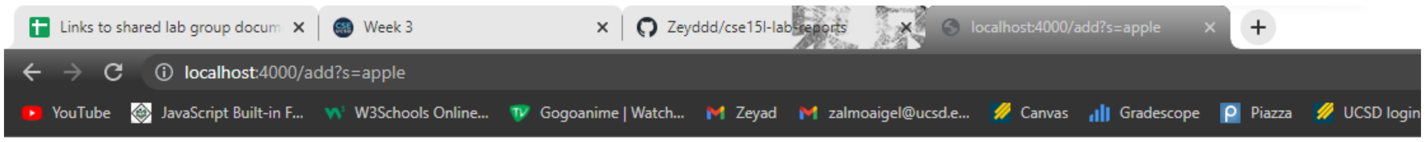
cse15l-lab-reports

Part 1

Simplest search engine code

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


class Handler implements URLHandler {
    // The one bit of state on the server: a number that will be manipulated by
    // various requests.
    ArrayList<String> strArray = new ArrayList<>();

    public String handleRequest(URI url) {
        if (url.getPath().equals("/")) {
            return String.format("Search either 'add?s' or 'search?s'");
        } else if (url.getPath().equals("/add")) {
            String[] parameters = url.getQuery().split("=");
            strArray.add(parameters[1]);
            return String.format(parameters[1] + " added!");
        } else {
            if (url.getPath().contains("/search")) {
                ArrayList<String> output = new ArrayList<>();
                String[] parameters = url.getQuery().split("=");
                for(int i = 0; i < strArray.size(); i++){
                    if(strArray.get(i).contains(parameters[1])){
                        output.add(strArray.get(i));
                    }
                }
                return output.toString();

                }
            }
            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());
    }
}
```

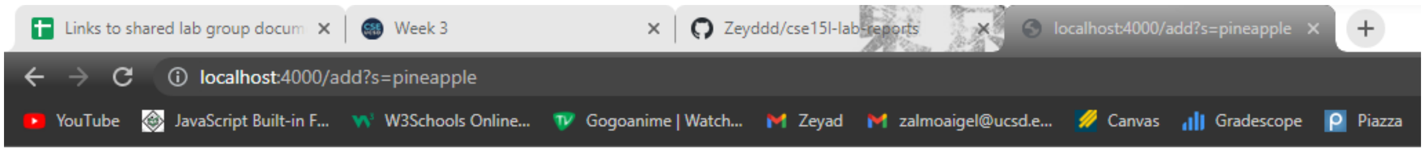
Adding to array



localhost:4000/add?s=apple

```
apple added!
```

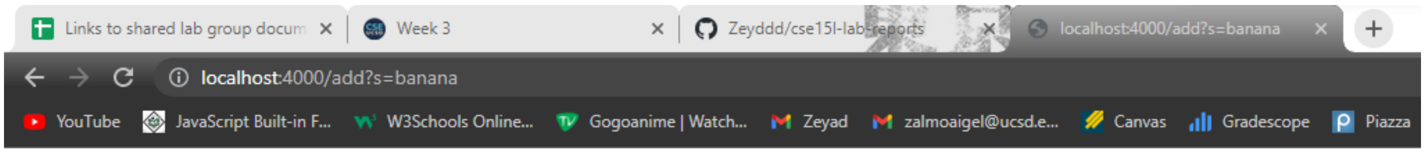
In this image, we are adding *apple* to the array. First, the code checks the path of the url. If it see's `/add` , it then runs another series of commands, which splits the query into an array, split by the equals sign =. After that it adds the second index of the array, in this case being apple. This is because the first value of the array will always be s, as I have coded this to have a dummy value in the beginning to avoid errors.



localhost:4000/add?s=pineapple

```
pineapple added!
```

In this image, instead of *apple*, I'm adding *pineapple*. The code goes through the exact same process as apple.

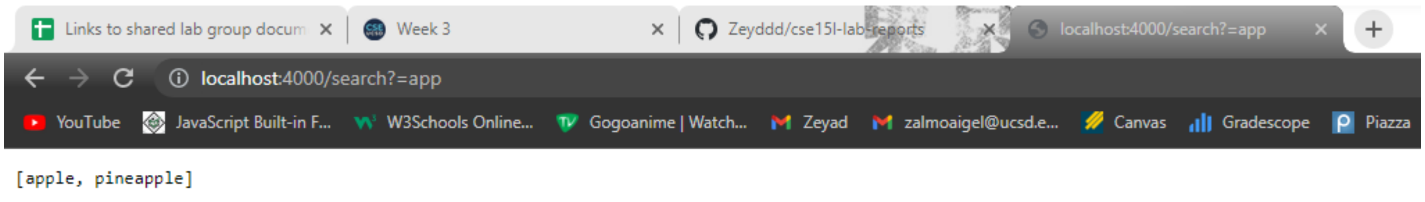


localhost:4000/add?s=banana

```
banana added!
```

In this image, instead of *pineapple*, I'm adding *banana*. The code goes through the exact same process as apple.

Searching the array



```
[apple, pineapple]
```

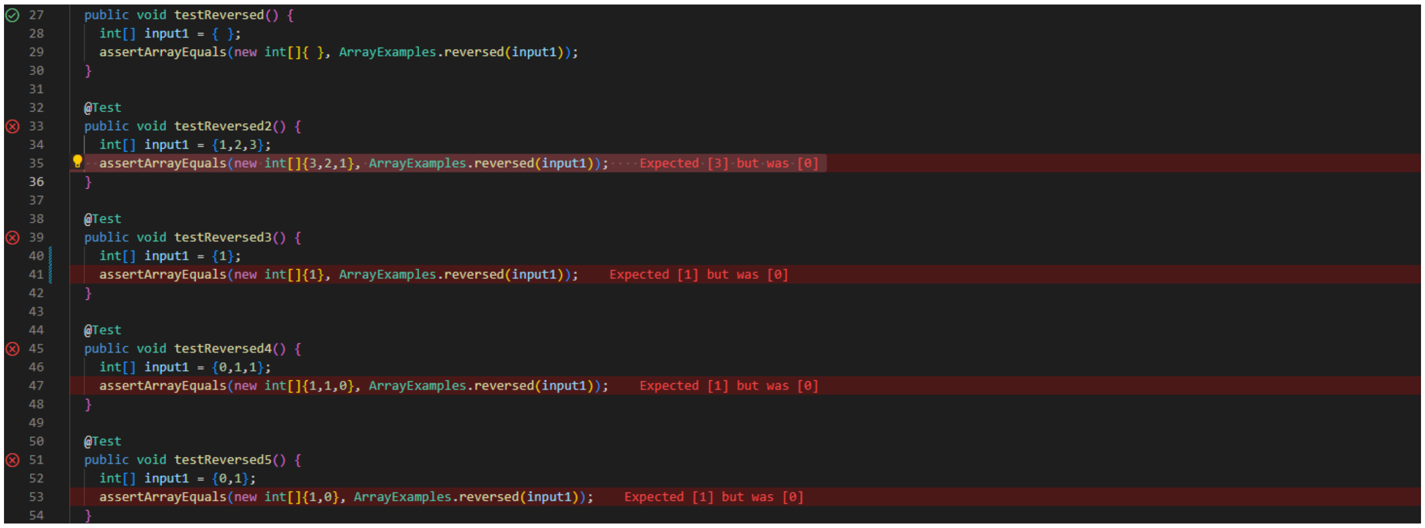
Here, we see that instead of adding something, I'm searching for a string. Just like Chrome, where you can use `CTRL + F` and search for things, either in its full name or part of it, my search method does the same thing. The code checks if the path is `/search`, and then splits the query into an array, just like the with `/add`. After that, it grabs the second index of the array, which is our search string, and compares it to every value in the original array, where we stored our added values, and checks to see if every index of that array contains our search. If it does, we add that value to a new array, and once we have gone through the entirety of the original array, we output the new array.

Part 2

Bug in array reversed

In *reversed*, no matter what values you put into the array to reverse, the ouput would always be an array of the same length just filled with zero's.

Below is an example screenshot of the symptom and test code



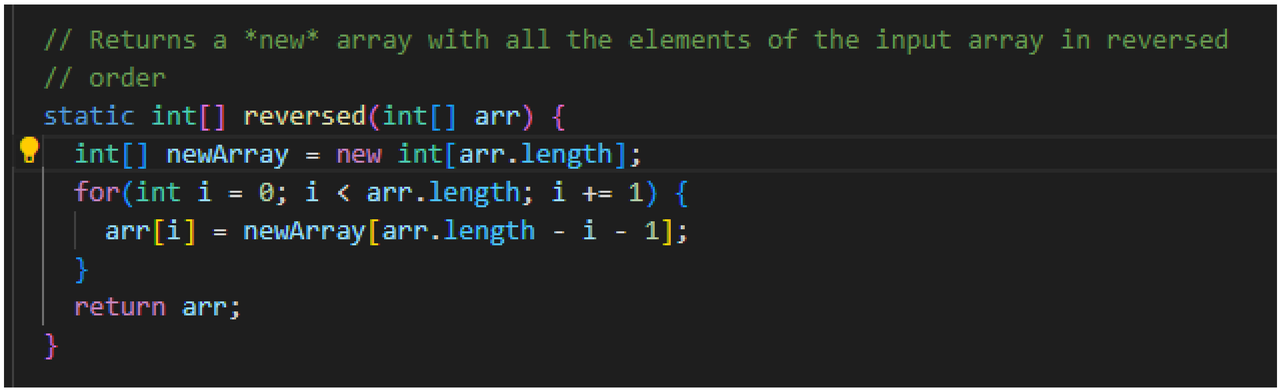
```
public void testReversed() {
  int[] input1 = { };
  assertArrayEquals(new int[]{ }, ArrayExamples.reversed(input1));
}

@Test
public void testReversed2() {
  int[] input1 = {1,2,3};
  assertArrayEquals(new int[]{3,2,1}, ArrayExamples.reversed(input1));    Expected [3] but was [0]
}

@Test
public void testReversed3() {
  int[] input1 = {1};
  assertArrayEquals(new int[]{1}, ArrayExamples.reversed(input1));    Expected [1] but was [0]
}

@Test
public void testReversed4() {
  int[] input1 = {0,1,1};
  assertArrayEquals(new int[]{1,1,0}, ArrayExamples.reversed(input1));    Expected [1] but was [0]
}

@Test
public void testReversed5() {
  int[] input1 = {0,1};
  assertArrayEquals(new int[]{1,0}, ArrayExamples.reversed(input1));    Expected [1] but was [0]
}
```

Here, we see all but one test's failing, all showing the same symptom. The reason the first test doesn't fail is due to it being empty. The other tests all fail as the array becomes filled with zero's.

Fix the bug

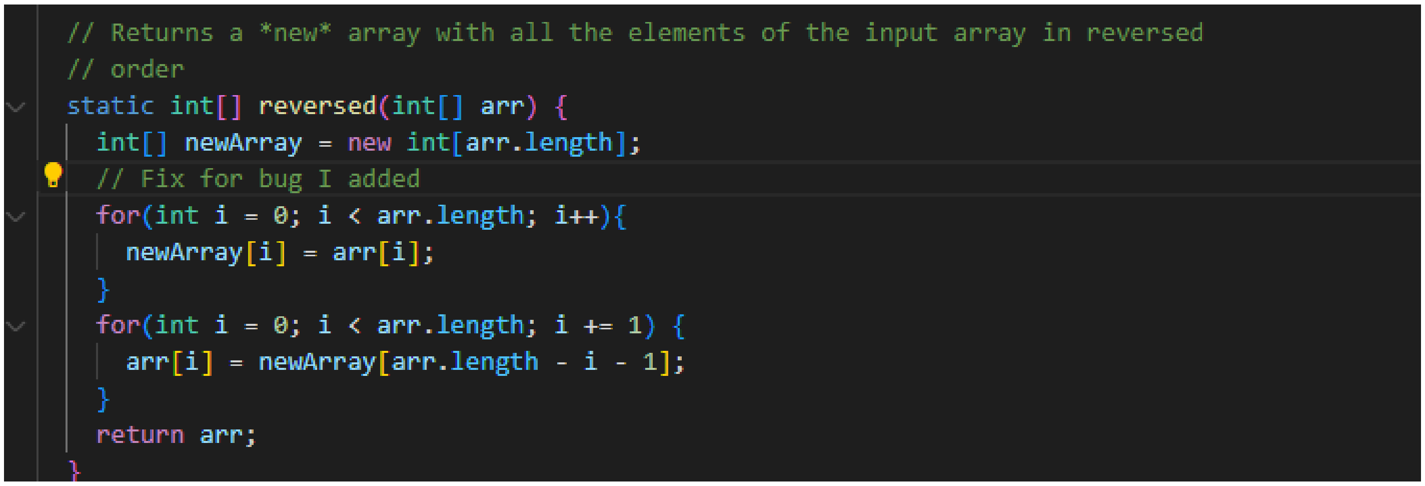
In order to fix the bug, I first identifed that the issue was that the new array used to reverse the old array was never filled with the old array's values. So, I added a for-loop to fill in the new array with the old array's values.

Before



```
// Returns a *new* array with all the elements of the input array in reversed
// order
static int[] reversed(int[] arr) {
  int[] newArray = new int[arr.length];
  for(int i = 0; i < arr.length; i += 1) {
    arr[i] = newArray[arr.length - i - 1];
  }
  return arr;
}
```

After

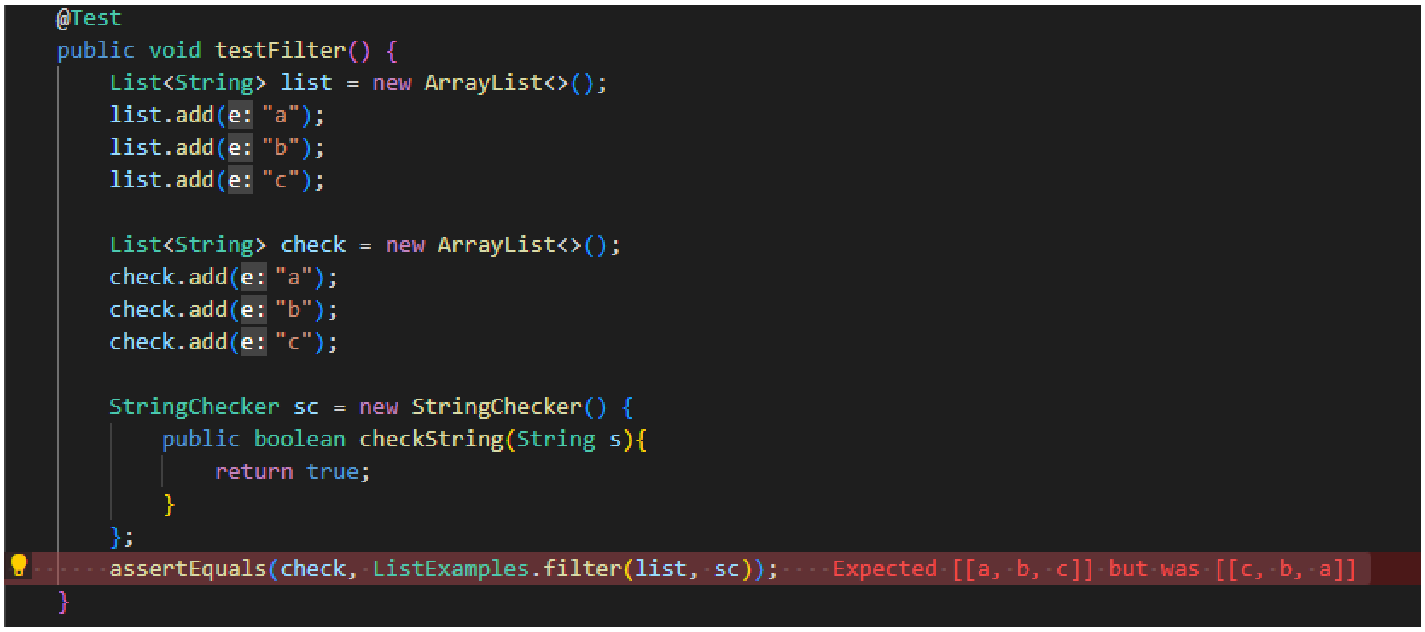


```
// Returns a *new* array with all the elements of the input array in reversed
// order
static int[] reversed(int[] arr) {
  int[] newArray = new int[arr.length];
  // Fix for bug I added
  for(int i = 0; i < arr.length; i++){
    newArray[i] = arr[i];
  }
  for(int i = 0; i < arr.length; i += 1) {
    arr[i] = newArray[arr.length - i - 1];
  }
  return arr;
}
```

Bug in List filer

In *filter*, the symptom is that the returned array is always reversed, no matter the filtered results

Below is an example screenshot of the symptom and test code

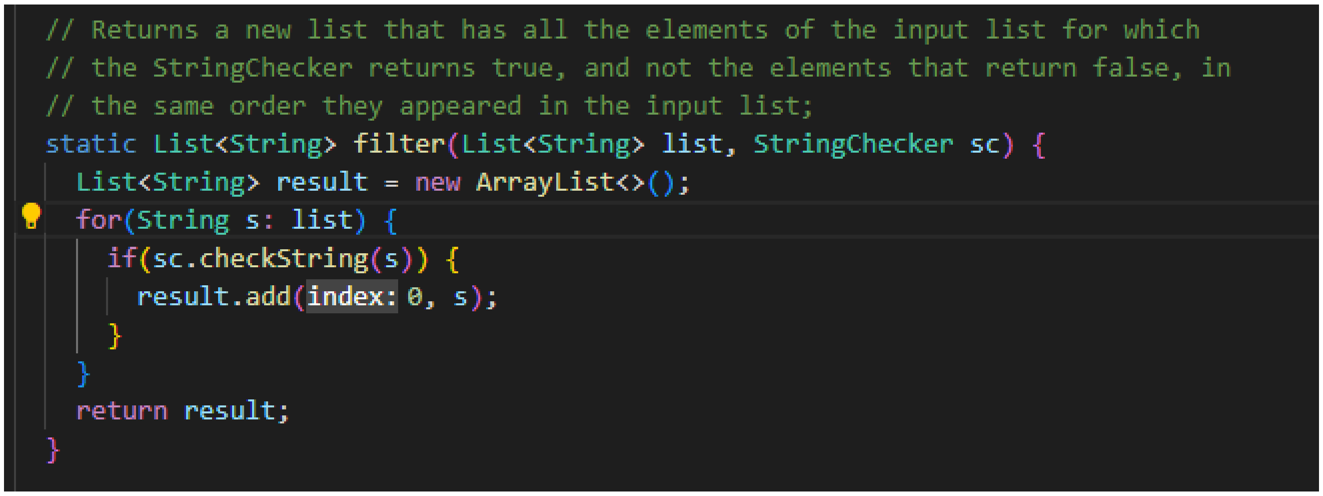


```
@Test
public void testFilter() {
    List<String> list = new ArrayList<>();
    list.add(e: "a");
    list.add(e: "b");
    list.add(e: "c");

    List<String> check = new ArrayList<>();
    check.add(e: "a");
    check.add(e: "b");
    check.add(e: "c");

    StringChecker sc = new StringChecker() {
        public boolean checkString(String s){
            return true;
        }
    };
    assertEquals(check, ListExamples.filter(list, sc));    Expected [[a, b, c]] but was [[c, b, a]]
}
```

Fix the bug

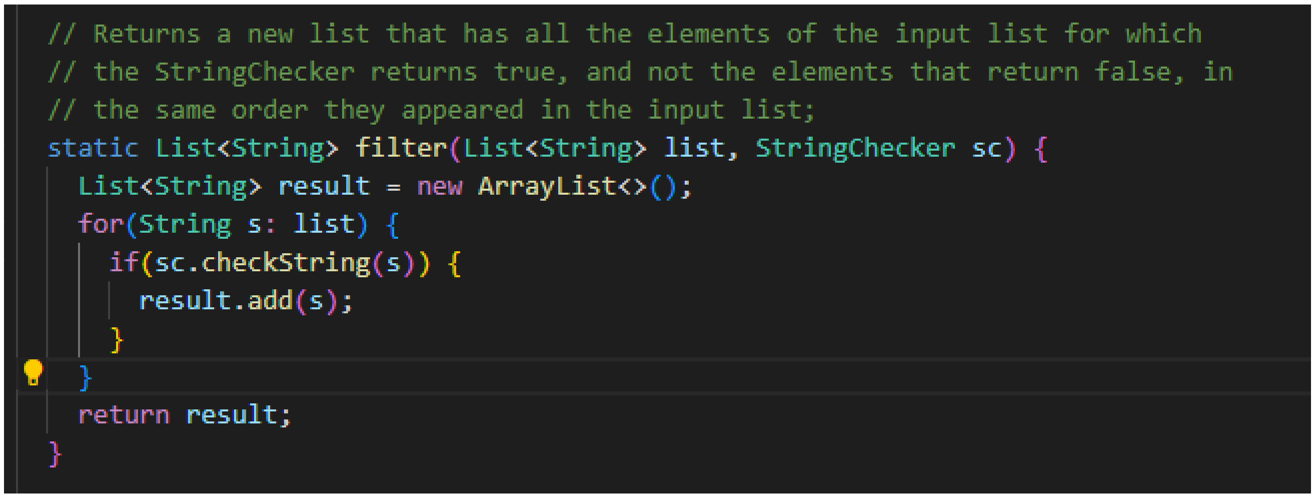
In order to fix the bug, I realised that the add method was always adding to the front of the list, so I changed it to add to the back of the list

Before



```
// Returns a new list that has all the elements of the input list for which
// the StringChecker returns true, and not the elements that return false, in
// the same order they appeared in the input list;
static List<String> filter(List<String> list, StringChecker sc) {
  List<String> result = new ArrayList<>();
  for(String s: list) {
    if(sc.checkString(s)) {
      result.add(index: 0, s);
    }
  }
  return result;
}
```

After



```java
// Returns a new list that has all the elements of the input list for which
// the StringChecker returns true, and not the elements that return false, in
// the same order they appeared in the input list;
static List<String> filter(List<String> list, StringChecker sc) {
  List<String> result = new ArrayList<>();
  for(String s: list) {
    if(sc.checkString(s)) {
      result.add(s);
    }
  }
  return result;
}
```