the cookie header value
URL Rewriting
Persistent Connections

## Servlets

**6. What advantages do Servlets provide over using in-built Java networking classes such as URLConnection?**

Servlets don't have to re-initialize for a new request, it can just start a new thread. So servlets get the benefit of being written in java but without the slower time.
**I thought it was mostly about the abstraction. Being abstracted from a lower level makes development faster and less prone to error (since there are less moving parts). Is this wrong?**

**7. What are some of the mechanisms available for passing parameters to a Servlet?**

(a) - via URL
i.e. http://localhost/myservlet/?myparam=foo
(b) - via web.xml as initial parameters,
i.e.:

```
<init-param>
        <param-name>email</param-name>
        <param-value>admin@email.com</param-value>
</init-param>
```

(c) - via request headers by setting arbitrary headers _____
(d) - via Request body example: Use 'query-parameters' doPost – Use RESTClient Firefox add-on

**8. Identify what is wrong/missing in <some_piece_of_Servlet_code>**

Needs answer.

**9. Show web.xml to satisfy some requirement related to Servlet configuration**

Needs answer.

**10. What are differences between Cookies and URL rewriting methods for session tracking?**

(a) - Cookies are included by the server in the HTTP Response header, and are included by the browser in every subsequent HTTP Request header, for as long as the session continues.

(b) - URL rewriting stores the session ID in the URl, i.e.

http://www.javapractices.com/topic/TopicAction.do;**jsessionid=863F3D316**?Id=191

According to that link, URL rewriting is more dangerous, since the session ID could be more easily intercepted by a third party.

**11. What does a specific Cookie attribute mean?**

Secure

only send if https page

aids in securing the cookie from being accessed by a client side script

Domain

Verify that the domain has not been set too loosely. As noted above, it should only be set for the server that needs to receive the cookie.

For example if the application resides on server app.mysite.com, then it should be set to "; domain=app.mysite.com" and NOT "; domain=.mysite.com" as this would allow other potentially vulnerable servers to receive the cookie.

Path

Verify that the path attribute, just as the Domain attribute, has not been set too loosely.

HTTPOnly

This attribute should always be set even though not every browser supports it. This attribute aids in securing the cookie from being accessed by a client side script so check to see if the "; HttpOnly" tag has been set.

Expires

Verify that if this attribute is set to a time in the future, that it does not contain any sensitive information.

**12. What are some of the use cases for Servlet filters?**

Servlet filters intercept the HttpServletRequest object before it reaches the service method, and intercept the HttpServletResponse object after it leaves the service method.

Servlet filters transform the request or response in a way that makes it easier to modularize certain functions, such as: Tracking users, blocking certain users based on their identity, scaling images, compression, and geographic localization.

More here: http://www.oracle.com/technetwork/java/filters-137243.html

**13. What would be printed when following filters are run?**

Needs answer.

## Layered Architecture

**14. What are the advantages of developing against an Interface?**

1. From a high-level point of view, interfaces are contracts - they guarantee that implementations of the interface implement all of the specified functionality. This lets other pieces of software interact with **just** the interface, and not care about the particulars of any given implementation.

This answer is really useful as an analogy: http://programmers.stackexchange.com/a/108681

2. Makes writing unit tests using mocks easier. There's a good explanation here in the top answer: http://stackoverflow.com/questions/4456424/what-do-programmers-mean-when-they-say-code-against-an-interface-not-an-object

**14.5 What is dependency injection?**

Code that you write has **dependencies**, which are just objects that your objects need references to. Dependency injection means you don't have to manually create those dependencies whenever you want to use the objects you've created , and comes in two flavors: setter injection, and constructor injection.

**15. How does setter injection work?**

The required dependency is set via a setter method of the object requiring the dependency.

**16. How does constructor injection work?**

The required dependency is passed as an argument to the constructor of (i.e. the dependent).

## Spring

**17. What are different components within a Spring-based application setup? In other words:** In order to write a spring based app, what are the things you need to consider?

Within the beans file - what kind of dependency injection you are using

... more

Needs answer.

**18. What is the purpose of servletContext.xml file?**

It is the Spring Web Application Context Configuration. It's for configuring your Spring beans in a web application.

**19. Write a Spring Controller for some given requirement: Ex: Hello World**

```
@Controller
public class helloWorldController {

        @ResponseBody
        @RequestMapping(value = "/helloworld")
        public String helloWorld()
        {
         return "Hello World!";
       }

}
```

**20. Write a unit test for a given Spring service method**

Needs answer.

**21. Use constructor injection to setup a Spring controller with a Spring bean**

Reference - http://www.tutorialspoint.com/spring/constructor_based_dependency_injection.htm

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <!-- Definition for textEditor bean -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
      <constructor-arg ref="spellChecker"/>
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
```

```
   </bean>

</beans>
```

**22. Use setter injection to setup a Spring controller with a Spring bean**

Reference - http://www.tutorialspoint.com/spring/setter_based_dependency_injection.htm

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <!-- Definition for textEditor bean -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
      <property name="spellChecker" ref="spellChecker"/>
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
   </bean>

</beans>
```

# REST and RESTEasy

**23. What is the uniformed constrained interface in REST design**

It's an interface that provides finite set of actions, ultimately providing familiarity and operability, such that anyone can use the application without having to know the internals of it. Actions are: CRUD.

**24. Identify whether following methods are idempotent or not**

Idempotent means that you can apply the operation a number of times, but the resulting state of one call will be indistinguishable from the resulting state of multiple calls. In short, it is safe to call the method multiple times. The only non-idempotent HTTP operation is POST.

From the RFC

**4.2.2. Idempotent Methods**

A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. Of the request methods defined by this specification, PUT, DELETE, and safe request methods are idempotent.

---

If it ends with POST, I'd say no. Any objections? Yeah sounds good too me.

Well it depends... If it requests were: GET a GET a PUT a DELETE a POST a, that would be idempotent, right?

The only non-idempotent HTTP operation is POST. - true, I see your point. This is an idempotent OPERATION, but when thrown in a sequence the system could end up the same as it started.

Oh, the answer is below.

A sequence is idempotent if a single execution of the entire sequence always yields a result that is not changed by a re-execution of all, or part, of that sequence.

If you did this sequence once, then re-did it only halfway through, the system would be changed from the state the first iteration put it in.

So I guess the moral of the story is "if evaluating a sequence for idempotency, finish the sequence once, then ask if you can do it again and stop at any point in the middle without ending up with a different result than the first time it finished."

Wouldn't that mean that any call which modified server data could be considered non-idempotent? A PUT, DELETE or POST all change the data on the server so if I was to perform a GET after making any of those calls the result would be different....

**25. Identify whether following sequence of methods is idempotent or not**

<Put, Put> is a non-idempotent sequence (see below), but <Get,Get>, <Put>, <Delete>, and <Delete,Delete> are all idempotent sequences.

Why is <put, put> non-idempotent?

Answer:

Given the definition of idempotent in the above question, <put, put> may or may not be idempotent depending on whether or not you put the same data twice.

So if it was <PUT, PUT> with the same input data it would be idempotent?

So for any single method called, only POST is idempotent but with multiple methods in a sequence either PUT or POST can be considered non-idempotent? What about <GET, DELETE> or <GET, DELETE, GET>?

What is confusing about this is that the RFC defines idempotent METHODS, not idempotent sequences.
But they are defined in our slides, copied below.

Idempotent sequences

– A sequence is idempotent if a single execution of the entire sequence always yields a result that is not changed by a re-execution of all, or part, of that sequence.

- Idempotent sequence:
  - <GET, GET, GET>,
  - <PUT>,
  - <DELETE>,
  - <DELETE, DELETE>
- Non-idempotent sequences:
  - <PUT, PUT>

## What about <GET, DELETE, GET> (all above sequences have the same method)?

Is a valid way to view if a sequence is idempotent imagining a GET call in-between each call of the sequence and confirming if all GET calls return the same thing?
Or is it if performing a sequence multiple times at the end of the sequence a GET call would return the same thing?
Or is it that if a sequence is called multiple times each step in each sequence must have the same return?

**26. In RESTEasy, describe what is the ApplicationClass**

The ApplicationClass tells the application server which JAX-RS components to register with the JAX-RS runtime.

Has two methods

public Set<Class<?>> getClasses()

•Get a set of root resource *classes*

•Default life-cycle for a resource class instances is 'per-request'

public Set<Object> getSingletons()

•Get a set of *objects* that are *singletons* within our application

–These objects are shared across different requests

## Logging

**27. Indicate what would be logged from <a_piece_of_code>**

Where can we see examples of this? -thanks - Reference - https://www.youtube.com/watch?v=BHNN_UI31_g

Ex.

```
Public class Nose{
```

```
        //get logger for the named subsystem. If logger has already been created with the given name it is
        //returned. Other wise a new logger is created.
        Private static Logger logger = Logger.getLogger("com.Wombat.nose");

        Public static void main(){
                //Log a FINE tracing message
                logger.fine("doing stuff");
                Try{
                        Wombat.sneeze();
                } catch (Error ex){
                        //Log warning message
                        logger.log(Level.WARNING, "trouble sneezing", ex);
                }
                logger.fine("done");
        }
}
```

The messages that appear will depend on what the level is set to.
If the level is only set to WARNING, then the only log will be "trouble sneezing" if it happens.

Log level descending in importance:

- SEVERE
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST

---

**28. As an application developer what logging level would you choose for <a_particular_situation>?**

SEVERE

should describe events that are of considerable importance and which will prevent normal program execution.

WARNING

should describe events that will be of interest to end users or system managers, or which indicate potential problem.

INFO

should only be used for reasonably significant messages that will make sense to end users and system admins.

CONFIG

intended to provide a variety of static configuration information, to assist in debugging problems that may be associated with particular configurations. For example, CONFIG message might include the CPU type, the graphics depth, the GUI look-and-feel, etc

FINE

should be used for information that will be broadly interesting to developers who do not have a specialized interest in the specific subsystem. FINE messages might include things like minor (recoverable) failures. Issues indicating potential performance problems are also worth logging as FINE.

FINER

indicates a fairly detailed tracing message. By default logging calls for entering, returning, or throwing an exception are traced at this level.

FINEST

indicates a highly detailed tracing message.

ALL

indicates that all messages should be logged.

**29. What is the difference between logging API and logging implementation?**

The logging API is an interface, while the logging implementation is the actual code for that interface.
With this approach you can easily swap out an implementation without having to change your code.

**30. How is a Logger's logging level set if its default level is "null"?**

If loggers level is null, it will inherit the level of its parent.
To set level: `log.setLevel(Level.INFO)`

## Unit Testing

**31. Given a method definition, identify all its dependencies from unit testing point of view**

Needs answer.

Includes for example the method's parameters like doGet (HTTPResponse response, HTTPRequest request)

**32. Given a method definition, write unit test(s) for it**

Needs answer.

**33. Given a piece of code, refactor it to enable writing of unit tests for it**

Tutorial on how to refactor code for testing :
https://www.kenneth-truyers.net/2012/12/15/how-to-unit-test-and-refactor-legacy-code/
Rule of thumb:
Don't create application-level domain objects such as controller classes, service classes, and so on as part of a method's execution
Create them outside of any method
Create them inside constructor, or init method (for Servlets), or let the container create them and inject into your class
**Points to remember when mocking:
Mockito does not allow mocking of Final classes
- Cannot Mock URL class

Cannot call private methods from the test class
- Need to make methods either public or protected

## Functional Testing

**34. What is difference between unit testing and functional testing?**

Unit testing is for individual layers - Functional testing is for the entire application

**35. Write a functional test for a particular REST resource**

Needs answer.

## XML/HTML Parsing

**36. In your assignment 1, what was the advantage of unmarshalling the response from the austin-social-media site?**

The advantage of unmarshalling was that you could query against Java Objects instead of the given JSON, making it easier to see which accounts had the needed information (such as type and website URL).

**37. Given a JSON/XML/HTML response, write code to parse it using <a_parsing_method>**

Xml parse example - http://www.tutorialspoint.com/java_xml/java_dom_parse_document.htm
HTML Parser examples - http://www.mkyong.com/java/jsoup-html-parser-hello-world-examples/
JSON parse example - https://examples.javacodegeeks.com/core-java/json/java-json-parser-example/

**38. Given a XML snippet, write an XPath expression to find it**

Reference - **http://viralpatel.net/blogs/java-xml-xpath-tutorial-parse-xml/**

| Expression | Description |
|---|---|

| | |
|---|---|
| nodename | Selects all nodes with the name |
| / | Selects from the root node |
| // | Selects nodes in the document from the current node that match the selection no matter where they are |
| . | Selects the current node |
| .. | Selects the parent of the current node |
| @ | Selects attributes |
| employee | Selects all nodes with the name “employee” |
| employees/employee | Selects all employee elements that are children of employees |
| //employee | Selects all book elements no matter where they are in the document |

**Use a combination of terms above to find a specific snippet. Ex.**

| Path Expression | Result |
|---|---|
| /employees/employee[1] | Selects the first employee element that is the child of the employees element. |
| /employees/employee[last()] | Selects the last employee element that is the child of the employees element |
| /employees/employee[last()-1] | Selects the last but one employee element that is the child of the employees element |
| //employee[@type='admin'] | Selects all the employee elements that have an attribute named type with a value of ‘admin’ |

**Did we even talk about this XPath stuff? I do not remember it at all…..**

# HW1

**1) Try out all the programs from HTTP directory in: https://github.com/devdattakulkarni/ModernWebApps**

**2) Read about following headers from the RFCs:**

- **Accept-Ranges : What partial content range types this server supports via byte serving**
- **Range-Request : Request only part of an entity. Bytes are numbered from 0.**
- **Age : The age the object has been in a proxy cache in seconds**
- **max-age : let caches know when an asset is expired by using the aptly-named "expires" header**

**3) Interact with 5 different sites/urls of your choosing using either the Java programs, curl, or REST Console. Make a histogram of response headers.**

**Which headers were always sent?**

| | |
|---|---|
| ACCEPT | text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 |
| ACCEPT_ENCODING | gzip, deflate |
| ACCEPT_LANGUAGE | en-us |
| CONNECTION | keep-alive |
| HOST | www.whatismybrowser.com |
| REFERER | https://www.google.com/ |
| USER_AGENT | Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/601.2.7 (KHTML, like Gecko) Version/9.0.1 Safari/601.2.7 |

Reference - https://www.whatismybrowser.com/detect/what-http-headers-is-my-browser-sending

**Which headers are unique to two or more sites?**

Needs answer.

**4) Find a site that sends back Transfer-Encoding: chunked response header.**
**Calculate the size of the content by considering each chunk.**
**Tally it with the size of the content by measuring it independently (using other means such as Unix word count (wc) command)**

# HW2

**1) Experiment with using If-Modified-Since request header against a site of your choosing.**
**What is the effect of using different date values in the header field?**

If the requested resource has not been modified since the time specified in the field, a copy of the resource will not be returned from the server; instead a '304' status will be returned

**2) Experiment with using If-None-Match request header against a site of your choosing.**
**What happens when you use the value that matches the value in the ETag response header?**

**What happens when you use a different value than what is the ETag response header?**

Perform the action only if the value specified in the header does not match that of the Etag of that resource on the server

**3) Repeat above with If-Match request header**

Perform the action only if the value specified in the header matches that of the Etag of that resource on the server

**4) Experiment with using combination of If-Modified-Since and If-None-Match request headers against a site of your choosing. Determine through experimentation which header takes precedence.**

If-Modified-Since takes precedence.

**5) Suppose that there is a proxy server between client and the origin server and suppose that the resource that client is requesting is present in proxy server's cache. Explain what actions should the proxy server take to respond to the client's request.**

Needs answer.

**6) Consider a University network where all the network traffic goes through a University-wide proxy server which implements caching according to HTTP 1.0 and 1.1.**
**Suppose that a user makes request to resource (r1) and the response sent by the origin server for that resource has a Cache-control header with max-age: 10**

Needs answer.

**Suppose another user makes request to the same resource after two minutes, what will happen?**

Needs answer.

**Suppose a user wants to ensure that the response is only received from the origin server. How can this be achieved?**

Needs answer.

**7) Explain what is a strong validator? Use examples to explain.**

Strong validators:
A server changes the validator for every change in entity
Weak validator:
A server changes the validator only for semantically significant changes to an entity

**8) Answer true or false:**

**(a) If-None-Match request header is a way for the client to control when new ETag value is calculated by the server**

FALSE reference - http://odino.org/don-t-hurt-http-if-none-match-the-412-http-status-code/

**(b) The reason for the origin server to include both Expires header and the max-age directive in its response is to allow proxy servers that understand HTTP 1.1 longer expiration time for that resource by specifying the max-age value that is greater than the time included in the Expires header.**

**9) Explain what is Transfer-encoding response header.**

Needs answer.

To make requests against a site you can use curl command line tool. You can add headers in your curl request with '-H' command line flag. Example:

curl -i https://www.cs.utexas.edu/~devdatta/index.html -H 'If-None-Match: "17a1e7d-18f-52abf16d5d15a"'

You can also use browser plugins such as REST Console in Chrome or Poster on Firefox for making REST calls.

Static variable for single session factory thing

# FINAL EXAM STUDY

https://docs.google.com/document/d/1YqT1hCVuf74tWNuUwkyn1N6ygeGYVh18NjndDuJ4KVQ/edit#