

cse15l-lab-reports

Lab Report 2

Simple Search Engine

Below is the code for the simple search engine I created based off of `NumberServer.java`

```
import java.io.IOException;
import java.net.URI;
import java.util.*;

class Handler implements URLHandler {
    ArrayList<String> searches = new ArrayList<>();

    public String handleRequest(URI url) {
        if (url.getPath().contains("/add")) {
            String[] parameters = url.getQuery().split("=");
            searches.add(parameters[1]);
            return String.format("String added");
        }
        else if (url.getPath().contains("/print")) {
            String output = "";
            for(String element : searches) {
                output = output + element + "\n";
            }
            return output;
        }
        else {
            System.out.println("Path: " + url.getPath());
            if (url.getPath().contains("/search")) {
                String[] parameters = url.getQuery().split("=");
                String output = "";
                for (String element : searches) {
                    if (element.contains(parameters[1])) {
                        output = output + element + "\n";
                    }
                }
                return output;
            }
            return "404 Not Found!";
        }
    }
}
```

```

    }
}


class SearchEngine {
    public static void main(String[] args) throws IOException {
        if(args.length == 0){
            System.out.println("Missing port number! Try any number between 1024 to 49151");
            return;
        }

        int port = Integer.parseInt(args[0]);


        Server.start(port, new Handler());
    }
}

```


As seen above, there are 3 types of functions the search engine is capable of performing: `add`, `search`, and `print`.

The `add` function is accessed by typing `"add"` into the URL as a path. To add a String, enter a query separated by the `"="`, anything on the right side of the `"="` will be added to the list as a String. 

In the screenshot above, `"add"` is shown as the path and the term being added is `"banana"`. The web page displays a message confirming that the term has been added to the list. In the code, the `add` function exists inside the large if-statement in the if true condition.

The `search` function is accessed by typing `"search"` into the URL as a path. To search for a String, like `add`, anything to the right side of the `"="` will be considered in the search. 

In the screenshot above, `"search"` is shown as the path and the search keyword is `"a"`, the search engine is seen returning `"apple"` and `"banana"` from the list, as both terms contain the letter `"a"`. In the code, the `search` function exists inside the large if-statement in the else condition.

The `print` function is accessed by typing `"print"` into the URL as a path. The output of the function is the whole list of Strings going on display on the web page. 

In the screenshot above, `"print"` is shown as the path and no query is needed for the function to work. The search engine prints out all of the Strings added to the list. In the code, the `print` function exists inside the large if-statement in the if-else condition. ***

ArrayTest: Average Without Lowest

For the `averageWithoutLowest()` method, the failure-inducing input was `{2, 3, 4, 2, 2}`, which should output 2.75 but instead output 1.75.

```
double[] input2 = {2.0, 3.0, 4.0, 2.0, 2.0};  
assertEquals(2.75, ArrayExamples.averageWithoutLowest(input2), 0.0001);
```

Terminal output

The bug turned out to be whenever the lowest value in the array was duplicated more than once, it caused the method to remove all the duplicate elements instead of just one. This was fixed by creating a boolean statement that determined whether or not one of the lowest elements was removed. If one is removed, the statement flips from false to true, and then streamlines the remaining elements to all be included in the final calculation.

The first code block shows the faulty code, while the second code block shows the fixed code:

```
static double averageWithoutLowest(double[] arr) {  
    if(arr.length < 2) { return 0.0; }  
    double lowest = arr[0];  
    for(double num: arr) {  
        if(num < lowest) { lowest = num; }  
    }  
    double sum = 0;  
    for(double num: arr) {  
        if(num != lowest) { sum += num; }  
    }  
    return sum / (arr.length - 1);  
}
```

```
static double averageWithoutLowest(double[] arr) {  
    if(arr.length < 2) { return 0.0; }  
    double lowest = arr[0];  
    for(double num: arr) {  
        if(num < lowest) { lowest = num; }  
    }  
    double sum = 0;  
    boolean removed = false;  
    for(double num: arr) {  
        if(num != lowest || removed == true) { sum += num; }  
        else { removed = true; }  
    }  
    return sum / (arr.length - 1);  
}
```

LinkedListTest: Append

The failure-inducing input for `append()` was by appending a 10, 20, and 30 respectively and proceeding to check if the last value in the linked list was a 30. Instead, an infinite loop was found in the terminal.

```
LinkedList tester = new LinkedList();
tester.append(10);
tester.append(20);
tester.append(30);
assertEquals(30, tester.root.next.next.value);
```

Terminal output

This time, the bug turned out to be in the condition when dealing with appending to a linked list with more than 1 existing element. Inside the while loop, whenever the end was detected, a new node would be added, thus creating the infinite loop. The solution was to move the creation of a new node outside the while loop:

```
public void append(int value) {
    if(this.root == null) {
        this.root = new Node(value, null);
        return;
    }
    // If it's just one element, add if after that one
    Node n = this.root;
    if(n.next == null) {
        n.next = new Node(value, null);
        return;
    }
    // Otherwise, loop until the end and add at the end with a null
    while(n.next != null) {
        n = n.next;
        n.next = new Node(value, null);
    }
}
```

```
public void append(int value) {
    if(this.root == null) {
        this.root = new Node(value, null);
        return;
    }
}
```

```
// If it's just one element, add if after that one
Node n = this.root;
if(n.next == null) {
    n.next = new Node(value, null);
    return;
}
// Otherwise, loop until the end and add at the end with a null
while(n.next != null) {
    n = n.next;
}
n.next = new Node(value, null);
}
```