```
class Solution {
    public boolean isMatch(String s, String p) {
        return matchHelper(s.toCharArray(), 0, p.toCharArray(), 0);
    }

    private boolean matchHelper(char[] str1, int idx1, char[] str2, int idx2) {
        if (idx2 == str2.length) {
            return idx1 == str1.length;
        }

        if (idx2 < str2.length - 1 && str2[idx2+1] == '*') {
            if (idx1 < str1.length && (str2[idx2] == str1[idx1] || str2[idx2] == '.')) {
                if (matchHelper(str1, idx1+1, str2, idx2)) {
                    return true;
                }
            }

            return matchHelper(str1, idx1, str2, idx2+2);
        }

        if (idx1 < str1.length && str1[idx1] == str2[idx2] || str2[idx2] == '.') {
            return matchHelper(str1, idx1+1, str2, idx2+1);
        }
        return false;
    }
}
```

Given an input string (`s`) and a pattern (`p`), implement regular expression matching with support for `'.'` and `'*'`.

```
'.' Matches any single character.
'*' Matches zero or more of the preceding element.
```

The matching should cover the **entire** input string (not partial).

Note:

- `s` could be empty and contains only lowercase letters `a-z`.
- `p` could be empty and contains only lowercase letters `a-z`, and characters like `.` or `*`.

```
public class Solution {
    public boolean isMatch(String s, String p) {
        int lens = s.length(), lenp = p.length();
        boolean[][] res = new boolean[lens+1][lenp+1];
        res[0][0] = true;

        for (int i = 0; i < lenp; i++) {
            if (p.charAt(i) == '*' && i > 0 && res[0][i-1])
                res[0][i+1] = true;
        }

        for (int i = 0; i < lens; i++) {
            for (int j = 0; j < lenp; j++) {
                if (s.charAt(i) == p.charAt(j) || p.charAt(j) == '.'){
                    res[i+1][j+1] = res[i][j];
                }
                if (p.charAt(j) == '*'){
                    if (j > 0 && s.charAt(i) != p.charAt(j-1) && p.charAt(j-1) != '.') {
                        res[i+1][j+1] = res[i+1][j-1];
                    } else {
                        res[i+1][j+1] = (j > 0 ? res[i+1][j-1] : false) || res[i+1][j] || res[i][j+1];

                                        // a* becomes empty       a* becomes a    a* becomes multiple a
(expand a)
                    }
                }
            }
        }

        return res[lens][lenp];
    }
}
```