Longest Common Prefix Java

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "". Give java code and explain

Here's a Java function to find the longest common prefix amongst an array of strings, along with an explanation:

Java Code

```
java


public class LongestCommonPrefix {
    public static String findLongestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) {
            return ""; // Return empty string if input is null or empty
        }

        String prefix = strs[0]; // Start with the first string as the prefix

        for (int i = 1; i < strs.length; i++) {
            // Reduce the prefix by checking with the current string
            while (strs[i].indexOf(prefix) != 0) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) {
                    return ""; // Return empty string if no common prefix is found
                }
            }
        }

        return prefix; // Return the final common prefix
    }

    public static void main(String[] args) {
        String[] strs = {"flower", "flow", "flight"};
        System.out.println("Longest Common Prefix: " + findLongestCommonPrefix(strs)); // Ou
tput: "fl"

        String[] strs2 = {"dog", "racecar", "car"};
        System.out.println("Longest Common Prefix: " + findLongestCommonPrefix(strs2)); // O
utput: ""
    }
}
```

Explanation

1. **Input Validation**:
    - If the input array is `null` or empty, the function returns an empty string since no prefix can exist.
2. **Initialization**:

- Use the first string in the array (`strs[0]`) as the initial prefix.

3. **Iterative Comparison**:
   - Loop through the rest of the strings in the array.
   - Use the `indexOf()` method to check if the current prefix is at the start (`indexOf(prefix) == 0`) of the string being checked.
   - If not, reduce the prefix by removing the last character (shorten it using `substring`).
   - Keep reducing the prefix until it matches the start of the current string or becomes empty.
4. **Empty Prefix Check**:
   - If the prefix becomes empty at any point, return `""` immediately since no common prefix exists.
5. **Return the Prefix**:
   - If the loop completes, the `prefix` contains the longest common prefix for all strings in the array.

Example Walkthrough

Input: `["flower", "flow", "flight"]`

1. Start with the first string: `prefix = "flower"`.
2. Compare with `strs[1] = "flow"`:
   - `"flower"` is not a prefix of `"flow"`.
   - Shorten `prefix` to `"flow"`.
3. Compare with `strs[2] = "flight"`:
   - `"flow"` is not a prefix of `"flight"`.
   - Shorten `prefix` to `"flo"`, then `"fl"`.
   - `"fl"` is a prefix of `"flight"`.
4. Final result: `"fl"`.

Input: `["dog", "racecar", "car"]`

1. Start with `prefix = "dog"`.
2. Compare with `strs[1] = "racecar"`:
   - `"dog"` is not a prefix of `"racecar"`.
   - Shorten `prefix` to `"do"`, then `"d"`, then `""`.
   - Return `""` as no common prefix exists.

Complexity Analysis

1. **Time Complexity**:
   - Let `n` be the number of strings and `m` be the average length of the strings.
   - The worst-case scenario involves comparing all characters in each string, leading to $O(n \times m)$.
2. **Space Complexity**:
   - The space complexity is $O(1)$ as no extra space is used apart from the prefix string.

This approach ensures efficiency and clarity for the problem.