Sousanna Chugunova

https://www.simplilearn.com/tutorials/java-tutorial/queue-in-java

The article "An Introduction to Queue in Java with Example" provides an in-depth overview of the Queue data structure in Java, emphasizing its first-in-first-out (FIFO) order. The article mentions using Java.Util.Queue interface which supports multiple operations, including insertion, inspection, and deletion of elements.

Queues in Java are useful for managing ordered groups of elements with FIFO behavior. They are suitable for scenarios where data does not require real-time transfers. Java provides multiple implementations of the Queue interface, each with unique characteristics and suitable for different scenarios.

Additionally, the article mentions different types of queues. The LinkedList allows for elements to be added and removed from both ends. Another queue type is a Priority Queue which is useful for scenarios where elements need to be processed based on priority rather than the order of insertion. Overall, this article did an excellent job of describing queues and providing coding examples to cement the knowledge.

Challenges Encountered

Managing edge cases, such as empty stacks or unexpected input, required careful consideration to ensure the program behaved correctly under all scenarios. Also, devising an algorithm to count characters using stack operations instead of straightforward loop proved challenging. I solved these problems by doing thorough research about edge cases and algorithms from numerous articles and notes provided to us to gain a deeper understanding of stack functionality and problem-solving skills.

Observations Gained

-Working on this project helped solidify my understanding of the Stack Abstract Data Type (ADT) and its fundamental operations like push, pop, and peek.
- This project improved my ability to think algorithmically, by requiring me to make a method to count characters using stack operations rather than straightforward iteration.

Additional Shares

Undersatanding how to use stack operations to count elements in a string instead of straightforward iteration was a bit challenging for someone (me) who is still new to algorithmic thinking.

Probing Questions for Classmates

- How would you check if a stack is empty?
- How would you compare the time complexities of two algorithms to determine which is more efficient?

Code

```
/*
 * Student name: Sousanna Chugunova
 * Class: CMSC204 (CRN 40439)
 * Instructor: Dr. Thai
 * Platform/compiler: Eclipse
 *
 * I acknowledge that this is my own work, and does not include
 * source code from the internet or from any other student.
 */

package discussion_S2;
import java.util.Scanner;
import java.util.Stack;

public class binaryCounter {

    // Method to analyze a binary string
    public static void analyzeBinaryString(String binaryString) {
        // Create a stack to store characters
        Stack<Character> stack = new Stack<>();

        // Push each character of the binary string onto the stack
        for (char ch : binaryString.toCharArray()) {
            stack.push(ch);
        }

        // Initialize counters for 0s and 1s
        int count0 = 0;
        int count1 = 0;

        // Pop characters from the stack and count 0s and 1s
        while (!stack.isEmpty()) {
            char ch = stack.pop();
            if (ch == '0') {
                count0++;
            } else if (ch == '1') {
                count1++;
            }
        }

        // Output the counts of 0s and 1s
        System.out.println("Total number of 0s: " + count0);
        System.out.println("Total number of 1s: " + count1);

        // Determine which value occurs more frequently and by how much
        if (count0 > count1) {
            System.out.println("0 occurs more frequently by " + (count0 - count1) +
" times.");
```

```
        } else if (count1 > count0) {
            System.out.println("1 occurs more frequently by " + (count1 - count0) +
" times.");
        } else {
            System.out.println("0 and 1 occur with equal frequency.");
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean tryAgain = true;

        while (tryAgain) {
            System.out.println("Enter a binary string (0s and 1s only):");
            String binaryString = scanner.nextLine();

            // Analyze the binary string
            analyzeBinaryString(binaryString);

            // Ask user if they want to try again
            System.out.println("Do you want to try again? (yes/no):");
            String choice = scanner.nextLine().trim().toLowerCase();
            if (!choice.equals("yes")) {
                tryAgain = false;
            }
        }

        // Say goodbye
        System.out.println("Goodbye!");
        scanner.close();
    }
}
```

```
Enter a binary string (0s and 1s only):
0101010101011110000110011
Total number of 0s: 12
Total number of 1s: 13
1 occurs more frequently by 1 times.
Do you want to try again? (yes/no):
yes
Enter a binary string (0s and 1s only):
01010101010101
Total number of 0s: 7
Total number of 1s: 7
0 and 1 occur with equal frequency.
Do you want to try again? (yes/no):
no
Goodbye!
```