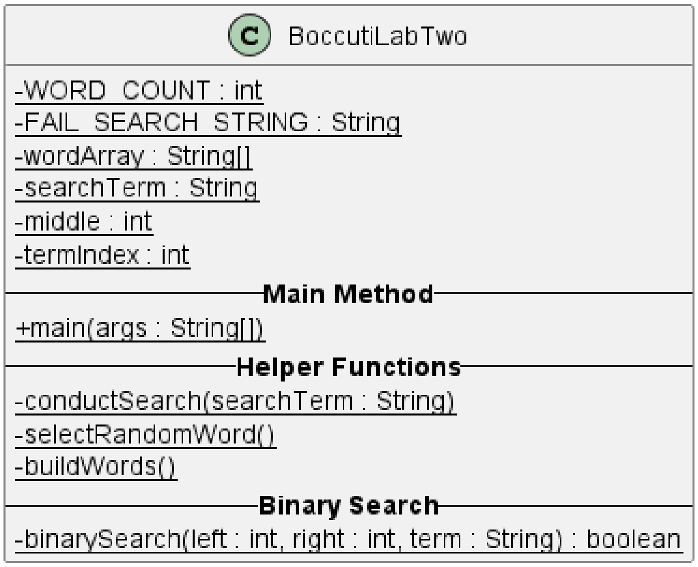
Jason Boccuti
CISC 213
Lab 2

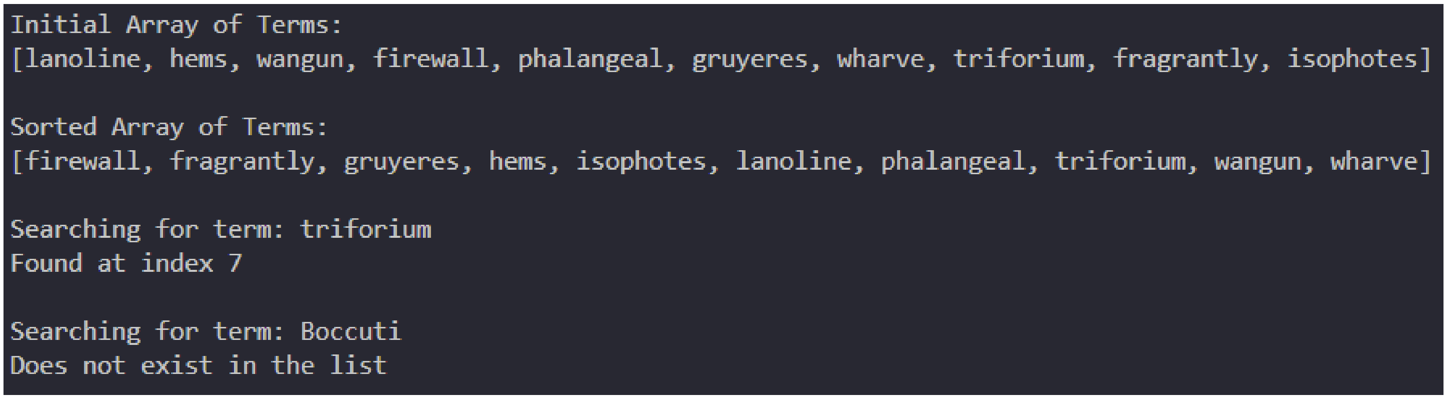
Pseudocode

A recursive binary search:

```
function binarySearch(array, left, right, term)
    middle = (left + right) / 2
    if right >= left
        if array[middle].compareTo(term) == 0
            return true
        else if array[middle].compareTo(term) > 0
            return binarySearch(left, middle - 1, term)
        else
            return binarySearch(middle + 1, right, term)
    return false
```

UML Class Diagram + Console Output





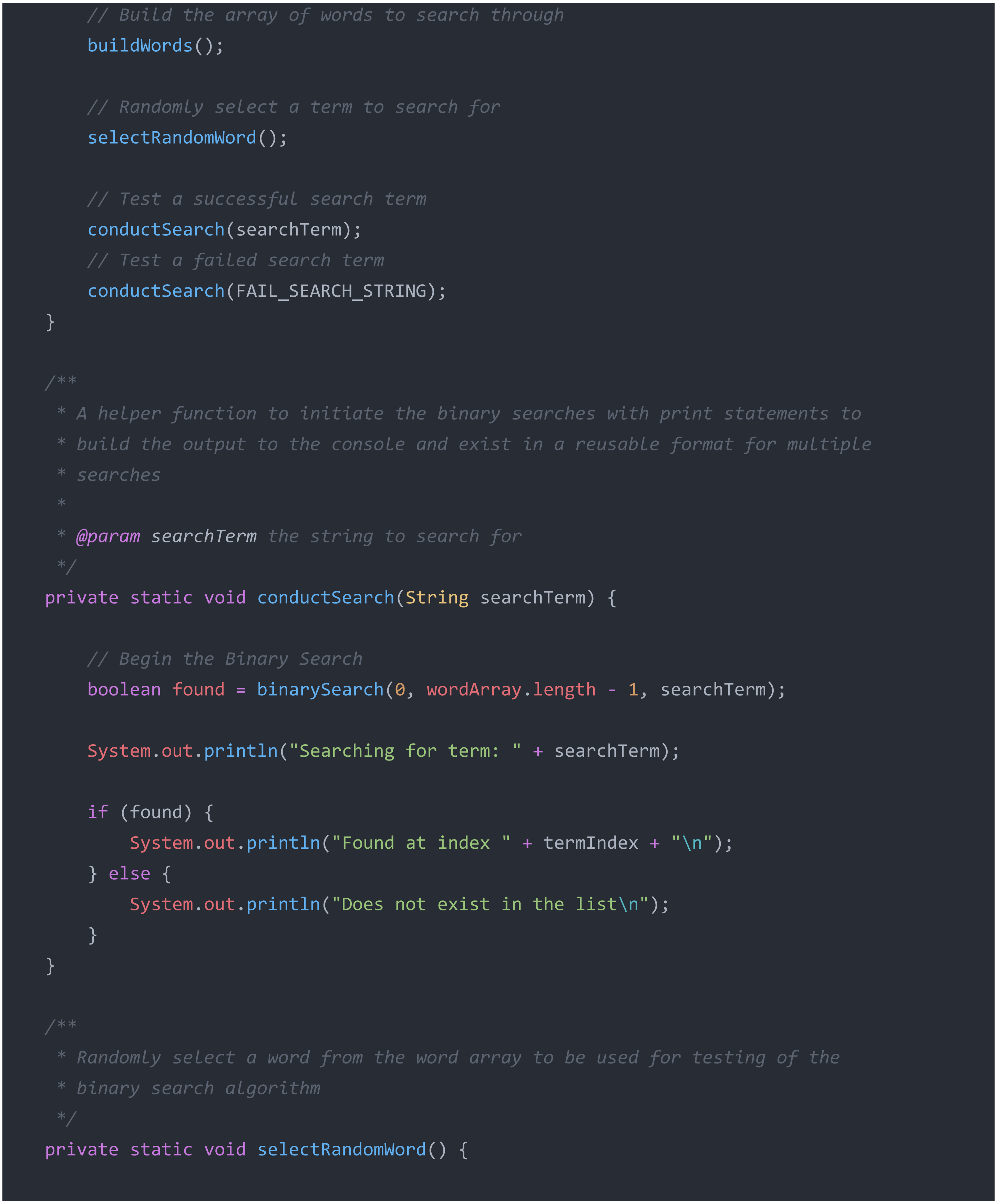
```
Initial Array of Terms:
[lanoline, hems, wangun, firewall, phalangeal, gruyeres, wharve, triforium, fragrantly, isophotes]

Sorted Array of Terms:
[firewall, fragrantly, gruyeres, hems, isophotes, lanoline, phalangeal, triforium, wangun, wharve]

Searching for term: triforium
Found at index 7

Searching for term: Boccuti
Does not exist in the list
```

Jason Boccuti
CISC 213
Lab 2

Source Code

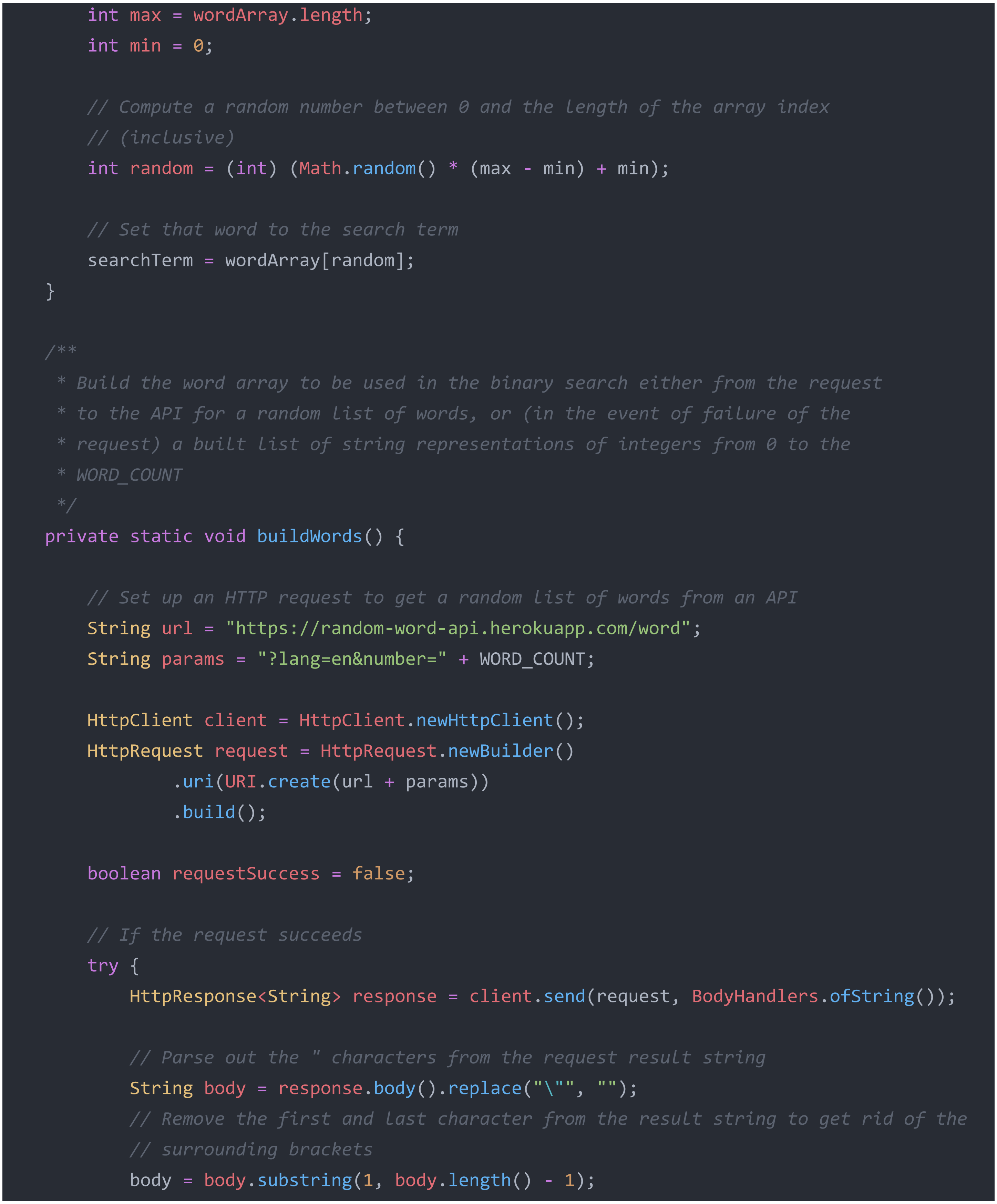
Source file also uploaded on Canvas

```
package dev.boccuti.www.cisc213.lab2;

import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;
import java.util.Arrays;

/**
 * Implementation of Lab #2 for CISC 213 that contains an example of a Binary
 * search on Strings and utilizes a web API to generate a list of random words
 * for input. Quickly increase and decrease input size by adjusting the
 * WORD_COUNT value.
 *
 * @author Jason Boccuti | jason@boccuti.dev
 */
public class BoccutiLabTwo {

    // Constants used to decrease copy-pasta code and reduce errors
    private static final int WORD_COUNT = 10;
    private static final String FAIL_SEARCH_STRING = "Boccuti";

    private static String[] wordArray = new String[WORD_COUNT];
    private static String searchTerm = "";
    private static int middle = 0;
    private static int termIndex = 0;

    /**
     * Main method of the lab that produces the binary search tests for one
     * successful and one failed attempt.
     *
     * @param args[] command line arguments that are not used in this program
     */
    public static void main(String args[]) {

```



```
        // Build the array of words to search through
        buildWords();

        // Randomly select a term to search for
        selectRandomWord();

        // Test a successful search term
        conductSearch(searchTerm);
        // Test a failed search term
        conductSearch(FAIL_SEARCH_STRING);
    }

    /**
     * A helper function to initiate the binary searches with print statements to
     * build the output to the console and exist in a reusable format for multiple
     * searches
     *
     * @param searchTerm the string to search for
     */
    private static void conductSearch(String searchTerm) {

        // Begin the Binary Search
        boolean found = binarySearch(0, wordArray.length - 1, searchTerm);

        System.out.println("Searching for term: " + searchTerm);

        if (found) {
            System.out.println("Found at index " + termIndex + "\n");
        } else {
            System.out.println("Does not exist in the list\n");
        }
    }

    /**
     * Randomly select a word from the word array to be used for testing of the
     * binary search algorithm
     */
    private static void selectRandomWord() {
```

Jason Boccuti
CISC 213
Lab 2

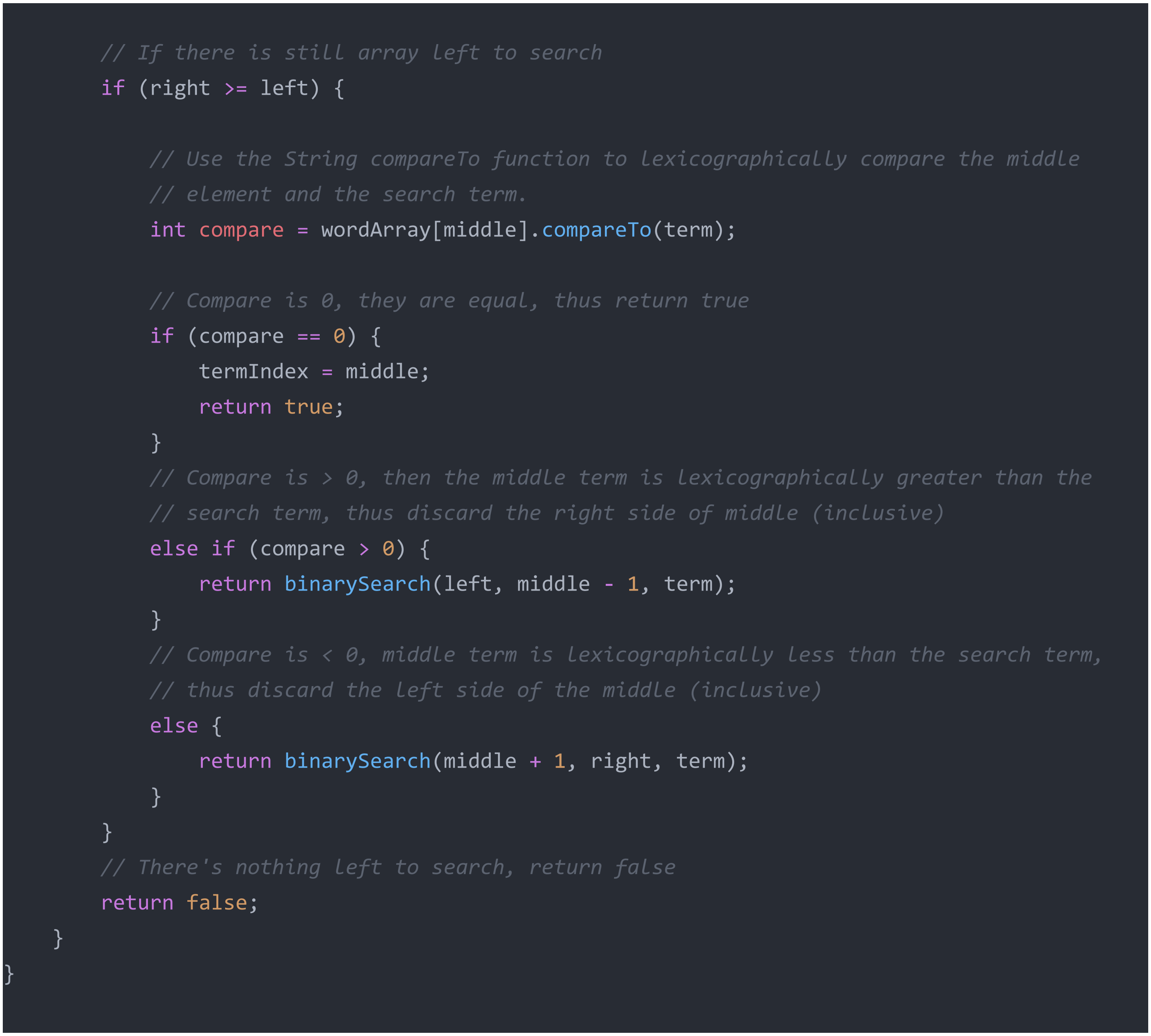


```
        int max = wordArray.length;
        int min = 0;

        // Compute a random number between 0 and the length of the array index
        // (inclusive)
        int random = (int) (Math.random() * (max - min) + min);

        // Set that word to the search term
        searchTerm = wordArray[random];
    }

    /**
     * Build the word array to be used in the binary search either from the request
     * to the API for a random list of words, or (in the event of failure of the
     * request) a built list of string representations of integers from 0 to the
     * WORD_COUNT
     */
    private static void buildWords() {

        // Set up an HTTP request to get a random list of words from an API
        String url = "https://random-word-api.herokuapp.com/word";
        String params = "?lang=en&number=" + WORD_COUNT;

        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url + params))
                .build();

        boolean requestSuccess = false;

        // If the request succeeds
        try {
            HttpResponse<String> response = client.send(request, BodyHandlers.ofString());

            // Parse out the " characters from the request result string
            String body = response.body().replace("\"", "");
            // Remove the first and last character from the result string to get rid of the
            // surrounding brackets
            body = body.substring(1, body.length() - 1);
```

Jason Boccuti
CISC 213
Lab 2

```
            // Split the comma separated result string into the wordArray
            wordArray = body.split(",");
            // Signal the request succeeded so the default word array does not need to be
            // built
            requestSuccess = true;
        } catch (IOException | InterruptedException e) {
            System.out.println("Issue contacting server for word list, using built in word
array.");
        }

        // A simple loop to build the wordArray list using the String representation of
        // numbers going up to the WORD_COUNT size to build mock data if the API is
        // unavailable
        if (!requestSuccess) {
            for (int i = 0; i < WORD_COUNT; i++) {
                wordArray[i] = String.valueOf(i + 1);
            }
        }

        System.out.println("Initial Array of Terms:\n" + Arrays.toString(wordArray) + "\n");
        // Sort the word array
        Arrays.sort(wordArray);
        System.out.println("Sorted Array of Terms:\n" + Arrays.toString(wordArray) + "\n");
    }

    /**
     * The actual binary search algorithm that recursively searches on strings using
     * the global array of words and a provided search term.
     *
     * @param left  the starting index of the word array to check
     * @param right the ending index of the word array to check
     * @param term  the string to search for
     * @return the boolean of whether the search term was found
     */
    private static boolean binarySearch(int left, int right, String term) {

        // Calculate the midpoint of the array (integer division will handle flooring
        // the result)
        middle = (left + right) / 2;
```

Jason Boccuti
CISC 213
Lab 2



```
        // If there is still array left to search
        if (right >= left) {

            // Use the String compareTo function to lexicographically compare the middle
            // element and the search term.
            int compare = wordArray[middle].compareTo(term);

            // Compare is 0, they are equal, thus return true
            if (compare == 0) {
                termIndex = middle;
                return true;
            }
            // Compare is > 0, then the middle term is lexicographically greater than the
            // search term, thus discard the right side of middle (inclusive)
            else if (compare > 0) {
                return binarySearch(left, middle - 1, term);
            }
            // Compare is < 0, middle term is lexicographically less than the search term,
            // thus discard the left side of the middle (inclusive)
            else {
                return binarySearch(middle + 1, right, term);
            }
        }
        // There's nothing left to search, return false
        return false;
    }
}
```