NameScoreCalculator Test with HTTP Get Function

Francisco Javier Castro Márquez

November 24, 2024

The objective of this Java program, named as NameScoreCalculator is to consume a web service, retrieving a list of names for sorting them alphabetically and calculate a score based on their position based on the value of the letters, summitting this score to another web service.

1 Libraries:

Libraries used in the code, providing the necessary tools to handle the communication, data process and the manage of collections in a simple but efficient program.

Library:	Methods used:
package org	example
java.io	BufferedReader, InputStreamReader, OutputStream
java.net	URL, HttpURLConnection
java.nio	charset.StandardCharsets
java.util.*	List, Optional, Map
org.json.	JSONArray, JSONObject
java.util.stream.Collectors	toList

Table 1: Package and Libraries imported.

2 Class-Level Constants:

```java
//URL of the web service that provides the names.
private static final String SOURCE_URL = "";

//URL of the web service where the calculated result is sent.
private static final String TARGET_URL = "";

//Bearer token for authenticating the GET request to fetch names.
private static final String SOURCE_AUTH = "";

```

```
//Bearer token for authenticating the POST request to submit
    the result.
private static final String TARGET_AUTH = "";
```
Listing 1: Class-Level Constants for the program

3 main Method:

The main entry point of the program, having the function to manage the classes needed or inform if there was an Exception in the process. As shown in the Listing 2

```
public static void main(String[] args) {
    try {
        // Fetch a list of names from a data source
        List<String> names = fetchNamesFromSource();

        // Preprocess the list of names
        names = preprocessNames(names);

        // Calculate the total score based on the processed
            names
        long totalScore = calculateTotalNameScore(names);

        // Print the total score to the console for
            debugging purposes
        System.out.println("Total Score: " + totalScore);

        // Send the calculated score to a target destination
        sendResultToTarget(totalScore, "Your Name", false);
    } catch (Exception e) {
        // Print an error message if an exception occurs
        System.err.println("An error occurred: " + e.
            getMessage());

        // Print the stack trace for debugging purposes
        e.printStackTrace();
    }
}
```
Listing 2: main()

4 fetchNamesFromSource Method:

The purpose of this function is retrieve the list of names by making an HTTP GET request to the source URL using the Bearer token for authentication. As shown in the Listing 3

```
private static List<String> fetchNamesFromSource() throws
    Exception {
    // Construct the URL string with the required parameters
    String urlString = SOURCE_URL + "?archivo=first_names&
        extension=txt";
```

```java

    // Execute the GET request and process the response
    return executeGetRequest(urlString, SOURCE_AUTH)
            // Parse the JSON response to extract names
            .map(NameScoreCalculator::parseNamesFromJson)
            // Throw an exception if the names cannot be fetched
            .orElseThrow(() -> new Exception("Failed to fetch names from source"));
}
```

Listing 3: fetchNamesFromSource()

5 executeGetRequest Method:

The purpose of this function is the execution of the HTTP GET request mentioned above, sebdubg tge authentication token in the request header. As shown in the Listing 4

```java
private static Optional<String> executeGetRequest(String urlString, String auth) throws Exception {
    // Create a URL object from the provided URL string
    URL url = new URL(urlString);

    // Open an HTTP connection to the URL
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    // Set the Authorization header for the request
    con.setRequestProperty("Authorization", auth);

    // Try-with-resources to ensure the BufferedReader is closed after use
    try (BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()))) {
        // Read the response lines and join them into a single String, then wrap it in an Optional
        return Optional.of(in.lines().collect(Collectors.joining()));
    } catch (Exception e) {
        // Print an error message if an exception occurs and return an empty Optional
        System.err.println("Error executing GET request: " + e.getMessage());
        return Optional.empty();
    } finally {
        // Disconnect the HTTP connection
        con.disconnect();
    }
}
```

Listing 4: executeGetRequest()

6 parseNamesFromJson Method:

The purpose of this function parsing the JSON response from the web service into a list of names. As shown in the Listing 5

```java
private static List<String> parseNamesFromJson(String jsonString) {
    // Convert the JSON string into a JSONArray
    JSONArray jsonArray = new JSONArray(jsonString);

    // Convert the JSONArray to a List, then stream through the list
    return jsonArray.toList().stream()
            // Map each object in the list to its "NAME" field and convert it to a string
            .map(obj -> ((Map<?, ?>) obj).get("NAME").toString())
            // Collect the results into a List of Strings
            .collect(Collectors.toList());
}
```

Listing 5: parseNamesFromJson()

7 preprocessNames Method:

The purpose of this function is the preprocessing of the list names by performing the operations from crop whitespace from names, removing non-alphabetical characters, converting all names to uppercase to ensure consistent process and sorting the names alphabetically. As shown in the Listing 6

```java
private static List<String> preprocessNames(List<String> names) {
    // Convert the list of names to a stream for processing
    return names.stream()
            // Remove leading and trailing whitespace from each name
            .map(String::trim)
            // Remove all non-alphabetic characters from each name
            .map(name -> name.replaceAll("[^a-zA-Z]", ""))
            // Convert each name to uppercase for consistent processing
            .map(String::toUpperCase)
            // Sort the names in alphabetical order
            .sorted()
            // Collect the processed names back into a list
            .collect(Collectors.toList());
}
```

Listing 6: preprocessNames()

8 calculateTotalNameScore Method:

The purpose of this function is calculating the total score for the name based on the alphabetical value and its position. As shown in the Listing 7

```java
private static long calculateTotalNameScore(List<String> names) {
    long totalScore = 0;  // Initialize total score to 0

    // Iterate through each name in the list
    for (int i = 0; i < names.size(); i++) {
        String name = names.get(i);  // Get the name at the current position
        long nameValue = getAlphabeticalValue(name);  // Calculate the alphabetical value of the name

        // Calculate the score for the current name and add it to the total score
        totalScore += nameValue * (i + 1);

        // Print debug information for the current name
        System.out.println("Name: " + name + ", Value: " + nameValue + ", Position: " + (i + 1) + ", Score: " + (nameValue * (i + 1)));  // Debug print
    }

    return totalScore;  // Return the total score of all names
}
```

Listing 7: calculateTotalNameScore()

9 getAlphabeticalValue Method:

The purpose of this function the compute of the alphabetical value from a given name, starting from 'A'=1, using chars() method to get the ASCII value. As shown in the Listing 8

```java
private static int getAlphabeticalValue(String name) {
    // Convert the name to a stream of characters
    return name.chars()
            // Map each character to its alphabetical value
            .map(ch -> ch - 'A' + 1)
            // Sum the values
            .sum();
}
```

Listing 8: getAlphabeticalValue()

10 sendResultToTarget Method:

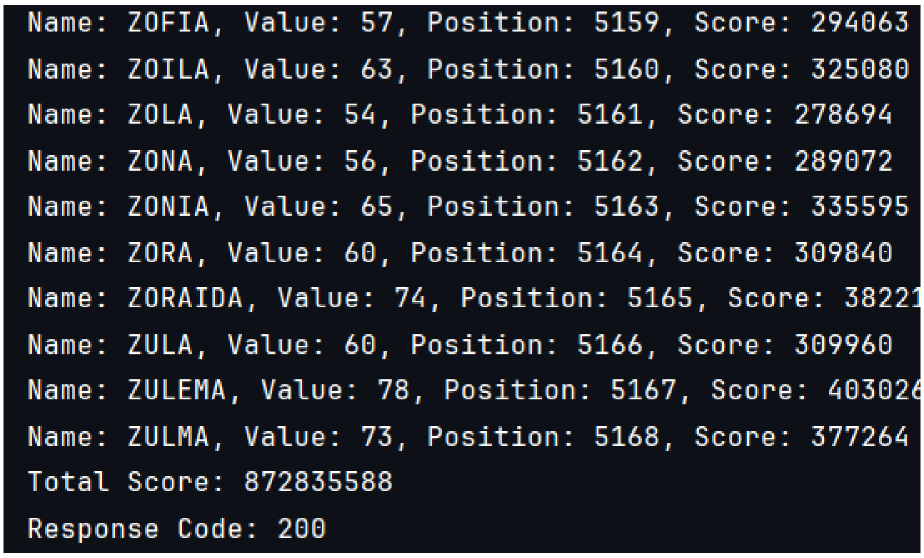
The purpose of this function is sending the total calculated score to the target web service by HTTP POST request, propagating the exception if the HTTP POST fails. As shown in the Listing 9

```java
private static void sendResultToTarget(long totalScore, String name, boolean isTest) throws Exception {
    // Construct the URL with query parameters
    String urlString = TARGET_URL + "?archivo=first_names&extension=txt&nombre=" + name + "&prueba=" + (isTest ? 1 : 0);

    // Create the JSON input string
    String jsonInputString = "{ \"ResultadoObtenido\": " + totalScore + " }";

    // Create a URL object
    URL url = new URL(urlString);

    // Open a connection to the URL
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    // Set the request method to POST
    con.setRequestMethod("POST");

    // Set the request properties
    con.setRequestProperty("Authorization", TARGET_AUTH);
    con.setRequestProperty("Content-Type", "application/json; utf-8");

    // Enable output for the connection
    con.setDoOutput(true);

    // Write the JSON input string to the output stream
    try (OutputStream os = con.getOutputStream()) {
        byte[] input = jsonInputString.getBytes(StandardCharsets.UTF_8);
        os.write(input, 0, input.length);
    }

    // Get the response code from the server
    int responseCode = con.getResponseCode();
    System.out.println("Response Code: " + responseCode); // Debug print

    // Read the response from the server
    try (BufferedReader br = new BufferedReader(new InputStreamReader(con.getInputStream(), StandardCharsets.UTF_8))) {
        StringBuilder response = new StringBuilder();
        String responseLine;
        while ((responseLine = br.readLine()) != null) {
            response.append(responseLine.trim());
        }
        System.out.println("Response Body: " + response.toString());  // Debug print
    }

    // Disconnect the connection
    con.disconnect();
}
```

Listing 9: sendResultToTarget()

11 Result:

With the successful run of the Java program the Total Score obtained is 872835588 as shown in Figure 1.



```
Name: ZOFIA, Value: 57, Position: 5159, Score: 294063
Name: ZOILA, Value: 63, Position: 5160, Score: 325080
Name: ZOLA, Value: 54, Position: 5161, Score: 278694
Name: ZONA, Value: 56, Position: 5162, Score: 289072
Name: ZONIA, Value: 65, Position: 5163, Score: 335595
Name: ZORA, Value: 60, Position: 5164, Score: 309840
Name: ZORAIDA, Value: 74, Position: 5165, Score: 38221
Name: ZULA, Value: 60, Position: 5166, Score: 309960
Name: ZULEMA, Value: 78, Position: 5167, Score: 40302
Name: ZULMA, Value: 73, Position: 5168, Score: 377264
Total Score: 872835588
Response Code: 200
```

Figure 1: Result of the Java Program