Day 8: Palindrome Check

Abhinav Yadav

"Programming isn't about what you know; it's about what you can figure out."
— Chris Pine

---

1 Introduction

A palindrome is a string that reads the same forward and backward, ignoring case and spaces. Checking for palindromes is a fundamental problem in string manipulation and helps build strong foundations in programming. This document provides an efficient approach to solve the problem.

2 Problem Statement

Problem: Check if a given string is a palindrome (case-insensitive). **Hint:** Compare characters from both ends using a loop. **Edge Case:** Handle spaces and special characters gracefully.

3 Algorithm

3.1 Steps to Solve the Problem

1. Normalize the string:
    - Convert all characters to lowercase.
    - Remove spaces and special characters to focus only on alphanumeric characters.
2. Use two-pointer technique:
    - Initialize two pointers, one at the beginning and one at the end of the string.
    - Compare characters pointed by the two pointers.
    - Move pointers inward if they match.
3. If all characters match, the string is a palindrome.

4 Code

```java
import java.util.Scanner;

public class PalindromeChecker {

    // Function to normalize the string by converting to lowercase and rem
    public static String normalizeString(String str) {
        StringBuilder normalized = new StringBuilder();
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                normalized.append(Character.toLowerCase(ch));
            }
        }
        return normalized.toString();
    }

    // Function to check if a string is a palindrome
    public static boolean isPalindrome(String str) {
        int left = 0, right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false; // Not a palindrome
            }
            left++;
            right--;
        }

        return true; // Palindrome
    }

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

        System.out.print("Enter␣a␣string:␣");
        String input = scanner.nextLine(); // Reading input from the user

        // Normalize the string
        String normalized = normalizeString(input);

        // Check if the normalized string is a palindrome
        if (isPalindrome(normalized)) {
            System.out.println("The␣given␣string␣is␣a␣palindrome.");
        } else {
            System.out.println("The␣given␣string␣is␣not␣a␣palindrome.");
        }
        scanner.close();
    }
}
```

5 Step-by-Step Explanation

1. **Normalize the string:**
   - Use `tolower()` to convert characters to lowercase.
   - Use `isalnum()` to filter out spaces and special characters.
2. **Check for palindrome:**
   - Use two pointers to compare characters from the start and end of the normalized string.
   - If all characters match, the string is a palindrome.

6 Complexity Analysis

- **Time Complexity:** $O(n)$ Normalizing the string and checking for a palindrome each require a single traversal of the string.
- **Space Complexity:** $O(n)$ Additional space is used for the normalized string.

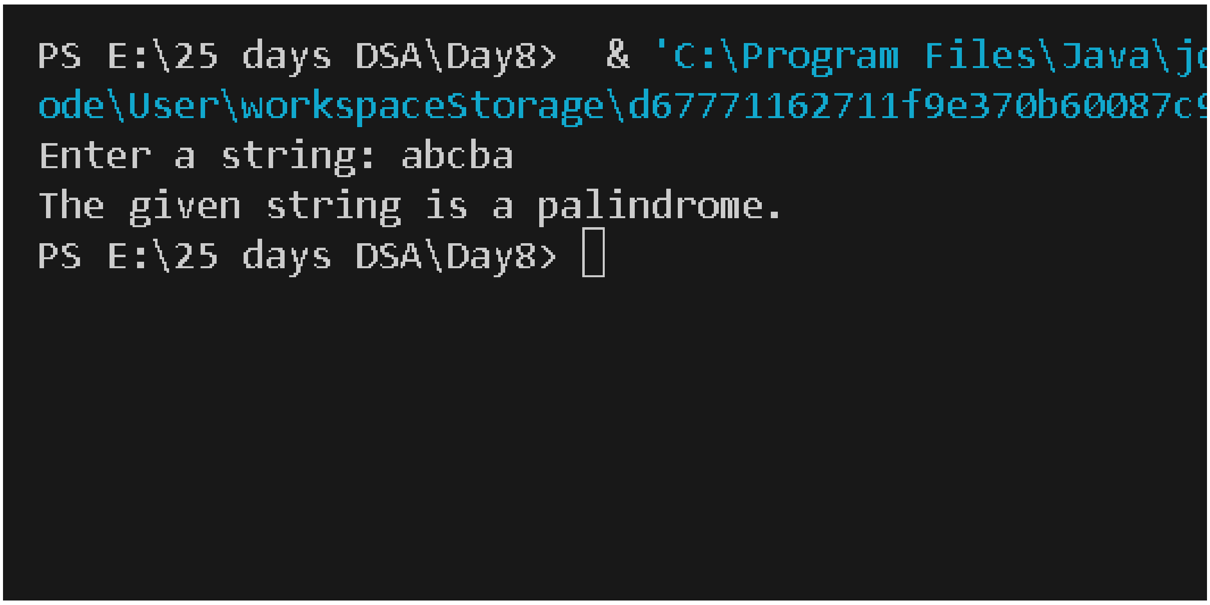
7 Examples and Edge Cases

Input String	Normalized String	Output
”A man, a plan, a canal, Panama”	”amanaplanacanalpanama”	Palindrome
”racecar”	”racecar”	Palindrome
”hello”	”hello”	Not a Palindrome
””	””	Palindrome (empty string)
”12321”	”12321”	Palindrome

8 Conclusion

This program efficiently checks for palindromes by normalizing the string and using the two-pointer technique. Handling case sensitivity, spaces, and special characters ensures robustness. This approach has a time complexity of $O(n)$, making it optimal for this problem.

9 Output



```
PS E:\25 days DSA\Day8>  & 'C:\Program Files\Java\j
ode\User\workspaceStorage\d67771162711f9e370b60087c
Enter a string: abcba
The given string is a palindrome.
PS E:\25 days DSA\Day8> 
```

Figure 1: Output