Web APIs

API = Application Programmer Interface

CS 240 – Advanced Programming Concepts

Internet Basics: TCP

- TCP (Transmission Control Protocol): The protocol on which the Internet is based
  - Allows programs running on different computers to connect and communicate directly with each other
  - Requires that each computer have a unique identifier called an "IP Address"
    - 128.187.80.20
    - 72.30.38.140

Internet Basics: Ports

- TCP uses Port Numbers to identify individual programs running on a computer
  - TCP Port Numbers are in the range 0 – 65535
  - Ports 0 – 1023 are reserved for system services (email, web, etc.)
  - Ports 1024 – 49151 are registered to particular applications
  - Ports 49152 – 65535 can be used for custom or temporary purposes
  - Email servers typically run on Port 25
  - Web servers typically run on Port 80
- The combination of IP Address and TCP Port Number uniquely identifies a particular program on a particular computer
  - (128.187.80.20, 25) => Email server on machine 128.187.80.20
  - (72.30.38.140, 80) => Web server on machine 72.30.38.140

Internet Basics

- Through TCP, a program on one computer can connect to a program running on another computer by specifying its (IP Address, TCP Port Number)
  - Connect to (128.187.80.20, 25) => Connect to email server on machine 128.187.80.20
  - Connect to (72.30.38.140, 80) => Connect to web server on machine 72.30.38.140
- Such a TCP connection is called a "Socket"
- Once a connection has been established, the two programs can pass data back and forth to each other (i.e., communicate)

Internet Basics: DNS

- IP Addresses are hard to remember
- Users prefer to reference machines by Name rather than by IP Address
  - pinky.cs.byu.edu instead of 128.187.80.20
  - www.yahoo.com instead of 72.30.38.140
- DNS (Domain Name System) is a protocol for looking up a machine's IP Address based on its (Domain) Name
  - Connect to (www.yahoo.com, 80)
  - DNS, what is the IP Address for "www.yahoo.com"?
  - 72.30.38.140
  - OK, Connect to (72.30.38.140, 80)

URLs (uniform resource locators)

`scheme://domain:port/path?query_string#fragment_id`

- **scheme** (case-insensitive) – http or https
- **domain** (case-insensitive) – The server’s domain name or IP address. The domain name google.com, or its IP address 72.14.207.99, is the address of Google's website.
- **port** (optional) – The port, if present, specifies the server’s TCP port number. For http URLs, the default port is 80. For https URLs, the default port is 443.
- **path** (case-sensitive) – The path is used to specify and perhaps locate the requested resource.
- **query_string** (optional, case-sensitive) – The query string, if present, contains data to be passed to software running on the server. It may contain name/value pairs separated by ampersands, for example `?first_name=John&last_name=Doe`.
- **fragment_id** (optional, case-sensitive) – The fragment identifier, if present, specifies a part or a position within the overall resource or document.

URLs

`http://www.espn.com:80/basketball/nba/index.html?team=dallas&order=name#Roster`

- **scheme** – `http`
- **domain** – `www.espn.com`
- **port** – `80`
- **path** – `/basketball/nba/index.html`
- **query_string** – `?team=dallas&order=name`
- **fragment_id** – `#Roster`

The URL Class

```
import java.net.URL;
…

URL url = new URL(
”http://www.espn.com:80/basketball/nba/index.html?te
am=dallas&order=name#Roster”);

String host = url.getHost();
int port = url.getPort();
String path = url.getPath();
String query = url.getQuery();
String fragment = url.getRef();

// Many more URL operations
```

HTTP (hypertext transfer protocol)

- Network protocol that drives the Web
- Built on top of TCP
- By default, Web servers run on TCP Port 80
- HTTP has a Request/Response structure
  - Client (e.g., web browser) sends a “request” message to the server
  - Server sends back a “response” message to the client

HTTP Request message format

```
<method> <request-URL> <version>\n
<headers>\n
\n
<entity-body>
```

```
<method> is the operation to perform on URL
<request-URL> can be full URL or just the path part
<version> is of the form HTTP/<major>.<minor>
<entity-body> is a stream of bytes (could be empty)
```

```
GET /test/hi-there.txt HTTP/1.1
Accept: text/*
Host: www.joes-hardware.com

```

HTTP Response message format

```
<version> <status> <reason-phrase>\n
<headers>\n
\n
<entity-body>
```

```
<version> is of the form HTTP/<major>.<minor>
<status> is a 3-digit number indicating status of request
<reason-phrase> human-readable description of status code
<entity-body> is a stream of bytes (could be empty)
```

```
HTTP/1.0 200 OK
Content-type: text/plain
Content-length: 18

Hi! I’m a message!
```

HTTP Request Methods

- GET – Retrieve document from server
- POST – Send data to server for processing
- PUT – Store document on server
- DELETE – Remove document from server
- HEAD – Retrieve document headers from server
- OPTIONS – Determine what methods the server supports
- TRACE – Trace the path taken by a request through proxy servers on the way to the destination server

HTTP Response status codes

100-199	Informational
200-299	Successful
300-399	Redirection
400-499	Client error
500-599	Server error

200	OK
401	Unauthorized to access resource
404	Requested resource does not exist

HTTP Headers

- List of name/value pairs
- `Name: Value\n`
- Empty line separates headers and entity body

- General headers (request or response)
  - `Date: Tue, 3 Oct 1974 02:16:00 GMT`
    - Time at which message was generated
  - `Connection: close`
    - Client or server can specify options about the underlying connection

HTTP Request Headers

- `Host: www.joes-hardware.com`
  - Host from the request URL
- `User-Agent: Mozilla/4.0`
  - Client application making the request
- `Accept: text/html, text/xml`
  - MIME types the client can handle
- `Authorization: dfWQka8dkfjKaie39ck`
  - Authorization credentials to identify the user
- `Referer: http://www.joes-hardware.com/index.html`
  - Page that contained the link currently being requested
- `If-Modified-Since: Tue, 3 Oct 1974 02:16:00 GMT`
  - Conditional request; only send the document if it changed since I last retrieved it

HTTP Response Headers

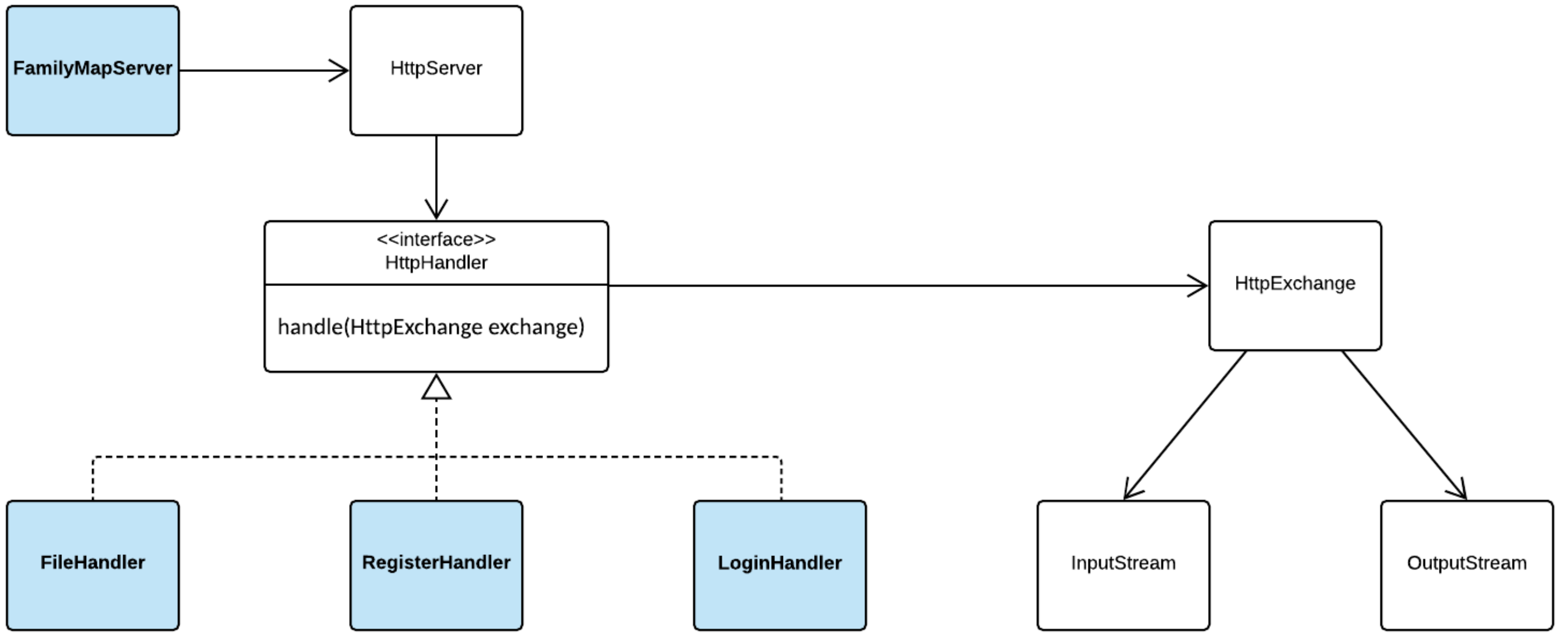
- `Content-length: 15023`
  - Length of response entity body measured in bytes
- `Content-type: text/html`
  - MIME type of response entity body
- `Server: Apache/1.2b6`
  - Server software that handled the request
- `Cache-Control: no-cache`
  - Clients must not cache the response document

HTTP

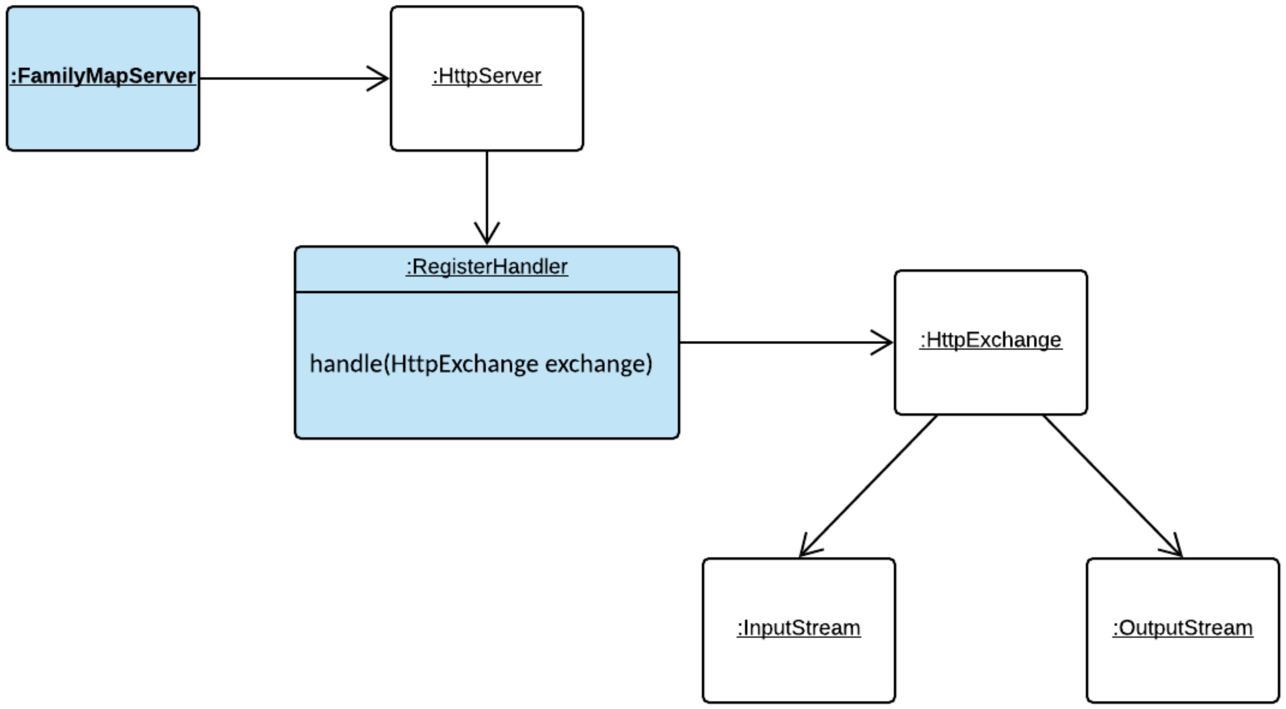
- Java's HttpServer class can be used to implement an HTTP server

- Java's HttpURLConnection class can be used by clients to make HTTP requests of a server and receive HTTP responses from the server

Creating a Server with the HttpServer Class



HttpServer Runtime View (handling a register request)



HttpServer Creation and Startup

```
private void startServer(int port) throws IOException {
    InetSocketAddress serverAddress = new InetSocketAddress(port);
    HttpServer server = HttpServer.create(serverAddress, 10);
    registerHandlers(server);
    server.start();
    System.out.println(”FamilyMapServer listening on port " + port);
}

private void registerHandlers(HttpServer server) {
    server.createContext("/", new FileRequestHandler());
    server.createContext("/user/register", new RegisterRequestHandler());
    …
}
```

The HttpExchange class

- The typical life-cycle of a HttpExchange is shown in the sequence below
  - getRequestMethod() - to determine the request method (i.e. GET, POST, etc).
  - getRequestHeaders() - to examine the request headers (if needed)
  - getRequestBody() - returns an InputStream for reading the request body.
  - getResponseHeaders() - to set any response headers, except content-length (returns a mutable map into which you can add headers).
  - sendResponseHeaders(int, long) - to send the response headers and response code. Must be called before next step.
  - getResponseBody() - to get an OutputStream to send the response body. When the response body has been written, the stream (or the exchange) must be closed to terminate the exchange.

Connecting to a Server with the HttpURLConnection Class

- HTTP GET Example
  - GetExample.java
- HTTP POST Example
  - PostExample.java

HTTP GET Request/Response Steps

1. **Client:** Create URL instance
2. **Client:** Open connection (url.openConnection()), set read timeout, set request method to GET, connect
3. **Server:** Handler's handle method is called and passed an HttpExchange instance
4. **Server:** Process request (use HttpExchange to get request method, URI, headers, etc if needed to process request)
5. **Server:** Send response code (exchange.sendResponseHeaders(responseCode, responseLength))
6. **Server:** Get output stream (exchange.getResponseBody())
7. **Server:** Write response to stream
8. **Server:** Close the exchange (exchange.close())
9. **Client:** Get Response code, get input stream
10. **Client:** Read and process response

HTTP POST Request/Response Steps

1. **Client:** Create URL instance
2. **Client:** Open connection, set read timeout, set request method to POST, **setDoOutput(true)**, connect
3. **Client:** Get output stream (connection.getOutputStream())
4. **Client:** Write request body to output stream

5. **Server:** Handler's handle method is called and passed an HttpExchange instance
6. **Server:** Process request (use HttpExchange to get request method, URI, headers (e.g. authorization), etc if needed to process request)
7. **Server: Get input stream (exchange.getRequestBody())**
8. **Server:** Process request (**convert json to object**, do business logic)
9. **Server:** Send response code (exchange.sendResponseHeaders(responseCode, responseLength))
10. **Server:** Get output stream (exchange.getResponseBody())
11. **Server:** Write response to stream
12. **Server:** Close the exchange (exchange.close())

13. **Client:** Get Response code, get input stream
14. **Client:** Read and process response

HttpHandler Example: Ticket to Ride Web API

- Get list of games
  - Description: Returns list of currently-running games
  - URL Path: /games/list
  - HTTP Method: GET
  - Request Body: None
  - Response Body: JSON of the following form:

```
{ "game-list": [
        { "name": "fhe game", "player-count": 3 },
        { "name": "work game", "player-count": 4 },
        { "name": "church game", "player-count": 2 }
        ]
}
```

- ListGamesHandler.java

HttpHandler Example: Ticket to Ride Web API

- Claim route
  - Description: Allows player to claim route between two cities
  - URL Path: /routes/claim
  - HTTP Method: POST
  - Request Body: JSON of the following form:

    { "route": "atlanta-miami" }
  - Response Body: None

- [ClaimRouteHandler.java]()

Writing a File Handler

- Register “/” with your file handler
  - `server.createContext("/", new FileHandler());`
  - Will cause all requests but those that are registered with a more specific path to route to your file handler
- Ignore everything but GET requests
  - Could send a 405 (Method Not Allowed)
- Get the request URI from the exchange
  - `String urlPath = httpExchange.getRequestURI().toString());`
  - If urlPath is null or “/”, set urlPath to “/index.html”

Writing a File Handler (cont.)

- Append urlPath to a relative path (no leading slash) to the directory containing the files
  - `String filePath = ”web” + urlPath;`
    - Assumes there is a directory named “web” in the root of the project containing your server and the files are in the “web” directory
  - Create a file object and check if the file exists (`file.exists()`)
- Return a 404 (not found) error if the file does not exist
  - For Family Map Server, also send the provided custom 404.html page
- If the file exists, read the file and write it to the HttpExchange’s output stream

  -
    ```
    OutputStream respBody = exchange.getResponseBody();
    Files.copy(file.toPath(), respBody);
    ```