



TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATION

WEB PROGRAMMING
LABORATORY WORK #2

HTTP, caching and content negotiation

Author:
Liviu MOCANU
std. gr. FAF-203

Supervisor:
Alexei ȘERȘUN

Chișinău 2023

1 Task

Our task for this laboratory work was to:

1. Write a command line program, using go2web executable as a starting point.
2. The program should implement at least the following CLI:
 - go2web -u <URL> # make an HTTP request to the specified URL and print the response
 - go2web -s <search-term> # make an HTTP request to search the term using your favorite search engine and print top 10 results
 - go2web -h # show this help
3. The responses from request should be human-readable (e.g. no HTML tags in the output)

We were not allowed to use any third-party libraries for making HTTP requests.

2 Results

In order to complete this laboratory work I used Java and the only third-party libraries besides core Java were *Jsoup* for parsing and presenting HTML as well as *JsonNode*, *ObjectMapper* and *SerializationFeature* for dealing with JSON when scrapping. I used *JSONArray* and *JSONObject* to handle the response when searching for a query in **Google** as the chosen search engine.

2.1 Executable

When `./go2web` with argument `-u` is used like we see in Listing 1, the **Scraper** from package `utm` is called from within the jar, with the `url` and `location` where the Java project is located as arguments, in order to be able to cash the result.

```
1  case $opt in
2  u)
3      url=$OPTARG
4      java -cp "$(pwd)/target/pw-2-1.0-SNAPSHOT-jar-with-dependencies.jar" utm.Scraper
5      "$url" "$(pwd)"
6  ;;
```

Listing 1: Implementation of `./go2web -u` command

When `./go2web` with argument `-s` is used like we see in Listing 2, the **Searcher** from package `utm` is called from within the jar, with the only argument being the `query`. As previously mentioned, I used the Google search engine. `shift` and `query="$*` were used to allow multiple words as prompt other than just one.

```
1  case $opt in
2  s)
3      # Use shift to remove the '-s' option from the argument list
4      shift
5      # Join all the remaining arguments into a single string separated by spaces
6      query="$*"
7      java -cp "$(pwd)/target/pw-2-1.0-SNAPSHOT-jar-with-dependencies.jar" utm.Searcher
8      "$query"
9  ;;
```

Listing 2: Implementation of `./go2web -s` command

The overall configuration of the go2web executable is showcased in Listing 3. Besides the help, I decided to add a message in case of an Invalid option or if the option is valid but no arguments were added even when required as demonstrated in [1].

```
1 #!/usr/bin/env bash
2
3 while getopts "u:s:h" opt; do
4     case $opt in
5         u)
6             url=$OPTARG
7             java -cp "$(pwd)/target/pw-2-1.0-SNAPSHOT-jar-with-dependencies.jar" utm.Scrapper
8                 "$url" "$(pwd)"
9             ;;
10        s)
11            shift
12            query="$*"
13            java -cp "$(pwd)/target/pw-2-1.0-SNAPSHOT-jar-with-dependencies.jar" utm.Searcher
14                "$query"
15            ;;
16        h)
17            echo "go2web -u <URL>          # make an HTTP request to the specified URL and
18                print the response"
19            echo "go2web -s <search-term> # make an HTTP request to search the term using
20                your favorite search engine and print top 10 results"
21            echo "go2web -h              # show this help"
22            exit 1
23            ;;
24        \?)
25            echo "Invalid option: -$OPTARG" >&2
26            exit 1
27            ;;
28        :)
29            echo "Option -$OPTARG requires an argument." >&2
30            exit 1
31            ;;
32    esac
33done
```

Listing 3: The executable go2web

2.2 Scraper

The `Scraper` class has 4 important functions: `main`, `requestResponse`, `establishConnection` and the `getResponseBodyAsString` function.

Besides some validation calling the `requestResponse` function, `main` has actually the caching functionality as well.

`Main` first creates a **cache** directory by calling `Files.createDirectories()` with a `Path` object created from string concatenating `cacheDirectoryPath` and "cache". It then creates a cache file by creating a `File` object within the cache directory, using the URL string with all *non-alphanumeric* characters removed as the filename.

If the cache file already exists and is a regular file (not a directory), the code reads the cached response from the file using `Files.readString()` after which requests a new response by calling the `requestResponse()` method with the URL string. If the new response is not null and is different from the cached response, the code updates the cache file by overwriting it with the new response using a `BufferedWriter`.

Otherwise, the code requests a new response using the `requestResponse()` method with the URL string. If the new response is not null, the code saves it to the cache file.

Then, the code either prints the cached response or the new response to the console, depending on whether a cached response was retrieved or a new response was requested. If there is an `IOException` while performing any of these operations, the code catches the exception, prints an error message to the console, and does not return any response as seen in Listing 5:

```
1 public static void main(String[] args) throws IOException {
2     // ...
3     String cacheDirectoryPath = args[1];
4
5     File cacheDirectory = Files.createDirectories(Paths.get(cacheDirectoryPath, "
cache"));
6     File cacheFile = new File(cacheDirectory, urlString.replaceAll("[^a-zA-Z0-9]", "")
+ ".txt");
7
8     // if request is already cached
9     if (cacheFile.exists() && !cacheFile.isDirectory()) {
10         String cachedResponse = Files.readString(cacheFile.toPath());
11         String newResponse = requestResponse(urlString);
12         if (newResponse != null && !newResponse.equals(cachedResponse)) {
13             // update cache
14             try (BufferedWriter writer = new BufferedWriter(new FileWriter(cacheFile)
)) {
15                 writer.write(newResponse);
16             } catch (IOException e) {
17                 System.err.println("Error: " + e.getMessage());
18             }
19         }
20         System.out.println(cachedResponse);
21     } else {
22         String newResponse = requestResponse(urlString);
23         if (newResponse != null) {
24             // save to cache
25             try (var writer = new BufferedWriter(new FileWriter(cacheFile))) {
26                 writer.write(newResponse);
27             } catch (IOException e) {
28                 System.err.println("Error: " + e.getMessage());
29             }
30             System.out.println(newResponse);
31         }
32     }
```

Listing 4: Scrapper main function

`requestResponse` takes `urlString` as an input parameter and returns a `String` representing the formatted response obtained from the HTTP connection.

The method works by first calling the `establishConnection` method to establish an HTTP connection to the URL specified by the `urlString` input parameter. It then retrieves the content type of the HTTP response using the `getContentType()` method of the `HttpURLConnection` object, and reads the response body into a `String` using the `getResponseBodyAsString` method.

Next, the method checks the content type of the HTTP response. It is exactly here that **content negotiation** was implemented:

If the content type is **"application/json"**, it uses the Jackson JSON library to parse the response body into a `JsonNode` object, formats the JSON using the `ObjectMapper` class, and converts it to a `String` using the `writeValueAsString` method of the `ObjectMapper` class.

If the content type is **"text/html"**, the method uses the `Jsoup` library to parse the response body into an HTML document and extract the text content using the `text()` method.

If the content type is not supported, the method prints an error message to the console.

The method disconnects the HTTP connection and returns the formatted response `String`. If there is an `IOException` while performing any of these operations, the method catches the exception, prints an error message to the console, and returns `null` as shown in Listing 5:

```

1 private static String requestResponse(String urlString) {
2     try {
3         HttpURLConnection connection = establishConnection(urlString);
4         String contentType = connection.getContentType();
5         String responseString = getResponseBodyAsString(connection);
6         String formattedResponse = null;
7
8         if (contentType.contains("application/json")) {
9             ObjectMapper objectMapper = new ObjectMapper();
10            objectMapper.enable(SerializationFeature.INDENT_OUTPUT);
11            JsonNode jsonNode = objectMapper.readTree(responseString);
12            formattedResponse = objectMapper.writeValueAsString(jsonNode);
13        } else if (contentType.contains("text/html")) {
14            formattedResponse = Jsoup.parse(responseString).text();
15        } else {
16            System.out.println("Content type not supported.");
17        }
18
19        connection.disconnect();
20        return formattedResponse;
21    } catch (IOException e) {
22        System.err.println("Error: " + e.getMessage());
23        return null;
24    }
25 }
```

Listing 5: Scrapper requestResponse function

`establishConnection` takes a `String urlString` as an input parameter and returns an `HttpURLConnection` object representing the established connection.

The method works by first creating a new `URL` object from `urlString`. Then, it enters a loop where it opens a new `HttpURLConnection` using the `openConnection()` method of the `URL` object. It also retrieves the status code of the HTTP response using the `getResponseCode()` method of the `HttpURLConnection` object.

It is exactly here that the function deals with **HTTP request redirects**. If the status code indicates that the server has issued a redirect (`HTTP_MOVED_TEMP`, `HTTP_MOVED_PERM`, or `HTTP_SEE_OTHER`),

the method retrieves the URL for the redirect from the Location header field of the HTTP response using the `getHeaderField()` method of the `URLConnection` object. It then disconnects the current connection and creates a new URL object from the redirect URL.

The loop continues until the status code does not indicate a redirect. The method then returns the `URLConnection` object representing the established connection.

The method throws an `IOException` if there is an error establishing the connection as shown in Listing 6:

```
1 private static URLConnection establishConnection(String urlString) throws
  IOException {
2     URL url = new URL(urlString);
3     URLConnection connection;
4     int status;
5     do {
6         connection = (URLConnection) url.openConnection();
7         status = connection.getResponseCode();
8         if (status == URLConnection.HTTP_MOVED_TEMP || status ==
  URLConnection.HTTP_MOVED_PERM
9             || status == URLConnection.HTTP_SEE_OTHER) {
10            String redirectUrl = connection.getHeaderField("Location");
11            connection.disconnect();
12            url = new URL(redirectUrl);
13        }
14    } while (status == URLConnection.HTTP_MOVED_TEMP || status ==
  URLConnection.HTTP_MOVED_PERM
15            || status == URLConnection.HTTP_SEE_OTHER);
16
17    return connection;
18 }
```

Listing 6: Scrapper `establishConnection` function

`getResponseBodyAsString` is a static function that takes an `URLConnection` object as an input parameter and returns a `String` representing the response body of the HTTP connection.

The method works by first creating a new `Scanner` object, which reads the response body of the HTTP connection using the `getInputStream` method. The `Scanner` is set to use the UTF-8 character set using the `StandardCharsets.UTF_8` constant.

The `Scanner` is then used to read the entire response body into a single `String` using the `hasNext()` and `next()` methods. The `hasNext()` method checks if there is any input left to read, and the `next()` method reads the next input token as a `String`. If there is no input left to read, an empty `String` is returned.

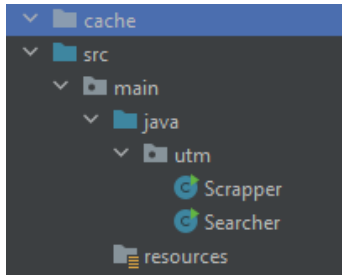
The method throws an `IOException` if there is an error reading the input stream as seen in Listing 7:

```
1 private static String getResponseBodyAsString(URLConnection connection) throws
  IOException {
2     try (Scanner scanner = new Scanner(connection.getInputStream(), StandardCharsets.
  UTF_8).useDelimiter("\\A")) {
3         return scanner.hasNext() ? scanner.next() : "";
4     }
5 }
```

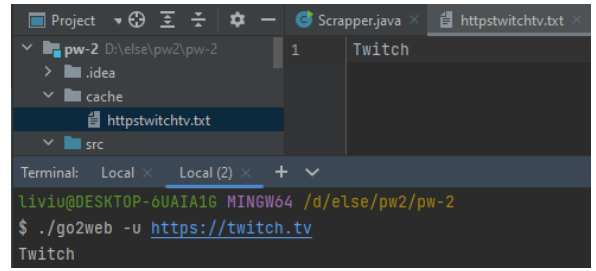
Listing 7: Scrapper `getResponseBodyAsString` function

An example of using cache can be seen in Figure 1 by sending an HTTP request to <https://twitch.tv>:

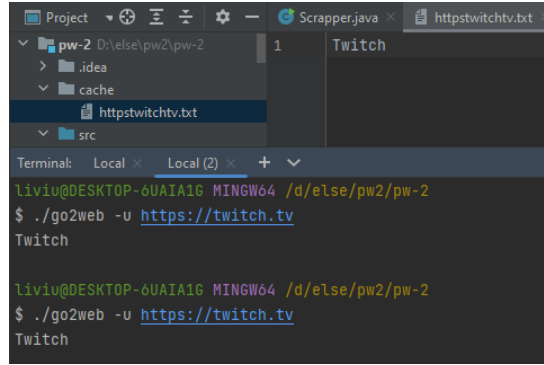
Unfortunately in cases such as with <http://worldtimeapi.org/api/ip>, cache won't help too much since every request will be different as seen in Figure 2:



(a) No cache files before any requests



(b) Cache directory after a request



(c) Reusing cache

Figure 1: Cache directory after `./go2web -u https://twitch.tv` command



Figure 2: Cache will change if the request is different

2.3 Searcher

Moving onto the `Searcher` class, we have 3 other important functions: `main`, `getResults`, and the `getResponseBodyAsString` function.

The function `main` performs a Google search using the Google Custom Search API and prints the top `MAX_RESULTS` (In our case 10) first results.

For that however, we had to set up our own `API_KEY` and `CX` by going to Google Developers Console and setting up a new Custom Search API project. In my case they can be seen in Listing 8:

```
1 private static final String API_KEY = "AIzaSyBKAkng_5Qu43Yg4aBlvEHKb-A2V6l2UBI";
2 private static final String CX = "75e13f83e334443b6";
```

Listing 8: Setting up a new Custom Search API project

Firstly, we encode the query passed as an argument using `URLEncoder.encode()` and the `UTF_8` charset.

Then we create a URI string using `String.format()`, with placeholders for the `GOOGLE_URL` constant, `API_KEY` and `CX`, and query variables. We create a `URI` object using the resulting string, and open a `HttpURLConnection` to the URL specified by the URI.

Afterwards, the request method is set to "GET" using `connection.setRequestMethod()`, and it retrieves the response code using `connection.getResponseCode()`.

If the response code is `HttpURLConnection.HTTP_OK`, the code reads the response body using `getResponseBodyAsString()` and extracts the search results using `getResults()` after which printing the search query and the top `MAX_RESULTS` first results.

Otherwise, if the response code is not `HttpURLConnection.HTTP_OK`, the code prints an error message to the console indicating that the search results could not be retrieved.

The code then disconnects the `HttpURLConnection`. If any exception occurs, we print the stack trace to the console.

```
1 public static void main(String[] args) {
2     // first verify whether the Searcher is called correctly
3     if (args.length < 1) {
4         System.err.printf("Searcher expectations not met:%n" +
5             "Provide only the query%n");
6         System.exit(2);
7     }
8
9     try {
10        String query = URLEncoder.encode(args[0], StandardCharsets.UTF_8);
11        URI uri = URI.create(String.format("%s?key=%s&cx=%s&q=%s", GOOGLE_URL,
12        API_KEY, CX, query));
13        HttpURLConnection connection = (HttpURLConnection) uri.toURL().openConnection
14        ();
15
16        connection.setRequestMethod("GET");
17        int responseCode = connection.getResponseCode();
18
19        if (responseCode == HttpURLConnection.HTTP_OK) {
20            String response = getResponseBodyAsString(connection);
21            List<String> searchResults = getResults(response);
22            System.out.println("'" + args[0] + "' provided the following results:");
23            for (int i = 0; i < searchResults.size(); i++) {
24                String result = searchResults.get(i);
25                System.out.println((i+1) + ". " + result);
26            }
27        } else {
28            System.err.println("Failed to get search results. Response code: " +
29            responseCode);
30        }
31    }
32 }
```



```

28     connection.disconnect();
29 } catch (Exception e) {
30     e.printStackTrace();
31 }
32 }

```

Listing 9: Searcher `main` function

`getResults` takes a string `responseString` as input and returns a list of strings which represent the results to the query by:

- Creating a new `JSONObject` by parsing the `responseString` parameter.
- Retrieving the items JSON array from the `JSONObject`.
- Using `IntStream.range` to create a stream of integers from 0 to `MAX_RESULTS` and then `mapToObj` to map each integer to the corresponding `JSONObject` in the items array.
- Using `map` to convert each `JSONObject` to a string that concatenates the values of the *title* and *link* fields, separated by a hyphen.
- Collecting the resulting strings into a list and returning that list.

Thus we extract the title and link fields from the items array of a JSON object and return them as a list of strings as seen in Listing 10.

```

1 private static List<String> getResults(String responseString) {
2     JSONObject json = new JSONObject(responseString);
3     JSONArray items = json.getJSONArray("items");
4
5     return IntStream.range(0, Math.min(items.length(), MAX_RESULTS))
6         .mapToObj(items::getJSONObject)
7         .map(item -> item.getString("title") + " - " + item.getString("link"))
8         .collect(Collectors.toList());
9 }

```

Listing 10: Searcher `getResults` function

The `Searcher` can be seen in action in Figure 3:

```

$ ./go2web -s what is pw
what is pw
'what is pw' provided the following results:
1. P.W. - Urban Dictionary - https://www.urbandictionary.com/define.php?term=P.W.
2. I am a faculty member. A student withdrew from my course and the ... - https://gsu.my.site.com/support/s/article/I-am-a-faculty-member-on-PAWS-is-PW-What-is-PW
3. P.W. | English meaning - Cambridge Dictionary - https://dictionary.cambridge.org/dictionary/english/pw
4. Pw definition and meaning | Collins English Dictionary - https://www.collinsdictionary.com/us/dictionary/english/pw
5. DOR Pass-Through Entity Withholding - https://www.revenue.wi.gov/Pages/OnlineServices/pw-home.aspx
6. PW Pizza - https://pwpizza.com/
7. Instructions for 2019 Form PW-1 - https://www.revenue.wi.gov/TaxForms2017through2019/2019-FormPW-1-Inst.pdf
8. Pw - What is pw short for? - https://slang.net/meaning/pw
9. Home - P.W. Grosser Consulting, Inc. - https://pwgrosser.com/
10. PAPBS > Program wide PBIS - https://papbs.org/Program-wide-PBIS

```

Figure 3: Top 10 search results for prompt 'what is pw' using Google

3 Conclusion

This laboratory work for Web Programming involved building a Command Line program that made HTTP requests to specific URLs, including search engines, and printed the responses in a human-readable format.

The project also included implementing advanced features, such as :

- **HTTP request redirects**, which are important for ensuring that the program can follow links and request the correct resources.
- **HTTP cache mechanism**, which can help reduce network traffic and improve performance.
- **Content negotiation**, which is important for building modern web applications that can work with various types of data.

By completing this laboratory work, I gained valuable experience in working with HTTP requests and responses, as well as got hands-on experience on advanced web development concepts that are essential for developing advanced web-based applications.

References

- [1] Small getopts tutorial, https://wiki.bash-hackers.org/howto/getopts_tutorial
Accessed on March 4, 2023.
- [2] Do a Simple HTTP Request in Java, <https://www.baeldung.com/java-http-request>
Accessed on March 4, 2023.
- [3] How to read write this in utf-8?, <https://stackoverflow.com/questions/13350676/how-to-read-write-this-in-utf-8>
Accessed on March 5, 2023 .
- [4] How To Get Google Api Key & Search Engine ID Google CSE, <https://www.youtube.com/watch?v=Bxy8Yqp5XX0>
Accessed on March 5, 2023 .