Code Review Stack Exchange is a question and answer site for peer programmer code reviews. It only takes a minute to sign up.

Sign up to join this community

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

CODE REVIEW

Palindrome-testing Java program for an interview

Asked 3 years, 11 months ago Active 2 years, 11 months ago Viewed 11k times

12

2

In an interview I was asked to write a function that would take an input word and return true if the word is a palindrome. At first I used an approach using `StringBuilder` but the interviewer said that wasn't allowed and to use a `for` loop instead. I scored 7/9 so I'm guessing it's possible to improve this:

```
public static boolean isPalinedrome(String word) {

    for(int i = 0; i < word.length(); i++) {
        if(word.charAt(i) != word.charAt(word.length() - 1 - i)) {
            return false;
        }
    }
    return true;
}
```

I added the following to the `main()` section to test

```
String input = "racecar";
if(isPalinedrome(input)) {
    System.out.println(input + " is a palinedrome");
} else {
    System.out.println(input + " is not a palinedrome");
}
```

The way I test it seems to be ugly. Is there a standard way to test a new function? I guess if you're in an advanced enough environment you would have test cases for JUnit to run against it, but anything simpler that could be done in an interview with a cloud IDE?

java interview-questions palindrome

Share Improve this question Follow

edited May 23 '17 at 7:13
200_success
140k 21 182 461

asked May 23 '17 at 6:01
smartname1
131 1 1 5

4 Before you start writing code you need to define what the requirements are. Is it just suppose to just handle ASCII characters or does the palindrome checker need to handle anything more complicated like combining diacritics (and, if so, is it just looking for equality for a naive string reversal or is it expecting the diacritics to still be applied to the same character when reversed)? – MT0 May 23 '17 at 10:23

2 Your test isn't a test. You shouldn't test if `"racecar"` is a palindrome : you know it is. You should check that `isPalindrome("racecar")` returns `true` . If not, raise an exception or display a big fat error message. – Eric Duminil May 23 '17 at 19:04

1 They might be checking for spelling; is there a reason it's called `isPalinedrome()` ? – Milo P May 23 '17 at 20:22

Without knowing the "expense" in Java of reversing an array, would it be a valid test to simply convert the string to an array, and reverse it to another array and use an array compare? – Adrian Hum May 24 '17 at 2:58

4 Answers

Active Oldest Votes

20

It's enough to loop until `word.length() / 2` , as this will compare the first half with the second half, so no need to go until the end.

As you use `word.length()` multiple times, you could extract it to a helper variable.

There is a typo in the method name.

As for testing, yes, JUnit is the way to go. In a cloud IDE, you could create a helper method that takes a single string, calls `isPalindrome` and prints the result. That way you can test multiple cases easily, by adding one line per case. It's important to try to cover corner cases and potentially interesting cases, not just the "happy path". For example:

- palindrome with even length
- palindrome with odd length
- single letter
- empty string
- non-palindromes

During an interview, it might also be worth mentioning the trade-off between comparing characters using `.charAt`, or using the array of characters returned by `.toCharArray`.

Share Improve this answer Follow edited May 23 '17 at 12:15 answered May 23 '17 at 6:40

janos
106k 14 144 370

1 I'd also include a null check, unless the interviewer had said not to. – DaveyDaveDave May 23 '17 at 11:22

3 @DaveyDaveDave thanks, I added a paragraph at the end about the trade-off between `.charAt` and `toCharArray`. (The actual details of that left as an exercise for the reader ;-) – janos May 23 '17 at 12:16

1 I would add to check for char spacing also.. I know it specifies `word` however `nurses run` is considered a palindrome also.. /pedant – Pogrindis May 23 '17 at 14:27

4 Don't check for `null` unless for some reason you want to return true or false for it. If someone passes you a null reference, just let the NPE be thrown, cause neither return value makes much sense. – cHao May 23 '17 at 16:44

2 @cHao I **strongly** disagree. Failing early is safer and easier to debug that letting an NPE propagate from wherever it happens to. It is also likely, in the general case, that passing `null` might work *sometimes* due to control flow. Modern Java practices recommend checking all method argument invariants and throwing if something is wrong. I would recommend against your suggestion in the strongest possible way. – Boris the Spider May 24 '17 at 7:06

2

JUnit tests would be a rather nice way to go here, as you correctly assume. It does not take much:

- import `org.junit.Test`
- statically import your asserts (e.g. `org.junit.Assert.assertEquals`)
- annotate your test with `@Test`

and you're good to go.

Pity your interviewer went for the `for` loop, palindromes can be nicely solved by recursion (pointing out the performance and memory usage drawbacks, of course..)

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class PalindromeTesterClass {

    @Test
    public void shouldRecognizeNull() {
        assertEquals(false, PalindromeTesterClass.isPalindrome(null));
    }

    @Test
    public void shouldRecognizeEmptyString() {
        assertEquals(true, PalindromeTesterClass.isPalindrome(""));
```

```
    }

    @Test
    public void shouldRecognizeOneCharacterPalindrome() {
        assertEquals(true, PalindromeTesterClass.isPalindrome("a"));
    }

    @Test
    public void shouldRecognizeTwoCharacterPalindrome() {
        assertEquals(true, PalindromeTesterClass.isPalindrome("aa"));
    }

    @Test
    public void shouldRecognizeTwoCharacterNonPalindrome() {
        assertEquals(false, PalindromeTesterClass.isPalindrome("ab"));
    }

    @Test
    public void shouldRecognizePalindrome() {
        assertEquals(true,
PalindromeTesterClass.isPalindrome("amanaplanacanalpanama"));
    }

    @Test
    public void shouldRecognizeNonPalindrome() {
        assertEquals(false, PalindromeTesterClass.isPalindrome("noPalindrome"));
    }


    public static boolean isPalindrome(String word) {
        if (word == null) {
            // assuming a null value is no palindrome
            return false;
        } else if (word.length() < 2) {
            // assuming both "" and "x" are palindromes
            return true;
        } else {
            // a word is a palindrome if it starts and ends in the same letter..
            if (!word.endsWith(word.substring(0, 1))) {
                return false;
            }
            // .. and everything in between the first and the last letter is a
palindrome
            return isPalindrome(word.substring(1, word.length() - 1));
        }
    }
}
```

Share Improve this answer Follow

answered May 23 '17 at 16:01

glissi

241 2 6

3 In this case I'd use `assertTrue` , in my opinion `assertTrue(PalindromeTesterClass.isPalindrome("aba"));` is more readable and coincise. – user131519 May 23 '17 at 20:31

3 I really don't see why this recursion approach would be nicer than a simple for loop. It's wordier, slower, and (in my opinion) less intuitive. Tests look fine though. – tomsmeding May 24 '17 at 10:33

At the very least, you should compare *code points*, not `char`s. In Java, char doesn't necessarily

represent a whole character, so for out-of-order comparison it won't make sense.

1

Share Improve this answer Follow

answered May 23 '17 at 12:54

Display Name
171 6

---

fastest solution

0

```
public static boolean isPalindrome(String input) {
   int n = input.length();
   for (int i = 0; i < (n / 2); ++i) {
       if (input.charAt(i) != input.charAt(n - i - 1)) {
           return false;
       }
   }
   return true;
}
```

Share Improve this answer Follow

answered Aug 23 '17 at 10:57

Vazgen Torosyan
109 2
